Hardware-Oriented Stereo Vision Algorithm based on 1-D Guided Filtering and its FPGA Implementation.

Katsuki Ohata*, Yuki Sanada†, Tetsuro Ogaki*, Kento Matsuyama†, Takanori Ohira†, Satoshi Chikuda†, Masaki Igarashi†, Masayuki Ikebe†, Tetsuya Asai†, Masato Motomura† and Tadahiro Kuroda*
*Faculty of Science and Technology, Keio University
Hiyoshi 3-14-1, Kouhokuku, Yokohama 223–8522, Japan
†Graduate School of Information Science and Technology, Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo 060–0814, Japan

***Abstract*—This paper presents a novel hardware-oriented stereo vision system based on 1-D cost aggregation. Many researchers have implemented hardware efficient stereo matching to realize real-time systems. However, such methods require a large amount of memory. We proposed a system that is based on a hardware-software hybrid architecture for memory reduction. It consisted of grayscale 1-D cost aggregation HW and 2-D disparity refinement SW. The 1-D processing reduced the size of RAM in our HW to 266 kb with an input image size of 1024×768. We achieved the average error rate for the Middlebury datasets as 6.24%. The processing time was 56.6 ms for the 1024×768 images and an average of 8.6 ms for the Middlebury datasets which have an average size of 400×380. Using the resolution of Middlebury datasets, our system can perform real-time depth-aided image processing.**

I. Introduction

Stereo matching is a method to estimate a depth map from a stereo image pair. The depth map is used to aid various image processing techniques such as pedestrian detection, 3-D mapping, and refocusing. To run depth-aided image processing in real-time, depth estimation must be processed at speeds exceeding real-time. Therefore, many previous studies concerned on the processing speed.

Stereo matching is defined as a cost minimization problem. In general, the algorithm is divided into the four steps; i) Cost calculation: A cost map is calculated for each disparity according to the similarities of corresponding pixels in the stereo image; ii) Cost aggregation: Cost maps are filtered to suppress noises; iii) Disparity computation: Disparities for each pixel are selected according to the cost map; iv) Disparity refinement: Errors in the disparity map are corrected. Depth is calculated from the disparity using the camera parameter.

The most time-consuming step is cost aggregation. The aggregation using a boxfilter [1] confuses costs with multiple objects. Segmentation can handle each object separately [2], but it also increases the computational cost. Therefore, an edge-preserving filter is often employed (e.g., [3], [4], [5]). Most of the successful methods in terms of speed are hardware efficient. CostFilter [5] was implemented on a GPU and runs at near real-time speed with high accuracy. In [6], the author implemented a combination of hardware efficient functions on CUDA, and achieved near real-time speeds, while remaining second in accuracy. To the best of our knowledge, [9] presents a state-of-the-art system. With FPGA implementation, it achieved 199.7 fps with an image size of 1024×768 while others employed images that are 80% smaller.

However, these methods have a issue with the memory size. The cost aggregation of [5] requires a memory size that is six times larger than the image size. Jin's method [9] requires a memory size of 7 Mb, and it is based on [6]; therefore, [6] requires a memory size that is near to that of [9].

In this paper, we propose a hardware-software hybrid system for stereo matching. This system consists of a 1-D horizontal processing hardware and a 2-D image-refinement software. The most time consuming step, cost aggression, is processed on the 1-D hardware. This is similar to RDP [7], but our method included cost calculation in 1-D space, thus resulting in highly hardware efficient method which enables the reduction of memory size and highly parallel processing. The error rate increases due to the lack of 2-D information. However, our system can compensate the error using the 2-D software.

II. Proposed Method

Our method follows the four steps of stereo matching. As the core of our system, we employed [5]. The key differences with [5] are the cost aggregation with the grayscale adaptive 1-D guided filtering, which is highly hardware efficient, and the refinement specialized to the 1-D disparity map.

