International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 12, Number 2, pages 425–445, July 2022

Weighted Least Square Filter for Improving the Quality of Depth Map on FPGA

Renzhi, MAO
Amano Laboratory, Keio University
Yokohama, Japan

Kaijie, WEI
Amano Laboratory, Keio University
Yokohama, Japan

Hideharu, AMANO
Amano Laboratory, Keio University
Yokohama, Japan

Yuki, KUNO
Marelli Corporation
Saitama, Japan

Masatoshi, ARAI
Saitama University
Saitama, Japan

Received: February 15, 2022
Revised: May 5, 2022
Accepted: May 30, 2022
Communicated by Nobuhiko Nakano

Abstract

This paper proposes a post-filtering system for improving the quality of depth maps for 3D projection on FPGA. We propose to implement the Weighted Least Square (WLS) filter on Field-programmable Gate Array (FPGA), which can predict the disparities, which cannot be measured, by using the values of the neighboring pixels. In our design, we optimized the architecture of WLS filter at the algorithm level. For hardware acceleration, we use the High-Level Synthesis (HLS) description to accelerate the algorithm. To break through the bottleneck brought by the limited memory resources on FPGA, we used UltraScale Architecture Memory Resources (URAM) on board and reduced memory consumption of Block RAMs (BRAM) from 140% to 80%. Through our approach, we can improve the quality of the depth map on the FPGA named M-KUBOS. And, the WLS filter can smooth the depth map with 130 MHz operational frequency on M-KUBOS at the speed of 10 frames per second (FPS), which achieves a 76.43% performance acceleration compared to the software execution on the ARM Cortex A9 at 1.2 GHz clock.

Keywords: FPGA, Computer Graphics, 3D Projection, Smoothing Filter, Weighted Least Square

1 Introduction

3D projection provides great convenience for people's lives. By processing 2D data and turning it into a 3D format, 3D projection technology can derive the depth information in the image data. The 3D projection has been adopted in a wide range of fields, which includes detecting, analyzing, and measuring. For example, a human cannot intuitively sense the depth distance by 2D images. The self-driving car, which is one of the hot topics nowadays, needs a better depth map for computing the distance.

However, considering the needs of applications, 3D projection technology faces several challenges such as high speed and lightweight. Although several algorithms are proposed, which are helpful for the 3D projection, there are bottlenecks in improving the image quality of the depth map.

Based on the existing problems, we emphasize refining depth maps quality on the 3D scene display system. During the 3D projection process, the system produces two depth maps computed from the stereo camera as an intermediate product for 3D projection processing. The quality of the depth map affects the final product directly. Therefore, to enhance the quality of 3D image processing, improving the quality of the depth map becomes an effective option.

This paper proposes a post-filtering system for improving the quality of depth maps for 3D projection on FPGA. We propose to implement the WLS filter on FPGA, which can predict the disparities, which cannot be measured, by using the values of the neighboring pixels. In our design, we optimized the architecture of WLS filter at the algorithm level and completed the hardware acceleration. For evaluation, we compared the FPGA implementation with the Central Processing Unit (CPU) implementation.

1.1 Improving the Depth Maps for 3D Projection on FPGA

Regarding the invalid pixels causing the quality-drop of the depth map and indirectly worsening the quality of the product after 3D projection, we decided to focus on improving the quality of the depth map by reducing the adverse effects of such invalid pixels. We try to provide depth maps with better quality for Semi-Global Matching (SGM) [1] on a 3D scene display system. SGM is an eminent stereo method that can perceive the depth information using the stereo camera.

During the 3D projection process, to calculate the distance between a certain location and one of the specific points on the scene, the system captures two images from a stereo camera. An example for the stereo images is shown in Figure 1a. However, when the depth maps are generated, there are some mismatched pixels, which cannot be measured by the system. The result of the depth map without post-filtering is shown in Figure 1b.

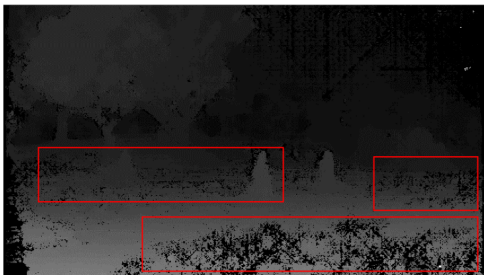
In the depth map before post-filtering, for convenience of the implementation, all pixels with invalid value have been considered as pixels with value 0 by the pre-processing of the system, which makes the pixel seem like a "black hole" that confuses the 3D image generation system. To improve the quality of the depth map and the final product, post-filtering technology based on OpenCV is introduced as a solution. The core algorithm of this post-filtering is Weighted Least Square Filter [2], which can fill the invalid region on the depth map by using the results of neighboring pixels.

In our study, the post-filtering system consumes a lot of computational and energy resources. Therefore, the most appropriate method to solve the problem is to implement it on FPGA to reduce its power consumption and accelerate the system. The two main reasons for choosing FPGA in this paper are listed below.

- FPGA with good energy efficiency is more suitable in specific application scenarios such as self-driving cars. The effect of this advantage is particularly emphasized in our case.
- In terms of flexibility, we can program hardware for specific applications on FPGAs.

For optimization, we modified the algorithm of the WLS filter to increase the parallelism of the program. We optimized the code of the WLS filter at the algorithm level, which contains splitting loops from functions, reducing the number of arrays in need and removing specific data dependency. Among them, the most notable optimization is to eliminate specific data dependencies. We trade

(a) Stereo camera captures the left and right image of one scene.



(b) Pixels with invalid value in the depth map without post-filtering.

Figure 1: Stereo image and the pixels cannot be measured in the depth map.

accuracy for reduced data dependencies within acceptable limits, resulting in further performance improvements.

In the hardware implementation, we used the HLS (High-Level Synthesis) description to accelerate the algorithm. To break through the bottleneck brought by the limited memory resources on FPGA, we used UltraScale Architecture Memory Resources (URAM) on board and reduced memory consumption of BRAM from 140% to 80%. Through our approach, we were able to improve the quality of the depth map on the FPGA called M-KUBOS. Although the optimization techniques are common ones, the implementation of the WLS filter on the FPGA has not been reported yet. So, they would be valuable case studies.

As far as our knowledge, it is the first trial to implement the WLS filter on FPGA. Thus, we compared it with the existing WLS algorithm on CPU. As the result, the WLS filter can smooth the depth map with 130 MHz operational frequency on M-KUBOS at the speed of 10 FPS, which achieves a 76.43% performance acceleration compared to the software execution on the ARM Cortex A9 at 1.2 GHz clock.

1.2 Overview of the Approach

In this paper, we propose an approach that builds a post-filtering system with a core function based on the WLS algorithm on an FPGA M-KUBOS [3]. Through the post-filtering system, we expect to improve the quality of depth maps generated from the SGM algorithm to achieve the final goal of improving the final products of the 3D projection system on FPGA.

However, we encountered many difficulties in practice, which shows the disadvantages of the WLS in FPGA implementation. First, the memory consumption of the whole system is quite large. Second, the dependency between the loops and functions impacts parallelism seriously. We spent most of our implementation time addressing these issues. Our approach can be divided into three stages. The overview of the approach is as below.

WLS Formulation Analysis In the first part, we analyzed the basic formulation of WLS filter to understand the features of the functions in the algorithm. We deconstructed the entire algorithm through mathematical analysis to construct the C++ language-based functions and preliminarily implemented the algorithm in a CPU environment.

HLS Code Transformation Based on the High-level Synthesis (HLS), we changed the algorithm of the C++ language-based functions and made them more suitable for FPGA implementation. To be more specific, we deconstructed various loops into new modules, which are inserted with some pragma to accelerate our work. Thus, more delicate optimizations can be done in these independent modules.

Hardware Acceleration Through Vitis HLS, we have done several hardware optimizations such as dataflow, pipelining, and resource specification to further improve the performance on FPGA.

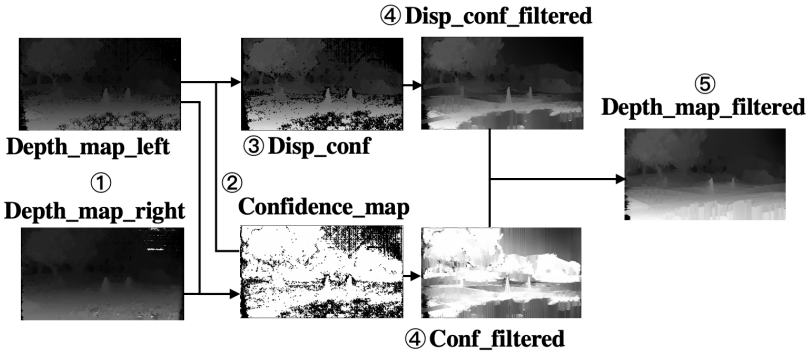


Figure 2: Overall processing of the post-filtering system.

