Splatty- A Unified Image Demosaicing and Rectification Method

Pranav Verma, Dominique E. Meyer, Hanyang Xu, Falko Kuester
University of California San Diego
{pverma, dom, hax032, fkuester}@ucsd.edu

Abstract

Image demosaicing and rectification are key tasks that are frequently used in many computer vision systems. To date, however, their implementations have been plagued with large memory requirements and inconvenient dataflow, making it difficult to scale them to real-time, high resolution settings. This has motivated the development of joint demosaicing and rectification algorithms that resolve the backward mapping dataflow for improved hardware implementation. Towards this purpose, we propose Splatty: an algorithmic solution to pipelined image stream demosaicing and rectification for memory bound applications requiring computational efficiency.

We begin by introducing a polynomial Look-up-Table (LUT) compression scheme that can encode any arbitrarily complex lens model for rectification while keeping the remapping errors below 1E-10 pixels, and reducing the memory footprint to $O(min(m, n))$ from $O(mn)$ for an $m \times n$ sized image. The core contribution leverages this LUT for a unified, forward-only splatting algorithm for simultaneous demosaicing and rectification. We demonstrate that merging these two steps into a single, forward-only splatting pass with interpolation, provides distinctive dataflow and performance efficiency benefits while maintaining quality standards when compared to state-of-the-art demosaicing and rectification algorithms.

1. Introduction

Robotic and mobile imaging systems often require image processing to occur in real-time, or near real-time [21]. Vision algorithms inherently depend on input pixel values and correctness of model assumptions that are used for higher-level cognitive and heuristic interpretations. A common use case of image preprocessing is stereo depth estimation where two sensor streams, as illustrated in Figure 1, are demosaiced and rectified before stereo matching can be used to extract pixel-wise depth estimates using the epipolar constraint [15].

The data dependency should in theory allow for the data to be operated on sequentially, one pixel row at a time, without the need to temporarily store whole frames to calculate the output. In reality however, most of the preprocessing pipeline implementations separate the stages whereby full images are buffered between the steps. Additionally, both the demosaicing and rectification can each be interpreted as a lossy interpolation step, which when stacked together result in an unnecessary loss of data [18], which can be mitigated if we used one interpolation step which did both of these tasks simultaneously. This work is motivated with a two-fold set of objectives. Firstly, it is to create an algorithm that approaches ideal color and re-mapping results, and secondly, it is to have it be optimized for a streaming, forward only mapping and First-in, First-out (FIFO) data flow. By achieving this proposed flow, the hardware mapping becomes inherently efficient. This results in a computational strategy that allows the data to be processed using minimal buffer memory, thereby reducing the processing and memory requirements which are often limited in embedded systems.

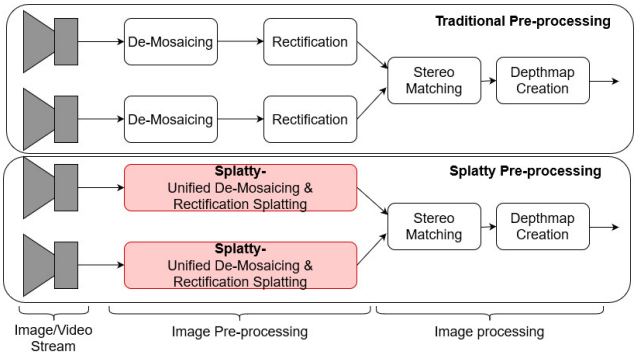

Figure 1: Stereo Pipeline- top: traditional preprocessing, bottom: Splatty preprocessing

This work proposes a novel preprocessing algorithm and architecture, Splatty, that simplifies rectification and demosaicing into a single forward mapping pass. At the time of this publication, this unification has not been explored and it is the goal that the advancement of video stream processing will enable accelerated image stream processing across sys-

tems with ever growing sensor resolutions and frame rates.

In Section 2, we provide an overview of Demosaicing and Rectification algorithms, as well as the previous work that has been done for each of them. In Section 3, we describe our approach in detail. In Section 4, we provide quantitative as well as qualitative comparisons between our method and existing methods for debayering as well as rectification, and show how our method yields a pipeline which is an order of magnitude more memory efficient than any other pipeline that can be constructed from current set of algorithms, while yielding competitive output quality. We finally provide discussion in Section 5, and our conclusions in Section 6.