A. 1-D Cost Aggregation

The key technique in our cost aggregation is guided filtering. The equation of the grayscale guided filtering is given below.

$$A(y,x) = \frac{\overline{I(y,x)C(y,x)} - \overline{I(y,x)}\,\overline{C(y,x)}}{\overline{I(y,x)^2} - \overline{I(y,x)}\,\overline{I(y,x)} + \epsilon} \quad (1)$$

$$B(y,x) = \overline{C(y,x)} - A(y,x)\overline{I(y,x)} \quad (2)$$

$$C'(y,x) = \overline{A(y,x)}I(y,x) + \overline{B(y,x)} \quad (3)$$

$\overline{X(y,x)}$ is the mean of X in the window centered at position (y,x), and ϵ is a regularizing parameter. This filter smooths the input C using the image I, and outputs C'.

This filtering is repeated for each disparities. Reusing the denominator of eq. (1) for each repetition reduces the computational cost. The denominator of RGB guided filter is a covariance matrix which has six values, thus requiring a memory size that is six times larger than the image size. For this reason, grayscale guided filtering can greatly reduce the memory size. Because RGB information is already included in the initial cost, the loss of accuracy is small. Moreover, the input of the 1-D method has only a 1-pixel height. Therefore, the memory size required for reusing the denominator of 1-D grayscale method is only the size same as the image width.

For optimal performance, the filter size needs to be fixed to the object size in an image. In our system, since the guided filter for each line works independently, parameters can be set individually. By precalculating the optimum parameter for each line, the performance of aggregation can be improved.

The object size can be estimated from the continuity of textures. Each line processing can operate with only a single filter size, so the tendency of continuity is used for parameter optimization. The calculation of continuity tendency T is given by an equation similar to the deviation,

$$T(y) = \sum_x |I^f_{rlarge}(y,x) - I^f_{rsmall}(y,x)| \tag{4}$$

I^f_r is the grayscale image 1-D boxfiltered with radius r, Both inputs are filtered to consider textures with repetitive patterns. The filter size of each line processing is selected using the tendency and thresholds. In our experiment, two threshold values th_1 and $th_2 = 2th_1$, and three rough radius values, r_s, $r_m = r_s + r_{step}$, and $r_l = r_s + 2r_{step}$ were determined.

The result of 1-D RGB aggregation and proposed aggregation are compared in Fig. 1. Streak effects were greatly reduced. The software compensates the remaining noise.

B. Preprocessing for Software Refinement

Before refining the disparity map on software, our hardware preprocesses the disparity map with error detection and noise reduction. This step requires only three line buffers for the hardware implementation. The upper FOR loop of Algorithm 1 shows the preprocessing algorithm.

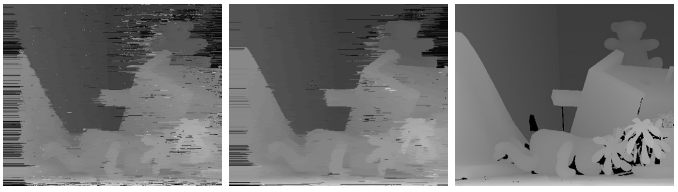

Fig. 1: Example of the Teddy dataset [8]. Left, right: RGB 1-D process without adaptive radius and grayscale 1-D process with adaptive radius without 2-D refinement. Right: Ground truth.

Algorithm 1: Refinement Algorithm

Data: Disparity map D, RGB image I, threshold th
for All (y,x) **do**
  if $|D(y,x) - D(y+1,x)| > 1$ **then**
    Label $(y,x),(y \pm 1,x)$ as an error
  if $|D(y-1,x) - D(y+1,x)| \leq 1$ **then**
    $D(y,x) = D(y-1,x)$
for All (y,x) Labeled as errors **do**
  R_{up} = five nearest non error points above (y,x)
  I_{up} = mean of $I \in R_{up}$
  d_{up} = max difference of $I(y,x)$ and I_{up} in RGB
  D_{up} = mode of $D \in R_{up}$
  C_{up} = count of D_{up} in $D \in R_{up}$
  if $C_{up} < 3$ **or** $d_{up} > th$ **then**
    $D_{up} = D(y,x)$, d_{up} = INT_MAX
  repeat procedures above using R_{down}
  (R_{down} = five nearest non error points below (y,x))
  $D(y,x) = D_{up}$ **or** D_{down} with lower d_{up} **or** d_{down}

Error detection labels errors using a vertical gradient. Because of 1-D method, vertical connections of disparities can be found only when the true disparity in a region is successfully recovered. In other words, errors occur as horizontal streaks which can be detected by obtaining a vertical gradient.

