Light Field Depth Estimation on Off-the-Shelf Mobile GPU

Andre Ivan† andreivan13@gmail.com
Williem‡ williem@binus.edu
In Kyu Park† pik@inha.ac.kr

†Dept. of Information and Communication Eng., Inha University, Incheon 22212, Korea
‡School of Computer Science, Bina Nusantara University, Jakarta, Indonesia 11480

Abstract

While novel light processing algorithms have been continuously introduced, it is still challenging to perform light field processing on a mobile device with limited computation resource due to the high dimensionality of light field data. Recently, the performance of mobile graphics processing unit (GPU) increases rapidly and GPGPU on mobile GPU utilizes massive parallel computation to solve various computer vision problems with high computational complexity. To show the potential capability of light field processing on mobile GPU, we parallelize and optimize the state-of-the-art light field depth estimation which is essential to many light field applications. We employ both algorithm and kernel-based optimization to enable light field processing on mobile GPU. Light field processing involves independent pixel processing with intensive floating-point operations that can be vectorized to match single instruction multiple data (SIMD) style of GPU architecture. We design efficient memory access, caching, and prefetching to exploit light field properties. The experimental result shows that the light field depth estimation on mobile GPU obtains comparable performance as on the desktop CPU. The proposed optimization method gains up to 25 times speedup compared to the naïve baseline method.

1. Introduction

Recently, research on light field has become one of the most active research areas in computer vision community, owing to its capability of storing richer information compared to conventional 2D images. Research on light field also thrives on the availability of portable 4D light field capturing devices, such as Lytro [1] and Raytrix [3] alongside synthetic light field dataset [30]. An exhaustive survey by Wu *et al*. [34] shows the overview of recent advance on light field research. Various researches also show exciting application of light field, such as digital refocusing [21], matting [12], detection and classification [20, 29], deblurring [23], editing [16, 33], and depth estimation [13, 19, 26, 28, 31, 32, 35, 37]. Among those applications, depth estimation is the most active research topic, as depth is essential for light field editing, display, and other vision-based applications.

On the other hand, light field processing requires abundant computing resources. Therefore, most development has been done in the desktop environment. However, light field processing in a mobile environment with limited resource poses challenging issues. The major issues are memory and computation complexity in handling light field high dimensionality. Yuttakonkit [36] explores the performance of light field application on development board with embedded GPU, but it is still limited to simple local correspondence computation. Agus *et al*. [4] and Jones *et al*. [17] use GPU for light field display and rendering. It indicates that recent development of middle-level light field processing and its application are not yet fully explored on most resource-constrained device, such as mobile phone. Note that, in this paper, we refer GPU on mobile phone as mobile GPU, while GPU on field programmable gate array (FPGA) and development board as embedded GPU.

Recently, the performance of mobile GPU has increased dramatically both in computation capability and memory bandwidth. However, mobile GPU still has its drawbacks compared to existing embedded GPU. The freedom of hardware-based optimizations is limited when dealing with off-the-shelf mobile GPU. Therefore, in this paper, the proposed optimization approach is built upon algorithms and kernel approach (software-based optimization). We believe mobile phone offers extended portability and availability by enabling light field to reach a broader spectrum of users. Che *et al*. [7] and Park *et al*. [22] give better understanding and guideline on maximizing desktop GPU capability on various problem including image processing.

In this paper, we explore challenges, findings, and solutions of employing light field depth estimation on off-the-shelf mobile GPU. We employ the recent state-of-the-art data cost for light field depth estimation, *i.e.* constrained angular entropy (CAE) introduced by Williem *et al*. [32]. We exploit the angular patch to enforce spatial locality and

memory caching for fast pixel access. CAE makes use of RGB channels and histogram to compute the correspondence cost, which can be vectorized to maximize the SIMD traits of GPU. To improve the efficiency of memory access, we use coalesced memory access exhibited by angular patch and histogram. We observe that the histogram computation has slow memory access without enough floating-point operation, yielding bottleneck of using CAE. Therefore, to reduce the memory access latency, binned histogram [27] is employed to compute the correspondence cost. We combine both works and introduce the binned CAE which is a novel data cost for fast light field depth estimation on mobile GPU. Finally, our experimental results demonstrate the capability of fast and robust light field depth estimation on an off-the-shelf mobile phone.

2. Related Works

Light Field Depth Estimation A number of depth estimation methods have been proposed using various properties of light field, such as angular patch [28], epipolar plane image (EPI) [30], defocus cue [25], and learning based method [13]. Works that integrate both correspondence and refocus cue exhibits more robust results. Wang *et al*. [28] proposed an occlusion-aware depth estimation algorithm, enforcing angular photo-consistency through finding a line that separate occlusion and non-occluded region. Tao *et al*. [25] combined both correspondence and defocus cues to utilize EPI and Laplacian operator to compute correspondence and defocus responses, respectively. Then, it was extended [26] by adding shading constraint to improve the original correspondence and defocus data cost. Williem *et al*. [32] proposed an angular patch based data cost called constrained angular entropy (CAE) and constrained adaptive defocus (CAD), which is an extension of their initial work [31]. These data costs employ a constrained histogram and color similarity constraint to find correspondence and defocus responses, respectively. The integration of both data costs demonstrates a dense depth estimation which is robust to occlusion and noise.