2 Background

2.1 Post-Filtering Process

There are five steps in this post-filtering system with the WLS algorithm. They take charge of processing the input depth maps, generating the confidence maps, filling the invalid pixels, and producing the filtered depth map. The post-filtering system is a post-processing system, and it receives the left and right depth maps derived from SGM algorithm.

In the post-filtering process, two depth maps are used as input. Also, the original left image captured by the stereo camera is used as a guide image for the whole system. After receiving the input depth maps, the system first computes the variance of each pixel in both left and right depth map. And then, it gets the discontinuity of each pixel in both left and right depth map by using the variance (Figure 2 ①). The variance of each pixel in both left and right depth maps is defined as below. Here, p represents the number of pixels in each depth map. $x_1, ..., x_p$ mean all the pixels in each image and x_p is the value of pixel p.

$$variance = \frac{x_1^2 + x_2^2 + ... + x_p^2}{p} - \left(\frac{x_1 + x_2 + ... + x_p}{p}\right)^2 \tag{1}$$

The discontinuity basically means the credibility of value of the depth map. It can be defined as $max\,(0, 1 - 0.001 \times (variance))$.

Based on the definition, when the variance is small, the discontinuity is close to 1. Otherwise, it is close to 0. If the variance is significantly large, the discontinuity becomes 0. Since the values of neighbouring pixels are usually equal to each other, the smaller variance can be trusted. Then, the confidence map is produced as the result(Figure 2 ②). The definition of the confidence map is shown below.

$$confidence = min\,(disc_L, disc_R) \times C \tag{2}$$

Here, $disc_L$ is the discontinuity of the left image and $disc_R$ is the discontinuity of the right image. By considering both left and right images, the output of the function can represent the credibility of each pixel in the confidence map. In the function, C is an appropriate coefficient. When $C = 1$, the confidence is limited between 0 and 1. In this study, we set $C = 255$, so the confidence should be limited from 0 to 255. After getting the confidence map, the value of each pixel in the confidence map multiples with the disparity data of the left depth map. Here, the disparity data means the pixel values of the left depth map. The result of this process generates a new disparity confidence image (Figure 2 ③).

Then, the system inputs both of them into an independent WLS algorithm to fill the invalid pixels by using the value of neighboring pixels. After the processing, the confidence map and the disparity confidence image produce a filtered image, respectively (Figure 2 ④). At last, we can get the filtered depth map by calculating the filtered disparity confidence image and the filtered

confidence map. The overall processing is shown in Figure 2, and the equation of the last processing (Figure 2 ⑤) is shown below. Since the disparity data is the pixel value of the left depth map, to make the equation clearer, we use depth to represent disparity in it.

$$\frac{(depth \times confidence)_{filtered}}{confidence_{filtered}} = depth_{filtered} \tag{3}$$

2.2 Weighted Least Square

Weighted least squares (WLS) optimization [2] is an algorithm widely used in data computation to optimize the computation process, which is an extension of the ordinary least squares (OLS) regression [4] method. It is implemented by adding weight to a specific data point in the dataset by attaching a non-negative constant to a specific data point. For an optimization algorithm, weighted least squares is an efficient way to obtain data with specific properties. In many application scenarios, the conditions of using weighted least squares can satisfy one of the following points.

Central Processing As mentioned in [5], in some special datasets, data points in certain regions often have a special meaning or higher value than other data points. When these data points need to be centrally processed, WLS can be used to add weights to these data points to distinguish them from general data points.

Inconsistent Variance Normally, to create a dataset with evenly dispersed data, the dependent variables in the dataset need to be clustered in such a way that the variances are substantially similar. However, as shown in [6], when the variance difference of the dependent variable in the datasets is significant, WLS can be useful. For example, noise in the image.

Particular Justification Processing using WLS has a particular justification for treating data points in the dataset unfairly when affected by reality.

WLS can be used in a variety of application scenarios, especially those who meet the above conditions. In our study, due to the existence of invalid pixels, the variance difference between each data point of the depth map is quite large. And, according to the resolution of the image, the size of the dataset is as large as 672×376, with a total of $252,672$ data points (pixels), so WLS is relatively suitable for this study. Besides, WLS can effectively optimize the entire process, effectively distinguish invalid pixels and valid pixels, and perform targeted processing on invalid pixels.

2.2.1 Basic Weighted Least Square Filter Formulation

Among the filters used for image smoothing, WLS filter is a relatively basic algorithm. According to [7], the basic formulation for WLS is to minimize the energy function, and we can get the solutions through a linear function. At the beginning of the WLS formulation, the effect of the energy function is explained. Here, d is the input depth image and g represents the original left image of the stereo camera as a guide image. Based on the explanation of energy functions in [2] and [7], minimizing the WLS energy function can export the desired output u. In Equation (4), we have d_p, g_p as the gray scale value of a single pixel, and $N(p)$ is the value for the pixels around the target pixel p. The λ in Equation (4) keeps the balance of the two terms. The smoothing effect becomes more obvious with the increase of λ.

$$W(u) = \sum_{p} \left((u_q - d_p)^2 + \lambda \sum_{q \in N(p)} \omega_{p,q}(g)(u_p - u_q)^2 \right) \tag{4}$$

By solving $\omega_{p,q}(g)$ in the formula above, we can obtain the similarity between two pixels p and q for further processing. $\omega_{p,q}(g)$ defines a range parameter σ and uses the base-e exponential function of the two pixels in the guide image to compute the similarity of the target pixel as $\mathbf{exp}(-||g_p - g_q||/\sigma_c)$.

WLS achieves the final result by weighting the data points in the dataset by minimizing the energy function. In the above equation expression, the similarity between pixels plays an important

role in the whole WLS energy function. Therefore, calculating the similarity by establishing a matrix method can effectively minimize the WLS energy function and obtain the final result. From above, we obtain the energy function to minimize. To minimize parameter u, we make the gradient of $W(u)$ become zero and solve a large linear function as follows.

$$(\mathbf{I} + \lambda \mathbf{A})\,\mathbf{u} = \mathbf{d} \tag{5}$$

Where $\mathbf{u}$ and $\mathbf{d}$ indicate $S \times 1$ vectors containing the gray values of each u and d. $\mathbf{I}$ is an identity matrix and the $S \times S$ matrix $\mathbf{A}$ [8] is defined as below. Here, m and n represent an index of a pixel p and they should be limited in the size of the input image, that means $m, n \in (1, ..., S-1)$.

$$\mathbf{A}(m,n) = \begin{cases} \sum_{l \in N(m)} \omega_{m,l}(g) & n = m \\ -\omega_{m,n}(g) & n \in N(m) \\ 0 & otherwise \end{cases} \tag{6}$$

By using the matrix $\mathbf{A}$ above, the filtered result $\mathbf{u}$ is finally obtained as shown in Equation 7.

$$\mathbf{u}(m) = ((\mathbf{I} + \lambda \mathbf{A})^{-1}\mathbf{d})(m) \tag{7}$$

2.2.2 Acceleration to the Linear System

While performing the linear computation, WLS consumes the computational resources to a great extent and causes a loss in processing efficiency. While trying to resolve this huge linear system, we learned from related research [7] of an improved algorithm based on the WLS algorithm named Fast Global Smoother (FGS), which was developed specifically to solve the computational problem of large linear systems.

In FGS, the weighted least square function can be defined as follow. In this new function, d^h is the 1D input depth image and g^h is the 1D guide image. Both of them are in the x dimension. Here, $x \in (1, ..., S-1)$.

$$\sum_x \left((u_x^h - d_x^h)^2 + \lambda_t \sum_{n \in N_h(x)} \omega_{x,n}(g^h)(u_x^h - u_n^h)^2 \right) \tag{8}$$

As before, $N_h(x)$ are the neighbour pixels of this pixel x. To keep this function different with Equation (4), parameter λ_t is defined.

