IEEE Open Journal of the
Solid-State Circuits Society

Received 9 August 2022; revised 30 October 2022 and 24 December 2022; accepted 6 February 2023. Date of publication 14 February 2023; date of current version 10 April 2023.

Digital Object Identifier 10.1109/OJSSCS.2023.3244759

A 250-mW 5.4G-Rendered-Pixel/s Realistic Refocusing Processor for High-Performance Five-Camera Mobile Devices

PO-HAN CHEN (Graduate Student Member, IEEE), SHU-WEN YANG, AND CHAO-TSUNG HUANG (Member, IEEE)

Department of Electrical Engineering, National Tsing Hua University, Hsinchu 300, Taiwan

CORRESPONDING AUTHOR: P.-H. CHEN (e-mail: pohan@stanford.edu)

This work was supported in part by the Ministry of Science and Technology (MOST) under Grant 108-2622-8-007-017 and Grant 106-2221-E-007-120, and in part by the NOVATEK Fellowship.

ABSTRACT Digital refocusing in multicamera mobile devices is becoming crucial. Realistic refocusing, which is a subset of digital refocusing, provides physically correct quality; however, its intense computational complexity results in low processing speed and restricts its applicability. Moreover, its complex computation flow requires substantial DRAM bandwidth and a large SRAM area, making it more challenging to implement in hardware. In this article, we present a high-performance refocusing processor based on a hardware-oriented realistic refocusing algorithm. The proposed compact computation flow saves 92% of the DRAM bandwidth and 32% of the SRAM area without noticeable quality degradation. To support high-performance refocusing, we develop highly paralleled engines for view rendering. They deliver 5.4G rendered-pixel/s throughput. The hardware accelerator improves the processing speed by 100× to 350× that of the original refocusing algorithm running on a general-purpose processor. The chip is fabricated with 40-nm CMOS technology and comprises 271 kB of SRAM and 2.3M logic gates. The chip processes Full-HD light fields up to 40 frames/s under 250 mW power consumption.

INDEX TERMS DRAM bandwidth reduction, high parallelism, light field, multicamera, refocus, view synthesis.

I. INTRODUCTION

MULTIPLE cameras are deployed on modern smartphones. This multicamera setup enables more and more novel camera features. Among these features, digital refocusing is crucially important. Refocusing creates a shallow depth-of-field effect (known as bokeh) on images, which is often used to direct the attention of viewers within a photographic or cinematographic scene. In addition, many computer graphics technologies, such as virtual reality, need to provide a better sense of depth within a scene. This visual sense can also be rendered by digital refocusing.

Fig. 1 illustrates the algorithmic concept of refocusing. A conventional camera captures images with the bokeh effect by carefully adjusting its focus position and aperture size. On the other hand, refocusing algorithms allow users to take pictures first and render the bokeh effect later. Also, these algorithms do not need a thick lens to create the bokeh effect. This feature makes refocusing preferable on devices with mechanical constraints preventing a thick lens to be installed.

Refocusing can be implemented for various input signals. For 3-D models, view sampling techniques [1], [2], [3], [4] provide decent depth-of-field effects. However, the computation is intensive and GPU acceleration is needed to provide fast processing. With a single image and its depth map, refocusing can also be implemented by applying depth-dependent blur kernels to the image [5], [6]. This solution is widely used in current mobile devices to create a bokeh effect in the background. However, the single-image solution suffers from the problem of missing information in the occluded regions. As shown in Fig. 2, a blurred pixel is formed by summing light rays coming from different surfaces including those of the occluded objects. In a single-image setup, such information is lost and refocusing algorithms have to make

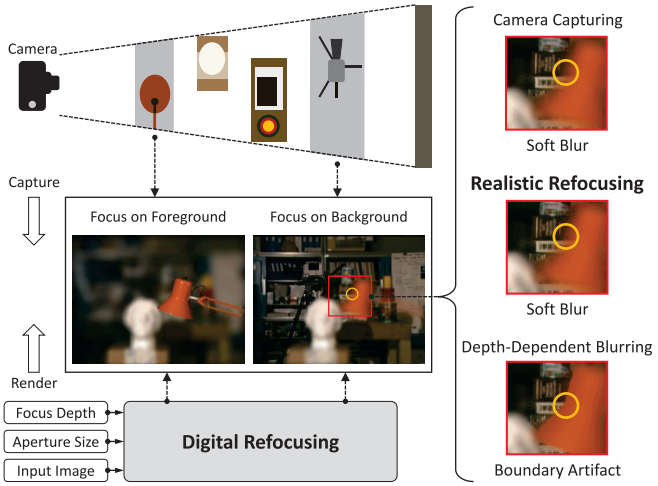


FIGURE 1. Images with a shallow depth-of-field effect can be captured by a camera with a big aperture. By using digital refocusing, such an effect can also be rendered after taking the pictures. Conventional single-image refocusing methods, like depth-dependent blurring, provide fast refocusing speed. However, it will present boundary artifacts on blurred foreground objects. In contrast, realistic refocusing uses multiple images and can achieve soft blur-like images captured by conventional cameras. This work targets the hardware implementation of realistic refocusing.

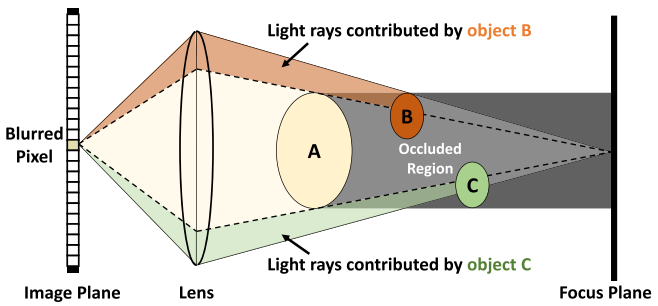


FIGURE 2. Objects A–C all contribute light rays to a blurred pixel when the focus plane is in the background. However, objects B and C are invisible in a single-image setup because they are occluded by object A. To perform refocusing, algorithms have a guess what is sitting in the occluded region created by object A.

a guess. A trivial guess or no guess (i.e., using foreground information only) provides faster processing speed but suffers from boundary artifacts like sharp edges (as shown in Fig. 1 bottom right) [7]. Although a complicated view synthesis algorithm can be adopted, it is still impossible to perfectly recover what is occluded behind an object for refocusing. Light-field images [8], [9], [10], [11] provide the missing information of occluded regions. Methods using light fields [12], [13] need densely sampled view points to avoid aliasing. The resulting huge storage requirement makes them impractical, especially for video applications. Refocusing using sparse light fields [14] reduces the storage. However, the computation of this method is too intensive because the method needs to render hundreds of new images to compensate for the sparsely sampled light field.

With more and more camera lenses being seen on mobile devices, physically correct refocusing using light-field images is becoming more achievable. This work aims at verifying the hardware acceleration approach in terms of

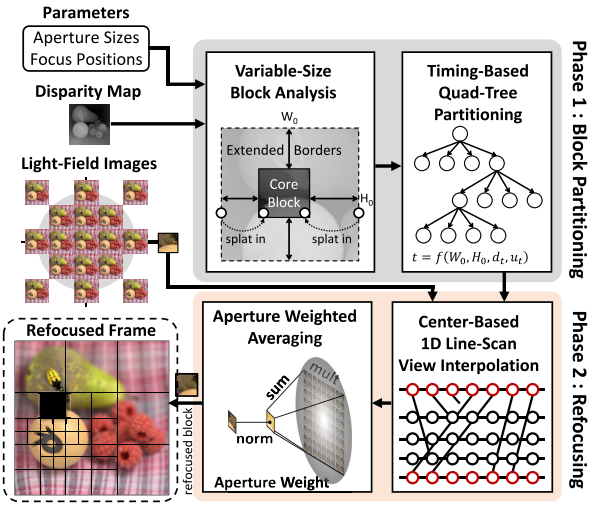


FIGURE 3. BBRR algorithm overview.

performance, energy efficiency, and memory bandwidth. This article extends the work of [15] and presents an integrated circuit based on the block-based realistic refocusing (BBRR) [14] algorithm using sparse light fields. Our main contributions are as follows.