Stereo Image Depth Estimation Stereo matching is similar to light field depth estimation, with the same goal of finding the disparity between different views. Choi *et al*. [9] proposed an efficient GPU based graph cut algorithm for stereo matching. Hofmann *et al*. [14] and Eibensteiner [11] proposed an architecture based stereo system on FPGA, semi-global matching (SGM), and event-based stereo matching, respectively. Park *et al*. [22] proposed the metrics to evaluate the effectiveness of parallel implementation on various image processing algorithm including stereo matching. Barron *et al*. [5] proposed a fast stereo matching in bilateral space for rendering a synthetically defocused image on mobile platform.

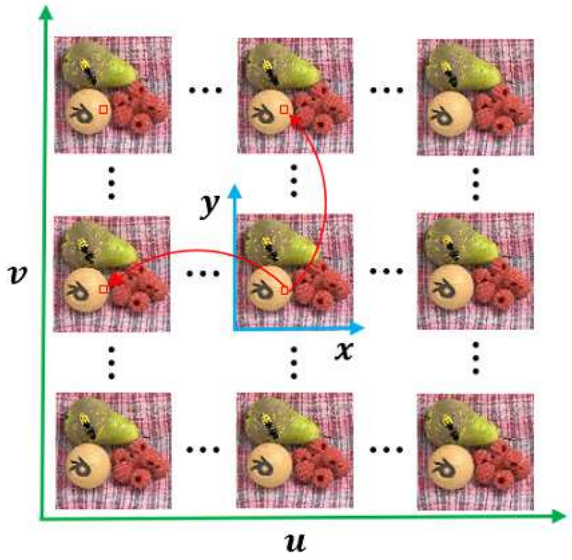


Figure 1: 4D light field representation $L(x, y, u, v)$. Red boxes are corresponding pixels across sub-aperture images.

Acceleration of Light Field Processing Agus *et al*. [4] proposed a volume ray casting system for multi-user light field display using single programmable GPU. Inspired by Hou *et al*. [15], Jones *et al*. [17] proposed a rendering technique for interactive 360° light field display using GPU vertex shaders. Lanman *et al*. [18] proposed a GPU-accelerated near light field display that employs real-time renderer for ray tracing and backward compatible fragment shader. Yuttakonkit [36] proposed a coarse grain re-configurable accelerator (CGRA) to handle short-burst memory transfer when accessing pixel value between each sub-aperture image. Another work by Chang *et al*. [6] employed a modified depth estimation method of Chen *et al*. [8] for light field captured by a pinhole-mask-array camera. Note that most GPU usages on light field were mostly for rendering purpose or a simple light field depth estimation. To the best of our knowledge, there has been no work of light field depth estimation on mobile GPU.

3. Background

3.1. Constrained Angular Entropy (CAE) Cost

CAE is an occlusion-aware data cost for light field depth estimation. In this paper, we use the 4D light field parametrization $L(x, y, u, v)$, as shown in Figure 1. $L(x, y, u, v)$ is sheared to $L_\alpha(x, y, u, v)$ for each depth candidate α described as follows.

$$L_\alpha(x, y, u, v) = L(x + \nabla_x(u, \alpha), y + \nabla_y(v, \alpha), u, v) \quad (1)$$

where (x, y) are the spatial coordinate and (u, v) are the angular coordinate.

$$\nabla_x(u, \alpha) = (u - u_c)(\alpha - \alpha_c)k \quad (2)$$

Algorithm 1 GPU Framework

procedure DEPTH ESTIMATION($L(x, y, u, v)$) ▷ Input 4D light field
  Image2D and buffer initialization
  Data transfer ▷ Host → Device
  for $\alpha :=$ depth search range **do**
    $L_\alpha(x, y, u, v)$ ▷ Light field shearing
    $\tilde{H}(\mathbf{p}, \alpha)$ ▷ CAE data cost
  end for
  $arg\,min(H)$ ▷ Winner takes all
  Data retreival ▷ Device → Host
  return $depth$ ▷ Output disparity map
end procedure

$$\nabla_y(v, \alpha) = (v - v_c)(\alpha - \alpha_c)k \tag{3}$$

where (u_c, v_c) denotes the coordinate of the center pinhole image. ∇_x and ∇_y are the shift value in horizontal and vertical direction according to α with disparity label k. α_c represents depth label α with zero disparity. Then, angular patch at pixel $\mathbf{p}(x, y)$ can be extracted from $L_\alpha(x, y, u, v)$ as follows.

$$A_\alpha^{\mathbf{p}} = L_\alpha(x, y, u, v) \tag{4}$$

To calculate the correspondence cost, Williem *et al*. [32] assume Lambertian reflectance without ignoring occlusion. The assumption also holds even in the presence of occlusion in angular patch, as long as the majority of pixels are still photo-consistent. The cost is computed using adaptive entropy which measures the randomness or the degree of intensity dominance inside angular patch.

The adaptive entropy cost is computed for each color channel using histogram. For each intensity i in the histogram, weight $w(i)$ is computed based on the intensity difference between pixel in the center and the rest of pixels in the angular patch A_α. The adaptive or constrained histogram is built as follows.

$$w(i) = \exp\left(-\frac{|i - A_\alpha(u_c, v_c)|^2}{2\sigma^2}\right) \tag{5}$$

$$g(i) = w(i)h(i) \tag{6}$$

where $h(i)$ is the probability of the intensity i in $A_\alpha^{\mathbf{p}}$. Then the adaptive entropy of each channel is computed by

$$\tilde{H}(\mathbf{p}, a) = -\sum_i \frac{g(i)}{|g|} \log\left(g(i)\right) \tag{7}$$

where $\frac{g(i)}{|g|}$ represents the normalized value of the constrained histogram. The cost for each channel is then integrated through average pooling as follows.

