Notice: This paper was presented at ASP-DAC 2015 but not by one of the authors.

Iterative Disparity Voting Based Stereo Matching Algorithm and Its Hardware Implementation

Zhi Hu[1], Yibo Fan[1*], Xiaoyang Zeng[1]
[1]State Key Lab of ASIC and System, Fudan University, Shanghai, China
[*]fanyibo@fudan.edu.cn

Abstract - Stereo matching is one of the key problems in computer vision. A large number of algorithms have been proposed but few of them achieve both high accuracy and short processing time on hardware. This paper presents a hardware-oriented stereo matching algorithm which is able to generate software-oriented-level results for 1920×1080 images at 48fps. Such performance prefigures new vistas of the applications of VLSI in stereo vision.

I. Introduction

Stereo matching is the corner stone of stereo vision. Generated disparity can be used to estimate the depth of every object in the image, making visual interactions with the real world feasible. Practical applications include free view point TV [1], robotic vision [2], automotive control [3] etc.

During the past decades, a number of algorithms implemented on different platforms have been proposed and roughly they can be divided into 2 groups, software-oriented and hardware-oriented. The former usually produce high quality results at the cost of long computational time due to complicated numerical computations and data structures.

Conversely, hardware-oriented algorithms aiming at real time processing often adopt straightforward strategies considering the constraints of limited resources and the direction of dataflow whereby in most cases generate incompetent results, being an obstacle to further effective processing. Many attempts have been made of building a hardware based stereo matching system. Jin *et al.*'s design[4] generates 64-level disparity map for a 640×480 image @230fps on XC4VLX200-10 FPGA; Zhang *et al.*'s design[5] produces 64-level map for a 1024×768 image@ 60fps on EP2SL150 FPGA; Jin *et al.*'s design [6] outputs 64-level map for a 640×480 image@507fps on XC6VLX240T FPGA. Even though substantial progress has been witnessed, there is much room for improvement: image resolution, quality of output disparities, and the consumption of hardware resources.

This paper proposes a hardware-oriented algorithm based on iterative weighted disparity voting and it possesses following advantages: 1) it produces results comparable to the software-oriented algorithms; 2) it is able to process 1080p images at low memory cost; 3) it can process 1920x1080 image streams at 48 fps, reaching the level of real-time processing, being a prospect of future 3D video processing.

II. Background and Related Works

The purpose of stereo matching is to extract horizontal disparities from a pair of largely-overlapping images, I_l and I_r, taken at the same level. In terms of methodology, prior algorithms can be divided into 2 groups, namely those adopting global and local matching methods. The former involves globally-optimizing strategies, for instance, graph-cut[7], belief propagation[8], dynamic programming[9] etc. These are memory resource exacting and time consuming intended for software implementation. Far from global strategies, local strategies focus only on the neighbouring pixels of current pixel being processed. Orderly data accessing and repetitive computation make it friendly for hardware and the proposed algorithm belongs to this category. A typical local matching method consists of following steps, though each step can be implemented in various ways. They are: 1) cost computation; 2) correspondence matching; 3) disparity refinement; 4) occlusion detection and filling.

Step 1 and 2 are instrumental to give an initial estimation of the disparity. The method of finding correspondence is to enumerate every candidate pixels p_{ri} in I_r within the disparity range ($0..d_{max}$) and choose the one with least matching cost between two pixels. Therefore we get, for a pixel p_l in I_l, its disparity can be expressed as:

$$d_{pl} = \arg\min_i \ \text{cost}(p_l, p_{ri}) \tag{1}$$

, where $p_l = I_l(x_l, y_l)$, $p_{r_i} = I_r(x_l, y_l - i)$ $(1 \le i \le d_{max})$. Different cost functions have been developed including census transform[10], pixel-based SAD (sum of absolute difference of luminance)[11], flexible-region-based SAD[12].

Step 3 is crucial since it largely determines the quality of the final result. The output of step 1 and 2 is a coarse disparity map with scattering misestimations. A non-content-based method is proposed by [4], which interpolates every disparity with its horizontally adjacent pixels using parabolic function without referring to the luminance of the input images. More effective methods are content-based, which refines every disparity considering the luminance of neighbouring pixels, such as bilateral filtering[13], its discrete simplification disparity voting [12], and the orthogonally decomposed version of voting[14].

Step 4 serves as detecting and extrapolating occluded disparities caused by perspective difference of 2 cameras.

III. Motivation

Most existent hardware-oriented local stereo matching algorithms are combinations of aforementioned strategies. They are inherently uncomplicated due to constraints of hardware resources. However, by optimization and iterative processing, artless methods can produce good results. Our motivation will be elaborated below within the framework of the processing steps of a local stereo matching algorithm.

A. Cost Computation and Correspondence Matching

Census transform is hardware-friendly but fails to be robust enough because small perturbations in a smooth region (with almost constant luminance) may lead to significantly different cost. Pixels-based SAD is computationally economical yet insufficient since values of neighboring pixels are ignored. Flexible-region-based gives good estimation but needs a 2D demarcation process to delineate the region which is incompatible with high-throughput pipeline structure.