2. Background Overview

2.1. Demosaicing

A majority of cameras in use today use Complementary Metal-Oxide-Semiconductor (CMOS) or a Charge-Coupled Device (CCD) based sensors, which only capture the intensity of light falling on each pixel. To capture information about color, a Color Filter Array (CFA) is usually placed over the sensor. This CFA is designed in such a way that each pixel gets light from only a single color. The most common color filter array used is the Bayer Pattern, where each pixel can measure the intensity of Red, Green or Blue color. An RGB image is obtained by interpolating the intensities of missing colors at each pixel from the intensities of pixels in the neighbourhood [30]. This process is usually referred to as debayering, or demosaicing. Several demosaicing methods have been proposed in the literature and have been compared in the works of [30].

Some algorithms (such as [31]) use interpolation methods, like bi-linear, bi-cubic, or spline interpolation, to calculate the missing color values at each pixel for each color channel. While these algorithms work very fast, these often fail at regions with sharp edges, and leave artifacts like color bleeding, and zipper effect. Other algorithms, such as [7], [27], [14] do a multi-pass approach, using interpolation to get an initial estimate of each channel, and then exploiting the correlation between each channel to improve upon their initial estimates. They produce better quality images, but this comes at the cost of increased processing: these methods need about double the computation as compared to simpler methods, to produce their outputs. Other methods (such as [10], and [28]) use frequency domain analysis to arrive at estimates for debayered images which have fewer artifacts. But frequency domain analyses have the disadvantage of being much more computationally intensive than solely spatial domain analyses, and as such, this limits their use in real-time vision pipelines, which require methods efficient in space and computational cost.

In the recent years, many neural network based approaches for demosaicing have also been suggested. [12]), [38], [25], [4], have proposed deep learning models for demosaicing. Others, such as [29], [9], [26], [39] have proposed deep CNN based approaches to super-resolution, and these have also shown success in demosaicing, since demosaicing can be reframed as super-resolution in the 3 color channels. [35], [22], [17] propose replacing Image Signal Processing (ISP) pipelines in vision systems with end-to-end learnt deep CNN models; however, they only perform demosaicing, denoising, and in some cases, intensity correction. None of these handle joint demosaicing and rectification. Almost all of the deep learning based solutions suffer from the same issue of large memory requirements: they need hundreds of MBs to store model parameters and intermediate feature maps. This makes them infeasible for use on edge devices with real-time processing requirements.

There has been some progress in recent years to transfer learnt neural network architectures to FPGAs for inference ([3], [37], [1]), by optimizing the model, datapath, or inference pipelines. Even so, with the advent of extremely deep convolutional neural networks used in various vision tasks, optimizing the best performing models so that they can process HD images in real-time hasn't been feasible up to now.

We adapt the ideas presented in [31], and [7] to design our own method of debayering that leverages a forward mapping, splatting based interpolation, which yields itself for ideal dataflow, and hardware parallelism.

2.2. Rectification

Rectification is a general image re-mapping process, used for correcting a set of distortions that are introduced in the image capturing process: lens distortion, camera tilts, offset from focal axes; as well as compensating for non-perfect placement of stereo camera pairs. This is common in stereo depth systems [19], where two cameras in a bi-focal stereo apparatus are not coplanar or row-aligned, and require re-mapping so that the epipolar lines in both the images are pixel row aligned. It is usually the most computationally demanding step in stereo processing pipelines (for eg., see Table 1 in [13]), and hence the focus of optimization, to improve resource utilization.

There are two methods to perform rectification: forward mapping, and backward mapping, with the latter being the dominant method. In forward mapping, each pixel in the input image is mapped to a location in the output image, and the input pixel's intensity value is used to calculate the intensities at the image pixels in the neighbourhood of the output pixel, as shown in top part of Figure 2. There are various methods to perform this extrapolation: the simplest method is to map the output pixel to the nearest pixel location in the output image. However, this approach leads to so called "holes" in the output image [6]. An improvement over this

approach is to use a bleeding or splatting technique, which extrapolates the pixel values from the forward-mapped pixel to all its neighbourhood pixels, and depending on the distance it bleeds over, it's value is more or less dampened using a decay function. For example, in the method described in [36], the intensity value falls off inversely with the square of the distance from the mapped location.