$$C(\mathbf{p}, \alpha) = \frac{\tilde{H}_R(\mathbf{p}, a) + \tilde{H}_G(\mathbf{p}, a) + \tilde{H}_B(\mathbf{p}, a)}{3} \tag{8}$$

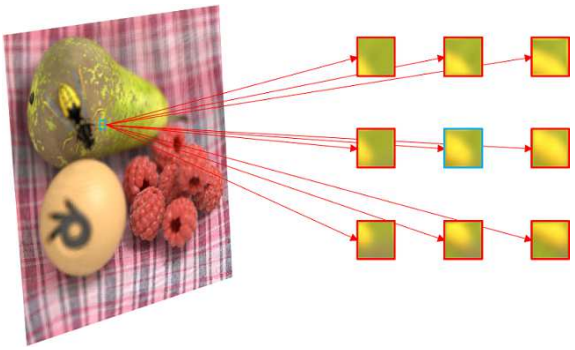


Figure 2: Angular patch extracted from $L_\alpha(x, y, u, v)$. Blue box denote the center of angular patch $\mathbf{p}(u_c, v_c)$, red box denote pixels from sub-aperture image $\mathbf{p}(u, v)$. Only 3×3 angular patch is visualized.

Finally, the depth map is obtained by finding the minimum cost across α for each pixel.

Note that Williem *et al*. [32] combine both CAE and CAD. However, we deem CAE is robust enough and we omit CAD to reduce the complexity of the whole framework. For more information regarding CAE and CAD we encourage the reader to refer to [32]. In this paper, we focus on a light field depth estimation framework that is suitable for mobile platform by modifying CAE data cost.

3.2. OpenCL

OpenCL [24] is an open standard parallel programming language for heterogeneous processors for CPU, GPU, and digital signal processor (DSP) on cross-platforms including desktop, mobile, and embedded devices. In OpenCL, code or function that executes on a device with multiple threads is called a kernel. Work-item is a unit of work that executes an instruction from kernel. A bundle of work-items is called work-group that synchronizes multiple work-item executions while communicating with each other using shared memory. Each work-item executes the kernel on a single thread and the whole threads processes a group of pixels in a data parallel manner. The size of the entire problem is referred as global size while work-group size is called local size.

To utilize mobile GPU (device), host (CPU) copies the whole data into GPU global memory and maps the partition of data onto each work-group. Host then stacks kernel execution into the GPU work queue which performs computation on the partitioned data. To perform instructed computation, each thread needs to access data in the global memory. Since access to global memory is slow, it is important to reduce the amount of global memory access by keeping the frequently accessed data onto the shared or local memory. After GPU finishes all of its work queues, the data can be retrieved back from GPU to host memory.

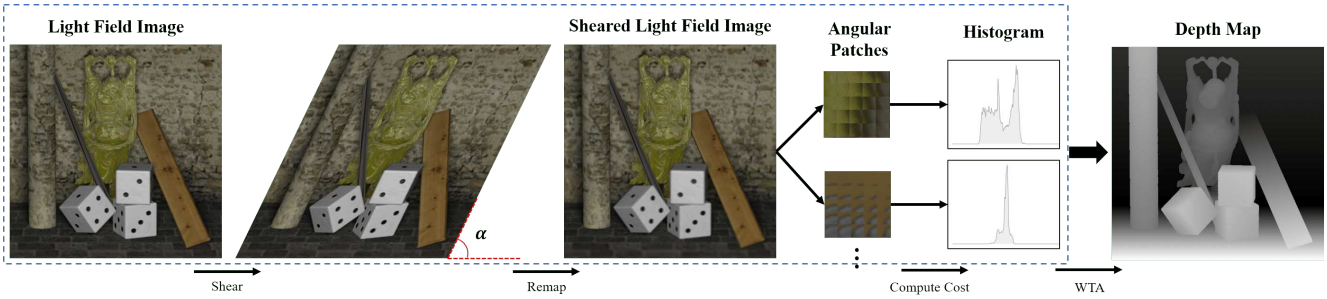


Figure 3: Visualization of our framework. Dashed blue region is computed multiple times based on depth search range.

4. Light Field Processing on Mobile GPU

4.1. Overview

The proposed light field depth estimation on GPU is divided into three kernels: light field shearing, per-pixel CAE computation, and disparity computation using winner-takes-all (WTA), as listed in Algorithm 1. Since the required memory space is larger than the amount of device memory, multiple times of kernel calls and the $L_\alpha(x, y, u, v)$ is processed iteratively. First, light field is sheared for each α and angular patch at α is obtained, as shown in Figure 2. Next, correspondence cost is calculated using CAE by building a constrained histogram for the pixels in the angular patch. Finally, WTA is applied to find the disparity with minimum cost across the depth candidates.

The initial GPU implementation of CAE is considered as the baseline implementation for the speedup evaluation of the proposed optimization method. We also use the available Matlab implementation on the desktop environment as another baseline for both qualitative and quantitative evaluation.

Figure 3 visualizes the general scheme of our framework. Each thread estimates depth for each pixel in center image pinhole $\mathbf{p}(u_c, v_c)$. Each thread accesses $(u \times v)$ pixels from remapped light field and computes the cost of that pixel. Therefore, it needs a considerable amount of global memory access. However, the histogram and entropy computation involve a significant amount of floating-point computation to hide the latency of global memory access.

In our approach, we enforce SIMD-style at thread level for efficient instruction and memory transaction discussed in Section 4.2. Furthermore, we reduce the latency of global memory access through efficient memory access explained in following Section 4.3, Section 4.4, and Section 4.5. Section 4.6 discusses the bottleneck due to the amount of branching operation and high latency memory access. Finally, the data mapping and algorithm complexity are discussed in Section 4.7.

