A Hardware Design of EDT Algorithm Applied to Binary Images

Hanxi Feng , Liping Tang
Department of Information Science and Technology,
Donghua University,
Shanghai, 201620, China
E-mail: ericfhx1988@163.com

Peifeng Zeng
Department of Computer Sciences,
Donghua University,
Shanghai, 201620, China
E-mail: zengpf@gmail.com

***Abstract*—In this paper, a hardware Euclidean distance transform (EDT) algorithm is proposed. The binary image data of each row are independent. For a binary image, the row distances and column distances are separately calculated. By taking the advantage of these two characteristics, a parallel EDT circuit based on the algorithm with the pipeline is designed and verified on the FPGA hardware platform. The experiment demonstrates that the time complexity of proposed EDT is $O(N)$ for binary image with the size of $N \times N$.**

Keywords-distance transform; Euclidean distance; parallelization; hardware; FPGA

I. Introduction

The Euclidean distance map on a binary image has been widely used in image processing, morphology, pattern recognition and artificial intelligence fields. Euclidean distance has the actual physical meaning as well as the accuracy. Compared with other commonly used distance definition, Euclidean distance transformation (EDT) has a higher computational complexity with a relatively high time complexity.

There are many methods to improve EDT computational complexity. When the accuracy requirement is not high, city distance and chessboard distance are commonly used as Euclidean distance approximation. For the chessboard and city-block distances, a raster-scanning algorithm works well and its time complexity is $O(N^2)$ [1].

Some algorithms are improved and optimized directly on the EDT algorithm. For a $N \times N$ binary image, the raster-scanning algorithm can not get precision Euclidean distance values for all pixels [2]. The order-communication algorithm can achieve the Euclidean distance for whole image, but some pixels will be calculated many times. So its time complexity can reach $O(N^3)$ in the worst case [3]. The divide-and-conquer algorithm computes the distance transform column by column and its time complexity reaches $O(N^2 \log N)$ [4]. The algorithm based on the construction of the Voronoi diagram reduces its time complexity to $O(N^2)$ [5]. The independent-scanning algorithm is suitable for Euclidean distance, city distance and chessboard distance. It firstly processes column scanning to acquire each distance of the column and row. Then it processes row scanning to obtain the Euclidean distance. Its time complexity is also $O(N^2)$ [6].

For the proposed algorithms in [4-6], their time complexity and input binary image are in the same order of magnitude, so they are named linear Euclidean distance transform algorithms. These algorithm can meet the needs of most applications, but in some high real-time image processing fields, such as pattern recognition and artificial intelligence, computing time is the shorter, the better. Such requirements prompt some parallel EDT algorithms. The parallel algorithm is based on hypercube computer and the grid interconnection tree network, and its time complexity reaches $O(\sqrt{N})$ [7]. The algorithm based on two-dimensional torus connected processor array makes the time complexity to be $O(N)$ [8]. But these two algorithms require a lot of independent processors, so the circuit size is large to go against system integration. In order to reduce circuit size and achieve real hardware implementation, there are also some researches based on VLSI. The algorithm proposes a hardware architecture in which the processing is divided into row processing and column processing. Each processing is performed by N processors and the time complexity is $O(N^2)$ [9].The algorithm proposes a $N \times N$ processing unit array and reduces the computing time to $(5N-4)$ clocks [10].

The construction of this paper is as follows: In Section 2, the independent-scanning linear EDT algorithm is briefly reviewed. The improved hardware algorithm is described in Section 3. In Section 4, some hardware system architecture improvement is detailed. Finally, some design details and the verification based on FPGA are showed in Section 5. In this paper, we can successfully avoid the use of multipliers and improve the system architecture to make the computing time to be $(2N+1)$ clocks.

II. Independent-scanning linear EDT algorithm

Euclidean distance is the important index to measure location relationship between different binary image pixels. It’s widely used in contour extraction, skeleton extraction and other image processing.