To enhance the robustness and improve accuracy and meanwhile considering feasibility, a fixed-size window based cost function with luminance and gradient terms is adopted in this paper. The proposed method hardly increases the processing time, since all the arithmetic operations here are additions and subtractions that will necessarily appear also in the later weighted disparity voting stage. Besides, the arithmetic units will not increase drastically since adjacent N×N windows share (N-1)/N data which can be reused.

B. Disparity Refinement

Bilateral filtering has shown strong competence among various disparity refinement strategies. Some GPU-based implementations [11, 13] are able to process low-resolution images in real time by using so-called O(1) filter. Yet bilateral filtering is not an easy task for hardware. Most bilateral filters need floating point operations to ensure the high accuracy which are extremely resource-demanding for hardware when sufficient accuracy is demanded.

Disparity voting doesn't need much complicated computation and thereby being a good choice for hardware implementation. Nevertheless, [14]'s usage of voting only serves as a minor fix to the yet high-quality disparity map generated by dynamic programming. Actually, disparity voting can be far more useful than the [14] has exhibited; our improvements made to voting enable it to transform a coarse initial estimation into a high-quality disparity map.

The first improvement is weighted voting. In [14], each pixel contributes the same weight to the result. However we discover that results can be better if pixels with different disparity and distance to the central pixel are evaluated with different weights, given the error distribution on the initially estimated disparity map. Plus, weighted doesn't add much complexity and overhead to hardware resources.

The second improvement is iterative voting. It will be shown that the quality of the disparity map increases remarkably after every weighted voting and that after 6 times of weighted voting the result converges to a fine level. The main problem of iterative voting is huge memory consumption resulting from row buffers for pipelining. A slice-based processing scheme is proposed to reduce memory consumption and make the iterative voting feasible.

Thirdly, in [14] the vertical voting results are directly sent out so there is no data dependence between adjacent pixels. But if the results can be immediately used (updated) in the voting of successive pixels in the current stage, the disparity map is effectively voted for another time. A memory write-back strategy is developed in our work.

C. Occlusion Detection and Filling

Occlusion filling process fills the occluded pixel with smallest horizontally adjacent disparity, which is not a purely raster-order operation. Considering investigated researches haven't given a detailed description of the implementation, this paper proposes a ping-pong buffer based architecture to accomplish the task.

IV. Proposed Algorithm

A. Cost Computation and Correspondence Matching

For each pair of pixels, the cost is computed by two respective 3×3 windows centering on them. For the sake of compact architecture, the size of window is fixed, and the choice of size is a trade-off between memory usage and accuracy (actually, as will be shown in Fig. 4, voting will satisfactorily mitigate the misestimations resulting from small local window).Our definition of the cost is:

$$\text{cost}\,(p_l, p_r) = \sum \left| I_{pl} - I_{pr} \right| + \sum \left| \frac{\Delta I_{p_l}}{\Delta x} - \frac{\Delta I_{p_r}}{\Delta x} \right| + \sum \left| \frac{\Delta I_{p_l}}{\Delta y} - \frac{\Delta I_{p_r}}{\Delta y} \right| \quad (2)$$

, where I denotes the luminance of pixels in the window and $\Delta I/\Delta x, \Delta I/\Delta y$ denotes respectively the vertical and horizontal luminance difference. Due to the need for cross-check, for every pixel p_r in I_r, the disparity should also be determined by which the cross-check is done:

$$d_{pr} = \arg\min_i \quad \text{cost}(p_{li}, p_r) \quad (3)$$

, where $p_r = I_r(x_r, y_r)$, $p_{l_i} = I_l(x_r, y_r + i) \quad (1 \le i \le d_{max})$. The cost function is equivalent to Eq.(2).

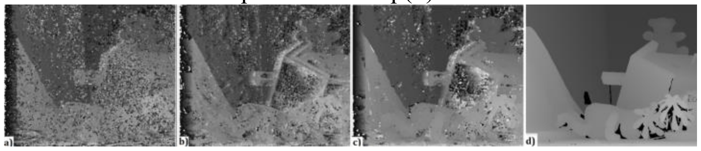


Fig 1. The result of initially estimated disparity map based on the cost computed by a) census transform; b) pixel-based SAD; c) proposed method. And d) is the ground truth (black areas are occluded areas).

The result of matching is shown in Fig. 1, comparing with the result of census transform and pixel-based SAD. Test image "Teddy" is obtained from [15]. Clearly, the proposed method outperforms the rest two, being more competitive for the ensuing disparity voting.

B. Iterative Weighted Disparity Voting

After correspondence matching, 3 rounds (generally after 3 rounds the result tends to stabilize, one test case is demonstrated in Fig. 4) of voting (each round is a vertical voting followed by a horizontal voting) are applied. The vertical voting, shown in the upper part of Fig. 2, proceeds as follows. For every pixel (scanned in a raster order) on the disparity map $p_0 = D(x, y)$, $\forall p_c \in \{ p \mid p = D(x+i, y) \}$ s.t. $\forall \text{channel}_X \left(\left| I_{\text{channel X}}(x+i, y) - I_{\text{channel X}}(x, y) \right| \le \tau \right), -r \le i \le r$ are the pixels for voting. We choose r = 10 to strike a balance between the capacity of error correction and the preservation of details. As for the criterion for pixel selection, $\forall \text{channel}_X \left(\left| I_{\text{channel X}}(x+i, y) - I_{\text{channel X}}(x, y) \right| \le \tau \right)$ (based on the assumption of the strong correlation between the luminance difference and disparity difference), we choose τ = 16 after tests. 3 channels are taken into consideration to prevent the unreliable selection resulting from low contrast in 1 channel; in implementation, the least 3 bits of RGB are discarded to cut down memory cost (we find quality of results only degrades slightly) and thus τ is set to $16/2^3=2$. After that, those selected pixels are sent to histogram accumulation for voting. The weight of each pixel $D(x+i, y)$ is defined as,

$$W_{D(x+i,y)} = \begin{cases} i/2 + D(x+i, y)/8 & i \ge 0 \\ -i + D(x+i, y)/8 & i < 0 \end{cases} \quad (4)$$

