AN EFFICIENT ALGORITHM FOR THE EXTRACTION OF CONTOURS AND CURVATURE SCALE SPACE ON SIMD-POWERED SMART CAMERAS

Paul J. Shin[1]*, Xinting Gao*[2]*, Richard Kleihorst*[2]*, Johnny Park*[1]*, and Avinash C. Kak*[1]

[1]School of Electrical and Computer Engineering, Purdue University
{paulshin, jpark, kak}@purdue.edu
[2] NXP Semiconductors, Corporate I&T / Research
{xinting.gao, richard.kleihorst}@nxp.com

ABSTRACT

SIMD (Single Instruction Multiple Data) processors have been shown to possess high capability for *vector*-based image processing due to their massively parallel architecture. However, it is not always easy to map the general-purpose processor implementations of high-level vision algorithms to such processors due to the hardware-imposed characteristics of SIMD processors. In this paper, our focus is on contour extraction algorithms for such processors. We present a novel real-time memory-efficient contour extraction algorithm that is suitable for the SIMD processor used in the WiCa camera developed by NXP. A further goal of this paper is to present a method for extracting the curvature scale space (CSS) on the same SIMD processor. For contour extraction, the detected contour points are first stored as they are detected, and then effectively reordered with low memory overhead. For extracting the CSS, we introduce a high-precision Gaussian filtering scheme using an extended number representation that is particularly suited for the SIMD processor of the camera. We demonstrate both in real time.

Index Terms— Contour extraction, contour tracing, curvature scale space, SIMD, smart cameras

1. INTRODUCTION

Wireless camera networks are expected to play a significant role in smart interior and exterior spaces of the future. For example, by tracking people in the environment, a camera network could become aware of the fact that an individual has slumped on the floor and is in need of medical help.

A camera network works best if the computing required for the interpretation of the images can be carried out locally at each node or by a neighboring set of nodes. This requires that the computer vision algorithms developed for general-purpose processors be mapped to the processors embedded in the individual camera units. Of course, when a vision problem must be solved by a neighborhood of cameras, the mapping must take the form of a distributed implementation in the presence of possibly lossy communications. Many such mappings have already been proposed for the case when the cameras use SIMD processors for fast image-based calculations. See, for example, [1] for the case of face recognition, [2] for the case of depth estimation with stereo, and [3] for gesture recognition. When such mappings are carried out on SIMD-powered (Single Instruction Multiple Data) smart cameras, such as the WiCa [4] developed by NXP Research (formerly Philips Research), one benefits from their massively parallel architecture to obtain high-speed implementations that, as it turns out, are also energy efficient.

The goal of this paper is to add to the vision algorithms that have so far been implemented on SIMD processor of the WiCa camera. In particular, we will demonstrate an SIMD implementation for extracting boundary contours. Contour extraction plays an important role in many computer vision algorithms as shape-based characterizations of objects are derived from the boundary contours. Such characterizations include Fourier descriptors [5], wavelet descriptors [6], the shape context [7], and the curvature scale space [8]. The applications of contour-based techniques including snakes [9] span video surveillance, medical image processing, and pattern recognition [10, 11, 12, 13]. In all these applications, contour extraction consists roughly of object segmentation followed by contour tracing.

A straightforward mapping of a contour extraction algorithm intended for a general-purpose processor to an SIMD processor goes against the grain of the SIMD architecture employed in the WiCa. For example, the WiCa does not make it easy to access the individual pixels of an image since only the ends of video scan lines are directly available, such lines being the basic data blocks that are processed in parallel. Thus, tracing a contour using either 4- or 8-neighborhood based algorithms is inefficient for such processors since such algorithms need direct access to the pixel data in the middle of the line memories. Thus the chain-code based contour tracing algorithms [14] are not suitable for the WiCa. Another challenge posed by the WiCa is the limited precision of the numbers that can be represented in the line memories. This

978-1-4244-2665-2/08/$25.00

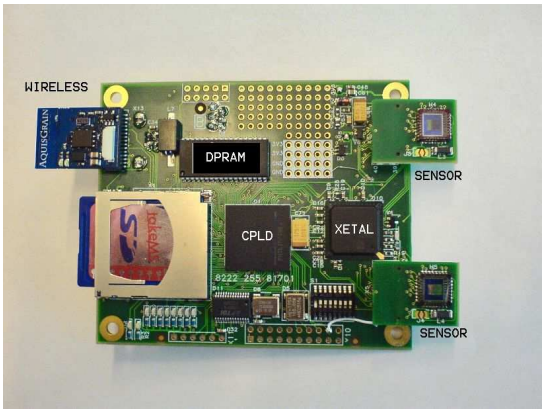


Fig. 1. Architecture of the WiCa 1.1 wireless smart camera.

limited precision creates difficulties for implementing linear filtering operations needed for creating curvature scale-space (CSS) representations.

Against a background of the above-mentioned challenges posed by the SIMD architecture of the WiCa, this paper presents a novel memory-efficient contour extraction and curvature scale space (CSS) algorithm for the WiCa. The contour extraction algorithm requires only six line memories for all of its four steps: object segmentation, contour tracing, contour reordering, and contour normalization. It does not require for the whole image to be stored. Only the detected contour points are stored in four line memories: two of the line memories are used for the x and y coordinates, one for storing the total number of boundary points in a video line which directs the point movement in vertical direction, and one for storing the horizontal direction of the next point movement. After detecting all the boundary points, they are reordered in such a way that they form a continuous counterclockwise contour of the object. The reordered x- and y-coordinates are stored in the remaining two line memories.