Noise reduction fixes streak errors isolated from other errors, using vertically adjacent pixels. This correction technique increases the accuracy of subsequent refinements.

C. 2-D Refinement

2-D refinement software consists of error correction and median filtering. In error correction, the five nearest non error pixels above and below the target pixel are used. The mode disparity of 5-pixel set which have colors closer to that of target pixel is used for the correction. The precise algorithm is shown in the lower FOR loop of Algorithm 1.

2-D refinement is completed with a 5×5 median filter for denoising. Our software is mostly a vertical 1-D process. However, implementing the algorithm on hardware requires buffers that have the sizes the same as that of the image. Therefore, this part was processed on software.

III. FPGA Implementation

The solid blocks of Fig. 2 shows a block diagram of our architecture. The maximum disparity ($\equiv 2^N$) determines the degree of the parallelism, where the output latency increases as N increases, while the throughput remains constant. The system consists of two radius estimators for left and right sequential images, two line buffers for controlling timings of the images, $2^{N+1} - 1$ delay registers (Z^{-1}) for generating binocular disparity, 2^{N+1} cost calculators, 2^{N+1} guided filters, two loser-takes-all circuits to find the minimum costs for left and right blocks, and an occlusion-check & refinement circuit.

Given input (sequential) left and right images are firstly sent to each radius estimator and outputs radius values for subsequent guided filters. The radius values are obtained after

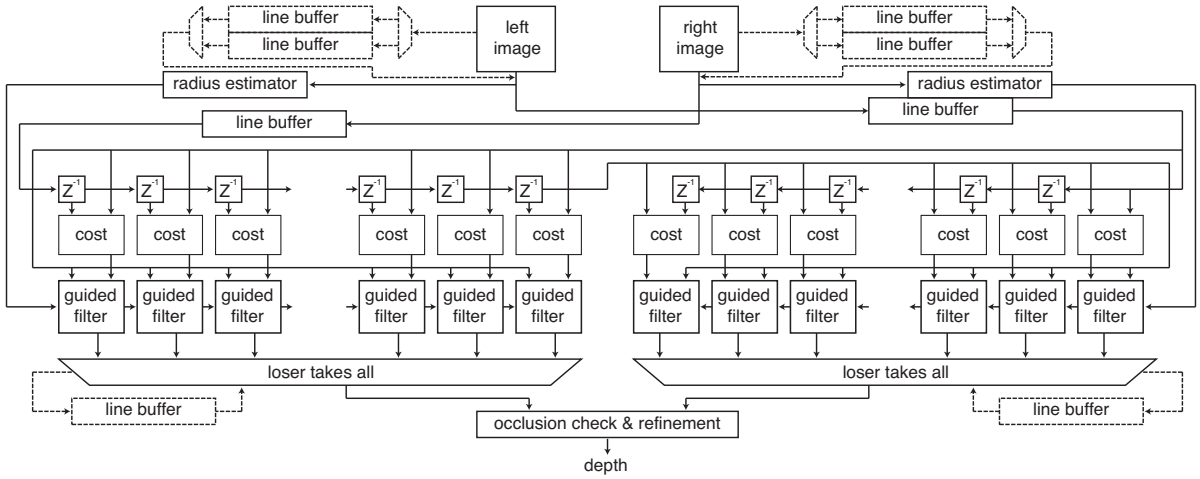

Fig. 2: Block diagram of proposed parallel stereo-matching architecture

ALUT	Register	Block memory	DSP block 9-bit	Input Res. & Disparity	Output Res. & Disparity	FPGA CLK
36,969	35,360	32,451	504	192×144 (24-bit RGB)	192×144(3-bit)	20 MHz