A. EDT applied to binary images

Let $B=\{b_{ij}\}$ be a $N\times N$ binary image. The pixel at row i and column j is denoted by (i,j). b_{ij} is the value of the pixel (i,j). For each pixel, the Euclidean distance is the straight-line distance between the pixel and the pixel of value 0 closest to it. Then the Euclidean distance map $D=\{d_{i,j}\}$ can be defined as follow:

$$d_{ij}=\min_{1\le p,q\le N}\{\sqrt{(i-p)^2+(j-q^2)}\mid b_{pq}=0\} \quad (1)$$

EDT has a high time complexity. For the order-communication algorithm, as the size of image increases, the computing time will increased nonlinearly and the real-time performance is poor. So the linear EDT algorithms are proposed. One is the independent-scanning algorithm proposed by Hirata, the other is the algorithm based on the construction of the Voronoi diagram. Their time complexity are both $O(N^2)$.

B. Hardware algorithm selection and analysis

A hardware algorithm need to take advantage of parallel computing and get rid of complicated calculation which is not practical for hardware. Compared with the algorithm based on the construction of the Voronoi diagram, the independent-scanning algorithm can be separated into relatively independent stages with few complex calculation. The hardware algorithm is based on the independent-scanning algorithm.

The basic idea of independent-scanning algorithm is to divide the EDT into two relatively independent stages. In the first stage, $T1$ calculates column distances $G=\{g_{ij}\}$ for an $N\times N$ binary image $B=\{b_{ij}\}$. In the second stage, $T2$ will add the squared values of row distance to the corresponding ones of column distance and obtain the Euclidean distance expressions. Draw the expressions as parabolic functions of row coordinates, then the lower envelope of these functions gives the Euclidean distance for each pixel of the row. These two stages have a sequential hierarchical relationship without direct correlation between the data. The characteristics are useful for realizing the hardware pipeline architecture.

The column distance can be calculated through the forward scanning and reverse scanning in $T1$. Its time complexity is $O(N^2)$. Calculating single pixel is changed to calculating row coordinates of the lower envelope in $T2$. The time complexity is also $O(N^2)$. So the whole time complexity of independent-scanning algorithm is $O(N^2)$.

III. THE IMPROVEMENT FOR HARDWARE

The independent-scanning algorithm has the advantage of independent stages to operate parallel calculation. However, it has certain limitations in hardware realization, so it is necessary to improve the algorithm according to the characteristics of the hardware design. In accordance with the independent-scanning algorithm, the improved hardware algorithm is also divided into two stages, $T1$ and $T2$.

A. The improvement on $T1$

In the independent-scanning algorithm, the column distance is the smaller one of the forward scanning result and the reverse scanning result. The calculating process is showed in Fig. 1. In Fig. 1, 'U' means that the column distance is undefined in the scanning. In the actual processing, 'U' can be replaced by a reasonable maximum. The data width of the column distance increases as one dimensional size of the image increases. This will take a lot of system storage resources. Furthermore, the method consists of the forward and reverse scanning and has a complicated calculation process. This will degrade the real-time performance.

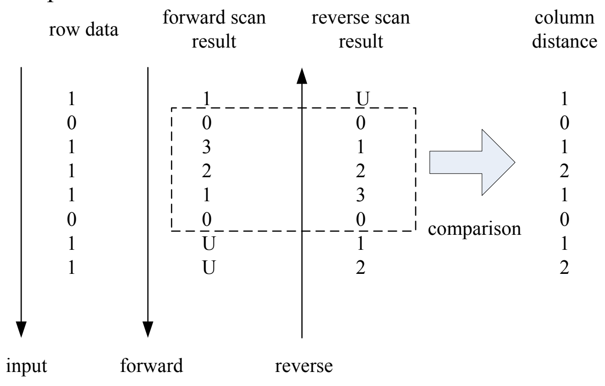

Figure 1. The procedure of bidirectional scans.

