GPU-Accelerated Single Image Depth Estimation with Color-Filtered Aperture

Yueh-Teng Hsu[1], Chun-Chieh Chen[2] and Shu-Ming Tseng[3]
[1] A-MTK Corp.
Taipei 236, Taiwan
[e-mail: mathew@a-mtk.com]
[2] Department of Electronic Engineering, National Taipei University of Technology
Taipei 106, Taiwan
[e-mail: jjchen0420@gmail.com]
[3] Department of Electronic Engineering, National Taipei University of Technology
Taipei 106, Taiwan
[e-mail: shuming@ntut.edu.tw]
*Corresponding author: Shu-Ming Tseng

Received November 19, 2013; accepted January 23, 2014; published March 31, 2014

Abstract

There are two major ways to implement depth estimation, multiple image depth estimation and single image depth estimation, respectively. The former has a high hardware cost because it uses multiple cameras but it has a simple software algorithm. Conversely, the latter has a low hardware cost but the software algorithm is complex. One of the recent trends in this field is to make a system compact, or even portable, and to simplify the optical elements to be attached to the conventional camera. In this paper, we present an implementation of depth estimation with a single image using a graphics processing unit (GPU) in a desktop PC, and achieve real-time application via our evolutional algorithm and parallel processing technique, employing a compute shader. The methods greatly accelerate the compute-intensive implementation of depth estimation with a single view image from 0.003 frames per second (fps) (implemented in MATLAB) to 53 fps, which is almost twice the real-time standard of 30 fps. In the previous literature, to the best of our knowledge, no paper discusses the optimization of depth estimation using a single image, and the frame rate of our final result is better than that of previous studies using multiple images, whose frame rate is about 20fps.

***Keywords*:** Real time, graphics processing unit, parallel processing technique, compute shade

http://dx.doi.org/10.3837/tiis.2014.03.020

1. Introduction

A GPU is installed in most computing devices such as the personal computer (PC) and smart phone, and it can provide almost all the data processing operations required in the field of digital signal processes and digital image processes. Although the modern multi-core central processing unit (CPU) has good performance, the speed of the other tasks originally assigned to it may be decreased while it implements the depth estimation task. As regards three-dimensional (3D) game graphics, however, GPUs are seldom used most of the time. The best advantage of using the GPU is its parallel processing capability, which means that it is able to process many data at the same time in parallel by using its large supply of powerful arithmetic logic units (ALUs). In [1], the author uses GPUs to accelerate the implementation of parameterized baseband models for two different orthogonal frequency division multiplexing (OFDM) protocols of 802.11a and 802.16 and achieve real-time throughput. In [2], the author shows a design example of a mobile WiMAX terminal implemented on the GPU platform which is nearly 90 times faster than the conventional DSP-driven modem. In [3], a novel scheme that accelerates password recovery of PDF files on GPUs using a compute unified device architecture (CUDA) is proposed, and the experimental results show that this scheme has higher speed performance at low cost.

There are several ways to realize depth estimation, generally divided into two categories: multiple image depth estimation [4][5][6] and single image depth estimation [7] [8] [9], the advantage of the former is fast implementation thanks to its simple algorithm but it has the drawback of higher hardware cost of multiple cameras. Accordingly, it is easier to achieve real time by utilizing multiple images to estimate depth density [5] [4]. On the other hand, the latter method has a lower hardware cost with only one camera, but it is quite difficult to achieve real time because of its numerous mathematical calculations. One of the recent trends in this field is to make a system compact, or even portable, and to simplify the optical elements to be attached to the conventional camera [10][11] and the smart mobile. Consequently, our research focuses on accelerating the depth estimation process with a single image until finally we achieve real time by employing our evolutional algorithm and complete the implementation on a GPU which has powerful computing capability.