This paper also presents a high-precision Gaussian filtering algorithm for extracting scale-space representations. It is based on an 18-bit number representation using two line memories; note that each element in the line memory in the SIMD processor of the WiCa consists of 10 bits. Based on this extended number representation, a high-precision arithmetic operation scheme is introduced, which plays a crucial role in accurate filtering. This allows for contour smoothing to be carried out iteratively with high precision.

The paper is organized as follows: Section 2 describes the hardware specification and the configuration of the WiCa. The new contour extraction algorithm is presented in Section 3. Section 4 presents the material related to the curvature scale space. We demonstrate our algorithms in Section 5 and give concluding remarks in Section 6.

2. SMART CAMERA HARDWARE PLATFORM

WiCa is a low-power, high-performance, and programmable smart camera platform equipped with four major components: one or two VGA color image sensors, a massively parallel SIMD processor known as IC3D, a general-purpose 8051 processor, and a Zigbee communication module, as shown in Fig.1. Both processors are connected through DPRAM (Dual-port RAM), which works as an asynchronous shared workspace.

The IC3D, one of the Philips' Xetal family of SIMD processors, consists of 5 specific internal blocks as shown in Fig.2. The video input and output processors stream 24-bit 3-channel input and output digital video signals to and from IC3D, respectively. The heart of the chip is a linear processor array (LPA) with 320 processing elements (PEs). Each PE shares with others both the memory address and the instructions in the SIMD sense and has simultaneous read and write access in one clock-cycle to the on-chip parallel memory of 64 lines of 320 positions. Each memory position consists of 10 bits. The global control processor (GCP) is a processor dedicated to performing global DSP operations, controlling the IC3D, and taking care of video synchronization while communicating with the LPA. The peak pixel performance of IC3D is about 50 GOPS at $80 \sim 100MHz$. The power consumption at the peak performance is low because of the shared instruction decoding and the ultra-wide memory access scheme.

The 8051 general-purpose processor is dedicated to performing high-level vision processing and decision making based on the data from IC3D. The 8051 processor communicates with IC3D through the DPRAM. It can share its workspace asynchronously with IC3D via the DPRAM, so that they can work in their own clock domains. The DPRAM has eight memory banks in WiCa 1.1, each of which has 64KB so that, at the maximum, one 256x256 image can be stored into one bank.

The wireless communication module of WiCa 1.1 is an Aquisgrain Zigbee module developed by Philips. It works as a wireless UART port of the system with limited capacity. The maximum data rate for wireless communications is around 10kB/sec. The data to be transmitted is sent by the 8051 processor to the communication module. The communication module is based on CC2430 transceiver from TI, which includes an additional 8051 controller for its own processing.

The Xetal chips including IC3D are programmed using an extended C (XTC) language. One of its features is a *vector* data type for the 320-element wide memory address representation in the LPA. A single line memory with its 320 data elements is processed as a single data unit.

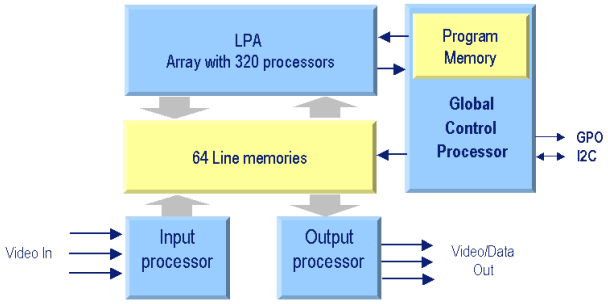


Fig. 2. Internal architecture of the IC3D, which is one of the Philips' Xetal family of SIMD processors.

3. THE PROPOSED CONTOUR EXTRACTION METHOD

Various contour extraction algorithms have been proposed for general-purpose CPUs based on pixel-level operations [15, 14, 16]. Contour extraction, commonly referred to as contour tracing, is generally carried out by finding the next point on a contour in a 4- or 8-neighborhood of the previous point. Such algorithms, however, cannot be implemented directly on the WiCa due to the unique characteristics of its hardware architecture. More specifically, the SIMD processor on the WiCa stores input data internally in line memories. As a consequence, only the first and the last values stored in a line memory can be read directly. Even if we use the DPRAM, the delay between a read request and when the data is available from the memory also makes it inefficient to read the individual pixels directly. These characteristics of the WiCa make it difficult to directly implement the conventional contour extraction algorithm designed for a general-purpose CPU. Moreover, due to the limited memory space with no dynamic memory allocation support in WiCa, complex data structures such as linked lists, which have been used in most of the existing parallel implementations of contour extraction [17, 18], cannot be employed in WiCa.

We propose an efficient contour extraction algorithm suitable for SIMD-powered architectures with limited memory space. The proposed algorithm entails three steps, in which we will describe in detail in the following subsections. We assume that the object of interest has been detected and its region identified using one of the available segmentation methods such as simple background subtraction, color segmentation, etc [19]. One could, for example, employ the color segmentation algorithm described in [3] that uses the k-means clustering method; this algorithm was developed specially for the WiCa.