1) A hardware-oriented realistic refocusing algorithm is devised based on BBRR. Our algorithm simultaneously considers DRAM bandwidth, SRAM area, and refocusing quality and is used to design a compact computation flow. The flow reduces 92% of the DRAM bandwidth and 32% of the SRAM area without noticeable quality degradation.
2) Two highly parallel view rendering engines address the computational intensity. Extensive parallel computing is achieved by combining pixel-level, line-level, and view-level parallelism. They together provide 5.4G rendered-pixel/s throughput to support high-quality and high-performance refocusing.

II. OVERVIEW AND PROFILING OF BLOCK-BASED REALISTIC REFOCUSING

Fig. 3 shows the algorithm overview of BBRR. For inputs, this algorithm takes a set of light-field images, a depth map, and a few user-defined parameters. Then, it executes the following two phases to generate a refocused image.

A. BLOCK PARTITIONING

We adopted block-based processing flow because it reduces the overall computational complexity by adapting to local blur statistics. The first step of BBRR derives the block partitioning map from the input depth map. Multiple block sizes of $8 \times 8, 16 \times 16,$ to 128×128 are used to construct the map. Note that a block consists of a core block region and its extended borders. Pixels in extended borders are needed to compute output pixels at the block boundaries. The size of the extended borders is calculated based on the

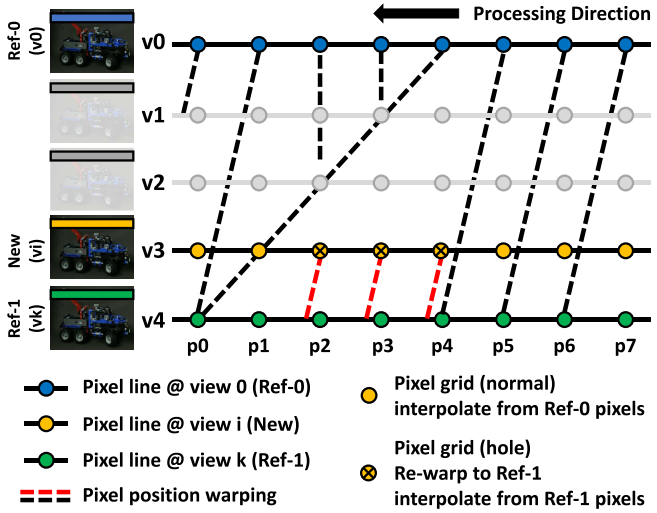


FIGURE 4. Illustration of 1-D line-scan view interpolation. Here, we show an example of interpolating one-pixel line at a new view *v*3 (where *v*0 and *v*4 are reference views). It warps the pixels from the reference view to the new view based on the depth map. It then uses the position information to interpolate the new values.

depth information to make sure it includes the needed pixels. BBRR then uses these variable-size blocks to build a tree structure and analyzes the tree to determine a runtime-optimized partition. The computation of this step is not intensive because the tree analysis is accelerated by using simplified math models. The images are then divided into blocks according to the partition map and sent to the next step for refocusing.

B. REFOCUSING

Refocusing is achieved by blending many views sampled from different angles. Because BBRR uses sparse light fields as input sources, view interpolation is required to render more views before blending to avoid aliasing. BBRR uses a center-based 1-D line-scan view interpolation algorithm to render new views between the input light fields. Fig. 4 illustrates the interpolation process. The algorithm first finds the positions of the reference pixels at the new view (called warping). If a new pixel at the integer position (called a grid pixel) is surrounded by two reference pixels, it interpolates from them. Otherwise, it is considered as a hole and we warp the grid pixel to the other reference view and do interpolation. Unlike many existing view synthesis methods [16], [17], [18], [19], a sophisticated hole-filling algorithm is unnecessary because the rendered views are not the final outcomes in refocusing. Besides, this algorithm only needs to reference the center depth map. It avoids storing multiple depth maps and reduces the storage. Finally, blending is achieved by averaging the views with an aperture weight. All of the refocused blocks then collectively constitute a refocused frame.

Table 1 shows the software runtime profiling results of BBRR executing on an Intel i7-8565U CPU. We executed each test case with ten different focus positions and recorded

TABLE 1. Runtime profiling result of BBRR.

Test Case	Width	Height	$\bar{t}_{P1}$	$\bar{t}_{P2}$	$\bar{r}_{P2}$	Throughput (pixel/sec)
Tsukuba[1]	384	288	0.01 s	2.09 s	99.76%	19.47 M
Horses[2]	1024	576	0.03 s	3.80 s	99.19%	14.91 M
Stilllife[2]	768	768	0.02 s	3.99 s	99.61%	21.04 M
Lego Truck[3]	1280	960	0.03 s	5.74 s	99.36%	22.32 M
Lego Knights[3]	1024	1024	0.05 s	6.42 s	98.97%	14.04 M
Camera[4]	1920	1080	0.10 s	17.06 s	99.24%	13.82 M
Kid and Cat[4]	1920	1080	0.10 s	44.17 s	99.60%	13.97 M
Kid and Ball[4]	1920	1080	0.08 s	7.12 s	98.81%	17.25 M
Girl and Pigeon[4]	1920	1080	0.14 s	22.19 s	99.26%	9.75 M

[1]Tsukuba (http://vision.middlebury.edu/stereo/data/scenes2001/data/tsukuba)
[2]HCI light field datasets [20] (https://lightfield-analysis.uni-konstanz.de/)
[3]Stanford light field archive (http://lightfield.stanford.edu/)
[4]Captured by Lytro ILLUM light field camera

the average execution time in $\bar{t}_{P1}$ (phase 1, block partitioning) and $\bar{t}_{P2}$ (phase 2, refocusing). Results showed that refocusing accounts for more than 98% of the overall runtime. For performance and energy concerns, this algorithm is unsuitable for direct software implementation on mobile devices. Although modern SoCs often integrate accelerators (e.g., GPUs or neural network accelerators) to speed up applications, they target workloads that have massive and independent computation so multiple data points can be processed in parallel. Our algorithm will not get much performance and energy benefit from these accelerators because its computation in phase 2 is heavily data dependent. Hence, this work builds a hardware accelerator that targets the refocusing phase of the algorithm. In this article, we propose high-performance circuit architectures for center-based 1-D line-scan view interpolation and aperture-weighted averaging. In terms of rendered pixel throughput, the accelerator chip is able to provide 5.4G pixel/s throughput. Compared to software implementation, which provides around 9–22M pixel/s throughput, our ASIC solution is 2 orders of magnitude faster.

III. HARDWARE-ORIENTED REALISTIC REFOCUSING ALGORITHM

The original refocusing algorithm is not feasible for direct hardware implementation for two reasons. First, it requires huge DRAM bandwidth to load the input light-field images. Second, its computation flow involves complex pixel scanning directions which increases the usage of on-chip memory. To overcome these design challenges, we propose a hardware-oriented refocusing algorithm that addresses the DRAM/SRAM issues and retains high refocusing quality.