TABLE I: Implementation & Performance Summary

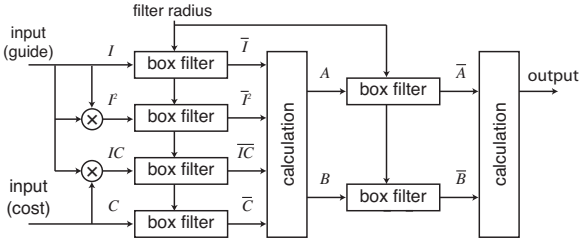

Fig. 3: Block diagram of guided filter

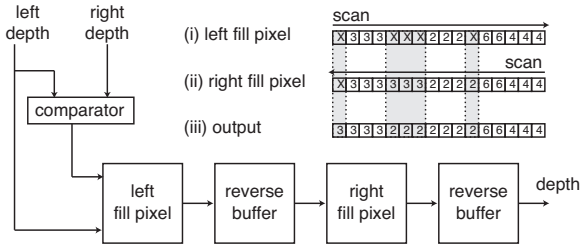

Fig. 4: Block diagram of occlusion check & refinement circuit

reading one line of input images. Therefore, the input image sequences are delayed by line buffers, as shown in Fig. 2.

After radius estimation, the delayed image sequences reach at inputs of delay registers (leftmost and rightmost of Z^{-1} in Fig. 2). Each register gives pixel values shifted from 0 to $2^N - 1$ pixels in both left and right directions. The cost estimator accepts both shifted and non-shifted pixel values and accumulate the cost between them in parallel.

The calculated cost maps are then smoothed by parallel guided filters. Our 1-D guided filter circuit consists of six box filters, and associate combinational arithmetic circuits (Fig. 3). The radius of box filter circuits is variable, and can be set at three different values (pixels). It should be remarked that in practical implementation, one may remove two box filters and one multiplier from the 1-D guided filter, because values of $\bar{I}$ and $\bar{I^2}$ in Fig. 3 are common over the line.

Among the smoothed cost maps, the minimum values for left and right views are selected by loser-takes-all circuits, and the corresponding disparity values are given to an occlusion check & refinement circuits. The circuit detects occlusion errors by comparing disparity values among left and right views, and corrects the errors by using disparity values on the border of the error region, which requires bidirectional scanning as shown in Fig. 4 (i, ii and iii); i) error region is filled by disparity values on the left border; ii) flow direction of the disparity sequences are inverted, and the resulting error region is filled by the smallest disparity values among the filled values in i) and disparity values on the right border; iii) the flow direction is further inverted. Fig. 4 shows the block diagram of processing i) to iii), where left and right fill pixel blocks represent the disparity filling circuits, whereas the reverse buffers invert the signal flows. Each reverse buffer consists of one line buffer and one up-down counter only.

IV. Experiments and Results

A. Hardware Experiment

The proposed system was implemented on a commercial FPGA board (Power Medusa MU200-SXII with Altera Startix II and onboard SRAMs). The system was coded by Verilog HDL, and the RTL model was synthesized by Quartus II. Parameters used are as follows: $\{\epsilon, r_{large}, r_{small}, th_1, r_s, r_{step}\} = \{2, 8, 6, 330, 5, 9\}$. Due to the limited number of logic elements, N was set at 3 and the

input image was shrunk to 192×144 pixels. Table I summarizes the implementation and performance. Fig. 5 shows the disparity results for the Tsukuba datasets from [8], processed by the FPGA.

Although the precision of the disparity values are degraded due to the reduction of maximum disparity, the FPGA could generate rough disparity map. Note that this is certainly due to the limited number of logic elements, because our RTL results perfectly matched with numerical results at any maximum disparity, and RTL results with 3-bit disparity matched the FPGA results as well. These results indicate that the proposed architecture is not a nominal, but is synthesizable and operates with acceptable clocks and number/scale of hardware resources.

To compare our method with [9], we compiled and estimated the method with image size of 1024×768. On compiling the method, maximum disparity was reduced to 16. We estimated the hardware specification by doubling up the number and CLK of disparity resolution dependent block. In this way, 32 guided filters work twice and generate 64 filtered cost maps. In order to double the CLK, buffers to hold the temporary lowest cost maps and two RGB input images were added. The dotted boxes of Fig. 2 are the additional buffers and they have the total size of 130 kb. As a result, the estimated size of RAM became 266 kb, which is only 3.7% of [9].

B. Software Experiment

We conducted the measurement on an Intel Core i3 2.53GHz PC using Middlebury benchmarks [8] . The threshold value was set to 40. The results are shown in Fig. 6 and Table III. Our method was as accurate as [9]. The average processing time was 56.6 ms for 1024×768 images, and 8.6 ms for the Middlebury datasets, which has the average size of 400×380. If the size of the Middlebury datasets is used, significant amount of time remains to place a depth-aided application.