Dense depth estimation with either a single view image or multiple images has been explored for decades, and a wealth of literature discusses the speed performance of the latter much more than that of the former. The reason is that it is still a challenge to obtain a good performance by using only a single image to estimate depth because of its complex calculation. The implementation optimized by two-dimension (2-D) generalization of dynamic programming (DP) [12] is precise; however, the optimized scheme is too complex to achieve real-time application. In [13], the authors use single instruction multiple data (SIMD) like MMX and SSE on a CPU to reconstruct depth maps via multiple images, utilizing parallel processing to achieve real-time performance but with lower precision. In [5][4], real-time performance utilizing multiple images based on GPU is achieved. The author in [5] presents an algorithm relying on the conventional sum of square difference (SSD) dissimilarity measure between correlation windows. The author in [4] extends the basic idea of simple plane sweep to generalized search spaces of any geometry and maintains performance.

All the literature discussed in the previous paragraph implements depth estimation using multiple images. To the best of our knowledge, there is still no literature on real-time depth map application using single image. In our approach, we follow the basic idea of depth estimation with a single image presented by [7]. The prefix sum [14] algorithm included in the box filter method [15] is modified to be suited to data-parallel processing in the GPU. With implementation on a compute shader, which is a programmable shader that expands Microsoft Direct3D 11 beyond graphics programming and provides high-speed general purpose computing to take advantage of the large numbers of parallel processors on a GPU, we achieve real-time performance of 53 fps with a single image scenario which is even more than the frame rate (20 fps) achieved by[4] with multiple images.

The remainder of this paper is organized as follows. We present a camera with color-filtered aperture in Section 2. In Section 3, we introduce the depth estimation algorithm. In Section 4, we explain the computational complex of the algorithm and present some methods to reduce it. In Section 5, we take simple examples to illustrate box-filter and prefix sum methods. Depth estimation results are presented in Section 6. Section 7 concludes.

2. Color-filtered Aperture

Our camera lens equipped with color filters is shown in **Fig. 1.** The RGB filters displace the captured image of the sensor with respect to the center. According to our placement of RGB filters, if an object is further than the focused depth, the captured image will have a rightward shift in the R plane, a leftward shift in the B plane, and an upward shift in the G plane. If an object is nearer than the focused depth, the captured image will be shifted in the opposite directions. **Fig. 2** shows the displacement of the R (top right), G (bottom right) and B (bottom left) planes, and the red lines highlight the shift between each of two RGB components.

We use a Canon EOS40D DSLR for implementation, and we cut out a disc with a triple square-shaped hole from a piece of black cardboard (the same fabrication process as in [7]), and make color filters (Fujifilter SC-58, BPB-53, and BPB-45) stick to it, then attach it in front of a Canon EF 50mm f/1.8 II lens. The fabrication process is simple and takes little time.

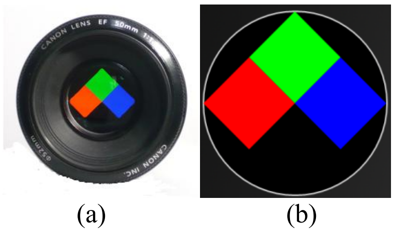


Fig. 1. (a) Camera lens equipped with color filters placed in front of the aperture (b) color filter placement