In order to reduce the time complexity, the bidirectional scans can be replaced by the unidirectional scan. Its queue structure is showed in Fig. 2. In the structure, each transfer cell has the construction shown in Fig. 3. Each rectangle represents a register. Registers $MB1$ and $MB2$ are 1-bit registers used for storing the pixel value. Register $DIFF$ is a 2-bit register used for storing the column distance increment. In the processing of row data, the column distance increment of the data before the first pixel of value 0 is set to -1, and the column distance increment of the data after the last pixel of value 0 is set to $+1$. For the pixels between the two pixels of 0 shown in the dashed box of Fig. 1, we observe the fact that the first half of these pixels get their column distances by the forward scanning and the others get the column distances by the reverse scanning. So we can distinguish the latter half pixels and modify the corresponding column distance increments by $MB1$ and $MB2$.

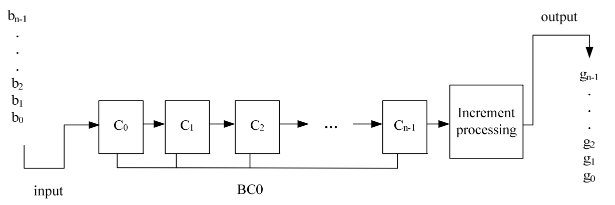

Figure 2. The structure of unidirectional scan.

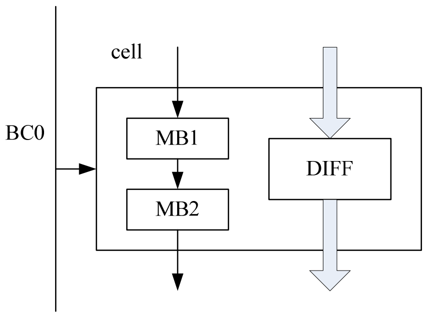

Figure 3. The structure of a transfer cell.

The unidirectional scan for a N-pixel row is showed in Fig. 4. First, the initial column distance value of increment processing cell is set to N. For all transfer cells, $MB1$ and $MB2$ are set to 1 and $DIFF$ is set to 0. The default column distance increment value of the input data is set to $+1$. When the input data is 0, the broadcast signal $BC0$ is set to be valid and the column distance increment of the current transfer cell is set to -1. Then the other transfer cells will modify their column distance increments according to the value of $MB1$ and $MB2$ when $BC0$ is valid. The modification rule is showed as follow:

$$DIFF = \begin{cases} -1, (MB1 = 1, MB2 = 1) \\ 0, (MB1 = 1, MB2 = 0) \\ unchanged, (MB1 = 0, MB2 = 0) \end{cases} . \quad (2)$$

After modification, $MB1$ and $MB2$ of all transfer cells will be set to 0. Then the column distance increments will be transferred to the increment processing cell to obtain the column distances.

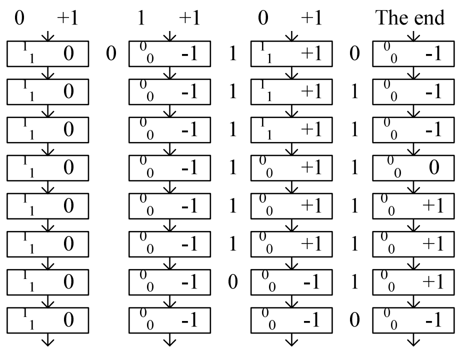

Figure 4. The procedure of unidirectional scan.

By the improvement of scanning method, calculating the column distance is changed to calculating the column distance increments to the adjacent pixels. The column distance increment value can be $+1$, 0 or -1, so only two-bit memory will be used to store the column distance increments. It reduces the cost of the system storage resources.