Based on Equation (8), the 1D output u^h that minimizes it is shown below.

$$(\mathbf{I}_h + \lambda_t \mathbf{A}_h)\mathbf{u}_h = \mathbf{d}_h \tag{9}$$

In this new solution, $\mathbf{I}_h$ is the identity matrix in the size of $S \times S$. And, $\mathbf{u}_h$ and $\mathbf{d}_h$ are the vectors of u^h and d^h. Based on Equation (9), an equation with three-point Laplacian matrix as follows is obtained.

$$\begin{bmatrix} j_0 & k_0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & 0 & 0 \\ 0 & i_x & j_x & k_x & 0 \\ 0 & 0 & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & i_{S-1} & j_{S-1} \end{bmatrix} \begin{bmatrix} u_0^h \\ \vdots \\ u_x^h \\ \vdots \\ u_{S-1}^h \end{bmatrix} = \begin{bmatrix} d_0^h \\ \vdots \\ d_x^h \\ \vdots \\ d_{S-1}^h \end{bmatrix} \tag{10}$$

From the element distribution of this matrix, we can see the largest different between Equation (5) and Equation (9).

- All non-zero elements exist only on the left and right diagonals of the matrix.

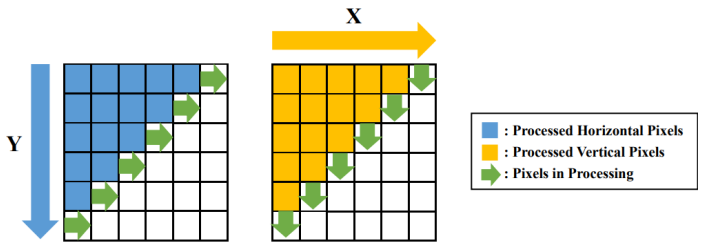


Figure 3: Forward data processing of WLS filter in vertical and horizontal directions.

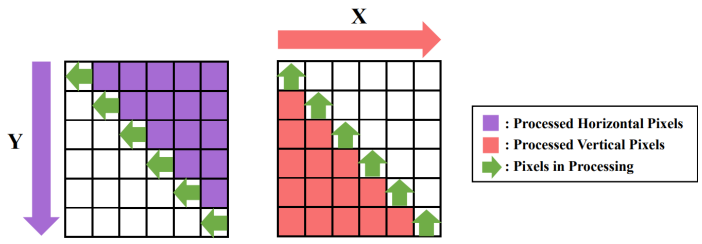


Figure 4: Backward data processing of WLS filter in vertical and horizontal directions.

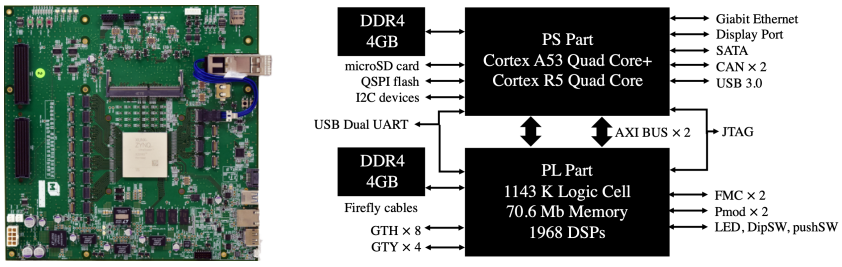
- The improvement of the algorithm allows the linear system of equations to be solved by Gaussian elimination, which reduces the time complexity of the entire algorithm.

That makes Equation (9) much easier to solve than Equation (5). In our case, the FGS algorithm based on WLS is used to fill in the invalid pixels in the depth map. Although the FGS algorithm is indeed superior to the basic WLS optimization in terms of processing speed, through practice, we found that it is difficult to further optimize the algorithm such as parallelization. Therefore, we propose to change the algorithm to remove specific data dependencies.

2.2.3 WLS Filter Processing

The processing of WLS Filter begins with the vertical data of the image. During the vertical data processing, every pixel on a current column is processed. By doing this, the matrix A in Equation (9) is created and the Equation (9) is solved. After finishing the vertical data processing, the WLS filter continues the processing with the horizontal data of the image. Same as before, during the horizontal data processing, every pixel on the current row is processed, either. By doing this, the matrix A in Equation (9) is created and the Equation (9) is solved again. The process of the forward data computation is shown in Figure 3. From the figure, in the forward processing of WLS filter, the program starts from the upper left corner of the image processes all the pixels to the right and downward, and finally converges to the pixels in the lower right corner of the image. In this process, WLS weighs each pixel upon the entire image.

After that, both the results from vertical and horizontal data are used to compute the final filtered output depth image. This processing is the backward processing which is the exact opposite of the forward processing as shown in Figure 4. In the vertical processing, the pixels in the upper right corner of the image are processed first and finally reach the pixels in the lower-left corner. In the horizontal processing, the pixels in the lower-left corner of the image are processed first and finally reach the pixels in the upper right corner. Based on the figures above, the WLS filter processes the image data from both the horizontal and vertical directions. Based on the analysis of Figure 3 and 4, the calculation of each pixel in the calculation process in the forward direction depends on the result provided by the calculation of the previous pixel.



(a) Image of M-KUBOS. (b) Block diagram of XCZU19eg.

Figure 5: Details of the target FPGA board.

2.3 Target FPGA Board

In our approach, we select PALTEK's M-KUBOS [3] as Figure 5a shown as our target FPGA board. It consists of Xilinx Zynq UltraScale+ MPSoC (XCZU19eg) [9] and two DDR4 DRAM modules each of which is connected to the Processing System (PS) and the Programmable Logic (PL). The DDR4 DRAM for PS is 4GB DDR4-2400 and for PL is 1x DDR4-2400 SODIMM socket. Three main specialties of M-KUBOS are listed below.

- It carries the largest XCZU19 in the Zynq ultraslace+ series. Among Multiprocessor systems-on-chips (MPSoCs), Zynq ultraslace+ has a relatively better price/performance ratio.
- It uses a high-speed bidirectional link of SAMTEC Firefly cables. This enables M-KUBOS to realize ultra-wideband communication between boards.
- It improves overall system performance by connecting Arm CPU with FPGA through a broad-band internal bus.

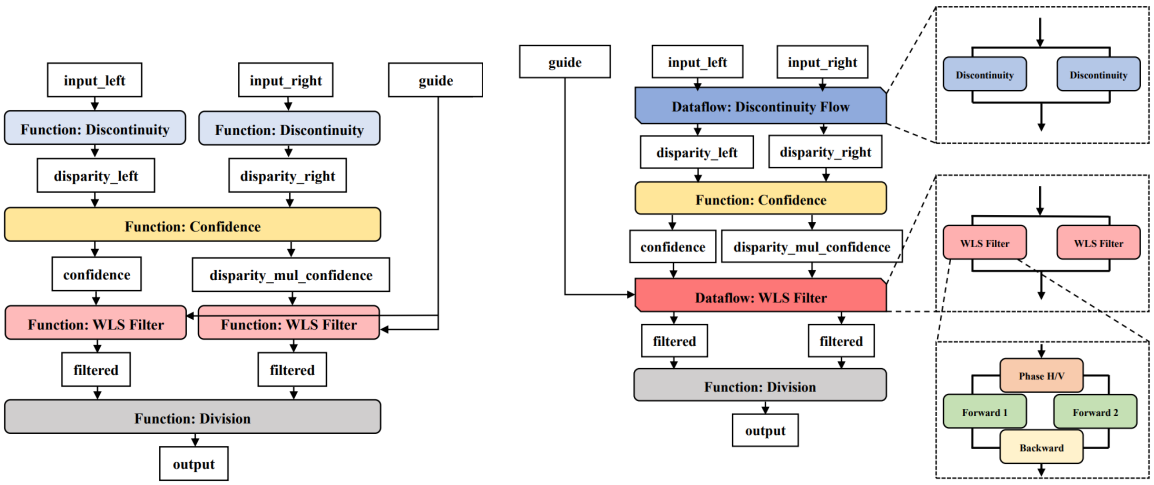
As shown in Figure 5b, various types of standard interfaces are provided for the PS in M-KUBOS. Eight GTH high-speed serial links and four GTY are provided for the PL to build an FPGA cluster. In the PS part, there is an ARM Cortex A53 quad-core and a Cortex R5 quad-core. In the PL part, there is an UltraScale+ FPGA with 1,143K logic cells, 70.6Mb block RAMs and ultra RAMs, and 1,968 DSPs. The PS part and the PL part are connected with dual AXI buses tightly within the package. These features make M-KUBOS suitable for image processing with large datasets. Moreover, the problems encountered in memory consumption and data throughput have been improved to a certain extent in terms of hardware performance.

