Jestr

Journal of Engineering Science and Technology Review 1 (2008) 19-24

Research Article

JOURNAL OF
Engineering Science and Technology Review

www.jestr.org

Hardware principles for the design of a stereo-matching state machine based on dynamic programming

J. A. Kalomiros*,[1] and J. Lygouras[2]

[1] *Department of Informatics and Communications, School of Technological Applications, Serres Institute of Education and Technology, Terma Magnisias, 62100, Serres, Greece*
[2] *Section of Electronics and Information Systems Technology, Department of Electrical Eng. & Computer Eng., School of Engineering, Democritus University of Thrace, Xanthi, Greece*

Received 10 December 2007; Accepted 15 January 2008

Abstract

This paper presents basic design principles for hardware implementation of a two-pass stereo-matching algorithm based on dynamic programming. For the first-pass a state-machine is proposed for the recursive calculation of the cost-function. The state-machine works along the diagonal of a 2-D disparity space for each epipolar pair of image scan-lines. On-chip local RAM stores tags that denote the minimum transition cost to every point in the disparity space among possible costs from all three neighboring points. All calculations are within a pre-determined useful disparity range. For the second pass, hardware rules are presented that produce the correct disparity per pixel, by backtracking stored cost values. Hardware stages are structured along a fully parallel pipeline, that outputs disparities in step with the input serial pixel stream at clock rate.

Keywords: Hardware Design, Stereo Vision, Dynamic Programming

1. Introduction

Dynamic programming is a general mathematical method that can reduce the complexity of optimization problems, by decomposing them into simpler and smaller problems [1]. It can be used in order to handle the stereo-correspondence problem by finding an optimized solution for the whole scan-line. In this sense the problem is solved globally, as opposed to local methods that find correspondences between local blocks [2,3]. In general, global methods produce better results and can resolve problems like lack of texture and occlusion [4,5]. However, such methods suffer because of their considerable computational cost.

Stereo vision dynamic programming exploits two geometrical constraints that apply to depth calculation by two parallel cameras. These are the epipolar constraint [2] and the monotonic ordering constraint [6]. The first means that in rectified stereo pairs the search for correspondences between stereo frames can be limited only to epipolar scan-lines. The second means that if a correspondence between two points A and B is established, then the next point on the right of A in one image can only correspond to a point to the right of B in the other image. This idea is shown in Fig. 1. It should be noted however, that the relative order of pixels between the two views is not always the same in real scenes.

The main idea behind a dynamic programming algorithm for stereo matching is to build a Disparity Space Image (DSI), attributing a cost value to all disparities and establishing best global disparities by backtracking the optimal path [7]. The cost of optimal path is the sum of the costs of the partial paths obtained recursively.

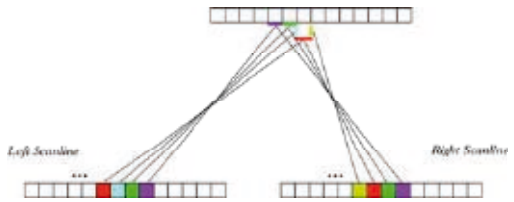

Fig. 1 The monotonic ordering constraint in stereo-vision. Finding the correspondence between two conjugate points in a stereo-pair means that all other correspondences lie to the right of the two points in both left and right images.

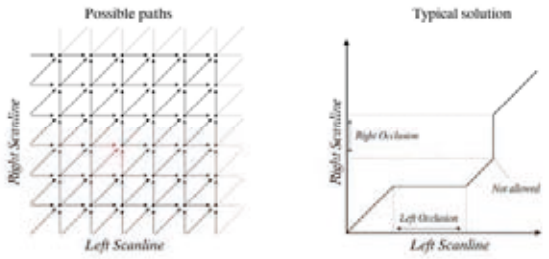

Fig. 2 Minimizing path (on the right) is determined by calculating recursively a cost function. Each point in the cost function (on the left) is derived from the three previous values, as shown by the arrows. Disparities are tallied in a next stage, by backtracking the path of minimum cost.

* E-mail address: ikalom@teiser.gr
ISSN: 1791-2377

One way to build the Disparity Space Image is to define x and y axis as the left and right scan-lines and then calculate the minimum cost path from the lower left corner to the upper right corner of the Disparity Space [8,9]. One always proceeds from left to right, as a result of the ordering constraint. This procedure is shown in Fig. 2, where each point of the 2-D cost function is determined by taking into account the previously calculated cost values of the three neighboring points to the left. Finding the correct disparities is now akin to finding the path in DSI which takes the shortest route through the cost values. Also, special rules can be added on how to transverse the search space in order to handle occlusion.