, where variable i denotes the distance to the central pixel. The purpose of adding an i term is to emphasize the pixels away from the central pixel. The reason is that in the initially generated disparity map wrongly-estimated disparities apt to agglomerate rather than scatter around evenly, as exhibited in Fig. 1-c. Give more weight to the peripheral pixels helps to amend areas filled with bad pixels; for areas devoid of bad pixels, emphasizing the peripheral pixels hardly affects the result since disparities of *selected* pixels barely change in a small region. And the reason for using *i*/2 for the upper pixels while using -*i* for the lower pixels is those upper pixels have just been updated thanks to write-back strategy so that they possess finer quality and deserve more weight. The $D(x+i,y)/8$ term is a minor adjustment to the weight to give less emphasis on pixels with extremely low disparity. Empirically, wrongly-estimated disparity values tend to be small, as shown in the dark areas of Fig. 1-c. However, for areas with almost constant low disparities (e.g. background), this term will not lead to an erratic result.

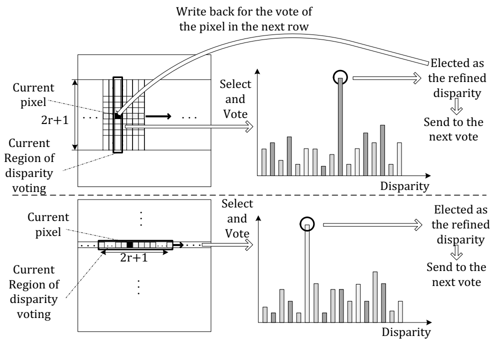

Write back for the vote of the pixel in the next row
Current pixel
2r+1
Current Region of disparity voting
Select and Vote
Elected as the refined disparity
Send to the next vote
Disparity
Current pixel
2r+1
Current Region of disparity voting
Select and Vote
Elected as the refined disparity
Send to the next vote
Disparity

Fig 2. Flow of disparity voting. The upper part is vertical voting and the lower part is horizontal voting.

After the weights of all pixels have been computed, they are accumulated into a histogram. Histogram consists of d_{max}+1 bins and each is accumulated according to Eq. (5) and then voted result can be obtained by Eq. (6).

$$H(i) = \sum_{\forall p \in \{p | D(p)=i\}} W_{D(p)} \tag{5}$$

$$D_{voted} = \max \ H(i) \ ,0 \le i \le d_{max} \tag{6}$$

Horizontal voting is at large similar to the vertical voting, as shown in the lower part of Fig. 2. $\forall p_c \in \{p \mid p = D(x, y+j)\}$ s.t. $\forall \text{channel}(|I_{\text{channel X}}(x,y+j) - I_{\text{channel X}}(x,y)| \le \tau), (-r \le j \le r)$ are selected for voting. The weight function, however, is slightly different from the vertical counterpart:

$$W_{D(x,y+j)} = |i| + D(i, y+j)/8 \tag{7}$$

.since the write-back strategy is not used in horizontal voting (discussed in the next section). The rest processing flow is equivalent to the vertical part.

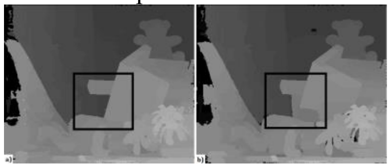


Fig 3. Comparison between a) weighted voting and b) non-weighted voting. Both disparity maps are generated after 3 rounds of voting, before occlusion detection and filling. Black squares emphasize the difference.

Fig. 3 shows the effect of weighted voting, which indicates weighted voting gives more accurate results, especially in terms of removing wrongly-estimated areas and delineating the exact contours of objects.

The voting process is not completed until 3 rounds of vertical and horizontal voting is done, i.e. V→H→V→H→V→H. Fig. 4 demonstrates step by step the remarkable improvement brought about by iterative voting.

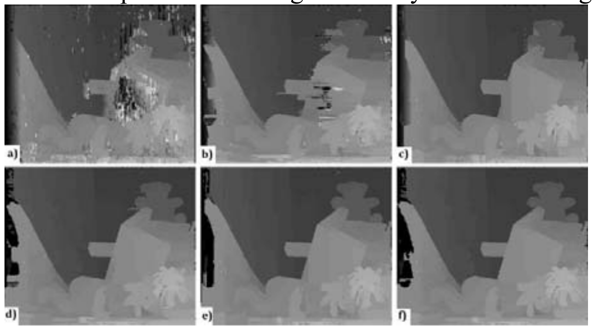


Fig 4. Disparity map after each vote. Image a) ~ f) respectively corresponds to the result of 1V (vertical voting in the 1st round), 1H (horizontal voting in the 1st round), 2V, 2H, 3V, 3H.