A. DRAM BANDWIDTH OPTIMIZATIONS

The system DRAM bandwidth is huge because each refocused frame needs to input and process 17 RGB light-field images. To address the bandwidth issue, we proposed three

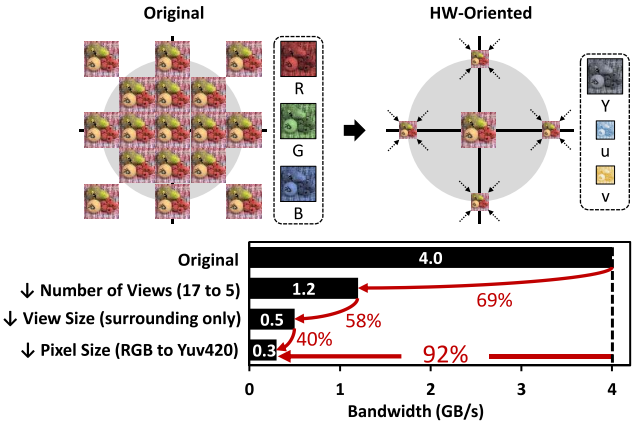


FIGURE 5. With HW-oriented algorithm optimizations, we reduced the DRAM bandwidth by 69%, 58%, and 40%, respectively. Overall it saves the off-chip traffic by 92%.

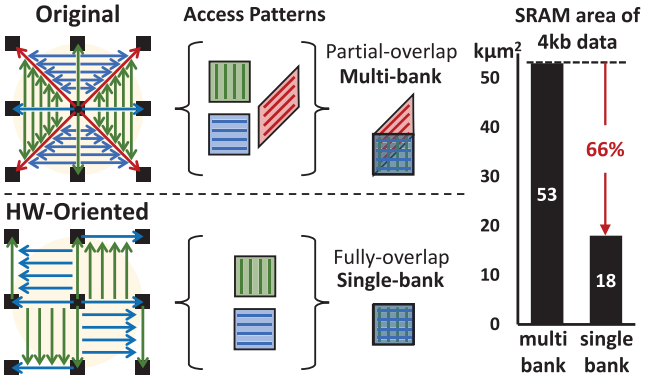


FIGURE 6. SRAM usage optimization by removing diagonal scanning directions.

techniques to reduce the input data size. As shown in Fig. 5, they together save 92% of total DRAM bandwidth.

1) REDUCTION OF INPUT VIEWS

The original algorithm requires 17 images to form a star-shaped light field for refocusing. However, the most important views are the surrounding images because they capture more occluded pixels from wider angles. Internal views can be interpolated by our algorithm with the depth map and the surrounding views. Our experiments show that we can achieve almost the same quality by using only five views. Hence, only the center, top, bottom, right, and left views are kept in our hardware-oriented algorithm. For the missing views that are essential during refocusing, we use a view extrapolator in our system to generate them beforehand.

2) REDUCTION OF SURROUNDING VIEW SIZE

The input views are used as references for view interpolation. In this work, a center-based view interpolation algorithm is adopted which means most of the rendered pixels are warped from the center view. The surrounding views (top, bottom, left, and right) are only referenced when hole filling is necessary. Because these views are referenced much less frequently compared to the center view, they are downsampled by a factor of 2 to reduce the size. Downsampling the surrounding views enables another 58% bandwidth reduction. Note that a 2× image upsampler is included in our system to recover the surrounding views' original resolution before processing.

3) REDUCTION OF PIXEL SIZE

To further reduce the input bandwidth, the size of the pixels is compressed by chroma subsampling because human eyes are less sensitive to chroma components. This technique is also widely used in image/video compression tasks. In this work, the images are stored and transmitted in YUV420 format. Compared to RGB444 format, it uses 50% less memory size. Because the depth map cannot benefit from chroma subsampling, the overall bandwidth reduction for this step is 40%. To retain sufficient quality, our system will upsample the chroma components back before refocusing.

B. SRAM USAGE OPTIMIZATIONS

In order to implement a high throughput system, multiple pixels are accessed in parallel and the access patterns depend on the scanning directions. In the original algorithm flow, pixels are scanned in eight different directions to synthesize a 2-D aperture. This flow induces additional memory usage because it needs multibank SRAMs to realize parallel pixel accessing. As shown in Fig. 6, for horizontal and vertical directions, their access patterns cover the same group of pixels. However, the access pattern for diagonal direction requires different groups of pixels. Those pixels must be stored in separate memory banks to support horizontal, vertical, and diagonal access patterns. It is possible to approximate diagonal scanning by horizontal scanning plus vertical scanning. The final refocused image using this approximation also has high quality because the algorithm blends all of the views in the aperture and the approximation error is amortized over these views. Hence, we proposed to remove the diagonal scanning directions. With only horizontal and vertical scanning, the access patterns can fully overlap and the pixels belonging to the same access can be stored to an entry of a single-bank SRAM. This saves the SRAM area by 66% through sharing the memory peripheral circuits. It also enables a view generation flow with simpler system scheduling.

C. QUALITY ASSESSMENT

All of the algorithm revisions made here are designed under strict quality constraints. Fig. 7 shows the quality assessment results of each modification. The PSNR value is calculated with respect to the frame-based algorithm using 17 input views in RGB444 format. Experiments show that each modification only introduces negligible quality degradation. The largest PSNR drop of 1.8 dB was found in the RGB444 to YUV420 conversion. Except for this, all the other PSNR drops are controlled within 1 dB.

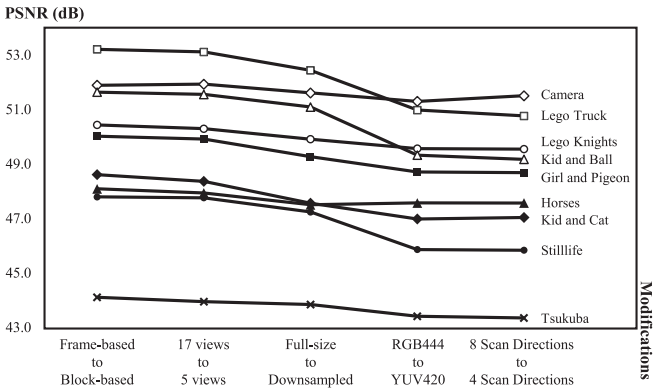


FIGURE 7. PSNR value after each modification of the refocusing algorithm. The experimental results show that our hardware-oriented algorithm offers almost the same quality compared to the original algorithm (all PSNR > 43 dB).