This paper presents some considerations about implementing parts of the dynamic programming stereo vision algorithm in reconfigurable hardware. Solutions are indicated to the main obstacles posed by the recursive nature of the algorithm. First, a state-machine is proposed, which is able to produce part of the Disparity Space plane at clock rate, in parallel with the pixel input stream of the left and right image. Also a RAM-based backtracking scheme is proposed that works at second pass and in step with the next scan-line input. These considerations can result in a real-time fully parallel hardware pipeline that produces disparities for a dense depth map at clock-rate.

2. Summary of the algorithm

Let us consider two scan-lines $I_l(i)$ and $I_r(j)$, in the left and right image, with $1 \le i,j \le N$, where N is the number of pixels in each line. While building the cost plane, pixels on each scan-line may be *matched* or *skipped* (considered to be occluded in either the left or right image). Therefore, two kinds of measures are considered. The first is the measure of matching two pixels *i, j* on the left and right scan-line respectively. The other is the measure of pixel *i* **not** matching pixel *j*, or in other words pixel *j* in the right scan-line being skipped in the search for a matching pixel for *i*. Correspondingly, pixel *i* could be skipped while looking for a matching pixel for *j*.

Let s_{ij} be the measure associated with matching pixel $I_l(i)$ with pixel $I_r(j)$. A squared error metric is considered between pixels given by:

$$s_{ij} = \frac{(I_l(i) - I_r(j))^2}{\sigma^2}, \quad (1)$$

where σ represents pixel noise. The measure of skipping a pixel in either scan-line is given by a constant *occl*. We take *occl*=0.2 and $\sigma = 0.1$.

Taking these measures into account, the minimizing alignment cost of two scan-lines can be calculated, according to the following recursive procedure:

a. $C(i,0)=i \times occl$ (2)

$C(0,i)= i \times occl$ (3)

$C(1,1)=s_{11}$ (4)

b. $C(i,j)=\min\{C(i-1,j-1)+sij, C(i-1,j)+occl, C(i,j-1)+occl\}$ (5)

Eqs. (2) and (3) calculate initial costs on the x and y axis. Eq. (5) defines the cost at each point of the DSI plane as the minimum between matching cost and left or right occlusion cost. For a particular point in the cost-plane, the cost of matching pixels *i,j* comes from adding s_{ij} to the left diagonal cost, while the costs of left or right occlusion come from adding the occlusion constant to the previous cost on the horizontal and the vertical (Fig. 2). In this way, the cost of the optimal path is the minimum sum of the partial costs. The total cost of matching two scan-lines is *C(N,N)*.

c. Depending on which of the three values in the parenthesis wins (see eq. (5)), a respective tag value 1,0,-1 is attributed to point *(i,j)* of the disparity space image. Intermediate values are stored in a matrix *M* with dimensions *NxN*. Given *M*, the optimal alignment and consequent disparities are found by backtracking.
d. Backtracking is defined as follows. Starting at *(i,j)=(N,N)*, corresponding stored values *M(i,j)* are examined. The case of *M*=1 corresponds to skipping a pixel in I_l and to a unit increase in disparity. The case of M=-1 corresponds to skipping a pixel in I_r and means a unit decrease in disparity, while M=0 matches pixels *(i,j)* and therefore leaves disparity unchanged. Beginning with zero disparity, the minimum cost path is followed backwards from *(N,N)*, and the disparity is tallied until point *(1,1)* is reached.

An efficient software implementation of dynamic programming can be found in Ref [10].

A variation of the above algorithm is used for our hardware considerations in the following paragraphs.

3. Hardware considerations

Designing with hardware all parallel calculations are ideally scheduled to be performed in step with the input stream of left and right image scan-lines. However the individual character of dynamic programming poses certain difficulties to such implementations. In particular, the recursive calculation of the cost function means that for each computation of a new subset of cost states, all necessary previous states need to have already been evaluated. Therefore, a proper course of calculations has to be established so that the computation of the cost plane proceeds without major errors. Also, an adequate slice of the cost-function along the optimal path has to be indicated, instead of the *NxN* total states, in order to render the hardware derivation of the cost-states a tractable task.