3 Implementation

3.1 Proposed Architecture of Post-filtering System

Based on the algorithm analysis in Section 2.2, the structure of the entire system without any optimization is the same as that in Figure 6a.

From Figure 6a, we can see that the whole system takes the depth map of the left image, the depth map of the right image and a guide image as input respectively at the beginning. The two input depth arrays are put into the same function to compute the disparity of each pixel in them. In this way, we get a pair of arrays that contains information about the discontinuities between the pixels. After this, these two arrays are fed together into a function to compute the confidence. From this computation, one array containing information about the confidence and another array containing information about disparity multiplied by the confidence are output. Then, these arrays are separately input into the WLS filter for smoothing. These smoothing processes are carried out under the guidance of the input guide map. Finally, the calculated values of corresponding pixels in the two smoothed images are divided to obtain a final smoothed image.



(a) Before (b) After

Figure 6: Architecture of post-filtering system before and after optimization.

3.2 Conflict and Incompatibility

After obtaining the C++ code that can run smoothly on the CPU, we verified the code on the ARM Core through the Vitis HLS 2020.1 tool. After that, the actual FPGA implementation is performed. In the actual FPGA implementation, we encountered many problems, most of which were caused by the conflict and incompatibility between the CPU executable code and the FPGA. These problems are listed below for analysis.

Dynamic Memory Allocation In C++ coding, dynamic arrays are very flexible to use and can be used to solve problems caused by the size of array boundaries. In the original code of the WLS filter, dynamic arrays are used a lot. However, as mentioned in [10], an FPGA contains a fixed set of memory resources, and it is not supported to dynamically create or release memory resources.

Data Dependency Then executing the original code on the CPU, we only need to think about the code being able to output results. But when implementing on FPGA, we also need to consider the optimization and execution speed of the code. From the analysis done in Section 3.1, we can see that the computation of the WLS filter in Figure 6a is based on a large linear system. This makes it difficult to parallelize the WLS filter.

Memory Consumption According to Figure 6a, we can see that in the execution of the entire system, we need to create six different new arrays to save the processing results of each function. In fact, each of these six new arrays is an image with a resolution of 672×376. This greatly occupies the memory space of BRAM.

3.3 Hardware Implementation and Optimization

3.3.1 Getting Discontinuity of Neighbouring Pixels

According to Figure 6a, after receiving the input depth maps, the two depth maps are put into a function called *Discontinuity* to calculate the disparity of each pixel, where the disparity is calculated by the variance of each pixel value . In this function, two levels of nested loops are required to traverse each pixel in the image. After a pixel is found, the information of the surrounding pixels is obtained through two layers of nested loops and the variance is calculated. Thus, this function contains a total of four levels of nested loops. In this algorithm, there is no data dependency between the two outermost loops. Therefore, in order to make the processing inside the whole function look clearer,

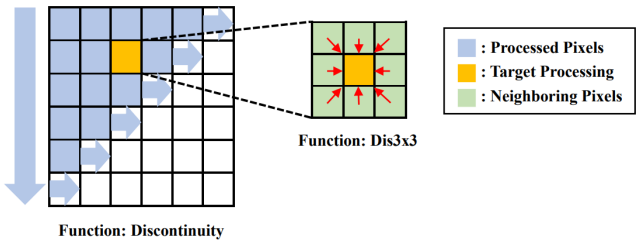
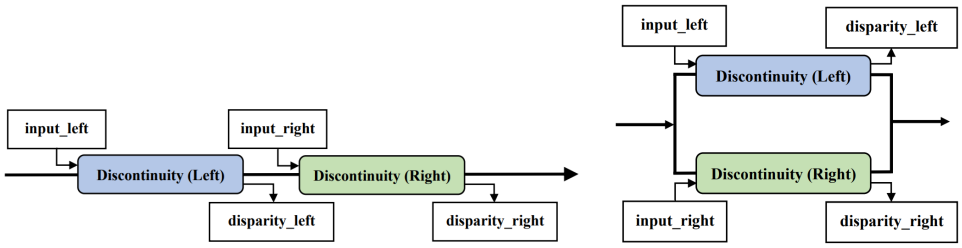


Figure 7: Getting Discontinuity of Neighbouring Pixels.



(a) Before (b) After

Figure 8: The dataflow optimization to discontinuity function.

we added a new function named *dis3x3* to the original function to complete the two inner loops, and pass the calculation results of the inner loops to outer loops. This modification makes the processing of the entire function like that shown in Figure 7.

According to Figure 7, there is no data dependency between the two outer loops in function *Discontinuity*, so they can be pipelined to speed up the process of solving the variance. Not only that, we have also done dataflow optimization for this function. From Figure 6a, we can see that the entire system needs to calculate the disparity of the input left depth map and right depth map respectively. Before optimization, these two processes are performed in order, that is, the left depth map is calculated first, and then the right depth map is calculated as shown in Figure 8a. However, this sequential execution does not make sense because there is no dependency between the disparity computations of the two depth maps. In other words, when we compute the right depth map, we do not need the result of the left depth map. Therefore, parallelizing these two processes through the dataflow approach becomes an effective optimization option. After the dataflow approach, the process becomes that shown in Figure 8b. According to Figure 8b, under the action of dataflow, the two processes are executed in parallel. In this way, the execution time of the *Discontinuity* function is reduced by half.

3.3.2 Filtering Images with WLS

In the optimization of the WLS filter, according to Figure 6a, the WLS filter needs to be used twice in the whole system to smooth the confidence map and disparity confidence map respectively. The same as for *Discontinuity* function in Section 3.3.1, there is no data dependence between the two of the WLS filter in the overall system. In other words, the processing of the left disparity confidence map does not rely on the result of the right confidence map. However, in the original code, the two uses of the WLS filter are performed in the order. Likewise, through the optimization of dataflow, two WLS filters are executed in parallel. Since WLS is the core function of the whole system, the execution time of this function accounts for most of the execution time of the whole system. Therefore, the execution speed is greatly optimized after the two WLS filters are executed in parallel.

To perform further optimization, we need to step into WLS filter processing. According to the explanation of WLS, it is a filter based on a linear system, has a huge number of data dependencies

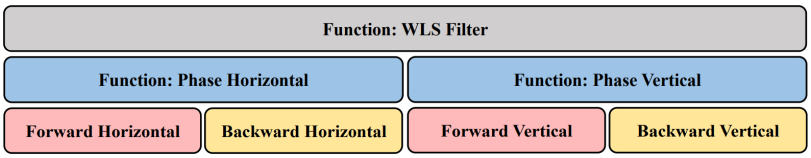


Figure 9: WLS filter split by processing steps.

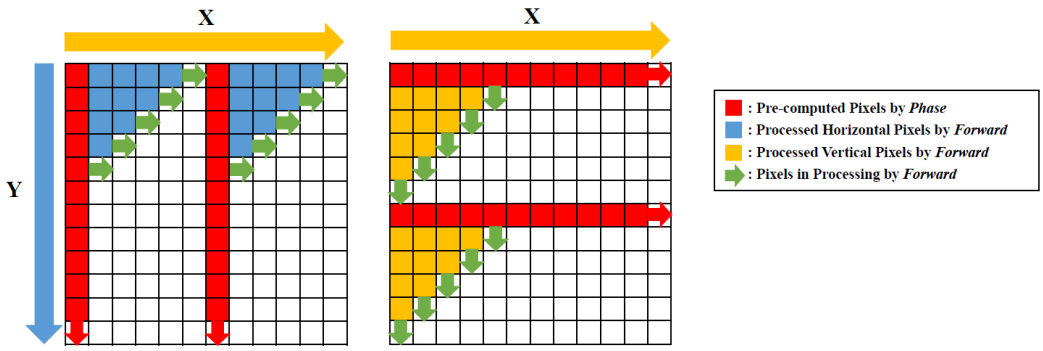


Figure 10: WLS filter horizontal and vertical forward processing after dividing.

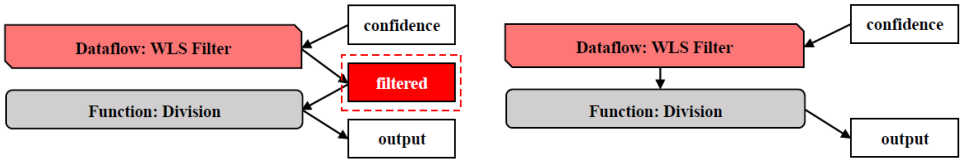
inside its processing. With such strong data dependencies, the optimizations that can be made for functions and loops within them are quite limited. Therefore, we decided to modify the algorithm to remove some data dependencies. Before the corresponding optimizations are performed, in order to make the processing within the function clearer, the WLS filter is split into seven different functions as shown in Figure 9.