B. *The improvement on* $T2$

The computation of $T2$ is to do column scanning for B. In $T2$ stage, the Euclidean distances are described as parabolic functions of row coordinates, then the lower envelope of these functions gives the Euclidean distance for each pixel of the row. In order to simplify the calculation of the intersections, the Hirata algorithm eliminates the row squared items of the original expressions and calculates the intersections for straight lines for further calculation. In this way, the nonlinear operation is converted to linear operation. Then in the lower envelope consisted of the intersections and parabolas, the Euclidean distance squared values can be calculated according to the corresponding row coordinates.

Due to the discrete characteristic of the binary image, all the parabolas can be discretized according to the row coordinates. By comparing the values of every discrete parabolic function in the row coordinate, the minimum value is the Euclidean distance squared value.

According to the parallel processing ability of the circuit and hardware algorithm features, taking the hardware architecture into consideration, the improvement of $T2$ is described as follow:

(1) The row distance squared value is obtained by accumulation level by level. The squared value of row distance for each level can be expressed as $(N-1)^2 + (2N-1)$ and N is the number of the level. When the input of $T2$ is the squared value of column distance, in the first level of $T2$, N is 1 and the row distance squared accumulation value is 1. In the second level, N is 2 and the row distance squared accumulation value is 3. The row distance squared accumulation value is $(2N-1)$ when the level number is N. This method can effectively reduce the bit wide of the adder input and reduce the consumption of system storage resources.

(2) The minimum distance squared value can be obtained by comparison level by level. In the N th level, the row distance squared value is N^2 by accumulation level by level. So the minimum distance squared value of the corresponding pixel is the Euclidean distance squared value of the pixel. By comparison level by level, the output is the Euclidean distance squared value of the pixel.

The hardware architecture of $T2$ based on the above two ideas is showed in Fig. 5. The arrows in the figure indicate the data flow of the comparison results. The value of the rectangle in the arrow connection is the row distance squared accumulation value for the next level.

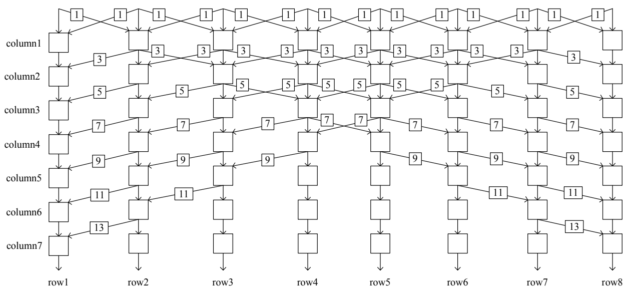

Figure 5. The structure of $T2$ procedure (for an 8×8 binary image).

Since the output of $T2$ is the distance squared value, the increment processing cell in $T1$ need to be adjusted to output the column distance squared value for hardware simplification. The binary image data are independent. The row distances and column distances are separately calculated. By taking the advantage of these two characteristics, if we use p processing units for parallel computation, the time complexity can be reduced to $O(N^2/p)$.

IV. HARDWARE SYSTEM ARCHITECTURE DESIGN

Based on the improvement of the independent-scanning algorithm and the design of hardware structure, a complete hardware calculation system can be designed. We can just connect $T1$ and $T2$ directly to form a complete hardware system shown in Fig. 6. The architecture of $T1$ is showed in the dashed box of Fig. 6.

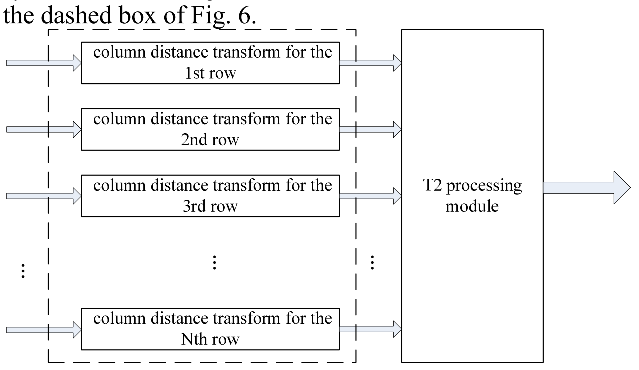
column distance transform for the 1st row
column distance transform for the 2nd row
column distance transform for the 3rd row
column distance transform for the Nth row
T2 processing module

Figure 6. The original architecture of computing system.