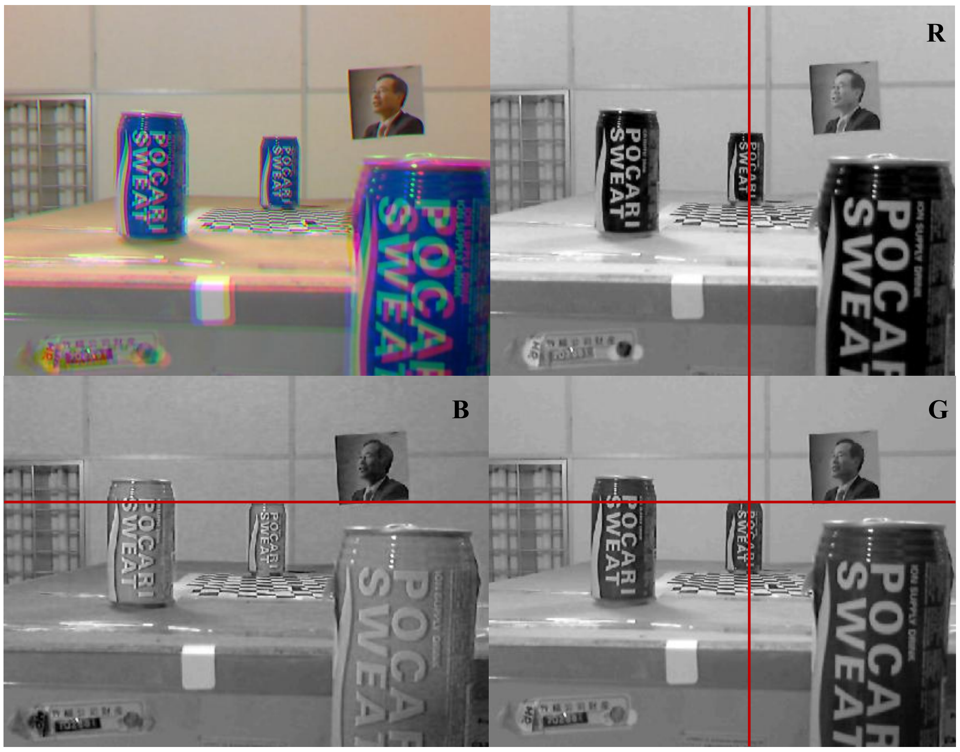


Fig. 2. Example of captured image (top left) and its R component (top right),G component (bottom right) and B component (bottom left). The red lines are superimposed to highlight the displacement of each component.

3. Depth Estimation Method

The image we have captured is a form of RGB plane consisting of I_r , I_g , and I_b , and because of the RGB color-filter placed in front of the lens, as shown in **Fig. 1**, we can observe that a scene pixel further than the focused depth has a rightward shift in the R plane, a leftward shift in the B plane, and an upward shift in the G plane. As a result, if we assume that pixel (x, y) has a disparity d , we can find the aligned pixel at $I_r(x+d, y)$, $I_g(x, y-d)$, and $I_b(x-d, y)$.

We determine the window size according to the captured image size and we slide the window through the whole captured image from left to right, from top to bottom. We denote $w(x, y)$ as a window around (x, y), and consider a set $S_I(x, y; d)$ of pixel colors with hypothesized disparity d as $S_I(x, y; d) = \{(I_r(s+d, t), I_g(s, t-d), I_b(s-d, t)) \mid (s, t) \in w(x, y)\}$. To rectify the misalignment, we should search for d to minimize the following color alignment metric [7]

$$L(x, y; d) = \frac{\lambda_0 \lambda_1 \lambda_2}{\sigma_r^2 \sigma_g^2 \sigma_b^2} \tag{1}$$

where λ_0, λ_1, and λ_2 are the descending positive eigenvalues of the covariance matrix $\sum$ of $S_I(x, y; d)$, and σ_r^2 , σ_g^2 and σ_b^2 are the variance of $I_r(x+d, y)$, $I_g(x, y-d)$, and $I_b(x-d, y)$, respectively, and also the diagonal elements of $\sum$. According to principal component analysis (PCA) [20], the product of eigenvalues is equal to the determinant of $\sum$, and the color alignment metric L in (1) is smaller when λ_0 is much larger than the λ_1 and λ_2; then the distribution of the pixel colors within the window in the RGB space will be elongated, which means that the pixels of RGB planes are more correlated after shifting*d* pixels toward the virtual center.