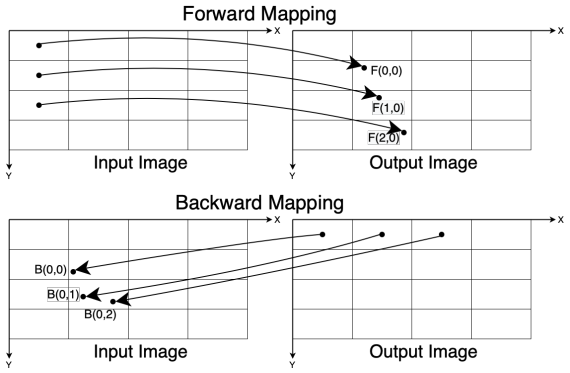

Figure 2: Forward and Backward Mapping

Backward mapping aims to overcome the problems of holes in the output image, by performing a reverse mapping: for each pixel in the output image, it finds the sub-pixel level accurate location of a point in the input image which maps to it. Then, it fills up the output image pixel by interpolating the intensity at the sub-pixel level accurate location from the intensities of the pixels in its neighbourhood [32]. This is shown in Figure 2. This approach yields good results, but is difficult to implement in a streaming setting, where the input image is coming in one row at a time, and buffering large number of rows is not feasible. The locations of backward mapped pixels may not vary uniformly, and this necessitates buffering multiple rows for both input and output images. [11].

Adapting rectification to a streaming setting to operate in real-time is shown to be a difficult endeavor by many academic and industry efforts. [2] attempts to create a real-time 3D vision system, where it uses rectification as a first step in their 3D vision pipeline. However, their results are limited to small image sizes (640x480 px). Larger image sizes at 30fps frame rates are not feasible on their systems, due to memory constraints. The work in [20] demonstrates that on state-of-the-art FPGA systems, it is still necessary to buffer image data to off-chip DDR memory instead of having it all on the on-chip BRAM memory, which means there's signification I/O overheads involved. Furthermore, the remapping step of the full stereo pipeline, is by a factor of 5, the slowest stage in the system throughput. One implementation by [34] has highlighted the feasibility to perform a radial only rectification with a backward mapping, but this implementation and architecture does not allow for more general rectifications to take place, and the output does not respect a FIFO order. [24] describe a lossy compression and subsampling based rectification method, which can handle more general cases of rectification, but it trades off reconstruction accuracy for reduced memory usage. We show in our results section that we utilise even less memory than [24] (we use $O(min(M, N))$ space for LUT storage vs $O(M \times N)$ for theirs, for an $M \times N$ pixels image), while keeping the reconstruction errors low.

3. Unifying Demosaicing and Rectification

We propose that combining rectification and debayering into one will provide improved dataflow as it reduces the in between buffering requirements, and may lead to improved accuracy. This is because both debayering and rectification can be interpreted as interpolation steps, and both these steps aren't perfectly lossless. In that scenario, performing one lossy, joint debayering+rectification step can be better than performing those two lossy steps, one after another, especially if our performance on both these tasks is as good as the existing methods for each task.

We start off by decoupling the three channels and performing rectification on each channel separately through one forward splatting pass. A Lookup Table informs the destination location of the splatting and a decay functions weighs the splatting to neighboring pixels in a set of output buffer rows. These get divided out depending on a count that is accumulated by the splatting contributions. Then, to improve the rectification result, we use the information of the green channel to improve the results of the blue and red channels, taking inspiration from the method proposed by [7]. This allows us to improve the reconstruction in areas with high frequency, such as sharp edges.

3.1. Implementation Overview

We implement rectification using a forward mapping based approach. Not only does this allows us to simplify the ordering of incoming stream of input pixels, it makes it easier for us to perform debayering along with rectification in one go on the stream of pixels, since they are ordered properly. It also allows us to map the input pixels as they come, and then discard them, allowing minimal buffering at the input side. For the output image, we need to buffer a few rows; the number depends on the extent of image distortion introduced by the rectification mapping. For most of the cases of stereo rectification that we've encountered, the maximum width of the output buffer does not exceed 50 rows. We present the algorithm flow in Figure 3, and lay out the complete algorithm in Algorithm 1

3.2. Look Up Table compression

We note that the LUT compression is important because the full look up table size grows with the size of the im-