In this architecture, a frame of image pixels has to be read row by row for EDT. So in the hardware system shown in Fig. 6, the squared values of column distance are calculated for each row, that is the calculation of $T1$. After the calculation of $T1$, the hardware system enters the processing of $T2$. The processing of $T2$ is based on accumulation and comparison level by level, so the calculation of $T1$ must be stopped at the same time for the sake of correctness. If $T1$ and $T2$ are at work at the same time, the output of $T1$ will influence the processing of $T2$ and cause unexpected errors. But in the original independent-scanning algorithm, $T1$ and $T2$ can work at the same. Obviously, $T1$ and $T2$ of the hardware system architecture shown in Fig. 6 work in a time-shared method. It is ineffective.

In order to improve the efficiency of the hardware system, the hardware system architecture is adjusted. In the new architecture shown in Fig.7, $T1$ consists of two data queue. One is the transmission queue of column distance increment, the other is the storage queue of column distance increment. The column distance increment storage queue will be refreshed in the control of the frame synchronization signal, $Flag$ .When a new frame comes into the system, the system give a $Flag$ signal. Then in $T1$, the column distance increments will be transferred to the storage queue of column distance increment from the transmission queue of column distance increment. The transmission queue of column distance increment will be ready for the new frame. At the same time, $T2$ and the storage queue of column distance increment can be worked for the previous frame. All these above forms pipeline architecture. The adjusted architecture of $T1$ is showed in the dashed box of Fig. 7.

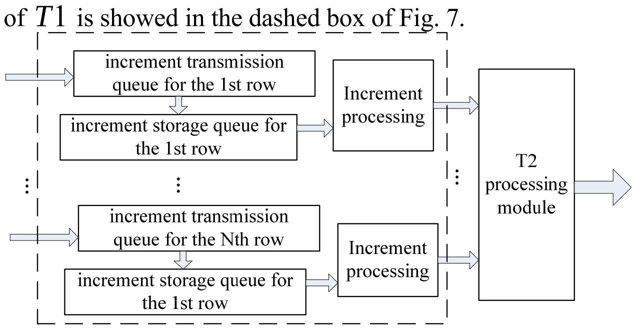
increment transmission queue for the 1st row
increment storage queue for the 1st row
Increment processing
increment transmission queue for the Nth row
increment storage queue for the 1st row
Increment processing
T2 processing module

Figure 7. The optimized architecture of computing system.

V. SOME DESIGN DETAILS AND ANALYSIS

Some design details are listed as follow based on an 8×8 hardware system.

(1) The layout of the pipeline architecture

As shown in Fig. 2, Fig.5 and Fig. 6, for a single row, $T1$ consists of 8 transfer cells and 1 increment processing cell. The number of calculation cells in $T2$ is 7. To improve the efficiency, $T1$ and $T2$ should work at the same time to form the pipeline architecture. $T1$ is the first stage. All transfer cells in $T1$ and $T2$ form the second stage. In this architecture, the number of cells in the first stage is 9, and the number of cells in the second stage is 16.As there is no multiple relation between the two numbers, it is hard to arrange clocks for different stages. Because there is no need to calculate column distance squared value in the first stage, the increment processing cell can be omitted. In this way, the number of cells in the first stage is 8, and the number of cells in the second stage is 16. Then the number of cells in the second stage is twice as that in the first stage.