On the other hand, the need to backtrack disparities means that the final disparities can not be calculated in step with the stream of their respective scan-lines, but only during a second pass, while the next pair of epipolar scan-lines is already streaming through the hardware pipeline. There follows the need for local memory that can store cost values until backtracking calculations end.

A third consideration arises from the need to map the 2-D cost-function plane (see Fig. 2) upon a 1-D array of disparity values for each scan-line. Working with a software serial algorithm, the optimal path on the DSI-plane would be followed in any number of steps, from the last point backwards. Since the optimal path is not a straight line, *N* pixels of disparity would require in software more than *N* cycles of computation. However, an efficient hardware design would require strict timing specifications, computing in parallel all steps needed for the derivation of one pixel of disparity. In this way *N* disparity pixels for each *N*-pixel

scan-line would require exactly *N* steps into the hardware pipeline, during the second pass. Actually, the nature of the cost-plane allows us to consider that if the possible range of disparities is equal to D, then for the i_{th} output disparity pixel in any scan-line, a maximum of D calculation steps need to be performed into the i_{th} column of the DSI plane (see for example Fig. 7). In this sense, we need to establish hardware computation rules that can implement in parallel all possible D steps within each column. Such parallel computations in the i_{th} column should output the total disparity change from the previous pixel $(i\text{-}1)_{th}$, in one clock cycle.

Some computational hints towards the above challenges are given in the following.

4. Design of a state machine for optimal path computations

In this section a simple state-machine is proposed that allows the recursive computation of the next cost-states from the previous ones, at least within a slice of the DSI plane. For this purpose the Disparity Space is calculated along the diagonal, as is shown in Fig. 3. Our preliminary results are based upon a seven-state machine that produces at each computation step a total of seven cost-states on both sides of the diagonal. This principle can be extended for more states, by increasing appropriately the computing elements.

With reference to Fig. 3, it can be noted that starting from a known initial state it is possible to calculate all states on both sides of the diagonal and between an upper and lower limit, up to the end of the plane. This computation is performed at each step by setting as next initial state the one that was computed at the previous step. This feedback between the output and the input can produce an efficient mechanism that derives all states between given limits up to the end of the plane, with the same hardware stage. Since the DSI is based upon the left and right scan-line pixels, the state machine is designed in such a way that produces a new set of seven states with each input pixel pair. When both scan-lines have streamed through the pipeline, the state-machine produces the final upper states of the cost plane.

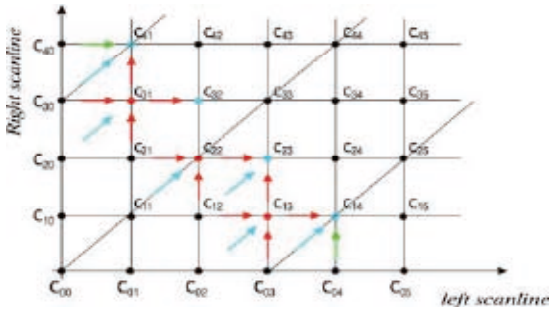

Fig. 3 Cost-function (DSI) plane along the diagonal

The seven states, shown in Fig. 3 as coloured points, are produced in two phases since they are found in two different levels in the cost-grid. Three states are computed first and four states follow in the next step. If the states c_{30}, c_{20}, c_{21}, c_{11}, c_{12}, c_{02}, c_{03} are given as input, then by taking three states together, as in Fig. 4, the state-machine computes the minimum cost of the transition to the states c_{31}, c_{22}, c_{13}. At the same time it passes the already known states c_{30}, c_{21}, c_{12}, c_{03} to the computation of the second phase (Fig. 4). At the second phase the remaining four states c_{41}, c_{32}, c_{23}, c_{14} are computed by using the three neighboring previous states. The calculation of the two limiting states of Fig. 4 (c_{41}, c_{14}) requires two marginal costs (shown in Fig. 3 as green arrows) that cannot not be derived from the actual set of the seven current states, but can be estimated approximately. Namely, the marginal costs are approximated by adding two times the occlusion cost to the last state on the diagonal. For example, the cost of the path from c_{04} to c_{14} is considered to be c_{03}+2x*occl*. This approximation is used for all the vertical and horizontal paths adjacent to the diagonal.