C. Occlusion Detection and Filling

Now 2 disparity maps are obtained, one generated with reference to the I_l according to Eq.(1) denoted as D_l and the other from I_r according to Eq.(3), denoted as D_r. For every pixel in D_l, it is non-occluded if

$$|D_l(x,y) - D_r(x, y - D_l(x,y))| \le \tau \tag{8}$$

To balance accuracy and robustness, we choose τ = 1.

For every labeled pixel, its disparity is extrapolated as the minimum horizontally adjacent non-occluded disparity. This operation is implemented by a two-pass row scan: firstly from left to right, if $D_l(x,y)$ is occluded, it is evaluated as the disparity of the previous (left) non-occluded pixel; then in the reverse order, if $D_l(x,y)$ is occluded, it is evaluated as the minimum of the currently filled disparity and the disparity of the previous (right) non-occluded pixel.

V. Hardware Architecture

Hardware architecture is established largely based on the algorithm with necessary hardware-oriented adaptations. An overall structure of the whole system is given in Fig. 5.

I_l and I_r are stored in 2 external memories, from where the 24-bit RGB values flow into the system. They are converted to 8-bit luminance and used to produce an initial coarse disparity map D_l and D_r in parallel. Meanwhile, truncated RGB values of pixels are delayed by a buffer to synchronize with the output of the correspondence matching module since they are concurrently needed by voting module. D_l and D_r are voted in parallel and then cross-checked. Finally the occluded pixels are filled.

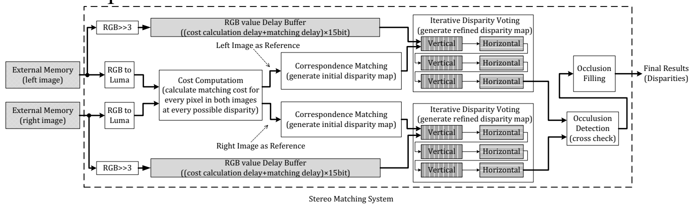
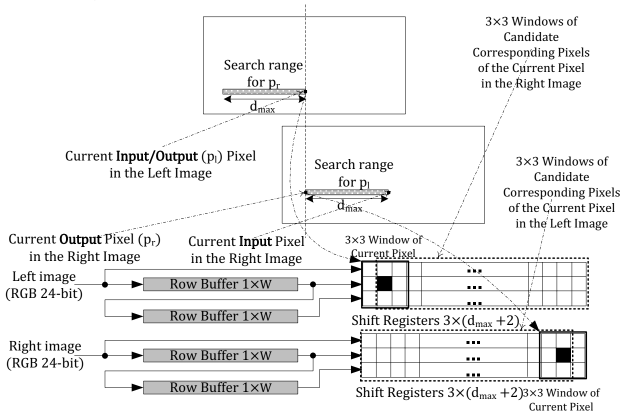

RGB>>3
RGB value Delay Buffer ((cost calculation delay+matching delay)×15bit)
Iterative Disparity Voting (generate refined disparity map)
Vertical
Horizontal
Left Image as Reference
External Memory (left image)
RGB to Luma
Cost Computatiom (calculate matching cost for every pixel in both images at every possible disparity)
Correspondence Matching (generate initial disparity map)
Occlusion Filling
Final Results (Disparities)
External Memory (right image)
RGB to Luma
Correspondence Matching (generate initial disparity map)
Iterative Disparity Voting (generate refined disparity map)
Occulusion Detection (cross check)
Right Image as Reference
RGB>>3
RGB value Delay Buffer ((cost calculation delay+matching delay)×15bit)
Stereo Matching System

Fig 5. Overall hardware architecture of stereo matching system.

A. Cost Computation and Correspondence Matching Module

The search range of every pixel of the I_l (or I_r) is determined respectively by Eq. (1) (or (3)). It can be deducted that the search range of the current pixel being processed of I_l

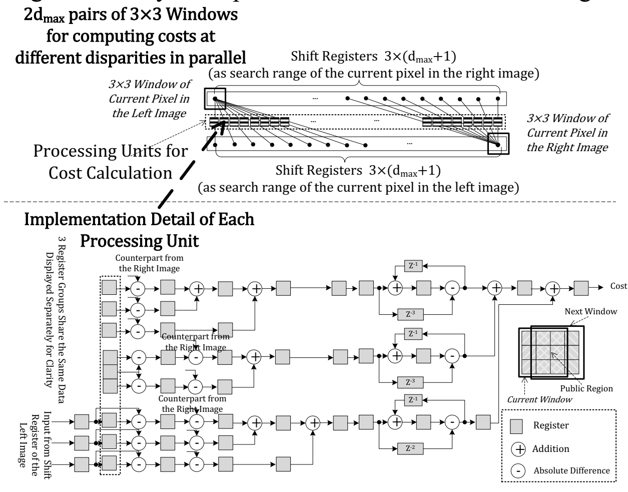
(denoted as p_l) extends to the current pixel from I_r (p_r) and vice versa (as shown in Fig. 6) only when the x-coordinates and the y-coordinates of the p_l and p_r are equal, which means that p_l and p_r can be processed without adding extra delay to either image source.