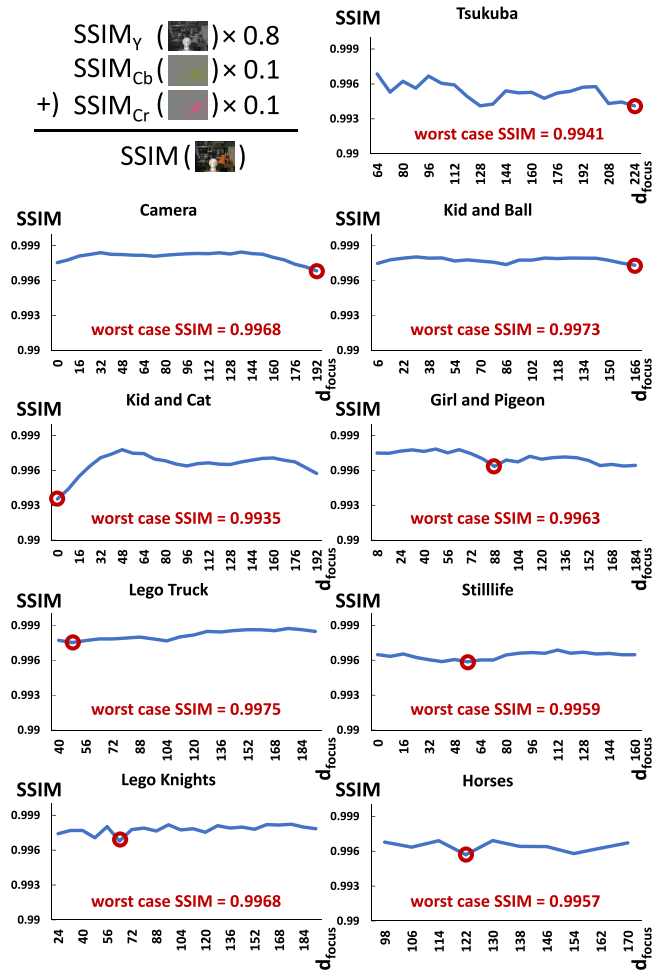


FIGURE 8. SSIM comparison between hardware-oriented algorithm and the original algorithm at different focus disparity (d_{focus}). The SSIM value of the color image is computed by the weighted sum of the individual SSIM value of its Y, Cb, and Cr components. The coefficients are 0.8 for Y, 0.1 for Cb, and 0.1 for Cr as suggested in the original SSIM paper. The worst SSIM in each test case is marked by a red circle and they all sit above 0.99.

To show the quality assessment that is closer to perceived quality, we compare our hardware-oriented algorithm to the original algorithm using structural similarity (SSIM) index [21]. Fig. 8 presents the SSIM comparison results

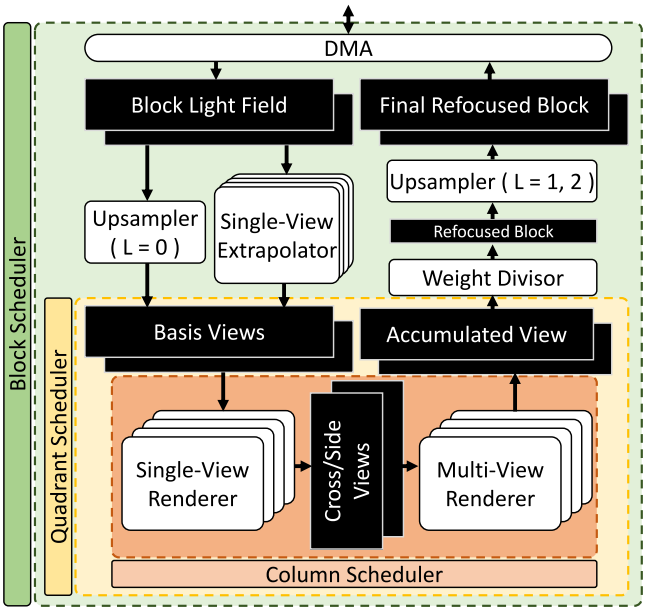


FIGURE 9. System architecture of the refocusing processor.

for each test case that focuses on different target disparity d_{focus}. The SSIM value is greater than 0.99 in all of our test cases (an SSIM value of 1.0 means a perfect match). The results show that our revised algorithm can generate refocused images that closely resemble those created by the original algorithm.

IV. SYSTEM ARCHITECTURE

The overall system architecture of the refocusing processor is shown in Fig. 9. It is a highly parallelized view rendering and accumulation accelerator that loads five view blocks and its depth map to output a block that focuses on a target depth. The refocused blocks are collected and assembled in external memory to become a complete refocused frame.

A. VIEW RENDERERS

View renderers are the core processing engines implementing the center-based 1-D line-scan view interpolation. They are highly parallelized to generate new views and are pipelined by single-port ping-pong SRAMs. To balance the system pipeline, the multiview renderer is designed with higher parallelism because it is in charge of generating around 70% of the new views. Note that the single-view extrapolor is a simplified version of the single-view renderer. It is used to generate the missing corner views in the first stage of view generation. A detailed discussion on the implementation of these highly paralleled view renderers will be given in Section V.

B. HIERARCHICAL PIPELINE SCHEDULERS

To enable efficient view rendering, the view generation flow breaks down into three hierarchies: 1) block; 2) quadrant; and 3) column, as shown in Fig. 10. These hierarchies allow parallel processing and are controlled by a pipeline scheduler.

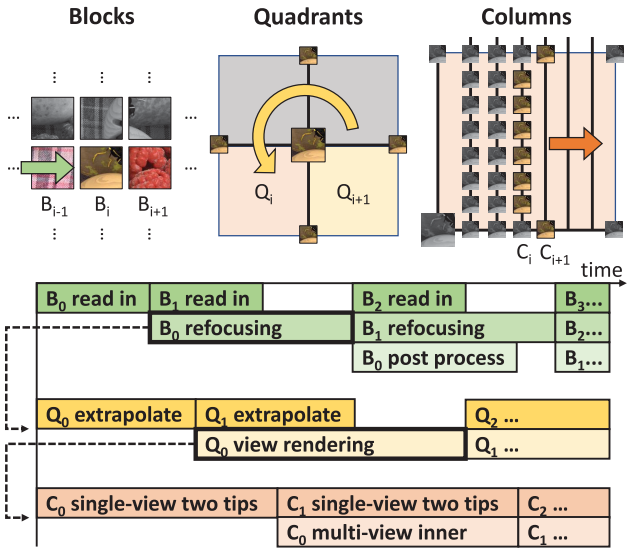


FIGURE 10. Hierarchy breakdown and corresponding timing diagrams.

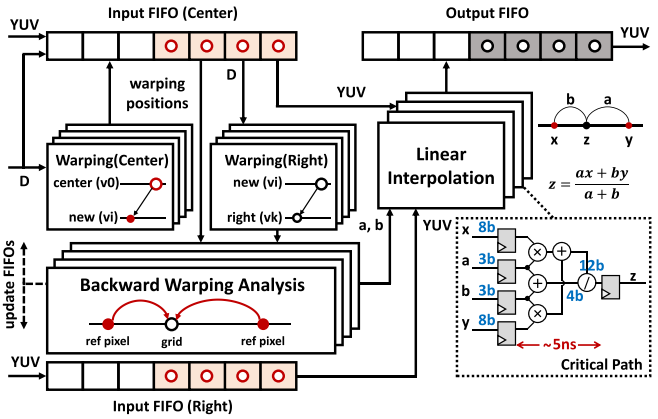


FIGURE 11. Algorithm concept and block diagram of single-view VSE. The view synthesis is implemented by pixel warping and linear interpolation in this engine.

1) BLOCK SCHEDULER

An image frame is partitioned into multiple blocks and processed sequentially. There are three stages for block processing. First, the light field and the depth map of a block are loaded into the chip via a DMA controller. Second, they are processed by the core refocusing engines to generate new views and accumulate the results. Finally, a refocused block is produced by a weight divisor and written out to external memory. Note that a view upsampler is required if the block is processed in the downsampled domain (i.e., $L = 1, 2$).

2) QUADRANT SCHEDULER

The view generation flow divides an aperture into four quadrants. They are processed in a counter clock-wised manner and each quadrant will go through two stages. First, an extrapolator creates a new corner view and two depth maps which will be used as references in the next stage. Furthermore, if this block is processed in its original resolution (i.e., $L = 0$), an upsampler should be used to recover the resolution of downsampled surrounding views. Second, all the views inside the quadrant will be generated column by column using the view renderers in the next hierarchy.