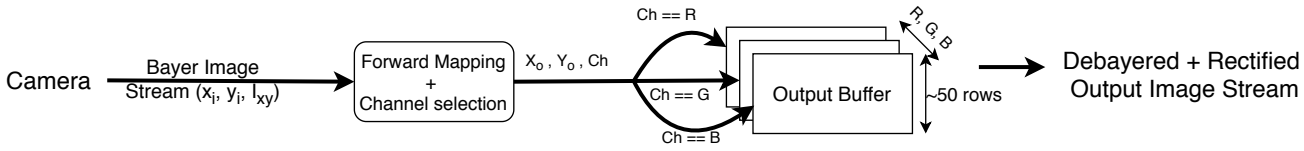

Figure 3: Proposed algorithm architecture

Algorithm 1 Forward mapping based debayering and rectification

for each row in input image **do**
  for each pixel, (x_i, y_i), in input row buffer **do**
    The color channel at (x_i, y_i) in the bayer pattern: c
    Find polynomial coefficients $P_x = LUT_x[x_i], P_y = LUT_y[x_i]$
    Compute output coordinates: $x_o = P_x(y_i), y_o = P_y(y_i)$
    Splat to the neighbourhood of (x_o, y_o), but only in the channel c
  end for
  if Top row in output buffer was not splatted to **then**
    Normalise row, adjust the values in 3 channels, and pop it out
  else
    Do nothing
  end if
end for

age. We compress LUT using a polynomial curve fitting scheme. In the most general setting, a LUT is the value of the function $f(x_{in}, y_{in})$ at discrete points x_{in}, y_{in}, which are the coordinates of the pixels in the input image. To perform LUT compression, we assume that the output y coordinate, (y_{out}) depends only on the input y coordinate (y_{in}) as an n^{th} order polynomial function of y_{in}: $y_{out} = a_0 + a_1 \cdot y_{in} + a_2 \cdot y_{in}^2 ... a_n \cdot y_{in}^n$. These coefficients will then be a function of X_{in}, the input x coordinate. We store the coefficients in a table whose size is $max(X_{in})$ For compression of the x coordinate LUT, we assume that even the output x coordinate x_{out} depends on the input y coordinate, y_{in}, as a n^{th} order polynomial, whose coefficients depend on X_{in}, and can be found as above. We have assumed here that $max(X_{in}) < max(Y_{in})$. If that's not the case, then we can flip the values and coefficient dependencies to be on x_{in} and y_{in}, instead of on y_{in} and x_{in}, respectively. This way, we will be able to compress the LUT from size $(x_{max} * y_{max})$ to $(2 * min(x_{max}, y_{max}) * (n + 1))$, where n is the order of the polynomial we use for curve fitting. For a full HD image (1080x1920), with even a 6^{th} order polynomial, we see the memory footprint decrease by $>$ 100x, with minimal increase in the computational expense for calculating

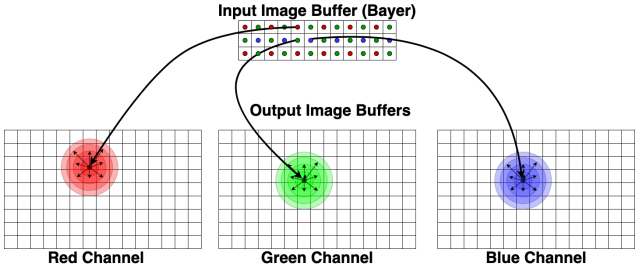

Figure 4: Illustration of splatting mechanism. Note that we splat each pixel only to it corresponding channel in the output image

the locations in the output image.

3.3. Splatting

We use a simplified version of the approach taken by [42], and [5] to define an isotropic splatting function, since computing the parameters for an anisotropic gaussian distribution at each pixel will be too computationally intensive for the demands of real-time processing on FPGAs. The splatting function, which describes how the a pixel's intensity contributes to the intensities of pixels in its neighbourhood, is defined as follows: For all pixels, P_o in the neighbourhood $N(P_o^{'})$ of pixel $P_o^{'}$, we can describe the contribution of P_o to the intensity of $P_o^{'}$ as

$$S(P_o^{'}, P_o) = I_{P_o} * f(d(P_o, P_o^{'})) \tag{1}$$

where I_{P_o} is the intensity of the pixel at location P_o, $f(.)$ is the fall-off function, which describes the variation in the contribution as a function of distance, $d(.)$.