By setting as input the above seven states c_{41}, c_{31}, c_{32}, c_{22}, c_{23}, c_{13}, c_{14} we can produce in the following two steps the next seven states c_{52}, c_{42}, c_{43}, c_{33}, c_{34}, c_{24}, c_{25}. The computation continues up to the point of maximum cost *C(N,N)*.

It can easily be shown that the computation of the cost-plane can start at the known states c_{30}, c_{20}, c_{10}, c_{00}, c_{01}, c_{02}, c_{03} with respective values 0.6, 0.4, 0.2, 0.0, 0.2, 0.4, 0.6. These values are derived from eqs. (2,3), with occlusion constant *occl*=0.2 and are always the same. Starting with these values the state machine is going to produce c_{30}, c_{20}, c_{21}, c_{11}, c_{12}, c_{02}, c_{03} etc.

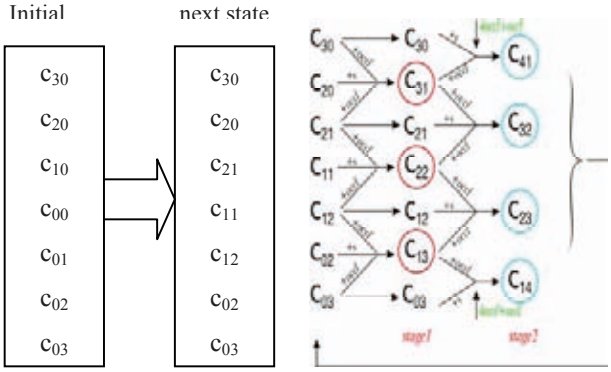

Fig. 4 Initial state and successive derivation of next states with a seven-state machine

Let it be noted here that starting from the triad c_{30}, c_{20}, c_{10} the three rival costs according to eq (5) are c_{30}+0.2=0.8 (where 0.2 stands for the occlusion constant), c_{20}+s=0.4+s (where s stands for the cost of eq. (1) with $s\geq 0$) and c_{10}+0.2=0.4. Obviously the minimum is 0.4 which stands for the state c_{20}. Similarly from the triad c_{01}, c_{02}, c_{03} the state c_{02} is derived. These are the launching steps of the proposed state machine.

The pixels of both epipolar scan-lines are inserted into the state machine as a stream of pixel pairs (left-right), each pair occurring simultaneously. The intensities are 8-bit, grayscale and are buffered appropriately in order to produce in parallel all necessary costs s_{ij}, according to eq. (1). This is used in eq. (5) to calculate the cost of the diagonal path from (i-1, j-1) point in the DSI to point (i,j). With reference to Fig. 3 the buffered intensities are *I1(1)*, *I1(2)*, *I1(3)*, *I1(4)* and *I2(1)*, *I2(2)*, *I2(3)*, *I2(4)*, where *I1* is the right scan-line intensities and *I2* are intensities on the left. In the parenthesis is the course of appearance of each pixel in the input stage. The cost of vertical and horizontal paths are computed by adding the occlusion constant to the previous value (see also Fig. 4).

The cost of diagonal, vertical and horizontal paths to each point are taken together and the minimum value is produced by an appropriate parallel-computation stage. Tag values are attributed to all three possible paths. A tag "1" is

attributed to the vertical path, tag "0" is for the diagonal path, while a tag value "-1" is attributed to the horizontal path. Wining tags at each point are stored and used during backtracking, according to rules measuring the change in the disparity value per pixel.

5. A numerical example

Let us consider two random test scan-lines, with eight pixels per line, each pixel 8-bit grayscale. The two "images" are shown in Fig. 5.

Image1:

105 153 88 92 120 54 207 222

Image2:

76 135 102 157 111 81 93 139

Fig. 5 Test scan-lines (8-bit, grayscale)

The rules of Section 2 are used in order to produce the 8x8 matrix of the cost function. The result is shown in Fig. 6. Also, the proposed state-machine is used in order to produce the cost function along the diagonal. These results are shown in Fig. 6 with colored numbers and are the same with the results of eqs. (1-5) within computation error. Successive outputs of the state machine are shown in red and blue. Total alignment cost is 1.315, at point *C(8,8)*. The respective matrix according to the wining tag-values (denoting vertical, horizontal or diagonal paths as corresponding to minimum cost) is shown in Fig. 7. Only the slice computed by the state machine is shown. Since the machine always produces seven states, some values flow over the boundary towards the end.