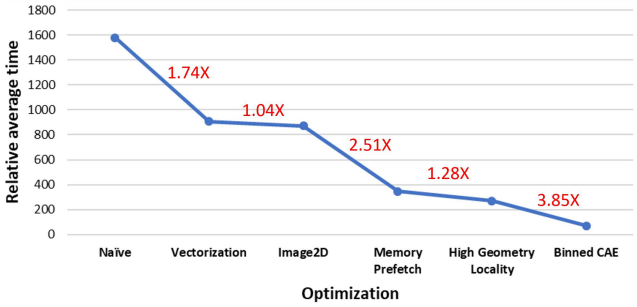


Figure 4: Cumulative execution time after each optimization. Y axis shows average execution time in seconds across 5 synthetic light field images from Wanner *et al*. [30].

4.2. Vectorization

Throughout the framework pixel-wise computation, such as light field shearing and CAE cost computation, is performed for each R, G, and B channels. In light field shearing, pixels are interpolated to the remapped light field for each color channel. CAE computes the cost in three channels independently and then the costs are integrated. The type of computations performed above for each color channel are identical and can be performed in SIMD-style.

In order to avoid redundant computation and memory access for each color channels, SIMD-style approach is employed at thread level to exploit data parallelism which is well supported in modern GPU architecture [7] [22]. Using OpenCL vector data type, same instruction is performed simultaneously for multiple data, yielding significant performance gain at instruction and memory level. In addition, memory access in vectorized approach exhibits less latency because it reduces the frequency of memory access. Note that this approach is equivalent to a small-scale memory coalescing.

Sheared pixels remapped in (1) are sampled using bilinear interpolation. The computation of weighted sum is performed using the vector type (*float-3*) discarding the alpha channel. Histogram computation in (5)∼(8) is also com-

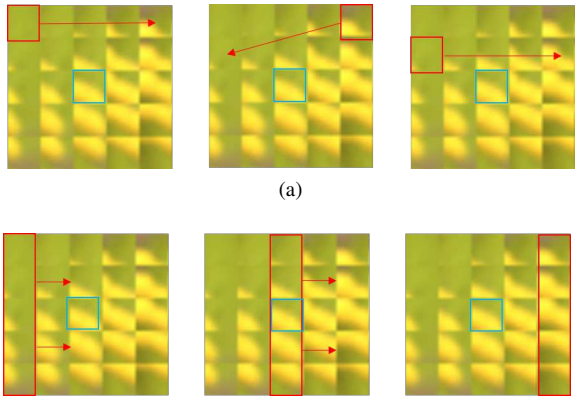


Figure 5: Comparison of pixel access within angular patch. (a) Shows common pixel accesses in angular patch. (b) Shows a column based parallelized pixel access. High local geometry exists between adjacent pixels in the same row or column.

puted using the vector type. The vectorization reduces the amount of instruction and memory transaction by a factor of three. Vectorization speeds up the framework by 1.74X compared to the baseline implementation. Cumulative improvement of computation time is shown in Figure 4.

4.3. OpenCL Memory Object and Texture Cache

One of the problems mentioned in [36] is the short-burst memory access when accessing pixels from each sub-aperture image. Access pattern in image processing is often sequential or structured. However, this does not apply to light field. As shown in Figure 1, long index jump is needed to access corresponding pixel from each sub-aperture image. Therefore, it limits the peak memory bandwidth due to incoherent memory access. However, to access pixels in angular patch, it only requires a simple iteration in horizontal and vertical direction, as shown in Figure 5. This trait is highly suited for OpenCL memory object called *Image2D*.

Image2D is regularly used to perform window-based filtering operation (*i.e*. Sobel and Gaussian) where pixels of interest are within the window or patch vicinity. *Image2D* thrives on this kind of operation due to the existence of texture cache. Texture cache is optimized for 2D spatial locality. Threads of the same warp that read texture addresses close together achieve the best performance [2]. Note that access pattern within angular patch can either be row-major or column-major order. *Image2D* improves the performance by 1.04X. While the speedup is not too significant, it plays an important role for the subsequent optimizations.

4.4. Memory Prefetch

Pixels in angular patch can be accessed through a simple iterative loop shown in Figure 5(a). Furthermore, histogram

Table 1: Average processing time (in seconds).

LF Input	Desktop CAE	Baseline GPU	GPU CAE	Binned CAE
Wanner	219.884	1578.644	273.668	62.686
Lytro Illum	64.581	515.153	98.5355	21.111

Table 2: MSE across 5 synthetic light field images.

Data Cost	Buddha	Buddha2	MonaRoom	Papilon	StillLife
CAE 9×9	0.0049	0.0112	0.0073	0.0162	0.0159
CAE 5×5	0.0111	0.0247	0.0143	0.0338	0.0856
GPU CAE	0.0105	0.0255	0.0138	0.0365	0.0572
Binned CAE	0.0335	0.0361	0.0289	0.0819	0.0997