3) COLUMN SCHEDULER

Inside each quadrant, the views are generated in a column-wised order. There are two stages for a column to be generated. First, the views on the two tips of a column are generated by the single-view renderer. Next, the inner views between the tips are generated using the multiview renderer and accumulated afterward.

V. HIGHLY PARALLEL VIEW RENDERER DESIGN

Based on the characteristics of the 1-D line-scan view interpolation algorithm, parallel computing can be implemented in pixel, line, or view level. A view synthesis engine (VSE) realizes the view interpolation algorithm. It implements the pixel-level parallelism by scanning and analyzing multiple pixels in each cycle. A VSE can also adopt view-level parallelism by generating the results at multiple view points. Such kind of VSE is called a multiview VSE; otherwise, it is called a single-view VSE. Multiple VSEs can be combined into a group called view renderer which implements the line-level parallelism by operating on multiple lines simultaneously.

The amount of required parallelism is determined based on the system throughput specification. In our test set, it typically needs to render 100–300 million pixels for one Full-HD refocused frame. In the worst cases, the system needs to render 9 billion pixels per second to achieve the frame rate goal of 30 frames/s. It means the chip has to render 45 pixels per cycle when it operates at 200 MHz. The final parallelism decision is 64 after considering the ramp up, ramp down, and idle overhead. The amount of pixel-level parallelism is restricted because of its data-dependent computation. The circuit cannot process more than 5 pixels in parallel in one line; otherwise, it fails to meet the timing specification. Finally, we chose parallelism to be 4 for each of the three (pixel, line, and view) parallelism levels. They together enable a $4 \times 4 \times 4 = 64$ parallelism in our system.

A. SINGLE-VIEW VSE

A single-view VSE (as shown in Fig. 11) renders a new line in three steps. First, it warps the pixels from the center view (view number 0) to the new view (view number i). A pixel with a disparity value of D means that it will shift D pixels toward the left in the right view (view number k). Warping is done by using the disparity of a pixel and the ratio of view numbers (i/k) to compute its pixel shift (Di/k) in the new view. Second, it analyzes the warping positions and determines how to generate the integer pixel grids. If a pixel grid is surrounded by two close warped pixels, then the value of the grid is computed by the pixels from the center view. Otherwise, the grid is considered as a hole and VSE will warp it again to the right view. The value of the grid is then computed by the surrounding pixel grids in the right

view. Finally, it uses linear interpolation $(ax + by/a + b)$ to find the values of the grids. Because the process of linear interpolation involves cascading a multiplier $(8b \times 3b)$, an adder $(11b + 11b)$, and a divider $(12b/4b)$, it becomes the critical path of the system (around 5 ns). Note that the full-function divider can also be implemented with a fixed entry lookup table followed by a multiplier, but we found they have very similar delay so we use a divider IP directly.

To realize pixel-level parallelism, the modules for warping, position analysis, and linear interpolation are duplicated four times. However, a VSE will not always consume four input pixels nor generate four output pixels in every cycle even if it wants to. Some pixels will warp to the occluded region and they do not contribute to the outputs. In this case, the input and output data throughput will not be four pixel/cycle. We denote the remaining pixels that are less than four as incomplete pixels. These pixels make parts of the engine idle and result in low hardware utilization. We proposed to include a 7-taps register FIFO at all I/Os to solve this problem. It allows smoother data streaming by buffering the incomplete pixels. Input FIFOs are used to constantly serve four pixels to the engine to keep it busy. Output FIFO is used to pack incomplete pixels and constantly streams four pixels out. In general, an N pixel parallel engine needs $N+(N-1)$ taps register FIFOs to enable high hardware utilization.

Pixel-level parallelism cannot be too aggressive because the operating frequency will be limited when it tries to compute too many pixels in parallel. Since the warping position analysis of a pixel depends on the result of the previous pixel, the analysis chain grows with the parallelism. Besides, the input FIFOs need the analysis results to update its content so there exists an iteration bound which prevents us from using pipelines to reduce the timing path. Finally, pixel-level parallelism is implemented in four so the combinational path can fit into a 5-ns clock period.

B. MULTIVIEW VSE

A multiview VSE is a downstream hardware of a single-view VSE. It takes two views generated by a single-view VSE to generate all the new views between them. To achieve higher data throughput, it combines pixel-level parallelism and view-level parallelism inside. One trivial way to implement view-level parallelism is to duplicate several single-view VSEs and assign them to process different views. However, each engine needs one exclusive SRAM because they operate independently and have different data throughput. This indicates lots of redundant storage because all those SRAMs are holding the same view data. We proposed to compute multiple views inside a VSE using the concept of pixel splatting. Instead of using two reference pixels to interpolate a new grid, the engine splats the value of each reference pixel to its nearby grids. Since the warping positions for neighboring views are similar, it is easy to splat a reference pixel to the grids in multiple views with little cost.

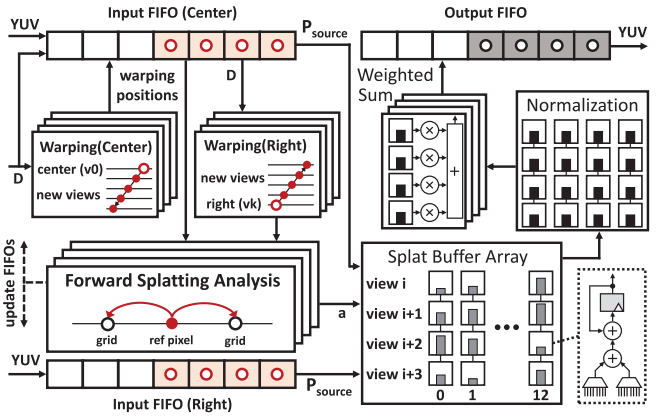


FIGURE 12. Algorithm concept and block diagram of multiview VSE. New views are generated by splatting a reference pixel's value to the neighboring pixels. The accumulated results should be normalized to ensure equal intensity.

As shown in Fig. 12, the FIFO interfaces are kept because pixel-level parallelism is still adopted. The reference pixels from the FIFOs are now warped to four adjacent views to realize view-level parallelism. The multiview warping positions are then used to analyze the splatting information, such as target splatting pixels and splatting weights. The splatting values are computed based on this information and accumulated in a splat buffer array. Inside the array, there are 4×13 accumulation units where 4 comes from the view parallelism and 13 comes from the maximum splatting range of a pixel. Because the accumulation units may receive contributions from multiple pixels, the results should be weighted averaged to have a normalized intensity. Finally, pixels are weighted summed across four views and streamed out. The proposed pixel splatting architecture integrates pixel- and view-level parallelism together and saves 73% SRAM area compared to trivial implementation.

Unlike the backward warping method used by the original 1-D line-scan view interpolation algorithm, pixel splatting uses the forward warping method. In most cases, forward warping is mathematically equivalent to backward warping. When more than two source pixels can contribute to a target pixel (known as racing), the forward warping method performs weighted averaging on all of the candidates, but the backward warping method only uses the nearest two source pixels to perform weighted averaging (i.e., linear interpolation). Both methods can be used in refocusing. To quantitatively show the perceptual difference of the refocused image, we generate refocused images using forward and backward warping and calculate their SSIM index with respect to the ground truth. Table 2 shows the worst-case SSIM for every test case. Results show that both forward and backward warping can generate images with SSIM scores greater than 0.99 and the difference between their SSIM scores is small. The results match our expectation that both methods produce almost identical refocused images because the small difference caused by the warping methods should be amortized over many other views to be blended.