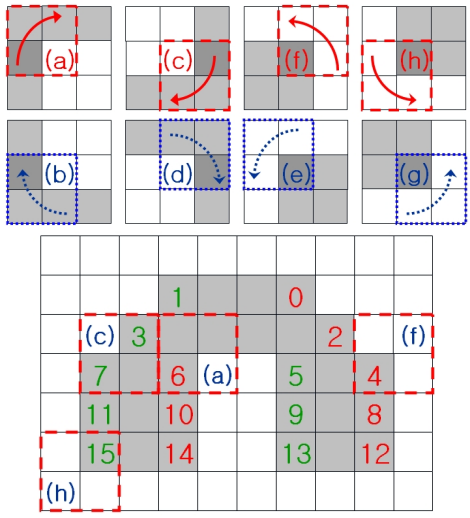
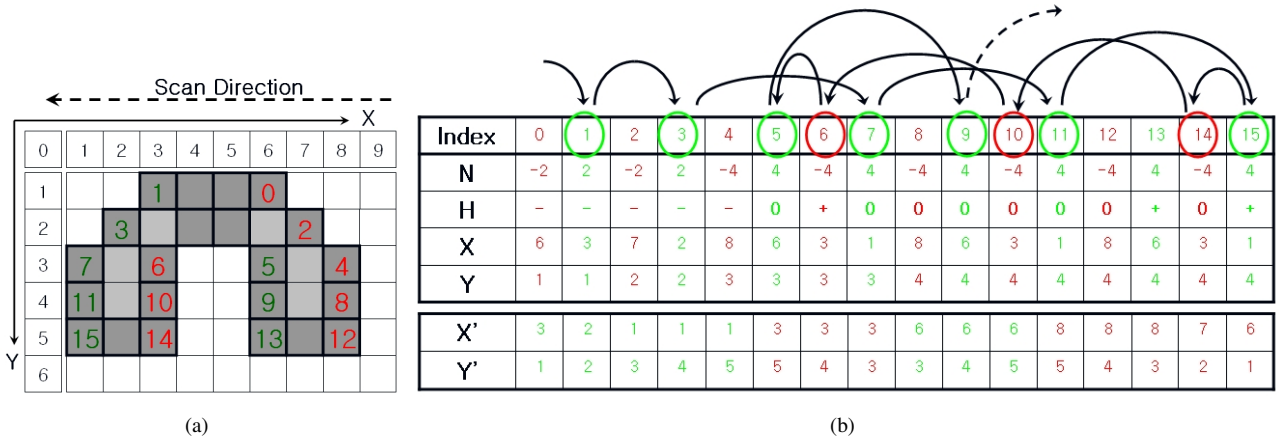


Fig. 4. The eight cases of point corners. The white pixel of the 2-by-2 mask corresponds to the background, and the black pixels the object. The darker gray pixel indicates the current point of interest. The red curved arrows in (a), (c), (f), and (h) indicate the point movement in counterclockwise direction. The x coordinates of the next contour point in (c) and (f) should be smaller than the current one, whereas those in (a) and (h) should be larger.

3.1. Detection of Left-Side and Right-Side Contour Points

Given a properly segmented region, contour detection refers to the *identification* of the outer boundary points of the region. In the proposed algorithm, instead of detecting all contour points, only the left- and the right-side contour points are initially detected. As each scan line of the image (i.e., image row) is processed in the right-to-left traversal direction, if a background pixel is followed by an object pixel, we declare the object pixel a right-side contour point. On the other hand, if an object pixel is followed by a background pixel, we declare the object pixel to be a left-side contour point. Figure 3(a) illustrates the idea of the left- and the right-side contour points where the segmented object region is represented by gray pixels, the background by white pixels, the contour points by darker gray pixels, the left-side contour points by those with green numbers, and the right-side contour points by those with red numbers. From now on, we will call the left-side contour points as *L-points*, the right-side contour points as *R-points*, and together as *LR-points*. We will loosely use the term LR-points in the sense that an LR-point could either be an L-point or an R-point. We will also assume that individual scan lines of the image are processed from top to bottom and that the pixels in each scan line is processed from right to left. The index numbers of LR-points shown in Figure 3(a) indicate the order at which each LR-point is detected.

The information of LR-points are stored in four line mem-



(a) (b)

Fig. 3. (a) A segmented object region in an image. The object region is represented by gray pixels, the background by white pixels, the contour points by darker gray pixels, the left-side contour points by those with green numbers, and the right-side contour points by those with red numbers. (b) The corresponding information of the LR-points in the object shown in (a). The information is stored in four line memories, X, Y, N, H. The arrows in the top illustrate the re-ordering procedure.The coordinates of the re-ordered LR-points are stored in the last two line memories *X'* and *Y'*.

ories that we call X, Y, N, and H in the order they are detected (see Figure 3(b)). We will call the immediate neighbor of a point in the storage as the *storage neighbor* and the neighbor along the contour the *contour neighbor*. Obviously, not all contour neighbors will be immediately connected in the image. For example, the points 1 and 2 in Figure 3 are storage neighbors and the points 3 and 7 are contour neighbors. In the next section, we will describe how we can re-order the storage so that two consecutive LR-points in the list are contour-neighbors. The line memories X and Y contain the x- and y-coordinates of the LR-points, respectively. The line memory N contains the total number of LR-points detected in the respective scan line. Note that when a scan line has an isolated object pixel, it is either because the scan line contains an extremal contour point or because the scan line cuts through the crossing point of an 8-like figure. In either case, the isolated object pixel will be counted in both the object-to-background and the background-to-object transition. Therefore, the value stored in N in this case will be 2. Including this extremal case, the value stored in N is always even. Moreover, we set the sign of the value in N to distinguish between L-points and R-points; we assign a positive sign to L-points and a negative sign to R-points as illustrated in Figure 3(b). Note that the sign of the N value also indicates the vertical direction of the next contour neighbor. The exceptions are the L-points having an R-point as the next contour neighbor and the R-points having an L-point as the next contour neighbor. With the counterclockwise contour tracing direction, the connected L-points move along the positive Y direction, thus the positive sign of the value in N. On the other hand, the R-points move along the negative Y direction, thus the negative sign.