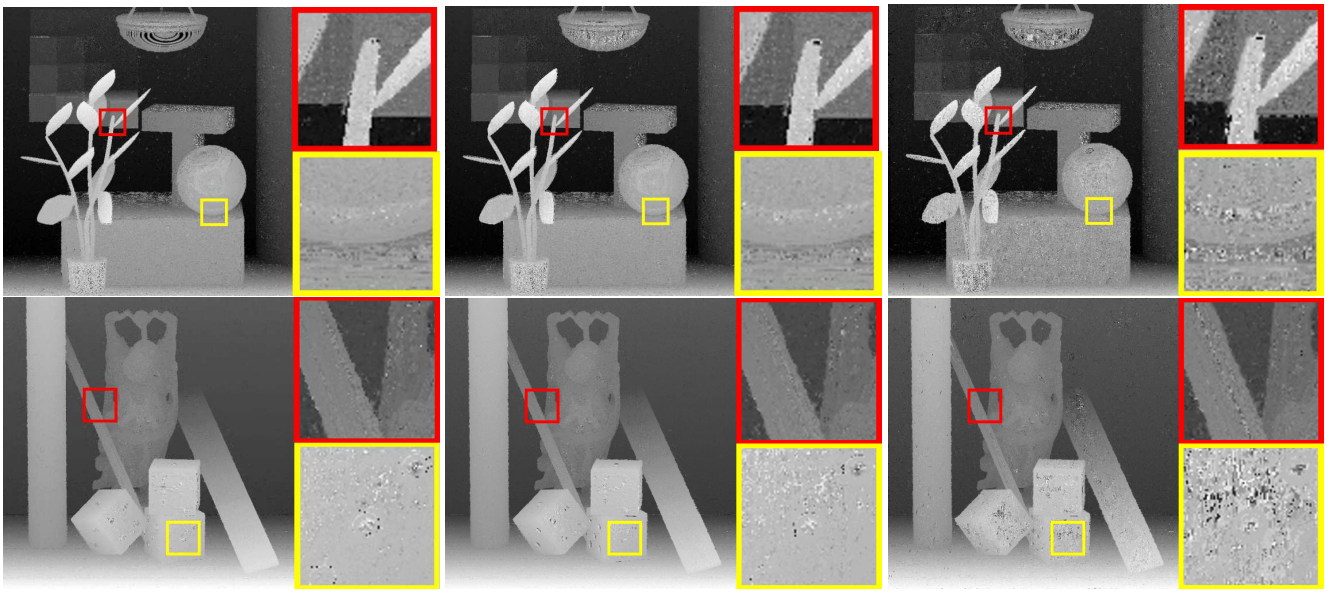
operation in (6) and (7) is highly iterative. We utilize memory prefetch for global and constant address spaces. Memory prefetch is beneficial only to a well-detected memory access pattern (*i.e*. memory access in explicit loop). The prefetched data is stored in L2 cache which provides faster access than the global memory or private memory access. OpenCL enforces auto prefetching as long as the compiler can detect the well-patterned memory access. Therefore, it is crucial to explicitly design the patterned memory access, especially on histogram operation in (7). The example is shown as follow.

```
▷ Option 1
g2[i] = g[i] / |g|.
Entropy_cost += (g2[i]) * log(g[i])
▷ Option 2
Entropy_cost += (g[i] / |g|) * log(g[i])
```

Option 1 shows the conventional programming approach that enhances code readability. However, the thread needs to load two different addresses. Option 2 enforces efficient memory access and memory prefetch with trade-off of readable code. The duplicate memory access instruction reduces the amount of transaction in memory side. The former histogram is loaded and stored in cache. Thus, the later histogram access is merely a load from cache. Memory prefetching improves the performance by 2.51X which is a significant speedup leading to the identification of bottleneck in our framework. The bottleneck in histogram computation is discussed in Section 4.6.

4.5. Global Memory Coalescing

Global memory coalescing conserves bandwidth and reduces effective latency. Instead of accessing pixels in stride from angular patch, we load a whole column in angular patch and process it simultaneously. Note that read instruction from global memory is aligned and contiguous. In addition, *Image2D* encourages memory access for pixels close together to gain more performance. Latency of effective global memory access is reduced through processing the whole column of angular patch simultaneously, as shown



(a) (b) (c)

Figure 6: Disparity map comparison. (a) Matlab 5×5. (b) GPU CAE. (c) Binned CAE. Binned CAE still holds the occlusion aware characteristic of CAE while slightly suffers in smooth region.

in Figure 5(b). The global memory coalescing improves the computation time by 1.28X.

4.6. Binned CAE

Inefficient memory access with high latency is performed for every histogram loop. For each memory access, only simple floating-point operation is performed. This is the bottleneck of CAE algorithm. In addition, *if*-branching interrupts the control flow of work-group [7]. This degrades the efficiency of parallel computing by causing threads of the same warp to diverge [22]. However, the usage of *if-else* is necessary to clear out *NaN* value and to avoid full histogram loop. It is rare for a single pixel to contain most of the intensity value inside histogram. Therefore, *if-else* is important to prune unnecessary operation.

The idea to deal with unnecessary 256 histogram loop is to bin the histogram, inspired by works of Vaish *et al*. [27]. Instead of applying uniform weight as in [27], we apply weight computed from (5) to the binned histogram. We bin the histogram to 64 intensity value, and modify the computation of $|g|$. Instead of standard iterative sum, we use dot product to compute $|g|$. The dot product is performed between histogram in vector of *float4* and matrix of ones. Note that OpenCL dot product operation is limited to only four elements.

Binned CAE limitation is when dealing with smooth region. Pixels with similar intensity might be rounded into the wrong bin, as shown in Figure 6. However, binned CAE still preserve the structure of the scene. Pixels at edge region have distinct intensity difference due to the presence of occluder. Therefore, binned histogram still can estimate the cost of those pixels accurately. It shows that the occlusion-aware characteristic still holds in binned CAE. Increasing σ helps alleviate the false estimated depth, while smoothing the depth slightly. We reduce the amount of binned histogram loop by a factor of four and gain 3.85X speedup.