4. Computational Complexity: Explanation and Reduction

For each window (described in Section 3) within the captured image, we have to calculate all the elements of the covariance matrix of each RGB plane with disparity d denoted by $S_I(x, y; d)$, including variance of $S_I(x, y; d)$ and covariance of each pair of $S_I(x, y; d)$, to obtain the color alignment metric L_d with disparity d within a local window. We search for all the disparity d (-5 to 10 in our implementation) to find the smallest color alignment metric $\hat{L}$

$$\hat{L} = \min_{d} L_d$$

Then we slide the window by one pixel and repeat the above process until we have gone through the whole captured image. It is easy to see that there is a large amount of computation, especially as we have to calculate the variance and covariance a number of times. Assuming R and G are random variables of R plane and G plane within a local window, the variance of R and covariance of R and G can be calculated as follows:

$$\begin{aligned} Var(R) &= E[R^2] - E[R]^2 \\ Cov(R,G) &= E[RG] - E[R]E[G] \end{aligned} \tag{2}$$

where *E[.]* represents the expectation operator. The four items of Eq. (2) need to obtain the sum of the pixels in either R or G plane within a window; therefore, we employ the box filter [15] to improve the speed of calculating the variance and covariance, so the complexity will be reduced. The box filter executes four operations for each output picture pixel and is independent of the box size. Using this method we can calculate the expectations of all pixel values belonging to each window in a captured image at one time. It greatly reduces the cycles of calculation and also significantly accelerates our implementation.

To use the box filter method, all the pixel's values of each RGB component must first be accumulated, which is a sequential operation. Unfortunately, such an operation is hard to realize in a GPU, which usually processes data in parallel. Therefore, we utilize the prefix sum algorithm, which is a parallel operation for cumulating pixels. The details are discussed in [14][16].

5. Explanation of Box-filter and Prefix Sum

In this section, we explain the effect of the box-filter [15] and prefix sum [14][16] by means of simple examples.

5.1 Box-filter

We take the simple case of a 5×5 image as shown in **Fig. 3(a)**, and the numbers inside the frame are assumed to be the intensity of the pixels. The sliding window size is assumed to be 2×2. First, we obtain the cumulation of **Fig. 3(a)** by a two-step operation which cumulates the value in the horizontal (**Fig. 3(b)**) and vertical directions (**Fig. 3(c)**). If we want to obtain the sum of the values inside the red rectangular window in **Fig. 3(a)**, it could be calculated by the following operation:

$$3+5+1+8=17=39-16-10+4$$

The numbers on the right-hand side are highlighted by a red rectangle in **Fig. 3 (c)**. It is easy to extend this to calculate the sum of each window, and could be done in parellel by a GPU.

1	3	4	2	8
3	1	3	5	2
2	6	1	8	3
4	4	9	2	2
1	6	5	1	4

(a)

1	4	8	10	18
3	4	7	12	14
2	8	9	17	20
4	8	17	19	21
1	7	12	13	17

(b)

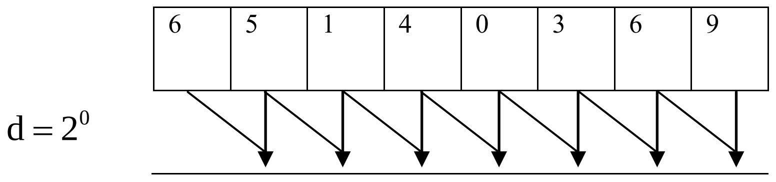
1	4	8	10	18
4	8	15	22	32
6	16	24	39	52
10	24	41	58	73
11	31	53	71	90

(c)

Fig. 3. (a) An example of a 5×5 image (b) The cumulation of (a) in the horizontal direction (c) The cumulation of (b) in the vertical direction

5.2 Prefix Sum

As described in Section 6.1, we should calculate the cumulation in the horizontal and vertical directions. A prefix sum algorithm is modified such that it can be data-parallel processing in the GPU. In the following, we use a simple example to explain this method. **Fig. 4** illustrates an example of the prefix sum algorithm when the sequence length is eigh. The parameter d is one initially, and it grows of a power of two in each stage until it achieves half the sequence length (=4 in **Fig. 4)**. The ith value ($i>d$) of the sequence in one stage is the sum of the ith and jth ($j=i-d$) values of the sequence in the previous stage. **Fig. 4** shows a simple case of eight values. Following the steps described above, we can see the bottom row is a cumulation of the top row. It is also easy to extend the process from 1-D to 2-D, and we can use the GPU to calculate the prefix sum in parallel, which greatly speeds up this operation.