V. Conclusion

This paper presented a hardware-software hybrid stereo matching constituted a 1-D hardware and fast software. The loss of accuracy caused by the loss of 2-D information was compensated for by changing the filter radius for each cost aggregation and by applying a new refinement technique specialized to 1-D processed disparity maps. Moreover, we implemented the hardware on an FPGA, and from the result of implementation, estimated the hardware specification needed to process a 1024×768 image. The result showed that proposed system operates with acceptable clocks and number/scale of hardware resources. If the image size of the Middlebury datasets was used, our system runs at speeds exceeding real-time while maintaining a sufficient accuracy.

Fig. 5: Experimental results for the Tsukuba dataset [8]. Left, middle: 192×144, 3-bit FPGA, and numerical disparity result. Right: 384×288, 4-bit numerical disparity result.

	Max. disparity	LUTS	RAM	CLK	FPS
[9]	60	122 k	7189 kb	318.3 MHz	199.7
Proposed	16	101 k	89 kb	20.0 MHz	25.4
Estimated	64	202 k	266 kb	40.0 MHz	25.4

TABLE II: Comparison of hardware with image size of 1024×768. Our method was only compiled and estimated.

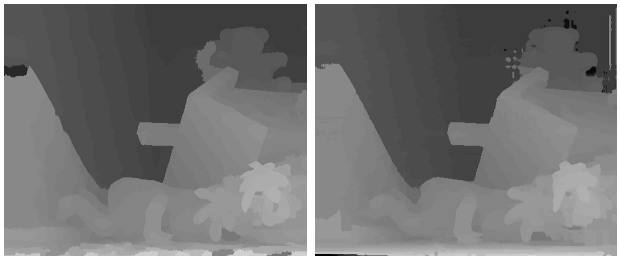

Fig. 6: Results of Teddy dataset [8]. Left: [5]. Right: Proposed.

Method	Tsukuba	Venus	Teddy	Cones	Average
ADCensus [6]	1.48	0.25	6.22	7.25	3.97
AdaptingBP [2]	1.37	0.21	7.06	7.92	4.24
CostFilter [5]	1.85	0.39	11.8	8.24	5.55
Fast [9]	1.84	0.48	12.7	9.19	6.13
Proposed	2.81	1.75	10.5	8.90	6.24
RealTimeBP [10]	3.40	1.90	13.2	11.6	7.68
MiniCensus [11]	4.34	1.68	12.6	11.0	8.20
RealTimeDP-Tree [12]	2.51	2.97	13.6	13.8	8.71

TABLE III: Comparison of the error rate of the *all* regions and the average of three regions defined in [8].

Acknowledgment

The authors would like to thank Semiconductor Technology Academic Research Center (STARC), Japan, for funding this research project.

References

[1] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *IJCV*, 2002.
[2] A. Klaus, *et al*. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. *ICPR*, 2007.
[3] Q. Yang. Recursive bilateral filtering. *ECCV*, 2012.
[4] C. Pham and J. Jeon. Domain transformation-based efficient cost aggregation for local stereo matching. *CSVT*, 2013.
[5] C. Rhemann, *et al*. Fast cost-volume filtering for visual correspondence and beyond. *CVPR*, 2011.
[6] X. Mei, *et al*. On building an accurate stereo matching system on graphics hardware. *ICCV*, 2011.
[7] X. Sun, *et al*. Stereo matching with reliable disparity propagation. *3DIMPVT*, 2011.
[8] D. Scharstein and R. Szeliski. http://vision.middlebury.edu/stereo/
[9] M. Jin and T. Maruyama. A fast and high quality stereo matching algorithm on FPGA. *FPL*, 2012.
[10] Q. Yang, *et al*. Real-time global stereo matching using hierarchical belief propagation. *BMVC*, 2006.
[11] L. Zhang, *et al*. Real-time high-definition stereo matching on fpga. *FPGA*, 2011.
[12] M. Jin and T. Maruyama. A real-time stereo vision system using a tree-structured dynamic programming on fpga. *FPGA*, 2012.