According to the algorithm of WLS filter, the processing can be classified into forward processing and backward processing. According to our investigation, before optimizing, the execution speed of the forward direction processing is relatively slow. Therefore, the forward processing has been modified at the algorithm level to optimize its execution speed.

3.3.3 WLS Forward Processing

Before the horizontal and vertical forward processing begins, the *phase_horizontal* and *phase_vertical* functions in Figure 9 compute the values of the pixels in the first column and row. Then, these pixels provide dependencies for the computation of following pixels. Here, in order to increase the data processing amount of the whole system, we divide the whole processing according to the center line of the image. To do this, after computing the values of the first column and the first row, we use the *phase_horizontal* function and *phase_vertical* function again to calculate the pixels on the vertical axis and the horizontal axis of the image. These pre-computed pixels on the central axis provide dependencies for the computation of pixels in the remaining half of the image. Based on this modification, the forward processing is divided into two parts as shown in Figure 10. The first part is processed based on the pixels of the first column or row, and the second part is processed based on the pixels of the column or row on the central axis, and the two parts can be executed in parallel. The idea of this solution is to sacrifice the data dependencies of a row and a column of pixels, and trade a certain amount of accuracy for faster execution time. After dividing, the original forward processing function is copied into two functions with half the original loop boundary and executed in parallel by dataflow.

In addition to dataflow, the loops in this function are pipelined for further optimizations. According to Section 2.2.2, four intermediate arrays required for WLS processing are created. These four arrays are respectively divided into two arrays by array partition method, so we can use loop unrolling with a factor of two to unroll and pipeline the loops.



(a) Before (b) After

Figure 11: Memory consumption strategy of reducing the number of arrays.

3.3.4 WLS Backward Processing

For the part of backward processing, it accounts for a lower proportion of system execution time than forward processing. Therefore, in order not to lose too much accuracy, it is not divided into two processing parts. However, as a beneficiary of the array partition mentioned in the previous subsection, the loops involved in the backward processing are also unrolled by the loop unrolling method with a factor of two.

3.3.5 Controlling Entire Processing

After all processing is done, the final smoothed result is returned to the main function. In contrast with the architecture of the post-filtering system in Figure 6a, the optimized system architecture is shown in Figure 6b. Overall, in the algorithm-level modification of the entire WLS filter, we first split the entire function into seven different processing stages, and then investigated the time consumption of each of these stages. Then, the forward processing that has the greatest impact on the execution time is divided into two parts, and they are executed in parallel. After obtaining the results of the two forward processing, the backward processing computes each pixel in the reverse direction to obtain the final smoothing result. Finally, the result is input into the WLS filter function and returned to the main function.

3.3.6 Memory Consumption Strategy

According to Figure 6b, except the input and output arrays, we need a total of six new arrays in the processing. Those arrays occupy too much memory space of BRAM. As the solution, we changed the algorithm in WLS filter and reduced the number of the arrays from six to four. The improvement done to the algorithm is shown in Figure 11.

Here, except the two input arrays for storing the values of depth maps, the system itself creates two new arrays in WLS filter function to store the filtered arrays of both confidence map and disparity confidence map before getting into the function of computing final filtered result.

This causes the problem that the arrays for storing the input data will be wasted after they are used in the previous function. Therefore, we changed the algorithm in the function of WLS filter for both confidence map and disparity confidence map to avoid creating new arrays and made the program keep using the arrays after the functions. Which means, the function of WLS filter in post-filtering system of our approach does not need to create new arrays to process the input arrays. That reduces the number of the arrays from six to four. Furthermore, storage binding is implemented to use URAM which has a total memory space of 288 KB×128. Using URAM to store the specific large array helps reduce the memory consumption of BRAM.

3.4 Overall Hardware Architecture

As explained in the previous subsections, the post-filtering system of our approach is implemented on M-KUBOS. The architecture of the entire hardware is shown in Figure 12.

First of all, the left and right depth maps and original stereo images are saved on off-chip DDR4 memory. Among them, the left and right depth maps are used as left and right inputs and the original stereo map as a guide map. Then, DDR4 passes the data to the AXI4 interface. The

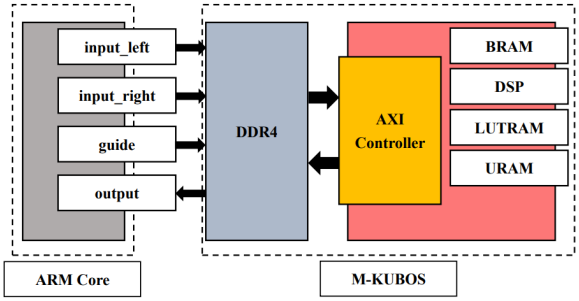


Figure 12: Overall Hardware Architecture

AXI4 interface is used to control the data stream to and from the entire FPGA. On FPGA, we use BRAM, DSP, LUTRAM and URAM to temporarily store the data needed for computation. For the strategy, we used URAM to reduce the memory consumption of BRAM. By doing this, the post-filtering system can be implemented on M-KUBOS successfully. After the processing in PL, the final result of filtered array is transferred back to the ARM core to generate a new filtered depth map for 3D projection system.

4 Evaluation

4.1 Evaluation Metrics

Before presenting the results, all evaluation metrics used in this study are explained in this subsection. These metrics include Peak Signal-to-Noise Ratio (PSNR) and energy consumption.

4.1.1 Peak Signal-to-Noise Ratio

According to the explanation from [11], PSNR is widely used in the evaluation of image processing results. It was originally used in engineering to express the ratio between the maximum power of a signal and the power of noise. These noises can adversely affect the fidelity of this signal. The signal to be evaluated tends to have a relatively wide dynamic range, so PSNR is measured in decibels. To understand PSNR, we need to know Mean Squared Error (MSE), which is a measurement tool that measures the average of the squares of the errors. Here, given an image $\mathbf{G}$ with a size of $row \times col$ and the image with noise $\mathbf{N}$, MSE is defined as follows.

$$MSE = \frac{1}{row \times col} \sum_{x=0}^{row-1} \sum_{y=0}^{col-1} (g(x,y) - n(x,y))^2 \tag{11}$$

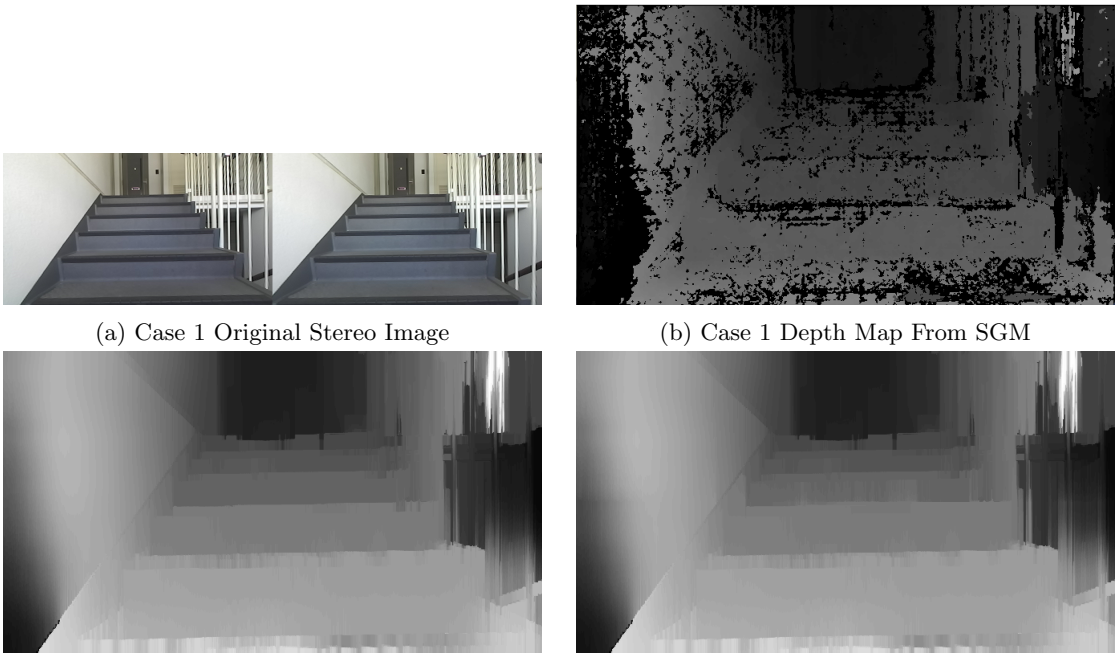
Based on the definition of MSE in Equation (11), PSNR is defined as follows.

$$PSNR = 10 \cdot \log_{10} \left(\frac{{MAX_{\mathbf{G}}}^2}{MSE} \right) \tag{12}$$

In Equation (12), $MAX_{\mathbf{G}}$ is the maximum pixel value of the pixels in image $\mathbf{G}$. According to the equation that when PSNR becomes higher, it becomes more difficult for human to recognize the difference between the images. That means, when PSNR is higher, the quality of the image or the video is better, except for some special cases. Based on the words in [12], PSNR values above 40 dB usually correspond to barely visible differences.