The fourth line memory H contains the information of the horizontal direction of the next contour neighbor, which can be determined by fitting a 2 by 2 window around the point and determine the corner information at the point of interest. There are eight possible corners as shown at the top of Fig. 4. Again, the gray pixels belong to the object region and white pixels to background. The point of interest is indicated by the darker gray pixel. Since we are only interested in the horizontal direction of the next contour point, we only need to consider four out of eight cases, namely the cases (a), (c), (f), and (h). Among these four cases, (a) and (h) represent the cases when the next contour point moves towards the positive X direction whereas (c) and (f) are the cases when the next contour point moves towards the negative X direction. If a point does not belong to any of these four cases, then there is no horizontal movement to the next contour neighbor (there is only a vertical movement). Some examples of these corners are shown in the bottom of Fig. 4.

In our current implementation, three line memories are used to store three image scan lines. For each point, a 2 by 2 window is fitted around the point to determine the corner information. Noticing that the cases (a) and (f) only occur to the R-points and (c) and (h) only to the L-points, this procedure is further speeded up by checking only two cases per point. For the points that were determined to have positive horizontal movement, '+' is assigned in H, and those with negative horizontal movement, '-' is assigned, and those with no horizontal movement, '0' is assigned.

3.2. Reordering Left-Side and Right-Side Contour Points

With the four line memories that contain the information of the LR-points, the sequence of LR-points is re-ordered in the

counterclockwise as follows. We will describe the re-ordering algorithm with the help of the example shown in Fig. 3. Let us first define some notations. Let $pt(i) = (x(i), y(i))$ be the x and y coordinates of the i-th point stored in the X and Y line memories, $n(i)$ and $h(i)$ the corresponding values in N and H, respectively, of the point.

First, the starting index point needs to be determined. Note that the starting index can be any number less than the total number of the detected LR-points. In our example, we will choose index 1 as the starting point. Thus, the x- and y-coordinates of $pt(1)$ is copied to the first position in X' and Y'.

The next step is to find the N value of the adjacent scan line of the current point. The adjacent scan line of an L-point is the scan line below (since the vertical direction of the contour-neighbor of an L-point is in the positive Y direction), and the adjacent scan line of an R-point is the scan line above. From the current point position in the storage, we can jump either 2 steps forward (in the case the current point is an L-point) or 2 steps backward (in the case of an R-point), until there is a change in the y-coordinate value. For example, starting from $pt(1)$, since it is an L-point, jumping 2 steps forward takes us to $pt(3)$. Since $y(1) \neq y(3)$, we know we have reached to a point in the adjacent scan line. The N value at this point is the number of LR-points in the adjacent scan line.

Once we have the N value of the adjacent scan line, denoted as m, the next step is to identify the next contour neighbor. It is not difficult to see that all possible candidates for the next contour neighbor are the points with even index numbers between $i + n(i)$ and $i + m$, along with the two storage neighbors of i. More formally, a set of the indices of all possible candidates can be expressed as: $K = \{i + k \,|\, k \in even\, numbers \wedge \min(n(i), m) \leq k \leq \max(n(i), m)\} \cup \{i + 1, i - 1\}$. Out of these possible candidates, we can discard some points by checking the values in H. Recall that $h(i)$ indicates the horizontal direction of the next contour neighbor of the i-th point. Thus, if $h(i) = -$, then the x coordinate of the next contour neighbor must be less than that of the i-th point. Similarly, if $h(i) = +$, then the x coordinate of the next contour neighbor must be greater than that of the i-th point. All candidate points that do not satisfy any of these conditions are discarded. For example, let's say the current point is $pt(3)$ in Fig. 3. All possible candidates for the next contour neighbor of $pt(3)$ is $\{2, 4, 5, 7\}$. Out of these candidates, only $pt(7)$ satisfies the constraint posed by $h(3)$ (i.e., $h(3) = -$ and $x(3) > x(7)$).

There may be some cases where the constraint posed by the H value does not eliminate all false candidates. In this case, we compute the distance between the current point and each of the remaining candidate points. The candidate with the smallest distance is chosen as the next contour neighbor. For example, let's say that the current point is $pt(6)$. Since it is an R-point, its adjacent scan line is the image row 2 (the third row of the image). Therefore, all possible candidates for the next contour neighbor of $pt(6)$ is $\{5, 7, 2, 4\}$. Out of these points, only $pt(7)$ is eliminated by the H value constraint. Among the remaining three points, $pt(5)$ is closest to $pt(6)$, thus $pt(5)$ is chosen as the next contour neighbor.

All the above steps are continued until the current point reaches to the original starting point. The pseudo code of the proposed algorithm is outlined in Algorithm 1.

Algorithm 1 Pseudo code of the proposed contour extraction algorithm

```
Initialize 6 line memories (X,Y,N,H,X',Y')
// LR-Points Detection (Section 3.1)
for each image scan line
   Hold the current scanline and two adjacent scanlines
        in the buffer
   Detect LR-points
   Fill in X,Y,N, and H
// LR-Points Re-ordering (Section 3.2)
i=i0 // i is the index of the current point
     // i0 can be any number less than the total number
     // of LR-points
j=0
do
   X'(j)=X(i)
   Y'(j)=Y(i)
   m = N value of the adjacent scan line
   K = {all even numbers between n(i) and m, i+1, i-1}
   for each k∈K
      if (x(i)-x(k) > 0 && h(i)=='+')
         Discard k from K
      if (x(i)-x(k) < 0 && h(i)=='-')
         Discard k from K
   for each k∈K
      compute the distance between pt(i) and pt(k)
   i = the index of the point with the minimum distance
   j++
while i≠ i0
```

In the detection and reordering procedures described so far, we have assumed that the total number of the detected LR-points fits in a single line memory size. If it is not the case, however, then we could either use additional line memories to store the remaining LR-points or downsample them by taking, for example, every second or third scan line. In the former case, the contour reordering algorithm should be able to deal with two or more line memories as a single circular array during its reordering procedure.