Once we have the contribution of all neighbouring pixels, to the intensity at pixel $P_o^{'}$, we normalize by dividing with the weights assigned to each of the neighbour's contribution. This can be expressed as follows:

$$I_{P_o^{'}} = \frac{\sum_{P_o \in N} I_{P_o} * f(d(P_o, P_o^{'}))}{\sum_{P_o \in N} f(d(P_o, P_o^{'}))} \tag{2}$$

We tested functions of the form $f(d) = e^{-(d^p)}$ and $f(d) = \frac{1}{1+d^p}$, for varying exponents p, and in section 4.2, we show how varying the functional forms affected the reprojection performance.

3.4. Rectification

The rectification step is automatically accounted for in the splatting destination. The decoded LUT table provides the destination pixel address to where the weighted values need to to be splatted to, and as such, there is no need for additional interpolation.

3.5. Debayering

To perform debayering, we start with the assumption that the channels can be decoupled, and each channel is debayered and rectified together independently. After getting an initial estimate of the channels, we correct our estimates for each channel by using the intensities of the other channels, in a fashion similar to the method proposed by [7] Since each channel is being filled out by the same input pixel, we can perform this correction as we are pushing the rows out of the buffer. This ensures that each pixel of the output image stream is coming out debayered and rectified.

3.6. Dataflow

The major novelty of Splatty manifests itself in the dataflow optimization of the algorithm that lends itself to streaming processing. Image streams are generally transmitted as pixel rows, which are buffered into row buffers. Our method operates sequentially on pixels as they are incoming, directly deriving forward mapping coordinates through the polynomial LUT decoding. The LUT memory footprint is of order $N \times O \times B \times 2$, where N is the image height, O is the polynomial order and B is the coefficient bit-depth. The decoding of the LUT is accomplished through a decoding block, which evaluates the polynomial at the pixel location. Using the destination pixel coordinates, a distance calculation block is used to calculate either the L1 or L2 distance norm. This gets passed to a decay function that calculates the splatting coefficient for the neighboring pixels, which in turn multiply accumulate the results in the output buffer. Once the output row has been completely contributed to, a row state releases the row. This implies that the number of output buffer rows needed is equal to the largest vertical remapping difference per row defined by the LUT. As this can be precomputed after a camera calibration, logic synthesis and architecture can be optimally adapted. This architecture lends itself to pipelining and block parallelism that scales due to the eliminated data dependency issue that is present with a backward mapping approaches.

4. Results

4.1. Look Up table compression

In order to find the polynomial order which best fits the rectification maps, we find the mean squared error

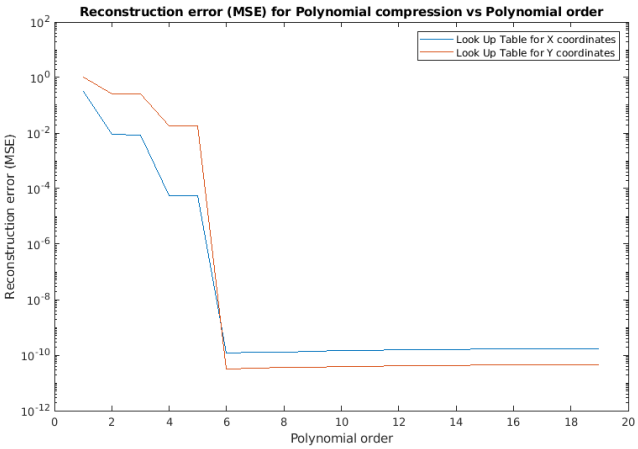

Figure 5: Mean squared Reconstruction Error vs polynomial order

distance between the values predicted by the polynomial-compressed rectification maps and the uncompressed maps for all pixels in the input image for each polynomial order. This way we can find the smallest order polynomial which fits the mapping well (i.e. which has MSE below a threshold, which we set to 1e-10).

Once we find the polynomial coefficients which best fit the mapping, we don't have to compute them again till we change the mapping itself. Therefore, the polynomial compression of the Look Up Table needs to happen only when we're calibrating the system.

In Figure 5, we present the mean squared reconstruction error as a function of the order of polynomial used for polynomial curve fitting. We observe that a 6^{th} order polynomial is sufficient for compression of LUT for this type of rectification. For orders greater than 6, we see that the errors do not decrease, and hence, in order to be as memory efficient as we can, we use order = 6.