TABLE 2. Quality evaluation of backward warping versus forward warping. Both methods build on top of the hardware-oriented algorithm in Section III. Their SSIM values are computed with respect to the frame-based unmodified algorithm in Section II. We pick the worst-case SSIM to show negligible quality impact brought by both warping methods.

Test Case	$SSIM_{worst}$	
	Backward Warping	Forward Warping
Tsukuba	0.9941	0.9892
Camera	0.9968	0.9961
Kid and Ball	0.9973	0.9940
Kid and Cat	0.9935	0.9926
Girl and Pigeon	0.9963	0.9910
Lego Truck	0.9975	0.9958
Lego Knights	0.9968	0.9957
Stilllife	0.9959	0.9857
Horses	0.9957	0.9883

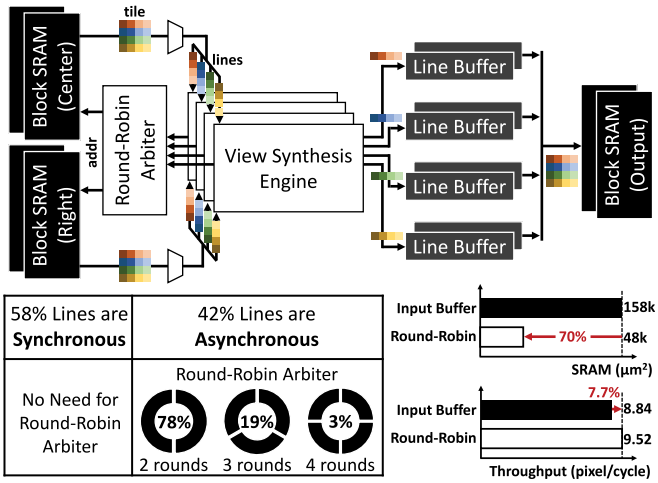
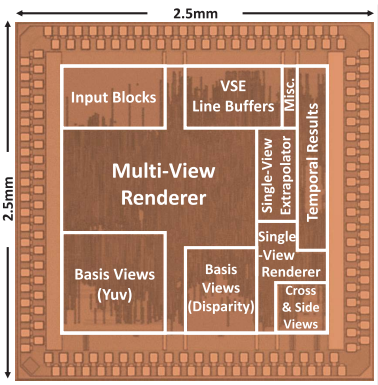


FIGURE 13. Architecture of view renderer. Multiple VSEs are grouped together to enable line-level parallelism.

C. VSE GROUP

To further boost the data throughput, we group multiple VSEs (they could be single-view or multiview version) into a view renderer as shown in Fig. 13. Each view renderer contains four VSEs and they are assigned to process different lines to achieve line-level parallelism. Since each VSE also implements four-pixel parallelism, a view renderer would require 16 pixels (4 pixels × 4 lines) per cycle. To serve this memory request, we glue these pixels together in one SRAM entry and call it a tile. Because a tile is accessed with different directions during view generation, a multiplexer is placed before each VSE to select pixels of the correct direction.

One biggest problem of grouping multiple VSEs is that they have different processing speeds so they may want to access different SRAM entries in the same cycle. Placing line buffers before and after the engines can solve the memory conflict issue. To retain data throughput, two single-port SRAMs can be used to implement double buffering.



Chip Specifications	
Technology	40 nm CMOS
Chip Area	2.5 mm x 2.5 mm
Core Area	1.9 mm x 1.9 mm
Gate Count	2.3 M (NAND2)
Total SRAM	271 KB
Voltage Supply	0.9 V
Clock Frequency	200 MHz
Average Power	250 mW
Rendered-Pixel Performance	5.4 G (pixel/sec)
Refocused-Pixel Peak Performance	Full-HD @ 40fps

FIGURE 14. Die photograph and chip specifications of the refocusing test chip.

However, these line buffers take too much area. In a single-view renderer, each VSE needs six SRAM instances which account for 40% of the renderer area. Double buffering also introduces additional control cycles when it switches between lines. These cycles are hard to remove because it cannot switch and prefetch the next line until all of the VSEs in a group finish processing. In terms of performance, these cycles create significant overhead because it only takes a few cycles to generate a line due to its parallel architecture.

In order to reduce the area and performance overhead, we removed the input line buffers. In replacement, the memory access conflicts are resolved by a round-robin arbiter. It analyzes the SRAM requests from all VSEs and serves distinct requests sequentially. When there are K distinct requests, the instant throughput of the renderer drops to $1/K$. However, based on the statistics, 58% of the lines have exactly the same requests and they do not need the arbiter. For those lines suffering from memory access conflicts, 78% of them only need 2 cycles to resolve the conflict. This is because neighboring lines tend to have a similar disparity distribution which results in the same memory access.

Removing the input line buffers reduces the SRAM area by 70% in a renderer. We also remove the control overhead by prefetching lines. Although this architecture suffers from infrequent throughput drop, the average throughput of a renderer increases 7.7% compared to the one using input line buffers.

VI. IMPLEMENTATION RESULTS

A. CHIP IMPLEMENTATION RESULTS

Fig. 14 shows the die photograph and the chip specifications of the refocusing test chip. The test chip is fabricated in a 40-nm CMOS technology comprising 2.3 million NAND2 equivalent logic gates and 271-kB on-chip memory. The chip can be operated at 0.9 V and 200 MHz to achieve its peak performance that renders full-HD resolution refocused images up to 40 frames/s. The test results show that it consumes 250 mW on average which justifies its applicability to most edge devices with strict energy constraints.

Fig. 15 shows the area and power breakdown of the test chip. Most of the chip area goes to the memory because lots

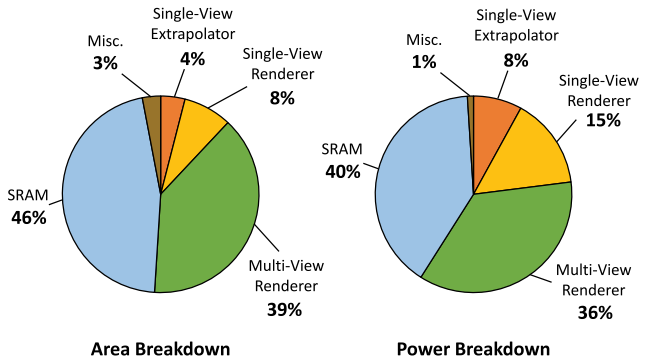


FIGURE 15. Area and power breakdown.

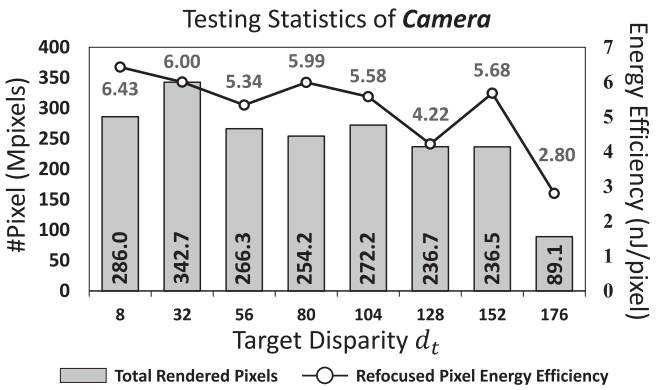


FIGURE 16. Statistics of total rendered pixels and refocused pixel energy efficiency of test case Camera. A big gap can be found at $d_t = 176$ because it focuses too close to the front and almost everything in the scene is blurred and downsampled by the algorithm.

of intermediate views are generated and stored locally. The multiview renderer implements 64× parallelism to generate most of the views and it represents 39% of the chip area. On the other hand, the single-view renderer implements 16× parallelism to generate the source views for a multiview renderer so it only takes 8% of the chip area. The single-view extrapolator is a simplified version of the single-view renderer which has approximately half of the logic gates. The power breakdown shows similar distribution to the area breakdown. The multiview renderer presents slightly less power because most of its accumulation units are not active when they are not splatted by the reference pixels.