3.3. Full Contour Extraction

Once we have the re-ordered list of LR-points, we can obtain the complete list of contour points by inserting additional points between two consecutive LR-points in the list that are not connected in the image. For example, although $pt(15)$ and $pt(14)$ are adjacent to each other in the re-ordered list, they are not pixel-wise neighbors. A complete list of contour points can be obtained by applying a set of rules for inserting additional points between two disconnected LR-points. However, depending on the point insertion rules, the resulting contour may be slightly different than the true contour of the object. For example, applying a simple rule of connect-

ing two disconnected LR-points on the same row can create correct contour points, for example, for the case of between $pt(15)$ and $pt(14)$ or incorrect contour points for the case of between $pt(6)$ and $pt(5)$. One possible way to alleviate this problem would be to utilize the corner information that we obtain during the LR-points detection step.

3.4. Discussion on the Proposed Contour Extraction Algorithm

The procedure we have described so far is for the case when the image contains a single object whose boundary needs to be extracted as a closed contour. However, the procedure extends easily to the case of multiple objects. Note that the contour tracing procedure we described above must always return back to the start point. If a contour returns back to its start point and there are still some boundary points unaccounted for, then starting a second boundary trace with any one of those previously unvisited points will extract the next boundary contour. We should also mention that image noise caused contours can be discarded during contour tracing by placing a threshold on the minimal acceptable length of a contour.

The contour extraction algorithm described above is summarized in Algorithm 1. It can detect and reorder all the contour points of the scene objects regardless of their shape and the starting point of the reordering. If desired, this algorithm can also be implemented on general-purpose processors.

Our proposed method is advantageous not only because it needs to hold only three consecutive input lines at once instead of storing an entire image and processes them in an SIMD fashion, but also because it reduces the number of boundary points to be processed to some extent depending on the shape of the object. Since it actually processes only the end points of the horizontal line segments detected as objects in a row, it just skips the processing of all the internal points of the horizontal line segment, directly going to the next boundary point of the next scanline. Thus, the more there are horizontal line segments in an image, the smaller the number of boundary points are stored and processed, which cause better performance with respect to the size of the input. It is because the performance of the contour tracing algorithm generally depends on the number of points to be processed.

4. EXTRACTING THE CURVATURE SCALE SPACE

We will now show how the contour extraction procedure of the previous section can be used to create a curvature scale space (CSS) for the boundary contours in an image. The curvature scale space is a multiscale, curvature-based shape representation for planar curves. It utilizes the curvature zero-crossings of a contour at multiple scales as shape features. The curves at lower resolution are obtained through smoothing with wider Gaussian kernels. The curvature scale space representation was tested by comparing it with other several shape description techniques on different image databases [20]. On the basis of such comparative evaluations, it was selected as a standard in the visual part of the MPEG7 standard.

4.1. Definition

Given a planar curve $\mathbf{\Gamma} = \{(x(u), y(u)) \mid u \in [0,1]\}$, a smoothed (also referred to as *evolved*) version of the curve is defined as:

$$\mathbf{\Gamma}_\sigma = \{r_\sigma(u) = (x_\sigma(u), y_\sigma(u)) \mid u \in [0,1]\}$$

where

$$\begin{aligned} x_\sigma(u) &= x(u) * w_\sigma(u) = \int_{-\infty}^{\infty} x(v) w_\sigma(u-v) dv \\ y_\sigma(u) &= y(u) * w_\sigma(u) = \int_{-\infty}^{\infty} y(v) w_\sigma(u-v) dv. \end{aligned}$$

In CSS, the weight function $w_\sigma(u)$ is a Gaussian of width σ:

$$w_\sigma(u) = \frac{1}{\sigma\sqrt{2\pi}} e^{-u^2/2\sigma^2}.$$

The curvature $\kappa_\sigma(u)$ of Γ_σ at a particular scale σ is given by

$$\kappa_\sigma(u) = \frac{\dot{x}(u)\ddot{y}(u) - \dot{y}(u)\ddot{x}(u)}{\left(\dot{x}(u)^2 + \dot{y}(u)^2\right)^{3/2}}.$$

The derivatives needed for the curvature calculation can be computed by:

$$\begin{aligned} \dot{x}_\sigma(u) &= \frac{\partial}{\partial u}\left(x(u) * w_\sigma(u)\right) = x(u) * \dot{w}_\sigma(u) \\ \ddot{x}_\sigma(u) &= \frac{\partial^2}{\partial u^2}\left(x(u) * w_\sigma(u)\right) = x(u) * \ddot{w}_\sigma(u) \end{aligned}$$

with similar forms for $\dot{y}_\sigma(u)$ and $\ddot{y}_\sigma(u)$. Since the derivatives of the Gaussian kernel can be precomputed, the curvature can be computed more easily by:

$$\kappa_\sigma(u) = \frac{\dot{x}_\sigma(u)\ddot{y}_\sigma(u) - \ddot{x}_\sigma(u)\dot{y}_\sigma(u)}{\left(\dot{x}_\sigma^2(u) + \dot{y}_\sigma^2(u)\right)^{3/2}}.$$

The curvature zero crossings are localized along the scale dimension. If we trace the location of these zero crossings in the (u, σ) space, we get what is referred to as the CSS image. In the CSS image, the zero crossings merge as the scale increases, as shown in Fig.5. *The CSS descriptor is defined as the coordinates of the merged points in the (u, σ) space if the points are not generated by noise.*

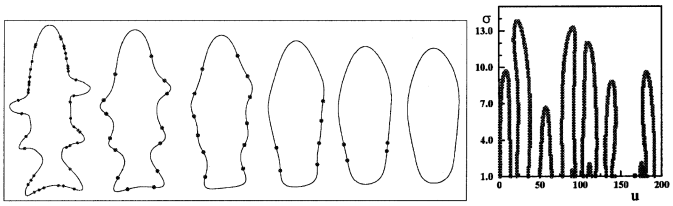


Fig. 5. Contour evolution and the CSS image: (a) Smoothing of a planar curve with wider Gaussian kernels; (b) The corresponding CSS image plotted in (u-σ) space. These are reprinted from [21].