4.7. Data Mapping on the GPU

We create $W \times H$ threads, where each thread computes the cost for each pixel. W and H denote spatial width and height of light field image. Each thread accesses $u \times v$ pixels in $L(x, y, u, v)$. We execute $O(D)$ call to the kernel except for WTA, where D denotes the depth search range. Within each call, the kernel handles $O(WH)$ pixels. WTA kernel needs one time $O(WH)$ call after cost is computed. Total computational complexity is $O(WH(D + 1))$. Mali G-71 has maximum 8192 threads (T_{max}), coming from 32 cores with 16×16 maximum work-group size. Thus, the actual computational complexity is $O(\frac{WH(D+1)}{T_{max}})$.

The input light field image is copied to the global memory giving read/write access to thread. The remapped light field is stored in texture memory. Angular patch in sheared light field image is stored in texture cache. The cost volume is stored in global memory, while histogram is stored in the register. Unfortunately, pixels in the angular patch do not have any spatial relationship. Therefore, shared memory has no significant impact on this work. The estimated depth is stored in global memory and retrieved back at the

Table 3: Bad pixel percentage (%) across 5 synthetic light field images.

Data Cost	Buddha	Buddha2	MonaRoom	Papilon	StillLife	Buddha	Buddha2	MonaRoom	Papilon	StillLife
	BP $\delta = 2$					BP $\delta = 4$				
CAE 9×9	4.6789	18.8822	16.0044	29.7201	3.9456	1.4957	8.0792	7.3861	11.9091	1.4208
CAE 5×5	12.9467	41.0256	32.4049	45.2433	12.6884	4.1909	20.4042	15.7398	24.2615	5.1293
GPU CAE	14.0523	39.7536	34.6369	46.2251	9.4842	4.3235	19.7045	16.8294	24.7382	3.3551
Binned CAE	32.7813	46.8736	55.6365	59.1146	14.0878	15.4853	26.0591	33.4552	37.9852	6.2902

end of work queues.

5. Experimental Result

We conduct the experiments on Samsung Galaxy Note 8, which equips Exynos 8895 Octa chipset and Mali-G71 GPU. Mali-G71 has 32 shader cores and 2.1MB L2 cache with 2.1GB global memory, 524KB memory cache, and up to 200 GFLOPS for single-precision *vec4* floating-point computation performance. Memory copy operation from host to device is up to 4.0GB/s bandwidth and 2.8 GB/s from device to host, respectively. For desktop environment, we use an Intel i7-7700 @3.6 GHz with 16 GB RAM PC. We utilize same parameter as used in [32], with depth search range = 75, and $\sigma = 10$.

For quantitative evaluation, we utilize a dataset generated by Wanner *et al*. [30]. Among the dataset, we select five synthetic light field images which have the same spatial size, with resolution of 5×5×768×768. We also use a real light field image (5×5×432×432) captured using Lytro Illum camera to shows the flexibility of our implementation. The angular dimension is cropped due to insufficient Android application memory when loading light field image. Only 5×5 sub-aperture images around the center are maintained. Another approach to crop the light field image is to sample the sub-aperture image sparsely. Note that light field image size is around 100 MB and 40 MB for original and cropped synthetic light field, respectively. We use toolbox provided by Dansereau *et al*. [10] to extract the light field images. For all three kernels, we use a total of 256 work-items (threads) for a single work-group which is the maximum capacity for Mali-G71. The number of work-item is decided through a series of empirical test. The global size is the spatial dimension of input light field image.

Since CPU implementation on mobile for comparison is not feasible, we compare our implementation on mobile GPU with original CAE running in our desktop environment. Our GPU implementation shows comparable performance to desktop GPU in terms of computation time with similar quantitative error. Note that binned CAE surpasses desktop computation speed. Table 1 shows the average processing time comparison across different implementations and inputs. Note that we exclude input (image load) and output (display) process in the computation time. The computation time difference between each input image is due to histogram calculation and branching. Image with more intensity variation needs more iteration to compute the histogram. Performance comparison between Android and desktop environment available in Table 2 and Table 3. CAE 9×9 and CAE 5×5 are Williem *et al*. [32] original Matlab implementation on desktop environment, where 5×5 and 9×9 represents light field angular dimension. The Matlab implementation is available on their project page. The drop in BP% and MSE from CAE 9×9 to CAE 5×5 is due to the loss of information in angular dimension.