B. EVALUATION

To assess the performance of the chip, we define two kinds of pixel as follows.

1) *Rendered pixels* refer to the pixels that are generated by the view renderers. Because these pixels are the direct outcome from hardware engines, they are primarily used to measure the chip's computing ability.
2) *Refocused pixels* refer to the final output pixels which are generated by blending the rendered pixels. For example, a full-HD refocused frame has around 2 million refocused pixels (1920 × 1080) and they are typically merged from hundreds of millions of rendered pixels.

The chip was tested by the full-HD image sequences in Table 1. Each of them was programmed to focus at eight different target depths distributed within its possible depth range. Testing results showed that the chip can generate 5.4 billion rendered pixels per second on average. For each refocused image, it requires the different number of rendered pixels to blend. This can be observed in Fig. 16 which shows the statistics of the test case *Camera*. The frame rate of a refocused image depends on how fast the chip can generate all the rendered pixels. Test cases with less rendered pixels tend to have better frame rate and refocused pixel energy efficiency. For example, *Camera* at $d_t = 176$ has best frame rate and energy efficiency compared to other d_t because it has lowest total number of rendered pixels. Although the refocused pixel performance is mainly determined by the total number of rendered pixels, sometimes the disparity map also affects the performance especially when there are too many occlusions in the scene. Some of the refocusing results are presented in Fig. 17.

It is possible to increase the frame rate and energy efficiency of a refocused image by reducing the total number of rendered pixels. This can be done by limiting the maximum blur radius ($\Delta d_{\max}$) of the input disparity map. However, the objects in the defocused region will be less blurred due to this restriction. An example is shown in Fig. 18. We can see the decrease in total rendered pixels and the increase in frame rate after changing $\Delta d_{\max}$ from 32 to 16. This change also makes the background (keyboard) become less blurred.

TABLE 3. Comparison with other view synthesis works.

	TCSVT 2011 [21]	ESSCIRC 2013 [22]	ISSCC 2011 [23]	This Work
Technology	90nm	65nm	40nm	**40nm**
Application	View Synthesis	View Synthesis	View Synthesis	**Refocusing**
Gate Count	0.3 M	2.3 M	1.4 M	**2.3 M**
SRAM Size	69 kb	440 kb	20 kb	**271 kb**
Frequency	200 MHz	260 MHz	240 MHz	**200 MHz**
Throughput (rendered-pixel/s)	0.1 G	0.5 G	1.9 G	**5.4 G**
Gate Efficiency (rendered-pixel/s·kgate)	0.34 M	0.22 M	1.36 M	**2.35 M**
Energy Efficiency (nJ/rendered-pixel)	-	1.10	0.04	**0.05***

*This number includes additional energy for refocusing.

C. COMPARISON

Because the main computation inside the refocusing processor is view synthesis, we compare this chip to other previous view synthesis works [22], [23], [24] and present the results in Table 3. Our highly paralleled engines enable superior rendered pixel throughput. For on-chip memory, because view synthesis results are not the final product of

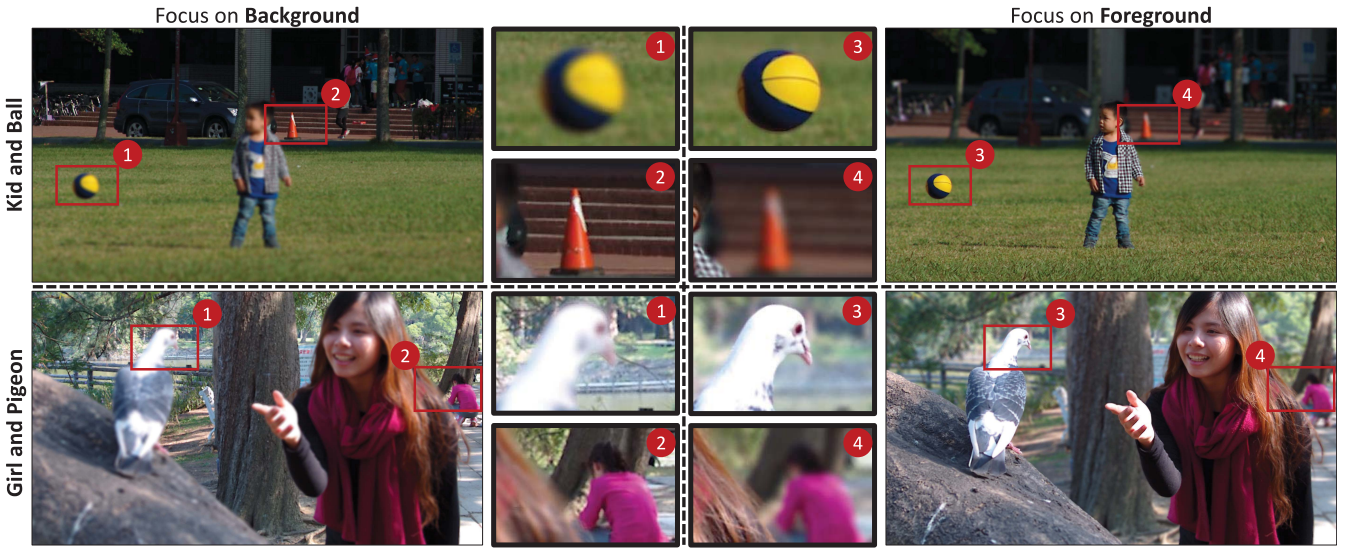


FIGURE 17. Realistic refocusing results generated by the test chip. The target focusing depth can be configured through the programmable parameters of the chip. Objects will be blurred if they are not in the target focus depth (number 1 and number 4 for example).

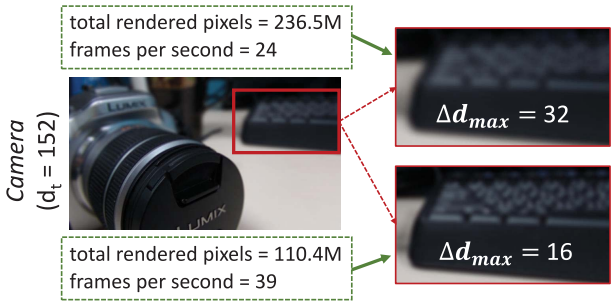


FIGURE 18. Limiting Δd_{max} can reduce total rendered pixels and improve the frame rate at the cost of refocusing quality.

refocusing, they need to be stored on the chip for later blending. Hence, more SRAMs are used compared to other view synthesis works whose synthetic views are their final outcome. In terms of gate efficiency, our work greatly surpasses the others because a customized and hardware-friendly view synthesis algorithm is adopted. Finally, our work also shows great rendered pixel energy efficiency even with additional energy spent on refocusing.

