Available online at www.sciencedirect.com

SCIENCE DIRECT®

Microprocessors and Microsystems 29 (2005) 405–410

MICROPROCESSORS AND MICROSYSTEMS

www.elsevier.com/locate/micpro

A pipelined array architecture for Euclidean distance transformation and its FPGA implementation

N. Sudha

Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India 600 036

Received 14 October 2003; revised 23 February 2004; accepted 5 October 2004
Available online 13 November 2005

Abstract

The Euclidean Distance Transform (EDT) is an important tool in image analysis and machine vision. This paper provides an area-efficient hardware solution to the computation of EDT on a binary image. An $O(n)$ hardware algorithm for computing EDT of an $n\times n$ image is presented. A pipelined 2D array architecture for harware implementation is designed. The architecture has a regular structure with locally connected identical processing elements. Further, pipelining reduces hardware resources. Such an array architecture is easily scalable to handle images of different sizes and is suitable for implementation on reconfigurable devices like FPGAs. Results of FPGA-based implementation shows that the hardware can process about 6000 images of size 512×512 per second which is much higher than the video rate of 30 frames per second.
© 2004 Elsevier B.V. All rights reserved.

Index terms: Euclidean distance transform; Image; Pipelining; FPGA

1. Introduction

Many problems in computer vision require the ability to extract and process *metric* and *radiometric* information that are intrinsic to images. Radiometric information would consist of gray-scale information in the form of histograms, edge and texture data. This paper focuses on metric information and in particular, that obtained via distance transforms on images.

A distance transformation converts a binary image consisting of foreground and background pixels into one where each pixel has a value equal to its distance to the nearest background pixel (alternatively, distances could be to the nearest foreground pixel). Applications of distance transform are numerous. These include shape analysis of objects [1], machine vision [2] and image matching [3]. The types of transforms used generally are city-block distance transform, chessboard distance transform and Euclidean distance transform (EDT). Of these, EDT finds widespread use in view of the natural metric employed.

Considerable research has been done on development of algorithms for computation of the EDT. Several sequential [4–7] and parallel [8–11] algorithms are available. Some work on parallel algorithms targeted to general-purpose processors is also known [12,13]. The time complexities of these algorithms for an $n\times n$ image on an $n\times n$ mesh are $O(n)$ and $O(\log^2 n)$. However, the actual implementation is not given and hence the real-time performance of these algorithms are not known.

Applications such as tracking of objects in a video sequence require real-time performance. The rapid growth in FPGA technology calls for hardware mappable algorithms and cost-effective FPGA-based solutions. Array-type architectures consisting of locally interconnected identical processors are most commonly designed to solve imaging problems in FPGAs. These architectures are easily scalable to handle different image sizes. The reconfigurability feature of FPGAs is favourable to such scalable designs. Further, FPGAs facilitate rapid prototyping of designs. However, Euclidean distance computation is hard to decompose into local neighborhood operations because it involves a nonlinear squaring and square root operations. Work on custom hardware for EDT is scarce. Ref. [10]

E-mail address: sudha@iitm.ac.in

presents an approach to the computation of EDT based on the morphological dilation operation. Some pointers to hardware mapping are provided.

In this paper, a pipelined array architecture is presented for the computation of EDT. The architecture comprises of a two-dimensional array of locally interconnected identical processing elements where each element is a sequential logic and all elements are operated synchronously. The computations within each element are with integers. Such a digital design with no floating point arithmetic is quite suitable for FPGA implementation. In particular, the idea is to assign, to each processing element, multiple pixels of the given binary image. The motivation is to reduce the hardware (space requirements) while keeping the speed of operation still at the desirable (real-time) level. Incorporation of pipelining is not straightforward and requires handling appropriately the data dependencies in the architecture.

The design details including the derivations to facilitate pipelining constitute an important contribution of this paper. The ideas presented for the case of 4 pixels per processing element readily extend to the case of more than 4 pixels per processing element (such as 9, 16 and so on). Results of Xilinx FPGA implementation suggest that the hardware can process fairly large images much faster than the video rate.

The organization of the paper is as follows. The next section gives the hardware mappable algorithm for EDT. Section 3 describes the pipelined architecture that implements the algorithm. Section 4 presents the Xilinx FPGA implementation of the architecture. Finally, Section 5 concludes the paper.

2. Hardware algorithm

In this section, a parallel algorithm which is amenable to VLSI implementation is presented. The salient feature of the algorithm is that the computation of EDT involves only integer arithmetic operations within a small neighborhood of each pixel and hence it is suitable for mapping onto a high-speed array architecture.