4.2. Computing the Derivatives on SIMD Processors

The extraction of the curvature scale space generally requires that Gaussian filtering be applied 50 to 100 times. With regard to the filtering calculations, a single multiplication or addition for 320 elements can be done in one cycle on IC3D. Exploiting this parallelity, an efficient filtering procedure with a single kernel can be implemented in a recursive fashion by using the output of the previous result.

The computation of curvature of the evolved contours entails the computation of the first and the second derivatives with multiple divisions. Since division is computationally expensive and inaccurate on IC3D, it should be avoided as much as possible. Moreover, even a small loss of precision in single filtering step may result in high inaccuracy in the output of a recursive filtering calculation. Therefore it is important that smoothing in linear scale space be carried out with high precision.

The WiCa has limitations on the precision with which numbers can be represented in the SIMD processor. The WiCa can only use a 10 bit number representation in its LPA, which gives it an integer range from -511 to 511. Also, there is no support for handling the overflow during the computations in LPA. Moreover, due to the real-time operation of WiCa, every processing element is driven by and synchronized to the inputs from the sensor. One video line is transferred from the sensor to the Xetal processor at each clock cycle. Thus, the basic storage block of the Xetal program is a single video line. Since Xetal runs at between $80 \sim 100MHz$ with 30 fps with 480 video lines per frame, the maximum number of clock cycles between the capture of the successive video lines is roughly 6000. If the number of instructions in a per-video line program block exceeds this number, the next input video line from the sensor will be missed, which may crash the whole program.

In what follows, we introduce a way of overcoming these limitations to achieve high-precision Gaussian filtering and the curvature zero-crossing localization on SIMD processors while completely avoiding division operations.

4.3. Gaussian smoothing

To implement Gaussian filtering with the required precision, we employ a number representation along with overflow control using two line memories: one for the higher bits, $Lmem_H$, and the other for the lower bits, $Lmem_L$. To carry the overflow from $Lmem_L$ into $Lmem_H$, one bit in $Lmem_L$ is used to mark the occurrence of the overflow. Also, since the sign bit in $Lmem_L$ is redundant with that in $Lmem_H$, only 8 bits are available for the lower bits. (Recall that each element in a line memory is represented by 10 bits.) Thus, the new two line-memory based number representation uses 18 bits. In this paper, the higher 10 bits are used for the integer part and the lower 8 bits for the fractional part. This gives us a range of -511.996 to 511.996 for representing numbers. This number representation entails a precision of $1/2^8 = 0.0039$.

With this extended number system, a suitable multiplication scheme for linear filtering is devised with overflow control. We assume that the initial input data, which are the coordinates of the boundary pixels in this application, are of integer type and that the result of the multiplication between the input data and the coefficients of a linear filter ranges from -511.996 to 511.996. Let *Signal* be an integer type *Lmem* data, and *Coef* one of the original coefficients of a linear filter. For high-precision computation, *Coef* is multiplied by 2^8, and split into the integer part, *intCoef*, and the fractional part, *fixedCoef*. The output will be two line memories, $Lmem_H$ and $Lmem_L$. The multiplication between *Signal* and *intCoef* is carried out using multiple additions and stored in $Lmem_L$. Whenever the value in $Lmem_L$ is greater than 255, we have a carry that is transferred to $Lmem_H$. *Signal* is also multiplied by *fixedCoef* and added to $Lmem_L$ with carry control. All these *Lmem*-based computations are carried out on an element-wise basis. If *Signal* is multiplied with all the filter coefficients using this scheme and added up, then filtering will be carried out without loss of precision (subject to our assumptions regarding the range of the multiplies). Since *Coef* is already multiplied by 2^8, the final result should be shifted to the right by 8 bit. Instead of doing it, if we take only $Lmem_H$ as the result, then it will be the integer part of the result of the multiplication while $Lmem_L$ will be the fractional part. The accurate multiplication algorithm is briefly summarized in Algorithm 2.

While this high-precision multiplication scheme benefits the accuracy of the filtering operations, it does involve a computational overhead. Checking whether or not there is a carry in each operation during the multiple additions required for multiplication demands is obviously burdensome. This additional computational overhead means that filtering operations related to one video line cannot be completed before the next video line must be scanned in — since it entails using more than 6000 machine cycles. One way to get around this problem is to multiply *Coef* by 2^7 or 2^6 instead of 2^8. That reduces the number of additions to be used for the multipli-

Algorithm 2 High-precision multiplication on LPA

```
AccurateMUL(Signal, intCoef, fixedCoef)
// Signal : integer-type data
// intCoef : integer part of 256*Coef
// fixedCoef : fixed point part of 256*Coef
// Note: Every computation is done in element-wise
Lmem_H = output Lmem for integer (higher 10 bits)
Lmem_L = output Lmem for fixed point (lower 8 bits)
// Compute integer multiplication
for i = 0...intCoef
    Lmem_L += Signal
    if Lmem_L > 255, then transfer carry to Lmem_H
// Compute fixed point multiplication
Lmem_L += Signal * fixedCoef
if Lmem_L > 255, then transfer carry to Lmem_H
```