1.400	1.214	1.038	0.877	**0.715**	***0.915***	***1.115***	**1.315**
1.200	1.014	0.838	***0.714***	**0.914**	**1.114**	***1.314***	***1.513***
1.000	0.814	**0.638**	***0.838***	***1.038***	**0.963**	**1.163**	***1.363***
0.800	***0.614***	**0.438**	**0.638**	***0.838***	***1.038***	**1.238**	**1.438**
0.600	***0.414***	***0.614***	**0.814**	**0.975**	***1.175***	***1.375***	1.575
0.400	**0.600**	***0.800***	***0.821***	**1.021**	**1.167**	1.367	1.567
0.200	**0.400**	**0.600**	***0.621***	***0.821***	1.021	1.221	1.421
0.000	0.200	**0.400**	**0.600**	0.800	1.000	1.200	1.400

Fig. 6 Cost-plane produced with the scan-lines of Fig. 5. Red and blue results are successive outputs of the state-machine.

6. The backtracking stage

The wining tag values produced recursively at each point of the cost function are stored in RAM memory, making use of on-chip RAM blocks existing today in most reconfigurable devices. For this particular test-design seven RAM blocks are needed with a depth of N bytes, where N represents the length of scan-line. RAM is written during the first pass and is read at the second pass, during the backtracking stage. Since scan-lines are streaming, a second set of RAM blocks is also needed where M-values are stored at second pass, while the first RAM set is read.

During backtracking, the stored tag sets are read in reverse order, seven values at a time. After several clock counts each set is rearranged by means of delay lines, in order to give the columns of Fig. 7. In this way, we now have "running columns" in the pipeline. Each column corresponds to one pixel of disparity image, given that disparities are found with respect to the pixels of the left image. We begin backtracking at the top of the last column and apply a set of rules in order to find the entry point to the next running column, which is the first on the left, in Fig. 7. In short, the rule is that if entry in the *i-th* column is at the *q-th* member of the column-set from top ($1<q\leq7$) and is denoted *entry=q*, then the exit is at the first row to the bottom that does not contain "1", that is exit is at the first "0" or "-1". Let this be the *p-th* member of the set, measuring from top. In case of "0" the next column (*i-1*) is entered by moving diagonally down to the left. In this way we land again on the *p-th* member of the next column-set, since the vertical set steps down as we move towards the beginning of the plane (see Fig. 7). Naming *"exit"* the entry point to the next running column, we denote *exit=p*. In case the *i-th* column is exited on "-1" at *p*, then we step horizontally and enter the (*i-1*) column at *exit=p-1*. For example, with respect to the sixth column of Fig. 7 (*i=6*), we suppose that we enter the column on the 4-th member of the vertical set (*q=4, entry=4*). Since the M-value of the 4-th member is "0", we would immediately exit diagonally to enter the next left column on the 4-th member (*p=4*, *exit =4*). However, if we entered at the top of the sixth column, where we have the 2-nd member of the vertical set (*entry=2*), we would immediately step to the left on *exit=1*. Next, we would move to the fourth column diagonally on *entry=1* and would continue to the third column (again *exit=1*) to land on M-value="1". Now, *entry=1*, but we should exit on *p=2*, diagonally to *exit=2*. The last step is diagonally to the left, again with *exit=2*.

Yellow shadow and arrows in Fig. 7 indicate the path of minimum cost followed by backtracking.

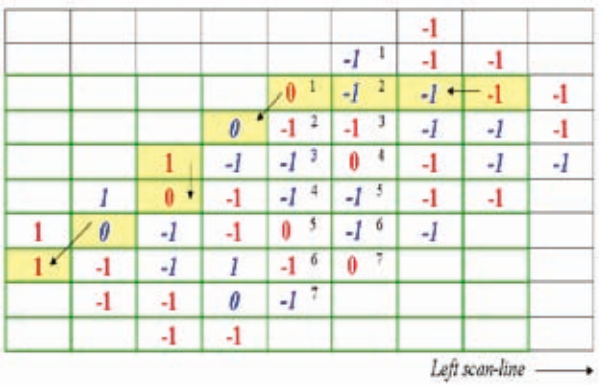

Fig. 7 Tag values (M-values), denoting horizontal, diagonal or vertical transition (-1,0,1, respectively) as corresponding to minimum cost. Green boundary corresponds to the 8x8 cost-plane. 7-member sets alternate in blue and red colour. Numbering from 1 to 7 represents the order of a particular M-state in the column. Arrows and yellow shadow indicate the path of minimum cost.