The algorithm computes distance vector (Δr, Δc) of pixels where Δr and Δc are the number of rows and columns by which a pixel is displaced from its nearest background pixel. The Euclidean distance is given by $\sqrt{\Delta r^2 + \Delta c^2}$. ($\Delta r$, Δc) of background pixels are initialized to (0,0) and those of foreground pixels are computed iteratively starting from the pixels nearby background and moving towards the far away pixels. At any iteration k, ($\Delta r(p)$, $\Delta c(p)$) of those pixels p whose nearest integer approximation to Euclidean distance $d(p)$ equals k are computed. That is, $d(p)$ lies within ($k-0.5$, $k+0.5$]. $d(p)$ is not an integer and hence we shall consider $d^2(p)$. $d^2(p)$ lies within (k^2-k, k^2+k] since $d^2(p)$ is an integer. However, $d^2(p)$ is quite large in magnitude and it requires a large storage space in hardware. A new integer quantity $\delta(p)$ which is much smaller than $d^2(p)$ is defined as $(k^2+k)-d^2(p)$. δ is used for the computation of (Δr, Δc). It is derived as follows.

$d^2(p)$ is derived first and then it is substituted in the definition for $\delta(p)$. $d^2(p)$ can be derived using the already computed (Δr, Δc) of eight neighbors p_i, $i=1$–8, surrounding p. It is given by $\min[\Delta r_i^2 + \Delta c_i^2]$ where $\Delta r_i = \Delta(p_i)$ if p_i is in the same row of p. Otherwise, $\Delta r_i = \Delta r(p_i)+1$. The increment by 1 is due to p being displaced from p_i by one row. Similarly, Δc_i is given in terms of $\Delta c(p_i)$. $d^2(p)$ can be rewritten as $\min[d^2(p_i) + \Delta R_i + \Delta C_i]$ where $\Delta R_i = 0$ if p_i is in the same row of p. Otherwise, $\Delta R_i = 2\Delta r(p_i)+1$. $\delta(p)$ is now derived as follows.

$$\begin{aligned}\delta(p) &= k^2 + k - d^2(p)\\ &= \max[k^2 + k - d^2(p_i) - \Delta R_i - \Delta C_i]\\ &= \max[\delta(p_i) - \Delta R_i - \Delta C_i] = \max[\delta_i] \qquad (1)\end{aligned}$$

$\delta(p) \geq 0$ means $d(p)$ lies below $k+0.5$.

The iterative computation of (Δr, Δc) proceeds as follows. δ of the background pixels are initialized to 0. At each iteration k, δ of foreground pixels whose (Δr, Δc) are not yet known are computed using already computed δ, Δr and Δc of neighbors. If $\delta(p) \geq 0$, then ($\Delta r(p)$, $\Delta c(p)$) corresponds to (Δr_i, Δc_i) where subscript i pertains to the pixel p_i that gives $\delta(p)$. That is, p_i that satisfies Eq. (1).

Once (Δr, Δc) of a pixel is known, its δ should be updated for at least two successive iterations as it depends on k. The updating allows use of δ for the computation of (Δr, Δc) of neighbors. $\delta_k(p)$ at iteration k is derived from $\delta_{k-1}(p)$ at iteration $k-1$ as follows.

$$\begin{aligned}\delta_k(p) &= k^2 + k - d^2(p) = 2k + [(k-1)^2 + (k-1) - d^2(p)]\\ &= 2k + \delta_{k-1}(p) \qquad (2)\end{aligned}$$

The iterative computation of (Δr, Δc) and δ values of pixels of an image with two background pixels is illustrated in Fig. 1. The pixels whose values have been computed at each iteration k are shown in the figure. Consider iteration $k=2$. δ of those twelve pixels whose (Δr, Δc) have been computed at $k=1$ are incremented by $2k$ (i.e. 4). Besides, some new pixels' (Δr, Δc) and δ are computed. One such pixel is at $(r,c)=(1,4)$. Its values are computed as follows. There are three valid neighbors at (2,3), (2,4) and (2,5). The δ_i values obtained from these neighbors are -1,2 and -2 and the maximum corresponds to the pixel at (2,4). (Δr, Δc) is hence given by (2,0).

To keep track of pixels whose (Δr, Δc) have been computed, a flag *done* is assigned to each pixel, whose value is set to 1 when the transform values of pixels are computed at any iteration. The algorithm for hardware implementation is given below.