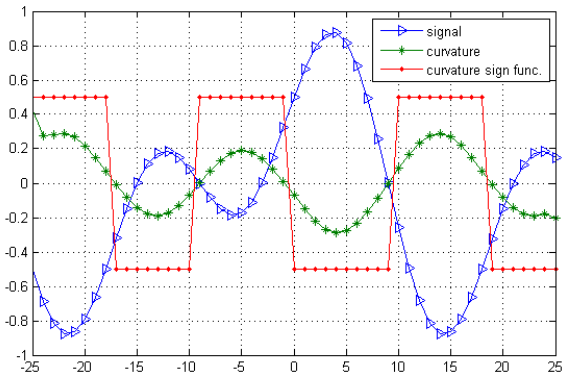


Fig. 6. The curvature zero-crossings from real curvature computation and its simplified version.

cation steps. Another way to deal with the problem is to distribute the computational load over several video lines. In this paper, the latter method is used so that completing one Gaussian filtering takes about 6 video lines in a recursive fashion.

4.4. Localization of Curvature Zero Crossings

The localization of curvature zero-crossings can be simplified by using an approximation scheme identifying the convexity and concavity of the curvature. The simplification is based on the fact that only the sign of the curvatures needs to be determined to obtain the curvature zero-crossings.

Curvature, $\kappa(i)$, is defined as the rate of change of the tangent angle $\phi(i)$. That is, $\kappa(i) = \lim_{h\to 0}\frac{\phi}{h} = \frac{d\phi(i)}{di}$.

Let $m(i)$ be the gradient of the tangent line at i_{th} point. Then $m(i) = \tan\phi(i) = \frac{dy}{dx}$ and

$$\begin{aligned}\frac{d\phi(i)}{di} &= \frac{d}{di}\arctan m(i) = \frac{1}{1+m(i)^2}\frac{d}{di}m(i)\\ &= f(i)\frac{d}{di}m(i)\,, where\ f(i) = \frac{1}{1+m(i)^2}.\end{aligned}$$

Since the differentiation of discrete data is defined as the following:

$$\frac{d}{di}m(i) = \lim_{h\to 0}\frac{m(i)-m(i+h)}{h} \approx \frac{m(i-h)-m(i+h)}{2h}$$

$$\frac{dx}{di} \approx \frac{x(i+h)-x(i-h)}{2h}\,, \frac{dy}{di} \approx \frac{y(i+h)-y(i-h)}{2h},$$

the first derivative of the gradient can be written as: $\frac{d}{di}m(i) = \frac{g(i)}{h(i)}$, where

$$\begin{aligned}g(i) &= (y(i)-y(i-2h))(x(i+2h)-x(i))\\ &\quad -(y(i+2h)-y(i))(x(i)-x(i-2h))\\ h(i) &= 2(x(i)-x(i-2h))(x(i+2h)-x(i)).\end{aligned}$$

Thus, combining these equations, the identification of the convexity and concavity of curvatures is simplified as the following without involving any divisions:

$$\begin{aligned}sign(\kappa(i)) &= sign(\frac{d\phi(i)}{di}) = sign(f(i)\frac{d}{di}m(i))\\ &= sign(\frac{d}{di}m(i))\,, since\ f(i) > 0\\ &= sign(\frac{g(i)}{h(i)})\\ &= sign(g(i)) \times sign(h(i)).\end{aligned}$$

The determination of the sign of the curvature using both real curvature computation and this simplification scheme was tested using MATLAB, as shown in Fig.6. The result shows that this simplified scheme is appropriate for localizing curvature zero-crossings.

Since this simplification is based on the gradient of tangent lines, the sign of the curvature becomes unreliable when a tangent line is close to either the horizontal or the vertical axis. In such cases, then the contour coordinates can be rotated by 45 degree to get stable results.

The convexity or concavity of the curvature may also be determined using the Gaussian derivatives as kernels. However, that would double the computational load since we would need to carry out two filtering operations at every scale: with the first and the second derivatives of the Gaussian. Since differentiation is much cheaper than high-precision filtering, we have used the the former in this paper.

5. DEMONSTRATION

The demonstration consists of two parts: one for contour extraction and the other for the curvature scale space. The integration of these algorithms with proper object segmentation

Fig. 8. The curvature zero-crossings on smoothed contours: The leftmost figure is the image of a 3D synthetic human model. Starting with the second left, the bold green contour segments indicate concave curvature and the thin red contour segments the convex curvature. From the left: σ= 8, 40, 120.

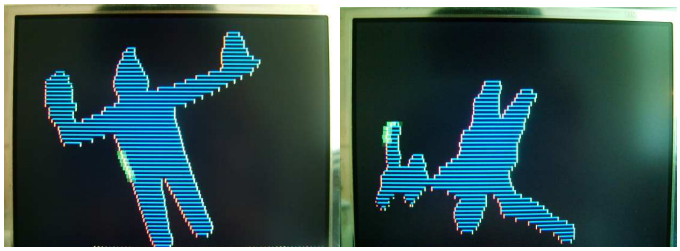


Fig. 7. The extracted contours of the segmented objects. The left-side contour points are marked in red whereas the right-side are in green. The highlighted bold green points move along the contour sequentially; we show these to help the reader visualize how the points are ordered.

and shape matching remains as future work. All programs are written in XTC language. The size of the image processed by the WiCa is 640 by 480 pixels (VGA). An LCD screen is connected to the camera for debugging purposes so that we can observe the intermediate results that are internally available. The display shows them in the VGA format.