3×3 Windows of Candidate Corresponding Pixels of the Current Pixel in the Right Image
Search range for p_r
d_{max}
Current Input/Output (p_l) Pixel in the Left Image
Search range for p_l
d_{max}
3×3 Windows of Candidate Corresponding Pixels of the Current Pixel in the Left Image
Current Output Pixel (p_r) in the Right Image
Current Input Pixel in the Right Image
3×3 Window of Current Pixel
Left image (RGB 24-bit)
Row Buffer 1×W
Row Buffer 1×W
Shift Registers 3×(d_{max}+2)
Right image (RGB 24-bit)
Row Buffer 1×W
Row Buffer 1×W
Shift Registers 3×(d_{max}+2)
3×3 Window of Current Pixel

Fig 6. Design of window generation. The upper part shows the position of the windows in the whole image.

Each image has 2 row buffers, functioning as capturing pixels from 3 adjacent rows of same y-coordinate synchronously. Then they are moved into respective shift registers which as a whole serve as the search range of the current pixel from another image and from which 3×3 window of each pixel in range can be extracted straightforwardly. Computation scheme is detailed in Fig. 7.



2d_{max} pairs of 3×3 Windows for computing costs at different disparities in parallel
Shift Registers 3×(d_{max}+1) (as search range of the current pixel in the right image)
3×3 Window of Current Pixel in the Left Image
3×3 Window of Current Pixel in the Right Image
Processing Units for Cost Calculation
Shift Registers 3×(d_{max}+1) (as search range of the current pixel in the left image)
Implementation Detail of Each Processing Unit
3 Register Groups Share the Same Data Displayed Separately for Clarity
Counterpart from the Right Image
Counterpart from the Right Image
Counterpart from the Right Image
Input from Shift Register of the Left Image
Cost
Next Window
Public Region
Current Window
Register
Addition
Absolute Difference

Fig 7. Design of cost computation from a pair of windows. Upper part shows how $2d_{max}$ windows are paired up. Lower part demonstrates the structure of cost computing.

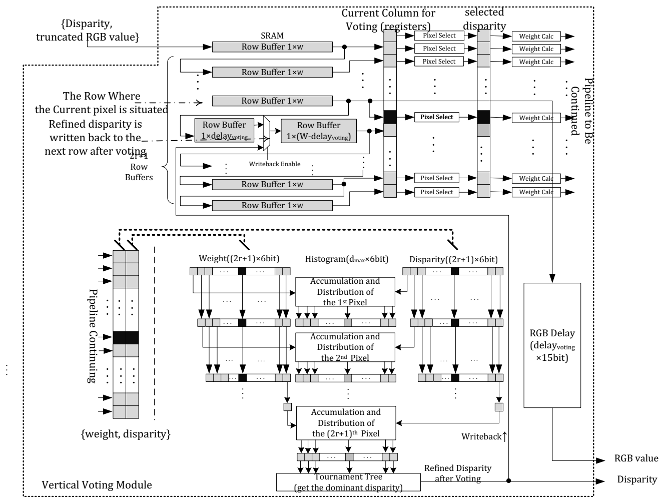
For each paired up pixels (windows) there is a processing unit (PU) to generate the matching cost between them so $2d_{max}$ costs can be computed at every clock cycle. A typical method is to extract data from 9 registers in the shift register group and then compute based on them. But since 2 adjacent windows overlap by 6 pixels (as is shown in the lower right part of Fig. 7) and that according to Eq. (2) the cost contributed by those 6 pixels is irrelevant to the position of the window. So part of the cost data can be reused. An improved structure is presented in the lower part of Fig. 7. At every clock cycle 3 pixels in a column are sent into the PU and the old cost is updated by adding three terms in Eq. (2) of newly-arrived 3 pixels, and subtracting these 3 values generated 3 clocks before.

Minimum Cost is then generated by a pipelined tournament tree in the correspondence matching module and meanwhile the disparity is generated.

B. Disparity Voting Modules

Among 6 sub voting process, 3 vertical/horizontal voting instances are structurally equivalent so only 2 different architectures, vertical and horizontal voting, are needed.

Vertical voting architecture is shown in Fig. 8. Pixels from row-buffers are selected following the aforementioned criterion and weighted according to Eq. (4). Thereafter the histogram is generated through the pipeline. At every stage, only 1 pixel is accumulated into the histogram considering the delay of combinational logic. The number of registers can be reduced by discarding accumulated elements after every stage. A tournament tree is used to pick out the dominant disparity in the histogram and send it to the next vote. RGB delay buffer serves as synchronizing RGB values with output disparity due to their presence in the next vote.



{Disparity, truncated RGB value}
SRAM
Current Column for Voting (registers)
selected disparity
Row Buffer 1×w
Pixel Select
Weight Calc
The Row Where the Current pixel is situated
Refined disparity is written back to the next row after voting
Row Buffer 1×$delay_{voting}$
Row Buffer 1×(W-$delay_{voting}$)
2r+1 Row Buffers
Writeback Enable
Pipeline to Be Continued
Pipeline Continuing
Weight((2r+1)×6bit)
Histogram(d_{max}×6bit)
Disparity((2r+1)×6bit)
Accumulation and Distribution of the 1st Pixel
Accumulation and Distribution of the 2nd Pixel
Accumulation and Distribution of the (2r+1)th Pixel
RGB Delay ($delay_{voting}$ ×15bit)
{weight, disparity}
Writeback
Refined Disparity after Voting
Tournament Tree (get the dominant disparity)
Vertical Voting Module
RGB value
Disparity

Fig 8. Design of vertical voting architecture.