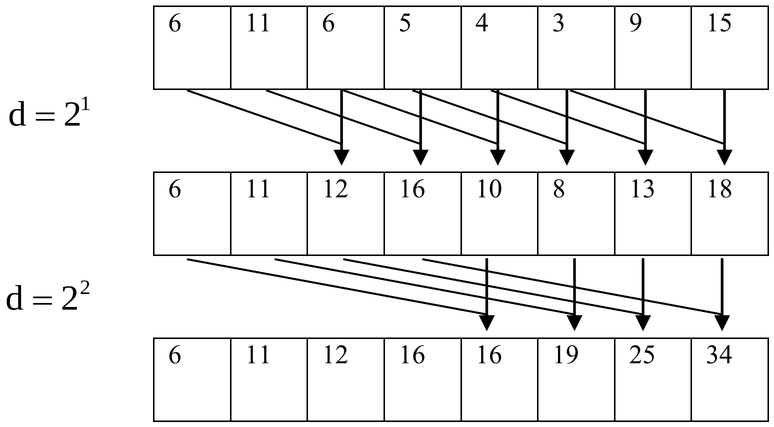


Fig. 4. Illustration of prefix sum

6. Depth Estimation Result

In this section, we implement the depth estimation in the hardware environment listed in **Table 1**. For software, we consider three languages: Matlab, C++ and compute shader. We use C++ along with Integrated Performance Primitives (IPP) library which is an extensive library of multicore-ready and highly optimized software functions with single instruction multiple data (SIMD) instructions. The compute shader we use is a shader language of DirectX 11. The languages' platform and version are listed in **Table 2**. The image in our implementation has a pixel resolution of 640×480.

As shown in **Table 3**, the CPU loading of the compute shader is almost zero, because the process is running on a GPU, which means that it does not affect the processes executed in a CPU. The frame rates of Matlab employing a box filter algorithm are almost 40 times greater than if it did not use a box filter algorithm. The performance of C++ using the IPP library is 5.3 times faster than it would be in Matlab. Finally, the performance of the compute shader is about 80 times faster than C++ and 420 times faster than Matlab. The implemented compute shader has a frame rate of 52.63fps, which exceeds the recognized real-time standard of 30fps. The standard is based on a well-known real-time television system, NTSC (National Television System Committee), whose frame rate is 29.97 fps [17] [18].

Table 1. Hardware list

Item	**Model**
CPU	Intel core i7 2.93-GHz

Memory	DDR3 4GB RAM
Mainboard	ASUS BM5295
Graphic Card	NVIDIA Geforce GT430

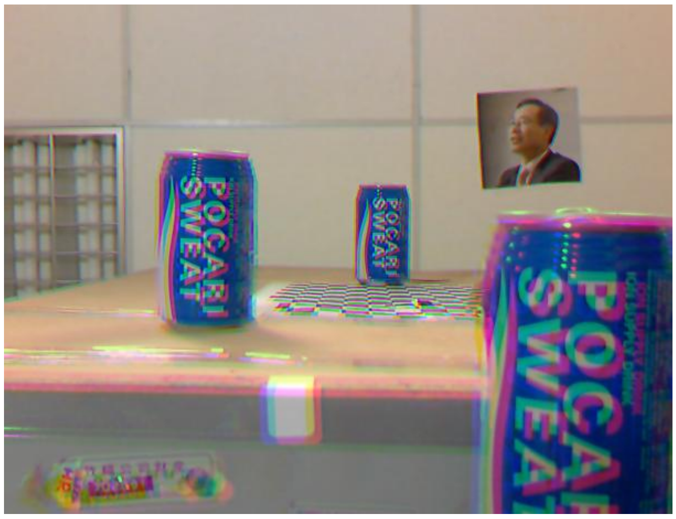
Table 2. List of languages, platform and version