4.2. Splatting Function

For our splatting function defined in Equation 1, we vary the falloff function f and the distance metric between two pixels, and choose the parameters which yield the highest average PSNR scores on the Kodak dataset [8].

For each value of any parameter, we find the average PSNR over all the possible values of the other parameters, over all images in the dataset. From these, the value of the parameter with the highest PSNR value is chosen. We show the results in Figure 6. From the figure, we find that the optimal splatting function would look like the following:

$$S(P_o^{'}, P_o) = I_{P_o} * exp(-\|P_o - P_o^{'}\|_1^4) \quad \forall P_o \in N_8(P_o^{'}) \tag{3}$$

where $\|.\|_1$ is the $l1$ norm, and N_k is the set of k nearest neighbours.

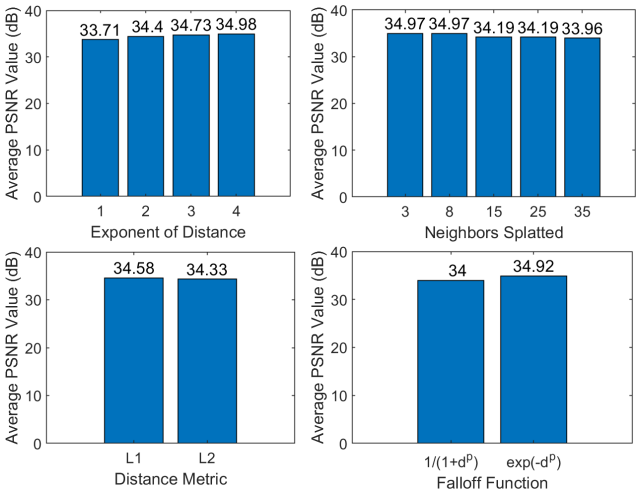

Figure 6: Variation of Average PSNR scores as we modify different parameters in the splatting function

4.3. Debayering

Since most of the debayering algorithms in use have been proposed for the general CPU+GPU architecture, we compare them with our C++ implementation in Table 1, comparing their reconstruction accuracy (via Peak Signal to Noise Ratio values) on the Kodak Images Dataset [8], along with the memory usage as a function of the image size, and the effective number of passes that each algorithm takes to produce the final output. The last two metrics are important, since multi-pass algorithms which need the full intermediate images in memory to produce outputs cannot be used effectively in a pipelined setting, where we would want to buffer the data, and use as few input and intermediate rows to produce an output row as we can. Because of this reason we only present the PSNR vs passes and memory usage order-of-magnitude for spatial domain based demosaicing algorithms, and skip the frequency domain and neural network based algorithms, since passes and memory usage order-of-magnitude make little sense in those scenarios. We also provide a more thorough comparison between all the different algorithms, by looking at average PSNR score on Kodak dataset vs the memory usage for processing 1920x1080 images, in Figure 8. The results show that our algorithm is better than the existing single pass algorithms, and some of the multi-pass algorithms ([40]), while being orders of magnitude more memory efficient than every other algorithm.

In the interest of space, we show the qualitative results from our algorithm and [31] on the complete Kodak dataset, and several close ups, in the supplementary material. The qualitative results show that our algorithm produces fewer, or about the same visual artifacts in high frequency regions (sharp edges) as the other approach ([31]).

Algorithm	Avg. PSNR (dB)	Num Passes	Memory
Bilinear	30.889	**1**	$O(mn)$
Cnst. Hue	33.259	2	$O(mn)$
Malvar	34.337	**1**	$O(mn)$
Li	34.388	2	$O(mn)$
Kimmel	35.61	2	$O(mn)$
Hamilton	**36.9**	2	$O(mn)$
Gunturk	35.8	2	$O(mn)$
Proposed	34.937	**1**	**O(min(m,n))**

Table 1: Debayering performance of various algorithms on Kodak Dataset, measured by PSNR in decibels, the effective number of passes to produce the output, and the order of memory consumed.

4.4. Rectification

To test our algorithm's rectification performance in a real world setting, we use a series of images from a stereo camera pair introduced in [33], and test our algorithm's performance on performing stereo rectification. As a implementation reference, we created a pipeline that uses the works by ([31], [16], and [41]), implemented in the OpenCV library debayering and rectification functions. We also use [16] to create our uncompressed LUT. The rectified outputs from the reference pipeline and from our Splatty algorithm are showing in Figure 7. From the qualitative results from Figures 7 along with the reconstruction errors in Figure 5 we can see that our rectification is on par with the standard rectification algorithm in use today ([16]), having a variation of less than 1e-10 in RMSE between pixel locations calculated by [16] and ours, even though we use a fraction of the memory for rectification (as seen from Table 4).