(2) The signal, $Flag$ and clock adjustment

Before processing a new frame, the increment processing cell need to be initialized for a correct calculation. So a frame synchronization signal, $Flag$ is used to let the increment processing cell initialize. At the same time, $Flag$ can be the sign of the increment transfer procedure and notifies the stage switch. In order to finish the process of the previous frame before $Flag$ arrives, the clock frequency in the second stage must be twice as that in the first stage.

(3) Replace the multiplier by the adder and shift register

As analyzed in 3.1, the input of increment processing cell is the column distance increment and the output is the column distance squared value. So the multiplier is need in the increment processing cell. The multiplier will not only degrade the real-time performance, but also occupy a lot of logic cells. For optimal performance, the multiplier can be replaced by the adder and shift register. Suppose the column distance and the squared value of the current pixel are m and m^2, then the column distance of next pixel may be $m-1$, m or $m+1$, the corresponding squared value may be $(m-1)^2$, m^2 or $(m+1)^2$. As $(m\pm1)^2 = m^2 \pm 2m + 1$, the column distance squared value of next pixel can be obtained by adding $(2m+1)$ or $(-2m+1)$ to that of the current pixel. In the expression, $2m$ can be obtained by moving m one bit to the left. So the multiply operation can be replaced by one shift operation and two add operations. The design has improvement on both real-time performance and hardware resources occupancy.

(4) The limitation to the output of the increment processing cell

Suppose the image scale is $N \times N$. For a single row, the column distance initial value in the increment processing cell is N. When there are more than one continuous pixels of value 0 in the row pixels, the column distance in the increment processing cell will be 0 for the first pixel of value 0. Because the register can only store unsigned number, the succeeding pixels of value 0 will produce continuous -1 increments. Then the underflow will happen. When the row pixels are all 1-pixels, all the increments will be $+1$, and the column distance will be beyond the reasonable range. These two conditions will cause unexpected errors. So some check and correction has to be done before the output. An extra 1-bit sign bit register is added as the high bit for the column distance, so the underflow can be checked by the register. Once underflow happens, reset the sign bit register and set the column distance to be 0. Set $2(N-1)^2$ to be the threshold value. When the column distance is beyond the reasonable range, the output will be set above the threshold value.

We establish an 8×8 hardware system on the EP2C5Q208C8 development board and finish the verification. To display the timing diagram clearly, the waveform of time order simulation for a 4×4 hardware system is showed in Fig. 8.

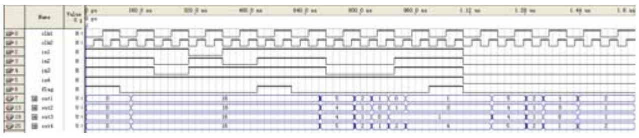

Figure 8. The waveform of time order simulation for computing system.

In Fig. 8, clk1 is the clock of the first stage and clk2 is the clock of the second stage. The rising edge is valid for the system. The in1, in2, in3 and in4 are the inputs of the row pixels (1101, 1010, 1011, 1111). The flag are the frame synchronization signal. The out1,out2,out3 and out4 are the output of the squared values of Euclidean distance. At the first clk1 clock, flag is set to 1 and all cells in the system are initialized. Then flag is set to 1 and the column distance increments are obtained after 4 clk1 clocks. When the 6th clk1 clock arrives, the second frame will come. So set flag to be 1 and initialize the first stage for the second frame. At the same time, the column distance increments are transferred to the storage queue of column distance increment and the second stage begins to calculate for the first frame. During the following 4 clk1 clocks, the column distance increments of the second frame are obtained by the first stage and the Euclidean distance squared values of the first frame are obtained by the second stage. The output of out1, out2, out3 and out4 are the squared value of Euclidean distance (2101, 1010, 1011, 2124). The average computing time per frame is $(2N+1)$ clk1 clocks as shown in the figure, and N is the one dimensional size of the image.