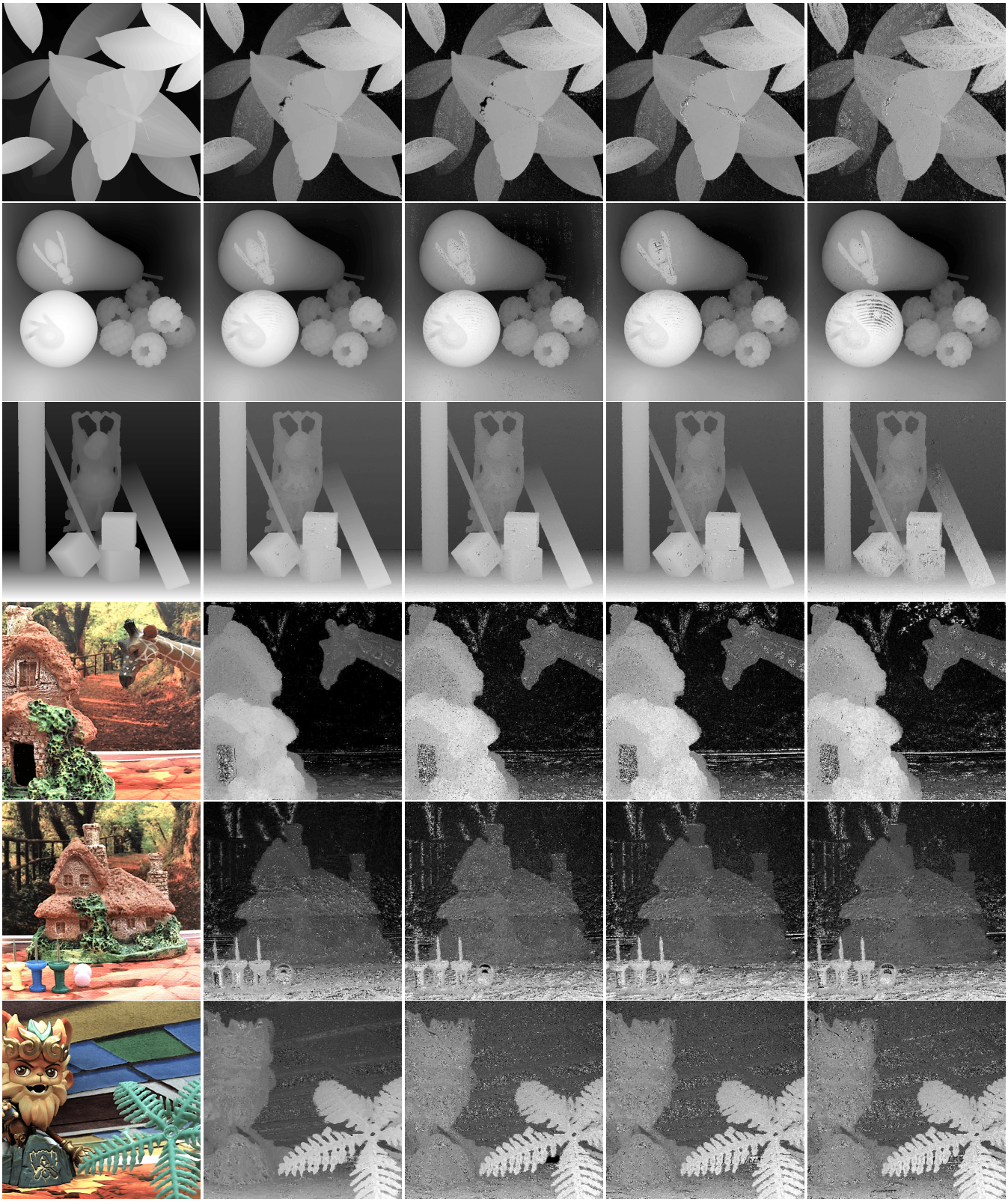
The difference between proposed mobile implementation and Matlab result is due to difference in floating-point precision on mobile and PC. The rounding convention of OpenCL function also affects the result. Our depth results for both synthetic and real light field images are available in Figure 7. The light field image contrast is enhanced only for visual purpose. Note that real light field images are more difficult to deal with since noise and occlusion frequently occur within the scene. In overall, the experimental results show that the proposed method could robustly estimate depth map in both synthetic and challenging real light field images.

6. Conclusion

In this paper, we explored the challenges in processing light field on off-the-shelf mobile GPU and proposed a series of algorithms and kernel-based optimizations. Our methods focused on efficient memory access in angular patch and histogram computation. SIMD-style approach using vector was performed throughout the framework. Coalesced memory access in angular patch was proposed. Novel data cost was proposed to allow fast light field depth estimation on mobile GPU. Finally, experiment results showed that CAE achieved a total of 5.76X and 5.22X speedup, while binned CAE achieved 25.18X and 24.40X speedup both for synthetic and real light field image, respectively.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korean government (MSIT) (2017-0-00142).



(a) (b) (c) (d) (e)

Figure 7: Depth map results from synthetic and real light field data. (a) Depth ground truth (row 1∼3) and real light field image (row 3∼6). (b) CAE 9×9. (c) CAE 5×5. (d) GPU CAE. (e) Binned CAE.

References

[1] Lytro 3D light field camera. https://www.lytro.com/. [Online]. 1

[2] OpenCL best practices. https://www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_OpenCL_BestPracticesGuide.pdf. [Online]. 5

[3] Raytrix 3D light field camera. https://raytrix.de/products/. [Online]. 1

[4] M. Agus, E. Gobbetti, J. A. I. Guitin, F. Marton, and G. Pintore. GPU accelerated direct volume rendering on an interactive light field display. *Computer Graphics Forum*, 27(2):231–240, April 2008. 1, 2

[5] J. T. Barron, A. Adams, Y. Shih, and C. Hernndez. Fast bilateral-space stereo for synthetic defocus. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4466–4474, June 2015. 2

[6] C. W. Chang, M. R. Chen, P. H. Hsu, and Y. C. Lu. A pixel-based depth estimation algorithm and its hardware implementation for 4-D light field data. In *IEEE International Symposium on Circuits and Systems*, pages 786–789, June 2014. 2

[7] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach. Accelerating compute-intensive applications with GPUs and FPGAs. In *Symposium on Application Specific Processors*, pages 101–107, June 2008. 1, 4, 6

[8] C. C. Chen, S. C. F. Chiang, X. X. Huang, M. S. Su, and Y. C. Lu. Depth estimation of light field data from pinhole-masked DSLR cameras. In *IEEE International Conference on Image Processing*, pages 1769–1772, September 2010. 2