The implementation of write-back strategy is also shown in Fig. 8. The width of the image (w) is far greater than the latency of voting for one pixel, so when one pixel has finished voting, its old disparity value has been moved to the next row buffer yet not shifted out again to vote for the pixel immediately below it. Therefore the newly updated disparity can be written back into the middle of the $(r+2)^{th}$ buffer, which is split into 2 segments to facilitate the operation. A multiplexor is used because the updated disparity is not available at the very beginning.

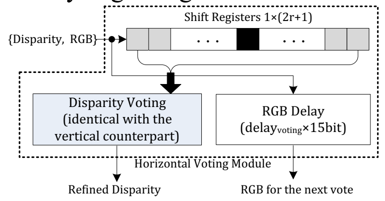


Fig 9. Design of horizontal voting architecture.

The implementation of horizontal voting demonstrated in Fig. 9, generally resembles the vertical counterpart except for the absence of write-back strategy. A set of shift registers are used instead of row buffers considering the horizontal input order. Write-back strategy is not used because when voting has completed, the updated disparity will not be used again during the current vote (latency > r).

Memory cost of vertical disparity voting is tremendous. An estimation can be made: 3 vertical voting processes, each requires a (2r+1)·w·(6+5×3)bits buffer (r=10;w=1920 for HD image;6 is the bit-width of disparity when d_{max}=63;5×3 stands for total bits of truncated RGB), therefore the total memory cost when the generation of D_r is also considered equals 21×1920×21×3×2bits=620kByte.

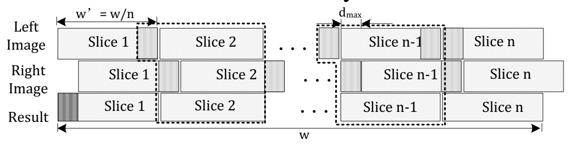


Fig 10. The concept of slice-based processing.

Actually when w is reduced, the memory cost is reduced proportionally. This inspires us to divide the input image into slices vertically generating sub-images with smaller width. But the concern is how the correctness of slice-based processing can be ensured. In fact the disparities of the leftmost $h×d_{max}$ area of I_l cannot be generated with full search range, so for every slice there is such a "unreliable zone", which degrades the quality of the final result. The resort is to extend the all the slices in I_l and I_r by d_{max} pixels so that reliable disparities can be produced from the otherwise unreliable zones. New "unreliable zones" brought about by extension can simply be discarded when disparity slices are finally appended together, as shown in Fig. 10.

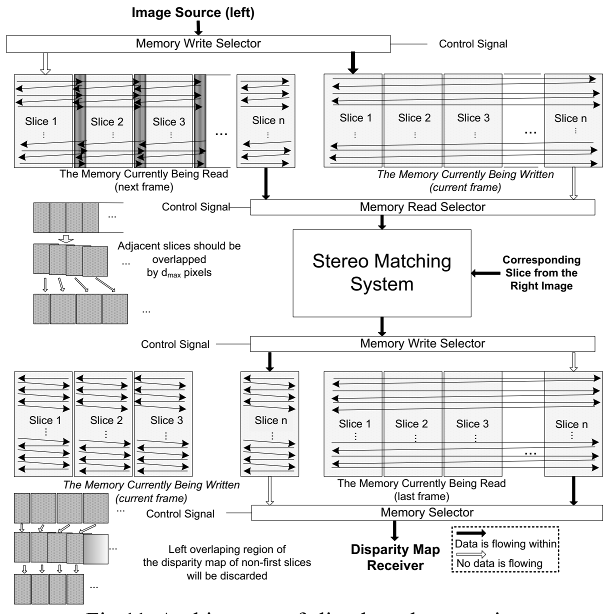


Image Source (left)
Memory Write Selector
Control Signal
Slice 1
Slice 2
Slice 3
Slice n
The Memory Currently Being Read (next frame)
The Memory Currently Being Written (current frame)
Control Signal
Memory Read Selector
Adjacent slices should be overlapped by dmax pixels
Stereo Matching System
Corresponding Slice from the Right Image
Control Signal
Memory Write Selector
The Memory Currently Being Written (current frame)
The Memory Currently Being Read (last frame)
Control Signal
Memory Selector
Left overlaping region of the disparity map of non-first slices will be discarded
Disparity Map Receiver
Data is flowing within
No data is flowing

Fig 11. Architecture of slice-based processing.

In this paper, we choose the width of the slice w' = 320, d_{max}=63, so the memory cost for vertical voting is cut down to 21×(320+64)×21×3×2bits = 124kByte, which is much more hardware-friendly. The price of this solution is the slight reduction of processing speed since it takes 320+64=384 clock cycles to process a row of 320 pixels (1.2 times longer), hence the throughput of overall system is 1.2 clock cycle per pixel. A trade-off can be made between memory cost and processing speed by choosing proper w'. The significance of this method is no matter how wide the image is, the memory cost of disparity voting remains constant.