The above rules follow from point c of the algorithm in Section 2. They are designed in hardware with a number of arallel stages that find *"entry"* and *"exit"* at each running column in one clock cycle. Knowing the entry point to the

current i_{th} column and the exit point to the next (i-1) column, the total change of disparity at the i_{th} pixel can be calculated:

$$\Delta d = exit - entry. \quad (6)$$

Starting with $d=0$, the system tracks disparities adding Δd at each step.

7. Hardware system and results

The stages detailed above are integrated into a hardware system, shown in Fig. 8 as a block diagram. A model of the system was designed using Altera's DSP Builder, which combines Simulink design tools with VHDL design flow. DSP-Builder contains bit- and cycle- accurate blocks which cover basic digital operations as well as complex functions [11].

Great care has been taken in order to resolve timing issues. Writing and reading RAM memory is a central part in the overall procedure and has to be clocked with detailed accuracy. In spite of such difficulties the design is feasible and can reproduce accurately the disparities calculated by a serial algorithm based on software.

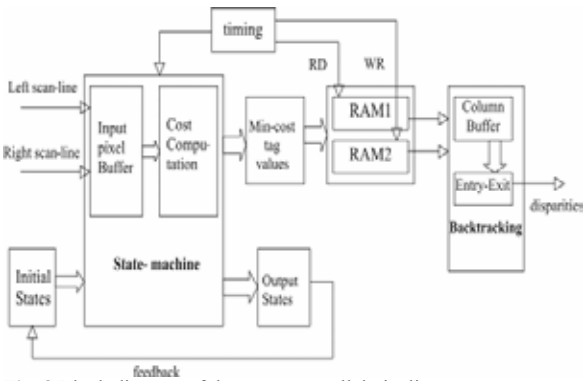

Fig. 8 Block diagram of the system parallel pipeline

Fig. 9 shows the results from the processing of the pair of scan-lines of Fig. 5. Two same scan-lines per image were used in order to check for timing consistency. It can be seen that both the software algorithm and the hardware model produce the same disparity values.

Preliminary synthesis results show that the above design can fit in a commodity FPGA chip, like the medium scale Cyclone II manufactured by Altera Co. or the Xilinx Virtex II series. Such devices are equipped with adequate on-chip storage capabilities that can accommodate the Random Access Memory (RAM) detailed in the previous paragraphs. They also possess embedded multipliers that suffice for the arithmetic operations needed for the state machine presented in section 4.

An advantage of the presented design is that images do not need to be stored in on-chip memory. Image scan-lines stream through the hardware pipeline and only a small part of them is buffered for the needs of the state-machine calculations. In this way memory requirements are limited to RAM needed for tag-storage. Almost any size of images can therefore be handled by the design. A large disparity-range is however demanding in resource usage, as it requires a more complex state machine, larger arrays for extracting the minimum costs and larger RAM for tag-storage.

It should be taken into account that FPGA fixed-point architecture is demanding with respect to hardware resources when high calculation accuracy is required.

The basic structure presented here is scalable and can expand to accommodate larger disparities. A state-machine able to produce approximately ten to sixteen states is thought to be adequate for real image applications.