VII. CONCLUSION

In this article, we proposed an ASIC solution to accelerate the view generation part and the view averaging part of the realistic refocusing algorithm [14]. A revised hardware-friendly algorithm is proposed to reduce DRAM bandwidth and SRAM usage without noticeable quality degradation. A hierarchical system pipeline is developed to maximize the hardware utilization during the view generation flow. The core VSEs are designed with three levels of parallelism to achieve ultra high pixel generation throughput. The test chip is implemented in 40-nm CMOS technology and achieves 100× to 350× runtime improvement compared to pure software implementation. The chip can generate full-HD realistic refocused images up to 40 frames/s while consuming 250 mW. This ability justifies its applicability to low-power edge devices with multicamera setups.

ACKNOWLEDGMENT

The authors would like to acknowledge chip fabrication support provided from Taiwan Semiconductor Research Institute (TSRI).

REFERENCES

[1] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *Proc. 11th Annu. Conf. Comput. Graph. Interact. Techn.*, 1984, pp. 137–145.
[2] M. Pharr and G. Humphreys, *Physically Based Rendering*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, Jun. 2010.
[3] J. Lehtinen, T. Aila, J. Chen, S. Laine, and F. Durand, "Temporal light field reconstruction for rendering distribution effects," in *Proc. ACM SIGGRAPH Papers*, 2011, pp. 1–12.
[4] K. Vaidyanathan, J. Munkberg, P. Clarberg, and M. Salvi, "Layered light field reconstruction for defocus blur," *ACM Trans. Graph.*, vol. 34, no. 2, pp. 1–12, Mar. 2015.
[5] P. Rokita, "Generating depth-of-field effects in virtual reality applications," *IEEE Comput. Graph. Appl.*, vol. 16, no. 2, pp. 18–21, Mar. 1996.
[6] Y. Taguchi, A. Agrawal, A. Veeraraghavan, S. Ramalingam, and R. Raskar, "Axial-cones: Modeling spherical catadioptric cameras for wide-angle light field rendering," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 1–8, Dec. 2010.
[7] J. Demers, *Depth of Field: A Survey of Techniques*. Boston, MA, USA: Addison-Wesley, 2004, ch. 23, pp. 375–390.
[8] M. Levoy, "Light fields and computational imaging," *Computer*, vol. 39, no. 8, pp. 46–55, Aug. 2006.
[9] B. Wilburn et al., "High performance imaging using large camera arrays," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 765–776, Jul. 2005.
[10] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan, "Light field photography with a hand-held plenoptic camera," Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Research Rep. CSTR 2005-02, Apr. 2005.
[11] K. Venkataraman et al., "*PiCam*: An ultra-thin high performance monolithic camera array," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–13, Nov. 2013.
[12] J. Fiss, B. Curless, and R. Szeliski, "Refocusing plenoptic images using depth-adaptive splatting," in *Proc. IEEE Int. Conf. Comput. Photogr. (ICCP)*, 2014, pp. 1–9.

[13] C.-K. Liang and R. Ramamoorthi, "A light transport framework for lenslet light field cameras," *ACM Trans. Graph.*, vol. 34, no. 2, pp. 1–19, Mar. 2015.
[14] C.-T. Huang, Y.-W. Wang, L.-R. Huang, J. Chin, and L.-G. Chen, "Fast physically correct refocusing for sparse light fields using block-based multi-rate view interpolation," *IEEE Trans. Image Process.*, vol. 26, pp. 603–618, 2017.
[15] P.-H. Chen, S.-W. Yang, S.-Y. Huang, L.-D. Chen, and C.-T. Huang, "A 250mW 5.4G-novel-pixel/s photorealistic refocusing processor for full-HD five-camera applications," in *Proc. Symp. VLSI Circuits*, 2019, pp. C154–C155.
[16] S. C. Chan, H.-Y. Shum, and K.-T. Ng, "Image-based rendering and synthesis," *IEEE Signal Process. Mag.*, vol. 24, no. 6, pp. 22–33, Nov. 2007.
[17] T. Georgeiv, K. C. Zheng, B. Curless, D. Salesin, S. Nayar, and C. Intwala, "Spatio-angular resolution tradeoffs in integral photography," in *Proc. 17th Eurograph. Conf. Rendering Techn.*, 2006, pp. 263–272.
[18] C.-K. Liang, T.-H. Lin, B.-Y. Wong, C. Liu, and H. H. Chen, "Programmable aperture photography: Multiplexed light field acquisition," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–10, Aug. 2008.
[19] B. Huhle, T. Schairer, P. Jenke, and W. Straßer, "Realistic depth blur for images with range data," in *Proc. DAGM Workshop Dyn. 3D Imag.*, 2009, pp. 84–95.
[20] S. Wanner, S. Meister, and B. Goldluecke, "Datasets and benchmarks for densely sampled 4D light fields," in *Proc. Vis. Model. Vis.*, 2013, pp. 145–152.
[21] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, pp. 600–612, 2004.
[22] Y.-R. Horng, Y.-C. Tseng, and T.-S. Chang, "VLSI architecture for real-time HD1080p view synthesis engine," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 9, pp. 1329–1340, Sep. 2011.
[23] M. Schaffner, P. Greisen, S. Heinzle, F. K. Gürkaynak, H. Kaeslin, and A. Smolic, "MADmax: A 1080p stereo-to-multiview rendering ASIC in 65 nm CMOS based on image domain warping," in *Proc. ESSCIRC*, 2013, pp. 61–64.
[24] P.-K. Tsung et al., "A 216fps 4096×2160p 3DTV set-top box SoC for free-viewpoint 3DTV applications," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2011, pp. 124–126.

PO-HAN CHEN (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering and computer science and the M.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2016 and 2018, respectively. He is currently pursuing the Ph.D. degree with Stanford University, Stanford, CA, USA.

From 2018 to 2020, he was a Design Engineer with MediaTek, Hsinchu, developing circuit architectures for camera ISP. His research interests include designing high-performance and energy-efficient circuits for computer vision and computational photography algorithms.

Mr. Chen was a recipient of the NovaTek Fellowship from 2016 to 2018.

SHU-WEN YANG received the B.S. and M.S. degrees in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2015 and 2018, respectively.

In 2018, she joined MediaTek Inc., Hsinchu, where she has been involved in the Silicon Product Development. Her research topic was photorealistic refocusing processing for Full-HD multicamera applications.

CHAO-TSUNG HUANG (Member, IEEE) received the B.S. degree in electrical engineering and the Ph.D. degree in electronics engineering from National Taiwan University (NTU), New Taipei, Taiwan, in 2001 and 2005, respectively.

He is an Associate Professor with the Department of Electrical Engineering, National Tsing Hua University (NTHU), Hsinchu, Taiwan. Prior to joining NTHU, he was a Postdoctoral Associate with the Massachusetts Institute of Technology, Cambridge, MA, USA, from 2011 to 2012 and then with NTU from 2012 to 2013. From 2005 to 2011, he was with Novatek Microelectronics Corporation, Hsinchu, for developing multi-standard image and video codecs. His research interests include light-field signal processing and deep convolutional networks, especially from algorithm exploration to VLSI architecture design, chip implementation, and demo system.

Dr. Huang was a recipient of the Young Scholar Innovation Award from Foundation for the Advancement of Outstanding Scholarship in 2020, the Outstanding Young Electrical Engineer Award from CIEE in 2019, the Outstanding Young Scholar Award from Taiwan IC Design Society in 2019, and the Junior Faculty Research Award from the College of EECS, NTHU, in 2017. He serves as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY and *Circuits, Systems and Signal Processing* (Springer). He is also a member of DISPS Technical Committee of IEEE Signal Processing Society.