5.1. Experiments with Contour Extraction

The segmented regions are fed from a PC into the DPRAM. The images are binarized so that the objects have white pixels and the background black. Then, the algorithm reads the image from the DPRAM at the same speed that would be the case if the image had been captured directly by the input sensor. (On the PC, a GUI program called WiCaEnv is used to send the images via a USB port for the experiment. This could also be done through wireless if we used the Zigbee module.) The contour extraction algorithm runs at 30 frames per second on the camera. Fig. 7 shows the extracted contours from two binarized images. To test the algorithm, the images are significantly deformed intentionally to be more complicated. Although it is hard to check how the contour points are ordered from the figures, we highlight each point sequentially as it is ordered. This creates an impression of the points moving along the contour.

5.2. Mapping of curvature Scale Space

We again feed the necessary data from a PC using the WiCaEnv program. A synthetic 3D human model is created and projected on the image plane of a hypothetical camera. Then we extract the ordered contour points using the OpenCV library — this will be replaced by our contour extraction algorithm in future experiments. After the ordered contour points are written in DPRAM, the curvature scale space algorithm retrieves them and performs Gaussian smoothing while localizing the curvature zero-crossings. In Fig.8, the leftmost image shows the projected 3D human model. Starting with the second-left image, we show the gradually smoothed contours of the model. The points in the concave curve segments are shown as bold and highlighted in green, whereas those in the convex curve segments are thinned and marked as red.

The algorithm runs on the WiCa at 15 frames per second because of the high-precision Gaussian smoothing. Currently, the contours evolve recursively using a single Gaussian kernel with a standard deviation of 4.0. (The smoothed contours will evolve more slowly if we used a narrower kernel.) The performance of this algorithm could be improved further by optimizing the code.

6. CONCLUSION

The implementation of real-time vision algorithms in embedded systems requires a careful balance between accuracy and speed. As to what challenges are created with regarding to the achievement of this balance depends on the architecture of the embedded system. Our paper presented the difficulties faced and the solutions thereof when one tries to implement contour extraction on the SIMD processor of the WiCa camera. Our paper also showed how it is possible to extract the curvature scale space on this processor and introduced an extended number representation for that purpose. The calculation of curvature scale space entails linear filtering — a step that is particularly prone to errors if implemented with limited precision.

7. REFERENCES

[1] H. Fatemi, R. Kleihorst, H. Corporaal, and P. Jonker, "Real time face recognition on a smart camera," in *Advanced Concepts for Intelligent Vision Systems, ACIVS'03*, 2003.

[2] X. Gao, R. Kleihorst, and B. Schueler, "Implementation of auto-rectification and depth estimation of stereo video in a real-time smart camera system," in *Fourth Workshop on Embedded Computer Vision in conjunction with CVPR, to be appeared*, 2008.

[3] C. Wu, H. Aghajan, and R. Kleihorst, "Mapping vision algorithms on simd architecture smart cameras," in *First ACM/IEEE International Conference on Distributed Smart Cameras, ICDSC '07*, Sep 2007, pp. 27–34.

[4] R. Kleihorst, A. Danilin, and B. Schueler, "Wireless smart camera with high performance vision system," *Telematik*, pp. 3:20–25, 2006.

[5] I. Bartolini, P. Ciaccia, and M. Patella, "Warp: Accurate retrieval of shapes using phase of fourier descriptors and time warping distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 142–147, 2005.

[6] G.C.H. Chuang and C.C.J. Kuo, "Wavelet descriptor of planar curves: theory and applications," *IEEE Transactions on Image Processing*, vol. 5, no. 1, pp. 56–70, 1996.

[7] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 4, pp. 509–522, 2002.

[8] F. Mokhtarian and A.K. Mackworth, "A theory of multiscale, curvature-based shape representation for planar curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 8, pp. 789–805, 1992.

[9] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[10] J. Malik and S. Russell, "A machine vision based surveillance system," *University of California, Berkeley, PATH Project MOU-83 Final Report*, 1994.

[11] M. Yokoyama and T. Poggio, "A contour-based moving object detection and tracking," in *2nd Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, oct 2005, pp. 271–276.

[12] N. Paragios and R. Deriche, "Geodesic active contours and level sets for the detection and tracking of moving objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 3, pp. 266–280, 2000.

[13] J. Liang, T. McInerney, and D. Terzopoulos, "United snakes," *Medical Image Analysis*, vol. 10, no. 2, pp. 215–233, apr 2006.

[14] H. Freeman and A. Saghri, "Generalized chain codes for planar curves," in *the Fourth International Joint Conference on Pattern Recognition*, Nov 1978, pp. 701–703.

[15] M. Ren, J. Yang, and H. Sun, "Tracing boundary contours in a binary image," *Image and Vision Computing*, vol. 20, no. 2, pp. 125–131, 2002.

[16] K. Ratnayake and A. Amer, "A real-time implementation of chaotic contour tracing and filling of video objects on reconfigurable hardware," in *IEEE International Conference on Systems, Man and Cybernetics, ISIC*, Oct 2007, pp. 1089–1094.

[17] David W. Capson, "An improved algorithm for the sequential extraction of boundaries from a raster scan," *Computer vision, graphics, and image processing*, vol. 28, no. 1, pp. 109–125, 1984.

[18] T.Chia, K. Wanga, L. Chenb, and Z. Chenb, "A parallel algorithm for generating chain code of objects in binary images," *Information Sciences*, vol. 149, no. 4, pp. 219–234, 2003.

[19] J. Freixenet, X. Munoz, D. Raba, J. Marti, and X. Cufi, "Yet another survey on image segmentation: Region and boundary information integration," *Computer Vision - ECCV 2002*, pp. 408–422.

[20] M. Bober, "Mpeg-7 visual shape descriptors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 6, pp. 716–719, 2001.

[21] S. Abbasi, F. Mokhtarian, and J. Kittler, "Curvature scale space image in shape similarity retrieval," *Multimedia Systems*, vol. 7, no. 6, pp. 467–476, 1999.