Language	**Platform**	**Version**
Matlab	Matlab	2008b
C++	Visual studio	2010
Compute shader	DirectX 11	CS5.0

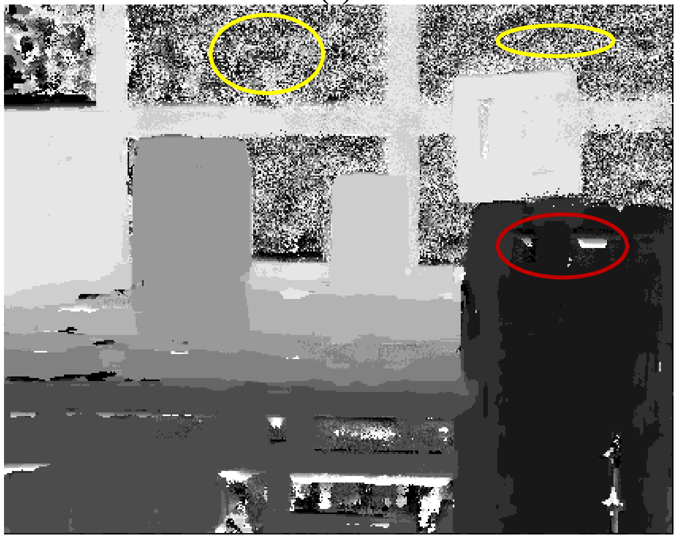
Table 3. Performance comparison of depth estimation among different languages for a 640×480 image

Language	**CPU loading (%)**	**Frame rates (fps)**
Matlab(no GPU)	14	0.003
Matlab (box filter)(no GPU)	14	0.125
C++ (IPP library) (box filter)(no GPU)	10	0.667
Compute shader (box filter)(GPU)	1	52.63

Fig. 5(a) shows the captured image taken with a lens attached by color filters and the camera focused on the background which is the farthest object from the image. We can see that the nearest object has the largest displacement. For example, the nearest beverage can is more misaligned than the next one, and that one is more misaligned than the farthest one. The depth map is presented in **Fig. 5(b)**, where the whiter color shows that the region has a smaller displacement. We can see that the nearer object in **Fig. 5(a)** corresponds to a darker region in **Fig. 5(b)**. The depth estimation which is a kind of local optimization has two major limitations, however. The first is that the texture of the local region should be reasonably obvious, as highlighted by yellow circles in **Fig. 5(b)**. The second is that objects should not have too high a proportion of single, pure R, G, or B colors, as does the bottle of the nearest beverage highlighted by the red circle in **Fig. 5(b)**. Local optimization such as normalized cross-correlation (NCC) [19] and PCA [20] cannot solve the problem of misestimation. Global optimization such as graphic cut [21][7] and dynamic programming [22] may partially solve this problem, but there is still the difficulty that objects must not be of one pure color. We would like to further investigate the problem of misestimation in future research.



(a)



(b)

Fig. 5. (a) The captured image. The foreground color is misaligned. (b) Estimated depth (the darker, the nearer)

7. Conclusions

We present an implementation of depth estimation from a single image. It is recognized that computational complexity is the main challenge to implementing depth estimation in real time, especially with a single image. In this paper, we propose an evolutional algorithm and modify it to fit the data-parallel processing of a GPU. The speed of the final result with a compute shader was 52fps. Previous research using multiple images obtained 20fps, so our result is about 2.5 times better. It is noteworthy that we use only a single image with one camera and lens but obtain higher frame rates.