4.1.2 Energy Consumption

Energy consumption refers to the electric energy provided for system operation per unit time, and its measurement is generally in watts (W) and kilowatts (kW). In this study, the energy consumption of



(a) Case 1 Original Stereo Image

(b) Case 1 Depth Map From SGM

(c) Case 1 Result from WLS on CPU

(d) Case 1 Result from WLS on FPGA

Figure 13: Comparison of depth maps in case 1 of CPU and FPGA.

each module on the FPGA is counted and analyzed. Through the analysis of energy consumption, we can understand the shortcomings of the whole system in terms of energy saving and where optimization can be focused in the future.

4.2 Image Processing Results

4.2.1 Effects of WLS Filter

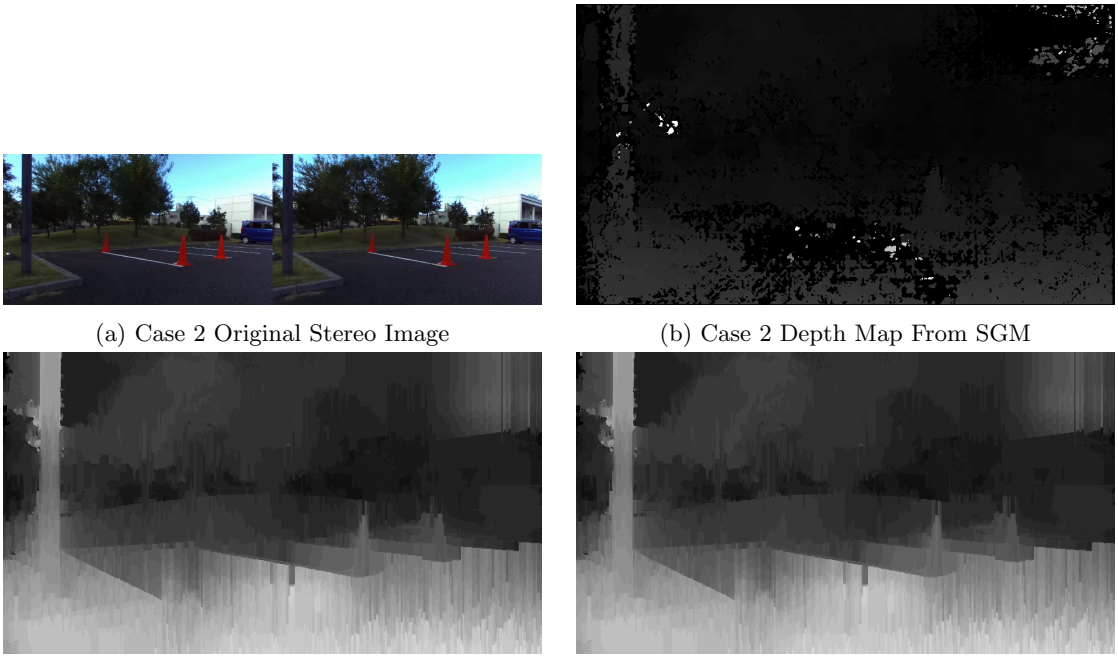
In the experiments of this study, two cases are used. Here, case 1 and case 2 correspond to the scenes of stairs and parking lots, respectively. The original stereo images of these two cases are shown in Figure 13a and Figure 14a.

According to the explanation of the post-filtering system in Section 1.1, SGM computes the depth maps of the left and right images of the stereo image during the 3D projection process. These depth maps based on the stereo image become the processing target of the post-filtering system on FPGA. In the filtering process, WLS filter in the system plays a crucial role. For comparison, the depth map provided by SGM, the processing result of the unoptimized WLS filter on the CPU and the processing result of the optimized WLS filter on the FPGA are shown in Figure 13 and Figure 14.

Based on the results, it is difficult to recognize objects in the depth map generated by SGM. The reason is that there is an extremely large number of black pixels that cannot be measured in the depth map before filtering. If not processed, these invalid pixels in the original depth map will have a great impact on the quality of the depth map, so that the depth map cannot accurately provide the depth information of the image. Same as the results in [13], whether it is processed by the WLS filter on the CPU or on the FPGA, the number of pixels that cannot be measured tends to be greatly reduced. Through this system, the 3D image processing for recognizing the distance between the subject and itself is enhanced.

4.2.2 WLS Filter on CPU and FPGA

After comparing the depth maps before and after WLS filter processing, we compare the processing results from the WLS filter on the CPU with results from the optimized WLS filter on the FPGA.



(a) Case 2 Original Stereo Image

(b) Case 2 Depth Map From SGM

(c) Case 2 Result from WLS on CPU

(d) Case 2 Result from WLS on FPGA

Figure 14: Comparison of depth maps in case 2 of CPU and FPGA.

Here in case 2, it is difficult to judge the difference between the non-optimized processing result and the optimized processing result with human eyes alone. However, in case 1, if we zoom in on the image, we can see some error pixels due to the optimization of speed at the expense of data dependence. The errors in the result is shown in Figure 15a.

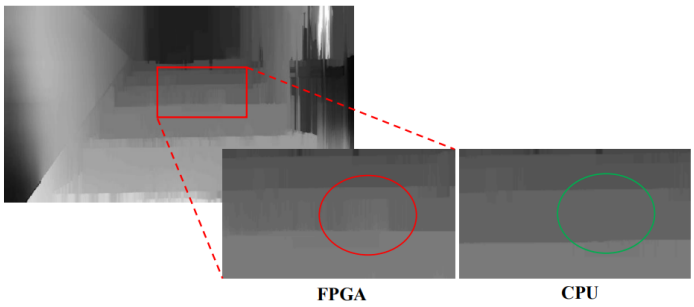
The formation of this type of error is due to the peculiarities of our optimization method. In the forward processing of the WLS filter, the data dependencies of the pixels on the horizontal and vertical central axes are removed, which means the system is weak on computing the pixel values near the central axis. Therefore, when the edge of the object in the image is close to the central axis, the incidence of errors increases, resulting the errors in the image. In Figure 15b, the situation of low accuracy is similar to that in case 1.

4.2.3 Accuracy of Optimized WLS Filter

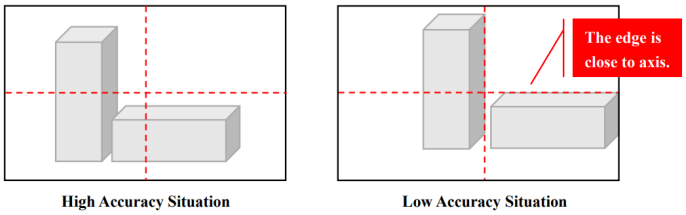
According to the optimization method explained in Section 3.3.3, in the forward processing of the optimized WLS filter, the data dependencies of the pixels on the horizontal and vertical central axes are removed. Although the execution speed of the system has been greatly improved, it is foreseeable that the accuracy will be affected to a certain extent. Therefore, how much impact the abandoned data dependencies have on the processing results needs to be assessed.

In the evaluation of accuracy, we used two specific evaluation indicators, one is how many pixels have different pixel values in the depth map generated by the optimized WLS filter compared to before optimization. One of the evaluation method is that we investigate the difference between the two depth maps directly by traversing all pixels. The other one is the PSNR mentioned in Subsection 4.1.1. We use the results generated by the non-optimized WLS filter on the CPU as the comparison standard, and the results generated by the optimized WLS filter on the FPGA as the comparison object to calculate the PSNR between the two images. After obtaining the PSNR, we combined all the data in Table 1 for evaluation.