Nonetheless, pixels from an image are input normally in a raster order rather than slice by slice, a ping-pong buffer based external memory structure is proposed, as is shown in Fig. 11. There are two banks of memory ahead of the matching system: one of them is being written by an image source in raster order while the other being read by matching system slice by slice; after the processing of the current frame has accomplished, the memory just being written is read slice by slice while the other memory now being written in raster order. Similarly if the receiver of the disparity map needs to read the image in raster order, another set of ping-pong buffer can be used at the output of the system.

C. Occlusion Detection Module

The criterion of an occluded pixel is given in Eq. (8). This can be implemented by shift registers filled with D_r which wait to be selected by D_l for cross-check. We set τ = 1 as threshold in Eq. (8); in practice we compare if $D_l/2$ and $D_r/2$ are equal and the error proves to be unnoticeable. Thus generates the occlusion bit which is a label for occluded pixels. The design is shown in Fig. 12.

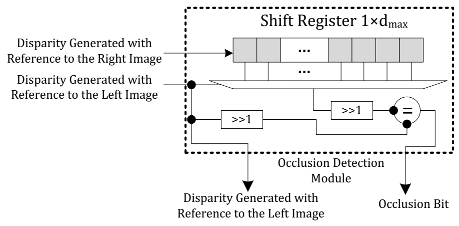

Shift Register 1×dmax
Disparity Generated with Reference to the Right Image
Disparity Generated with Reference to the Left Image
>>1
>>1
=
Occlusion Detection Module
Disparity Generated with Reference to the Left Image
Occlusion Bit

Fig 12. Design of occlusion detection module.

D. Occlusion Filling Module

We present here a ping-pong buffer based structure to accomplish the two-pass scan. Once the forward filling of a pixel is done, its value is stored in a buffer while at the same time another buffer stored with filled disparity of the previous row is being filled reversely. After the entire row has been processed in both buffers, 2 buffers exchange their roles. The design is shown in Fig. 13.

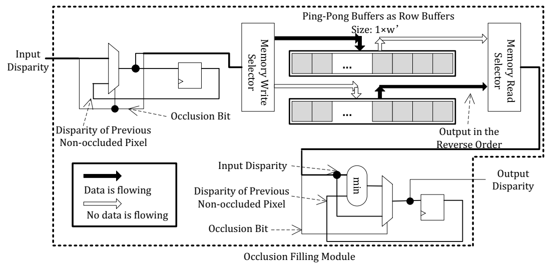


Fig 13. Design of occlusion filling module.

VI. Experimental Results

The quality of the results generated by the proposed algorithm is compared with several hardware-oriented algorithms. Here we use four images, "Tsukuba", "Venus", "Teddy", "Cones" from the [15] as the test bench. Our results are generated by the simulation of Modelsim 10.0c.

As can be seen in Fig. 14, the proposed algorithm generates high-quality results and especially outperforms other hardware-oriented algorithms in delineating the exact outline of objects in the image. This can be explained by iterative voting which repeatedly remove the misestimated disparities extending from correctly generated regions. Table

I gives a quantitative comparison by Middlebury's evaluator [15].

TABLE I.
Quantitative comparison (error rate of non-occluded pixels at error threshold=1.0) based on Middlebury test bench.

Algorithm	Tsukuba	Venus	Teddy	Cones	Platform
Proposed	2.21	1.73	5.74	3.64	FPGA
[14]	2.54	0.19	6.74	4.42	FPGA
[4]	9.79	3.59	12.50	7.34	FPGA
[16]	2.81	1.75	10.5	8.90	FPGA
[5]	3.84	1.20	7.17	5.41	FPGA
[17]	2.16	0.24	6.27	4.7	GPU
[18]	2.21	0.46	9.58	3.23	CPU

From the comparison, our algorithm outperforms listed hardware algorithms in most test images and is able to generate results comparable to the software-based implementation. Nonetheless, our algorithm doesn't perform well in "Venus" image since voting in horizontal and vertical directions can't produce a very accurate result over the large plain area with the gradient of disparity being diagonal. We hope aligning voting direction with gradient of pixels can be implemented efficiently on hardware in our future work.

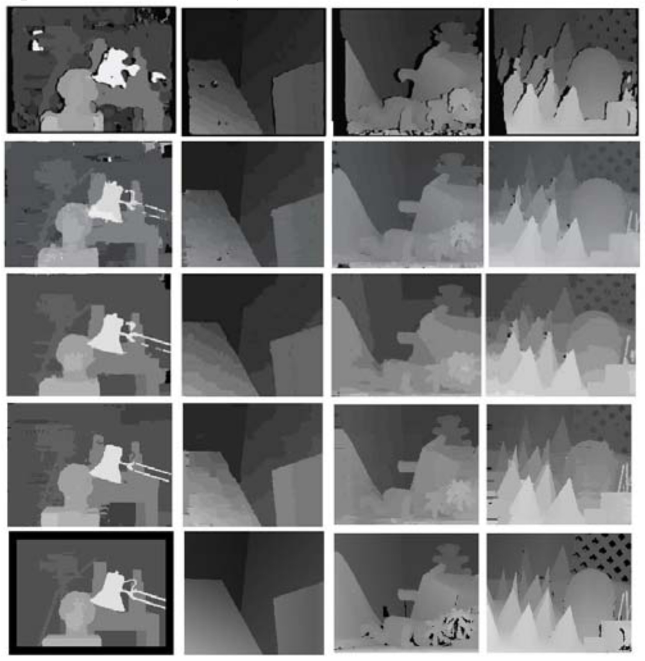


Fig 14. Disparity maps generated by the proposed algorithm comparing against other researchers' works. The 1st ~ 3rd rows show results from [4, 5, 14]. The 4th row presents results from the proposed algorithm. The 5th row shows the ground truth.