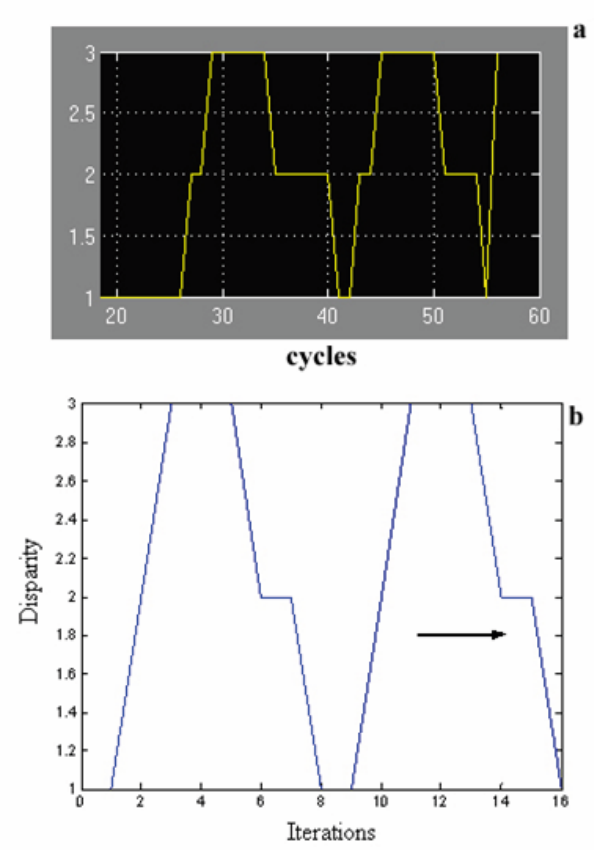

Fig. 9 (a) Disparity results from the model hardware design in Simulink. (b) Same scan-lines processed by a software serial algorithm.

8. Conclusions

This paper presents the basic principles for the design of a hardware system that can perform dense-depth calculations based on a two-pass dynamic-programming algorithm. A state-machine is proposed that can produce cost-states recursively, along the diagonal of the DSI plane. RAM blocks are used to store tag values and hardware-friendly backtracking rules are established. Model-based design is used in order to evaluate the system performance for small disparities and for small scan-lines. The first results show that the system is functional and can produce small disparities correctly.

Next steps in this project is to simulate the design for larger scan-lines and disparities and to use the resulting VHDL components in order to target a medium capacity FPGA chip. Some additional hardware design for I/O and system control is also needed. An efficient real-time hardware stereo machine can be implemented based on the principles presented above. Such an implementation can contribute to an already established tradition of accelerating computationally demanding image processing tasks with reconfigurable hardware [12,13,14].

References

1. T.H. Cormen, C.E. Leiserson, and R.L. Risvest, "Introduction to Algorithms", New York: McGraw-Hill, 1990.
2. M. Z. Brown, D. Burschka and G. Hager, "Advances in Computational Stereo", IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003, Vol. 25, No. 8, pp. 993-1008.
3. D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-frame Stereo Correspondence Algorithms", International Journal of Computer Vision, 2002, Vol.47, No 1-3, pp. 7-42.
4. P.N. Belhumeur, "A Bayesian Approach to Binocular Stereopsis", International Journal of Computer Vision, 1996, Vol. 19, No. 3, pp. 237-260.
5. S. Birchfield and C. Tomasi, "Depth Discontinuities by Pixel-to-Pixel Stereo", Proceedings of the IEEE International Conference on Computer Vision (ICCV), 1998, Mumbai, India, pp.1073-1080.
6. A.L. Yuille, and T. Poggio, "A generalized ordering constraint for stereo correspondence", A.I. Memo 777, AI Lab, MIT, 1984.
7. E.R. Davies, "Machine Vision: Theory, Algorithms, Practicalities", Elsevier, 3nd Edition, 2004.
8. Y. Ohta and T. Kanade, "Stereo by Intra- and Inter-Scanline Search using Dynamic Programming", IEEE Trans. On Pattren Analysis and Machine Intelligence, 1985, Vol.7, No.2, pp.139-154.
9. I.J. Cox, S.L. Hingorani, S.B. Rao,, and B.M. Maggs, "A Maximum Likelihood Stereo Algorithm", Computer Vision and Image Understanding, 1996, Vol. 63, No. 3, pp. 542-567.
10. S. Forstmann, Y. Kanou, J. Ohya, S. Thuering, and A. Schmitt, "Real-Time Stereo by using Dynamic Programming", Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'04), June 2004, Washington D.C., USA, p.29.
11. "DSP-Builder v. 7.2 Reference Manual", Altera Technical paper MNL-DSPBLDR-7.2, Altera Corporation, Oct. 2007.
12. A. Darabiha, J. Rose and W. J. MacLean, "Video-Rate Depth Measurement on Programmable Hardware", Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'03), June 18-20 2003, Vol. 1, Madison, Wisconsin, USA, p. 203-210.
13. J.A. Kalomiros and J. Lygouras, "Design and Evaluation of a Hardware/Software Architecture for Fast Image Processing", to be published in Microprocessors and Microsystems Journal, doi: 10.1016/ j.micpro. 2007.09.001.
14. M. Hariyama, Y. Kobayashi, H. Sasaki and M. Kameyama, "FPGA Implementation of a Stereo Matching Processor Based on Window-Parallel-and-Pixel-Parallel Architecture", IEICE Trans. Fundamentals, Dec. 2005, Vol. E88-A, No. 12, pp. 3516-3522.