In Table 1, *Diff* shows how many pixels have unequal pixel values in the results generated by the filters before and after optimization in one case. *Match* shows the matching rate between the depth maps generated by the filters before and after optimization in one case. In one case, the higher the



(a) The errors in Depth Map from optimized WLS on FPGA in case 1.



(b) The situations of high and low accuracy.

Figure 15: The errors in depth Map and the situations of high and low accuracy.

Table 1: Results of Accuracy of WLS Filter after Optimization

Case	Diff	Match	MSE	PSNR
Case 1	44790	82.3%	10.305	38.000
Case 2	15901	93.7%	5.150	41.013

matching rate, the smaller the accuracy impact of the algorithm modification. According to the table, out of a total of 252, 672 pixels, 44, 790 pixels have changed before and after the optimization in case 1. This results in an 82.3% matching rate between the depth maps generated by the WLS filter before and after optimization. In case 2, 15, 901 pixels have changed through optimization, which brings the matching rate to 93.7%. The PSNR between depth maps in case 1 is 38.000 dB and that in case 2 is 41.013 dB. Taking into account the acceptable PSNR range mentioned in [12], PSNR values above 40 dB usually correspond to barely visible differences, which means that our optimization method has certain feasibility.

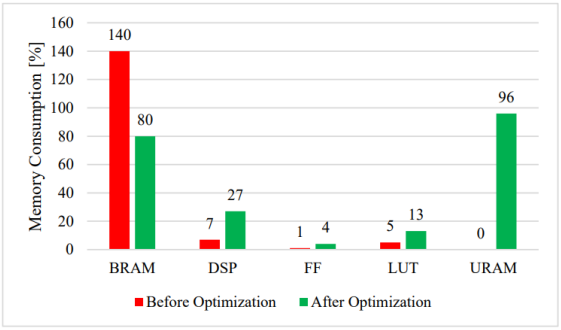
4.3 Resource Consumption

4.3.1 Memory Consumption

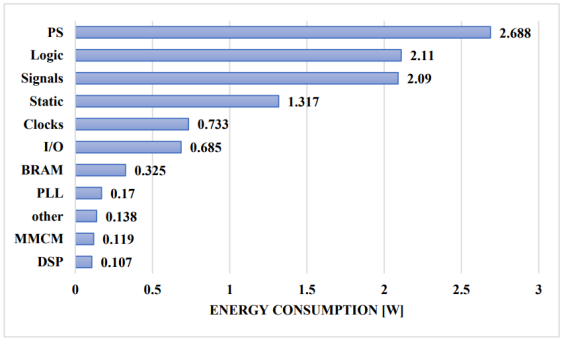
According to Figure 16a, before optimization, the memory consumption of BRAM is 140%, Digital Signal Processor (DSP) is 7%, Flip-Flop (FF) is 1%, Look Up Table (LUT) is 5% and URAM is 0%. According to the data, the total bottleneck of the implementation system is the memory. The largest obstacle is that the total usage of BRAMs is 140%, but in the FPGA implementation, the usage of each Ram should be controlled under 100%.

After applying the strategy targets to the memory consumption problem mentioned in Section 3.3.6, the memory consumption of BRAM is reduced to 80%, DSP becomes 27%, FF becomes 4%, LUT becomes 13% and URAM becomes 96%. Here, for improving the performance, the memory resource consumption of DSP increased from 7% to 27%. The increase of DSP is not related to moving memory from BRAM to URAM.

According to the result, the problem of excessive memory consumption of BRAM was successfully solved. On the one hand, this is due to the algorithmic elimination of two arrays used to store the



(a) Memory consumption before and after memory strategy.



(b) Energy consumption of each functions on M-KUBOS.

Figure 16: The results of resource consumption of WLS filter on M-KUBOS.

Table 2: Execution Time Comparison of Optimization on ARM CPU

Optimization Level	Time[s]	FPS	Accel.
None	1.06889	0.936	-
-O3	0.483206	2.07	54.8%
-Ofast	0.420845	2.38	60.6%

results of function computations. On the other hand, this is because URAM is enabled to share the memory consumption pressure of BRAM.

4.3.2 Energy Consumption

After the WLS filter is installed on the M-KUBOS, the energy consumption of each module during system execution is shown in Figure 16b. As can be seen from the figure, the energy consumption of PS reaches 2.69 W, accounting for 29% of the energy consumption of the entire system. The energy consumption of logic reached 2.11 W, accounting for 23%. In the signal module, the energy consumption reaches 2.09 W, which also accounts for 23% of the system.

4.4 Performance on ARM CPU and M-KUBOS

In the last evaluation experiment, we obtained the performance of the WLS filter on ARM CPU and the optimized WLS filter on M-KUBOS.

4.4.1 Optimization Effectiveness of Baseline CPU Implementation

On Arm CPU, we used the optimization options in g++ to compile the program. The results of the execution time with different optimization option are listed in Table 2. Here, the first option we used is the -O3 option. The result execution time of it is 0.483206s, which achieved a 54.8% performance acceleration compared to that without optimization. After that, we choose the -Ofast option to optimize the implementation further. The result execution time is 0.420845s, which achieved a 60.6% performance acceleration compared to that without optimization. The main reason of such low performance on CPU is lack of sufficient parallelisms. There are still functions in the program that can be parallelized.

4.4.2 Optimization Effectiveness of FPGA Implementation

On M-KUBOS, we took the latency of each loop in the WLS filter as the result. The result is shown in Table 3. From this table, we can see that the loop of forward processing in WLS that takes the longest time to complete is divided into four parts. Among them, the two horizontal forward

Table 3: Latency of Each Loop on M-KUBOS

Loop Name	**Lat.[ns]**
LOOP_PHASE_HOR	$3.74E+7$
LOOP_PHASE_VER	$3.41E+7$
LOOP_FORWARD_HOR1	$3.78E+4$
LOOP_FORWARD_HOR2	$3.78E+4$
LOOP_FORWARD_VER1	$2.16E+4$
LOOP_FORWARD_VER2	$2.16E+4$
LOOP_BACKWARD_HOR	$5.74E+4$
LOOP_BACKWARD_VER	$2.66E+4$
LOOP_DIS_1_2	$1.25E+7$
LOOP_CONF_1_2	$2.53E+6$
LOOP_INI_1	$7.58E+6$
LOOP_INI_2	$2.53E+6$
LOOP_OUTPUT_1_2	$2.53E+6$
LOOP_FUSION_1	$3.36E+3$
LOOP_FUSION_2	$1.88E+3$

Table 4: Execution Time Comparison of System on ARM CPU and M-KUBOS

Platform	**Time[s]**	**FPS**	**Accel.**
ARM CPU	0.420845	2.38	-
M-KUBOS	0.0991637	10.09	76.43%

processing take exactly same time to be completed. In other words, these loops are processed at the same time. That means, the optimization techniques in Section 3.3.3 has the expected effectiveness, same for the two vertical forward processing.

Besides that, we can see that the loops for getting the discontinuity of the two depth maps are successfully parallelized by dataflow optimization as the techniques shown in Section 3.3.1. That means, the execution time of the function is reduced by half.

4.4.3 Comparison of CPU and FPGA

The execution time comparison of the whole system executing on ARM Cortex A9 CPU and Zynq UltraScale+ SoC style FPGA, M-KUBOS is presented in Table 4.

Based on Table 4, the execution time of WLS filter on ARM CPU is 0.420845 second and on M-KUBOS is 0.0991637 second. Which means, the WLS filter of our approach is faster than before and 76.43% performance acceleration is achieved with HLS description with 130 MHz operational frequency on M-KUBOS compared to the software execution in the PS module ARM Cortex A9 at 1.2 GHz clock.

5 Related Work

As related work, the researches related to keywords such as 3D projection, smoothing filter, WLS optimization algorithm, FPGA implementation are introduced. In [7], the algorithm of FGS is proposed to accelerate the WLS smoothing filter. We leverage this algorithm in our FPGA implementation.

To understand 3D Projection Systems based on Stereo Matching, we take [14] as a guide for studying Global Matching-based 3D Projection System, such as [15, 16] which propose Dynamic Programming (DP), and Semi-global Matching-based 3D Projection System, such as [17, 18] which is more suitable for hardware implementation.