Next we give the result of synthesis and evaluate the practicability of the proposed system. We synthesize our RTL code with Quartus II 11.0 on device EP4S40G2. Maximum allowed frequency is 121.76MHz, which means that the system can process a pair of 1080p images at the frame rate of 121.76M/(1920×1080×1.2)=48fps.Thus real-time processing of HD video can be realized. The cost of hardware resources are: memory size 1,040kbit/14,283kbit (7.3%), total registers 96,398/182,400(52.8%), combinational ALUTs 104,632/182,400(57.3%), all are affordable. MDES(millions of disparities estimated/second, a widely-used index) of our algorithm reaches 1920×1080×64×48=6370M, greater than 4521M of [4], 3019M of[5], less than 9345M of [6](but their memory usage reaches 7128kbit).

VII. Conclusion

This paper presents an iterative weighted disparity voting based local stereo matching algorithm and its RTL architecture. Experimental results show that system is able to process 1920×1080-image at 48fps with low memory cost, being very competitive among prior hardware-oriented implementations.

VIII. Acknowledgement

This paper was supported by National Natural Science Foundation of China (61306023),Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP, 20120071120021), STCSM (13511503400), National High Technology Research and Development Program (863, 2012AA012001)

References

[1] M. Tanimoto, "FTV: Free-viewpoint Television," *Signal Processing: Image Communication*, Vol. 27(6), pp. 555-570, 2012.
[2] D. Murray, J.J. Little,"Using Real-Time Stereo Vision for Mobile Robot Navigation,"*Auton. Robots*,Vol.8(2),pp.161-171, 2000.
[3] S. Gehrig, C. Rabe, L. Krueger, "6D Vision Goes Fisheye for Intersection Assistance," Computer and Robot Vision, 2008,pp. 34-41,2008.
[4] J. Seunghun, C. Junguk, D.P. Xuan, L. Kyoung-Mu, P. Sung-Kee, et al., "FPGA Design and Implementation of a Real-Time Stereo Vision System," *Circuits and Systems for Video Technology, IEEE Transactions on*, Vol. 20(1), pp. 15-26 , 2010.
[5] L. Zhang, K. Zhang, T.S. Chang, G. Lafruit, G.K. Kuzmanov, D. Verkest, "Real-time high-definition stereo matching on FPGA," *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 55-64, 2011.
[6] M. Jin, T. Maruyama, "A fast and high quality stereo matching algorithm on FPGA,"*FPL 2012*,pp. 507-510,2012.
[7] V. Kolmogorov, R. Zabih, "Computing visual correspondence with occlusions using graph cuts," *ICCV 2001*, pp. 508-515, 2011.
[8] J. Sun, N. Zheng, H. Shum, "Stereo matching using belief propagation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 25(7), pp. 787-800, 2003.
[9] Y. Ohta, T. Kanade, "Stereo by intra-and inter-scanline search using dynamic programming," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol.2, pp.139-154, 1985.
[10] R. Zabih, J. Woodfill, "Non-parametric local transforms for computing visual correspondence," *ECCV 1994*, pp. 151-158, 1994.
[11] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, M. Gelautz, "Fast Cost-Volume Filtering for Visual Correspondence and Beyond," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 35(2), pp.504-511, 2013.
[12] Z. Ke, L. Jiangbo, Y. Qiong, G. Lafruit, R. Lauwereins, et al., "Real-Time and Accurate Stereo: A Scalable Approach With Bitwise Fast Voting on CUDA," *Circuits and Systems for Video Technology, IEEE Transactions on* , Vol. 21(7) , pp. 867-878, 2011.
[13] Q. Yang, "Hardware-efficient bilateral filtering for stereo matching," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 36(5), pp. 1026-1032, 2013.
[14] C. Liao, H. Yeh, K. Zhang, V. Geert, T. Chang, G. Lafruit, "Stereo Matching and Viewpoint Synthesis FPGA Implementation," *3D-TV System with Depth-based Rendering*, Springer New York, pp. 69-106, 2013.
[15] D. Scharstein, R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Computer Vision, International Journal on*, Vol. 47(1-3), pp.7-42, 2002.
[16] K.Ohata, Y.Sanada, T.Ogaki, K.Matsuyama, T.Ohira, et al., "Hardware-oriented stereo vision algorithm based on 1-D guided filtering and its FPGA implementation,"*ICECS 2013*,pp.169-172, 2013.
[17] V. Drazic, N. Sabater, "A precise real-time stereo algorithm," *IVCNZ 2012*, pp. 138-143, 2012.
[18] Y. Deng, X. Lin, "A fast line segment based dense stereo algorithm using tree dynamic programming,"ECCV 2006,pp. 201-212, 2006.