In order to evaluate and analyze the relation between hardware resources occupancy rate and image scale, we build six hardware systems in the EP2C5Q208C8 development board. The one dimensional size of the systems ranges from 4 to 10. The relation between the resources occupancy rate and the image scale is showed in Fig. 9. We can observe that the resources occupancy rate is not in strictly linear relation with the image scale. Along with the growth of image scale, not only the quantity of the processing cells grows, the bit width of the processing cells and other memory will also increase gradually. It leads to the nonlinear relation between the resources occupancy rate and the image scale.

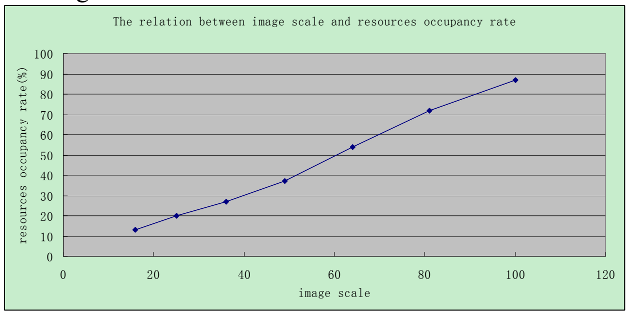

Figure 9. The relation between resources occupancy rate and image scale.

VI. CONCLUSIONS

A hardware algorithm for the Euclidean distance transform has been proposed in this paper. The algorithm consists of the row scanning $T1$ and the column scanning $T2$. Due to the introduction of signal $BC0$, the algorithm performs the unidirectional scan instead of the bidirectional scan, and this improves the computing time. Also, the multiplier has been replaced by the adder and shift register in $T2$. With the adjustment to the system architecture, the hardware system works in the pipelined way and computes the Euclidean distance map for a $N \times N$ binary image in $(2N+1)$ clocks. It is a great improvement on the computing speed.

ACKNOWLEDGMENT

The authors would like to thank anonymous reviewers for their critical comments, which help to improve the presentations of this paper.

REFERENCES

[1] A. Rosenfeld and J. Pfaltz, “Sequential Operations in Digital Picture Processing,” Journal of the ACM, vol. 13, pp. 471–494, Oct. 1966.

[2] P. Danielsson, “Euclidean Distance Mapping,” Computer Graphics and Image Processing, vol. 14, pp. 227–248, Nov. 1980.

[3] I. Ragnemalm, “Neighborhoods for Distance Transformation Algorithm Using Ordered Propagation,” CVGIP: Image Understanding, vol. 56, pp. 399–409, Nov. 1992.

[4] M. Kolountzakis and K. Kutulakos, “Fast Computation of Euclidean Distance Maps for Binary Images,” Information Processing Letters, vol. 43, pp. 181-184, Sep. 1992.

[5] H. Breu, J. Gil and D. Kirkpatrick, “Linear Time Euclidean Distance Transform Algorithms,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 17, pp. 529-533, .May 1995.

[6] T. Hirata, “A Unified Linear-Time Algorithm for Computing Distance Maps,” Information Processing Letter, vol. 58, pp. 129–133, May 1996.

[7] Y. Lee, S. Horng and T. Kao, “Parallel Computation of the Euclidean Distance Transform on the Mesh of Trees and the Hypercube Computer,” Computer Vision Image Understanding, vol. 68, pp. 109-119, Oct. 1997.

[8] L. Chen and H. Chuang, “An Efficient Algorithm for Complete Euclidean Distance Transform on Mesh-Connected SIMD,” Parallel Computing, vol. 21, pp. 841-852, May 1995.

[9] D. Paglieroni, “A Unified Distance Transform Algorithm and Architecture,” Machine Vision and Applications, vol. 5, pp. 47-55, 1992, doi: 10.1007/BF01213529.

[10] L. Chen and H. Chuang, “Designing Systolic Architectures for Complete Euclidean Distance Transform,” The Journal of VLSI Signal Processing, vol. 10, pp. 169-179, 1995, doi: 10.1007/BF02407034.