We also compare the performance of several program languages with and without additional algorithms, and the result shows that our evolutional algorithm can improve the performance of frame rates about 40-fold. Moreover, the final implementation in a compute shader based on a GPU, which is 80 times faster than one implemented in C++ employing an IPP library and 420 times faster than one implemented in Matlab (all three use our evolutional algorithm), has frame rates up to 52 fps, exceeding the recognized real-time standard of 30 fps. Consequently the implementation of depth estimation with a single image on a GPU with our evolutional algorithm has greatly improved performance and successfully achieved real-time application.

References

[1] Rongchun Li, Yong Dou, Jie Zhou, Baofeng Li and Jinbo Xu, "From WiFi to WiMAX: Efficient GPU-based Parameterized Transceiver across Different OFDM Protocols," *KSII Transactions on Internet and Information systems*, vol. 7, no. 8, pp. 1911-1932, August, 2013.
Article (CrossRef Link)

[2] J. Kim and S. Hyeon, "Implementation of an SDR System Using Graphics Processing Unit," *IEEE Communications Magazine*, vol. 48, no. 3, pp. 156-162, March, 2010.
Article (CrossRef Link)

[3] K. Kim, S. Lee, D. Hong and J. C. Ryou, "GPU-Accelerated Password Cracking of PDF Files," *KSII Transactions on Internet and Information Systems*, vol. 5, no. 11, pp. 2235-2253, November, 2011.
Article (CrossRef Link)

[4] J. Woetzel and R. Koch, "Real-time Multi-stereo Depth Estimation on GPU with Approximative Discontinuity Handling," in *Proc. of 1st European Conference on Visual Media Production (CVMP)*, pp. 245-254, March, 2004.
Article (CrossRef Link)

[5] R. Yang and M. Pollefeys, "Multiresolution Real-time Stereo on Commodity Graphics Hardware," in *Proc. of Conference Society on Computer Vision and Pattern Recognition*

(CVPR), vol. 1, pp. 211-217, June, 2003.
Article (CrossRef Link)

[6] C. Zach, A. Klaus, B. Reitinger and K. Karner, "Optimized Stereo Reconstruction Using 3D Graphics Hardware," in *Proc. of Workshop of Vision, Modeling and Visualization (VMV)*, pp. 119-126, August, 2003.
Article (CrossRef Link)

[7] Y. Bando, B. Chen and T. Nishita, "Extracting Depth and Matte Using a Color-filtered Aperture," *ACM Transactions on Graphics (TOG)*, vol. 27, no. 5, pp.1–9, December, 2008.
Article (CrossRef Link)

[8] B. Liu, S. Gould and D. Koller, "Single Image Depth Estimation from Predicted Semantic Labels," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1253-1260, June, 2010.
Article (CrossRef Link)

[9] S. H. Lai, C. W. Fu & S. Chang, "A Generalized Depth Estimation Algorithm with a Single Image," *IEEE Transactions on Pattern Analysis ans Machine Intelligence (PAMI)*, vol.14, no. 4, pp. 405-411, April, 1992.
Article (CrossRef Link)

[10] R.Ng, M. Levoy, M. Br´e Dif, G. Duval, M. Horowitz and P. Hanrahan, "Light Field Photography with Hand-held Plenoptic Camera," *Tech. Rep. CSTR* 2005-02, Stanford Computer Science, April, 2005.
Article (CrossRef Link)

[11] A. Veeraraghavan, R. Raskar, A. Agrawal, A. Mohan and J. Tumblin, "Dappled photography: mask enhanced cameras for heterodyned light fields and coded aperture refocusing," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, pp. 1-12, 2007.
Article (CrossRef Link)

[12] V. Kolmogorov and R. Zabih, "Multi-camera Scene Reconstruction via Graph Cuts," in *Proc. of Seventh European Conf. Computer Vision*, vol 3, pp. 82-96, May, 2002.
Article (CrossRef Link)