To select the proper filter, we relate the Bilateral Filter (BF), Guide Filter (GF) and WLS based filter. In [19], BF method is used to improve image interpolation for weighted least squares

estimation and the improved interpolation method has a the average PSNR that is 0.47 dB and 0.23 dB respectively higher than two similar approaches in [20, 21]. We take this work as a guide to understand the variance of error values in datasets of depth maps. In [22], a robust GF-based visual tracking system is proposed. The work shows that the ability to keep the edges of the image smooth is the advantage of the system. In [23], an alternative approximation whose image processing results are close enough to WLS filter is proposed. By relating this work, we understand that WLS based filter is more suitable for hardware implementation.

To sum up overall, different with related works, this study focuses on WLS implementation on FPGA in pursuit of better smoothing effect, lower power consumption and higher performance. Due to the particularity of the chosen algorithm and platform, we encountered various obstacles and challenges. Based on related work, we learned that WLS filter is based on a linear system, which means that there is a lot of data dependencies in the algorithm of WLS. For parallelization, data dependence is the largest obstacle. How to break through the limitation of data dependence and improve the parallelism of the system is the biggest challenge we encounter in algorithm modification. Followed by challenges brought by WLS, since we selected FPGA for light weight and low power consumption, we have to face the problems brought by the limited memory resources of FPGA. In image processing, it is necessary to use large arrays in order to store pixel values. This put great pressure on the limited memory resources of FPGA, and it is extremely easy to make memory consumption become a bottleneck for optimization. Although we have acquired basic knowledge and experience from related researches, the new challenges arising from algorithm and platform above still become difficulties that need to be overcome in this study.

6 Conclusion

This paper focuses on optimizing depth maps with better quality for the 3D scene display system. On 3D image processing, the quality of depth maps affects the final product directly. Therefore, to enhance the quality of 3D image processing, improving the quality of depth maps becomes an effective option. To improve the quality of the depth map and the final product, post-filtering technology is introduced as a solution. The core algorithm of this post-filtering is WLS filter, which can fill the invalid region on the depth map by using the results of neighboring pixels. In our study, the post-filtering system is a time consuming and energy sensitive system, therefore, the most appropriate method to solve the problem is to implement it on FPGA to reduce its energy consumption and accelerate the system.

At the beginning of the whole study, we obtained the corresponding WLS-based post-filtering algorithm from FGS and carried out a mathematical analysis on it. In this way, we have learned the role of each function upon the entire program and the relationship between functions. During the implementation, we recognize the conflict and incompatibility when the function is executed on CPU and on FPGA. Therefore, we optimized the code of WLS filter at the algorithm level, which contains splitting loops from functions, reducing numbers of arrays in need and removing specific data dependency. Among them, the most notable optimization is to eliminate specific data dependencies. Since WLS filter is a smoothing filter based on linear system, removing dependencies can lead to a certain loss in accuracy. After that, the optimizations made with HLS description for each function are enabled. These include dataflow, pipelining, unrolling, and memory allocation.

After implementing the optimized WLS filter on FPGA successfully, we have done several evaluations based on the image processing results, results from Vitis HLS, results from Vivado HLS, and a comparison of results on CPU and FPGA. Through the evaluation of results, we confirmed that WLS filter of our approach can improve the quality of the depth map on M-KUBOS. The accuracy evaluation of two cases in our experiments shows a PSNR of 38.00 dB and a PSNR of 41.01 dB, respectively. The memory consumption of BRAM is reduced from 140% to 80% by reducing the numbers of arrays in the system and enabling URAM.

In conclusion, through our approach, the WLS filter can smooth the depth map with 130 MHz operational frequency on M-KUBOS at the speed of 10 FPS, which achieves a 76.43% performance acceleration compared to the software execution on the ARM Cortex A9 at 1.2 GHz clock.

For future work, we will continue to explore the part of the algorithm of WLS filter that can be further optimized to improve the parallelism of the system. For example, after eliminating more data dependencies, we would like to add the simple interpolation algorithm to WLS filter to make up for the accuracy loss caused by eliminating data dependencies. In addition, although the memory consumption of BRAM has been greatly reduced, it is still as high as 80%, and the consumption of URAM as its replacement has also reached 96%. It is foreseeable that memory consumption may still be a bottleneck for future optimizations. Therefore, we would like to enable DRAM to continue to reduce BRAM consumption, and do more comprehensive array partitioning to improve the usage of distributed memory. Finally, the reduction of energy consumption and the improvement of cost efficiency is also one of our future work. The constant pursuit of making the entire WLS filter lighter enables it to be used in a wider range of applications.

References

[1] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2008.

[2] Zeev Farbman, Raanan Fattal, Dani Lischinski, and Richard Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. Graph.*, 27(3):1–10, 2008.

[3] PALTEK. Fpga computing platform m-kubos. Accessed on September 10, 2021.

[4] Ata Kaban. A new look at compressed ordinary least squares. In *2013 IEEE 13th International Conference on Data Mining Workshops*, pages 482–488, 2013.

[5] Han Du and Peter M Bentler. Distributionally weighted least squares in structural equation modeling. *Psychological methods*, June 2021.

[6] Xunyu Pan, Xing Zhang, and Siwei Lyu. Exposing image splicing with inconsistent local noise variances. In *2012 IEEE International Conference on Computational Photography (ICCP)*, pages 1–10, 2012.

[7] Dongbo Min, Sunghwan Choi, Jiangbo Lu, Bumsub Ham, Kwanghoon Sohn, and Minh N. Do. Fast global image smoothing based on weighted least squares. *IEEE Transactions on Image Processing*, 23(12):5638–5653, 2014.

[8] L. Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006.

[9] Vamsi Boppana, Sagheer Ahmad, Ilya Ganusov, Vinod Kathail, Vidya Rajagopalan, and Ralph Wittig. Ultrascale+ mpsoc and fpga families. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–37, 2015.

[10] XILINX. Vitis high-level synthesis user guide, March 2021. Accessed on October 25, 2021.

[11] Nikolay Chervyakov, Pavel Lyakhov, and Nikolay Nagornov. Analysis of the quantization noise in discrete wavelet transform filters for 3d medical imaging. *Applied Sciences*, 10(4), 2020.

[12] Sylvain Paris and Frédo Durand. A fast approximation of the bilateral filter using a signal processing approach. In *European conference on computer vision*, pages 568–580. Springer, 2006.

[13] Sergio Trejo, Karla Martinez, and Gerardo Flores. Depth map estimation methodology for detecting free-obstacle navigation areas. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 916–922, 2019.

[14] Matteo Poggi, Seungryong Kim, Fabio Tosi, Sunok Kim, Filippo Aleotti, Dongbo Min, Kwanghoon Sohn, and Stefano Mattoccia. On the confidence of stereo matching in a deep-learning era: a quantitative evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[15] Donald E. Kirk. An introduction to dynamic programming. *IEEE Transactions on Education*, 10(4):212–219, 1967.

[16] Yu-Cheng Fan, Yan-Hong Jiang, Hung-Kuan Liu, and Chieh-Lin Chen. Dynamic programming based disparity estimation circuit design for 3d image system. In *SiPS 2013 Proceedings*, pages 256–259, 2013.

[17] Xiangsheng Huang, Ziling Huang, Ming Lu, Pengcheng Ma, and Weili Ding. A semi-global matching method for large-scale light field images. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1646–1650, 2016.

[18] Jan Kallwies, Torsten Engler, Bianca Forkel, and Hans-Joachim Wuensche. Triple-sgm: Stereo processing using semi-global matching with cost fusion. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 192–200, 2020.

[19] Kwok-Wai Hung and Wan-Chi Siu. Improved image interpolation using bilateral filter for weighted least square estimation. In *2010 IEEE International Conference on Image Processing*, pages 3297–3300, 2010.

[20] Xin Li and M.T. Orchard. New edge-directed interpolation. *IEEE Transactions on Image Processing*, 10(10):1521–1527, 2001.

[21] Xiangjun Zhang and Xiaolin Wu. Image interpolation by adaptive 2-d autoregressive modeling and soft-decision estimation. *IEEE Transactions on Image Processing*, 17(6):887–896, 2008.

[22] Dandan Du, Huchuan Lu, Lihe Zhang, and Fu Li. Visual tracking via guided filter. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 1781–1785, 2015.

[23] Wei Liu, Xiaogang Chen, Chuanhua Shen, Zhi Liu, and Jie Yang. Semi-global weighted least squares in image filtering. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5862–5870, 2017.