[9] Y. K. Choi and I. K. Park. Efficient GPU-based graph cuts for stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 642–648, June 2013. 2

[10] D. G. Dansereau, O. Pizarro, and S. B. Williams. Decoding, calibration and rectification for lenselet-based plenoptic cameras. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1027–1034, June 2013. 7

[11] F. Eibensteiner, J. Kogler, and J. Scharinger. A high-performance hardware architecture for a frameless stereo vision algorithm implemented on a FPGA platform. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 637–644, June 2014. 2

[12] J. Fiss, B. Curless, and R. Szeliski. Light field layer matting. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 623–631, June 2015. 1

[13] S. Heber and T. Pock. Convolutional networks for shape from light field. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3746–3754, June 2016. 1, 2

[14] J. Hofmann, J. Korinth, and A. Koch. A scalable high-performance hardware architecture for real-time stereo vision by semi-global matching. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 845–853, June 2016. 2

[15] X. Hou, L.-Y. Wei, H.-Y. Shum, and B. Guo. Real-time multi-perspective rendering on graphics hardware. In *Eurographics Conference on Rendering Techniques*, pages 93–102, June 2006. 2

[16] A. Jarabo, B. Masia, A. Bousseau, F. Pellacini, and D. Gutierrez. How do people edit light fields? *ACM Trans. on Graphics*, 33(4):146, July 2014. 1

[17] A. Jones, I. McDowall, H. Yamada, M. Bolas, and P. Debevec. Rendering for an interactive 360° light field display. In *ACM Trans. on Graphics*, August 2007. 1, 2

[18] D. Lanman and D. Luebke. Near-eye light field displays. In *ACM Trans. on Graphics*, page 11:1, November 2013. 2

[19] J. Li, M. Lu, and Z. N. Li. Continuous depth map reconstruction from light fields. *IEEE Trans. on Image Processing*, 24(11):3257–3265, July 2015. 1

[20] N. Li, J. Ye, Y. Ji, H. Ling, and J. Yu. Saliency detection on light field. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 39(8):1605–1616, August 2017. 1

[21] R. Ng. Fourier slice photography. *ACM Trans. on Graphics*, 24(3):735–744, July 2005. 1

[22] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. Kim. Design and performance evaluation of image processing algorithms on GPUs. *IEEE Trans. on Parallel and Distributed Systems*, 22(1):91–104, June 2011. 1, 2, 4, 6

[23] P. P. Srinivasan, R. Ng, and R. Ramamoorthi. Light field blind motion deblurring. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3958–3966, July 2017. 1

[24] J. E. Stone, D. Gohara, and G. Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science Engineering*, 12(3):66–73, May 2010. 3

[25] M. W. Tao, S. Hadap, J. Malik, and R. Ramamoorthi. Depth from combining defocus and correspondence using light-field cameras. In *IEEE International Conference on Computer Vision*, pages 673–680, December 2013. 2

[26] M. W. Tao, P. P. Srinivasan, J. Malik, S. Rusinkiewicz, and R. Ramamoorthi. Depth from shading, defocus, and correspondence using light-field angular coherence. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1940–1948, June 2015. 1, 2

[27] V. Vaish, M. Levoy, R. Szeliski, C. L. Zitnick, and S. B. Kang. Reconstructing occluded surfaces using synthetic apertures: Stereo, focus and robust measures. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2331–2338, February 2006. 2, 6

[28] T. C. Wang, A. A. Efros, and R. Ramamoorthi. Depth estimation with occlusion modeling using light-field cameras. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 38(11):2170–2181, January 2016. 1, 2

[29] T.-C. Wang, J.-Y. Zhu, E. Hiroaki, M. Chandraker, A. Efros, and R. Ramamoorthi. A 4D light-field dataset and CNN architectures for material recognition. In *European Conference on Computer Vision*, pages 121–138, February 2016. 1

[30] S. Wanner and B. Goldluecke. Globally consistent depth labelling of 4D lightfields. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 41–48, June 2012. 1, 2, 4, 7

[31] Williem and I. K. Park. Robust light field depth estimation for noisy scene with occlusion. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4396–4404, June 2016. 1, 2

[32] Williem, I. K. Park, and K. M. Lee. Robust light field depth estimation using occlusion-noise aware data costs. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, in press. doi:10.1109/TPAMI.2017.2746858. 1, 2, 3, 7

[33] Williem, S. K. Won, and I. K. Park. Spatio-angular consistent editing framework for 4D light field images. *Multimedia Tools and Applications*, 75(23):16615–16631, December 2016. 1

[34] G. Wu, B. Masia, A. Jarabo, Y. Zhang, L. Wang, Q. Dai, T. Chai, and Y. Liu. Light field image processing: An overview. *IEEE Journal of Selected Topics in Signal Processing*, 11(7):926–954, October 2017. 1

[35] Z. Yu, X. Guo, H. Ling, A. Lumsdaine, and J. Yu. Line assisted light field triangulation and stereo matching. In *IEEE International Conference on Computer Vision*, pages 2792–2799, June 2013. 1

[36] Y. Yuttakonkit and Y. Nakashima. Performance comparison of CGRA and mobile GPU for light-field image processing. In *Fourth International Symposium on Computing and Networking*, pages 174–180, November 2016. 1, 2, 5

[37] S. Zhang, H. Sheng, C. Li, J. Zhang, and Z. Xiong. Robust depth estimation for light field via spinning parallelogram. *Computer Vision and Image Understanding*, 145:148–159, April 2016. 1