[13] H. Hirschmueller, "Improvements in Realtime Correlation-based Stereo Vision," in *Proc. of IEEE Workshop on Stereo and Multi- Baseline Vision*, pp. 141-148, December, 2001.
Article (CrossRef Link)

[14] S. Sengupta, A. E. Lefohn and J. D. Owens, "A Work-efficient Step-efficient Prefix Sum Algorithm," in *Proc. of the Workshop on Edge Computing Using New Commodity Architecture*, pp. 26-27, May, 2006.
Article (CrossRef Link)

[15] M. J. McDonnell, "Box-filtering Techniques," *Computer Graphics and Image Processing*, vol. 17, pp. 65-70, 1981.
Article (CrossRef Link)

[16] D. W. Hillis and G.L.Steele, Jr, "Data Parallel Algorithms," *Communications of the ACM*, vol. 29, no. 12, pp. 1170-1183, December,1986.
Article (CrossRef Link)

[17] Richard Williams, "All in Good Timecode," *Adobe Magazine*, pp. 57-59, 1999.
Article (CrossRef Link)

[18] Ze-Nian Li and Mark S.Drew, "Fundamentals of Multimedia," *China Machine Press*, p. 104, 2004.
Article (CrossRef Link)

[19] J. P. Lewis, "Fast Template Matching," in *Proc of Vision Interface*, pp. 120-123, May, 1995.

Article (CrossRef Link)

[20] I.T. Jolliffe, *Principal Component Analysis*, Springer-Verlag, New York, 1986.
Article (CrossRef Link)

[21] J. Kim, V. Kolmogorov and R. Zabih, "Visual Correspondence Using Energy Minimization and Mutual Information, " in *Proc. of International Conference on Computer Vision (ICCV)* vol.2, pp. 1033-1040, 2003.
Article (CrossRef Link)

[22] L. Wang, L. Miao, G. Minglun, R. Yang and D Nister,"High-quality Real-time Stereo Using Adaptive Cost Aggregation and Dynamic Programming," in *Proc. of Third International Symposium on 3D Data Processing, Visualization and Transmission*, pp. 798-805, 2006.
Article (CrossRef Link)

Yueh-Teng Hsu received an M.Sc. degree in electrical engineering from the National Taiwan University in 1997 and a Ph.D. in electrical engineering from Chang Gung University in 2007. From 2004 to 2013, he served as R&D manager at Liteon Co. and Mitac Co. at Taiwan. Presently he serves as a Director of R&D at A-MTK Co., developing mobile applications with GPU-based algorithms. His research interests include parallel computation algorithms, software basebands of DAB/DVB receivers and computational photography.

Chun-Chieh Chen received a B.Sc. degree in electronic engineering from the National Taipei University of Technology in 2012, and studied at the Graduate Institute at the National Taipei University of Technology.

Shu-Ming Tseng received a B.Sc. degree from the National Tsing Hua University, Taiwan, and an M.Sc. and Ph.D. from Purdue University, IN, USA, all in electrical engineering, in 1994, 1995, and 1999, respectively. He worked for the Department of Electrical Engineering, Chang Gung University, Taiwan, from 1999 to 2001. Since 2001, he has worked for the Department of Electronic Engineering, the National Taipei University of Technology, Taiwan, where he is currently a Professor. His research interests are MIMO, OFDM,queueing analysis, software defined radio, cooperative communications, and power-line communications. He has served as an editor for KSII Transactions on Internet and Information Systems, indexed in SCIE, since 2013. Prof. Tseng served as a Technical Program Committee member for symposia of the IEEE VTC Fall 2003, WirelessCom 2005, IWCMC 2006, M-CCN 2007, WiCON 2010, ISITA2010/ISSSTA2010, APWCS 2011, etc. He has been listed in Marquis Who's Who in the World since 2006. His team implemented a real-time PC-based software DAB receiver in 2006 and the real-time PC-based software DVB-T receiver in 2012.