We also present a quantitative comparison between [16] and our implementation, by comparing their performance on rectifying radial, tangential and projective distortions, and their combinations. We use OpenCV's initUndistortRectifyMap function to create uncompressed LUTs from lens distortion parameters, which we then invert for use in forward mapping setting, and [16] to create uncompressed LUTs from projective transformation matrices. We take a high density checkerboard pattern, apply said distortions, and use both rectification pipelines to undistort the images, and then compute the mean absolute distance between corner points in the ground truth and the undistorted outputs from both pipelines (Table 2). Images generated in this experiment are present in the supplementary material.

4.5. Memory footprint

To the best of our knowledge, no other algorithm performs joint debayering and rectification in a streaming based approach. Therefore, we consider multiple rectification and debayering algorithms, and create a pipeline by stacking the most memory efficient rectification and debay-

(a) Bayer Input (b) [31] + [41] (c) Our Algorithm

(d) RAW Bayer input (e) [31] + [41] (f) Our algorithm

Figure 7: (a),(d) Raw, unrectified image from right camera of a stereo pair, (b),(e) Debayered, then Stereo Rectified Image using [31], and [41] (c)Debayered + Stereo Rectified Output from our algorithm, (d)-(f) Close ups from respective images

Distortions\Algorithm	Bwd Mapping ([16])	Proposed
Radial	0.8473	**0.7177**
Radial + Tangential	0.831	**0.6998**
Tangential	0.4831	**0.2021**
Projective	**0.7678**	0.9013
Average	0.7323	**0.630225**

Table 2: Mean Absolute Error in corner detection after rectifying distortions using backward mapping based approach ([16]) and our forward mapping based approach

ering algorithms together. Since many of the debayering algorithms are multi-pass algorithms which cannot work efficiently in a buffering setting, it is infeasible to implement them on FPGAs and compare their memory utilisation directly. We present a theoretical estimate on the memory utilisation of each algorithm, taking into account only the memory needed to store the input, output (as unsigned 8 bit integer values) and intermediate stages or LUTs (if any) (as float32 values), and buffering when possible, to highlight the feasibility of each algorithm for use on an edge compute device. We then show the approximate usage for processing a 1920x1080 image for non-deep-learning based methods in Table 4. In order to also compare the recent deep learning based demosaicing methods, we consider the memory usage of storing the parameters, and any intermediate feature maps that need to be stored to produce output at inference time (for eg., feature maps used in skip connections). We compute the memory usage for these methods and present a plot of demosaicing + rectification performance (in terms of PSNR scores) vs memory utilization for processing 1920x1080 image, in Figure 8. We also show the total on-chip memory available on a commonly used reference FPGA, the Xilinx Ultrascale+ ZU15EG, and the commercially available FPGA with the largest amount of on-chip memory, the Xilinx Virtex Ultrascale+ VU19P. The algorithms on the right of these lines will need to store their parameters/intermediate outputs onto off-chip memory, which is a hinderance in real-time systems, since I/O from off-chip memory is slow. We validate the algorithm using High-Level-Synthesis (HLS) tools for synthesized and routed RTL on a Xilinx ZU15EG FPGA. The resource utilization is shown in Table 3, which delivers an overall throughput of 16FPS and can accommodate up to 3 full 1080p pipelines on a single device. No off-chip memory or UltraRAM is used for this implementation and the video streams are implemented using an AXi4-Stream. Our algorithm takes 4x less memory than the next most efficient combined, debayering and rectification pipeline, and performs better than the most memory-efficient debayering algorithms, evident from the PSNR scores in Table 1. This proves that for edge scenarios, where efficiency and accuracy need to be balanced, our algorithm is the best choice for debayering and rectification preprocessing steps.

Name	**BRAM**	**DSPs**	**FF**	**LUT**
Splatty	266	63	136954	113436
ZU15EG Total	1488	3528	682560	341280
Utilization %	17	1	20	33

Table 3: Splatty Implementation Resources on a Xilinx ZU15EG FPGA

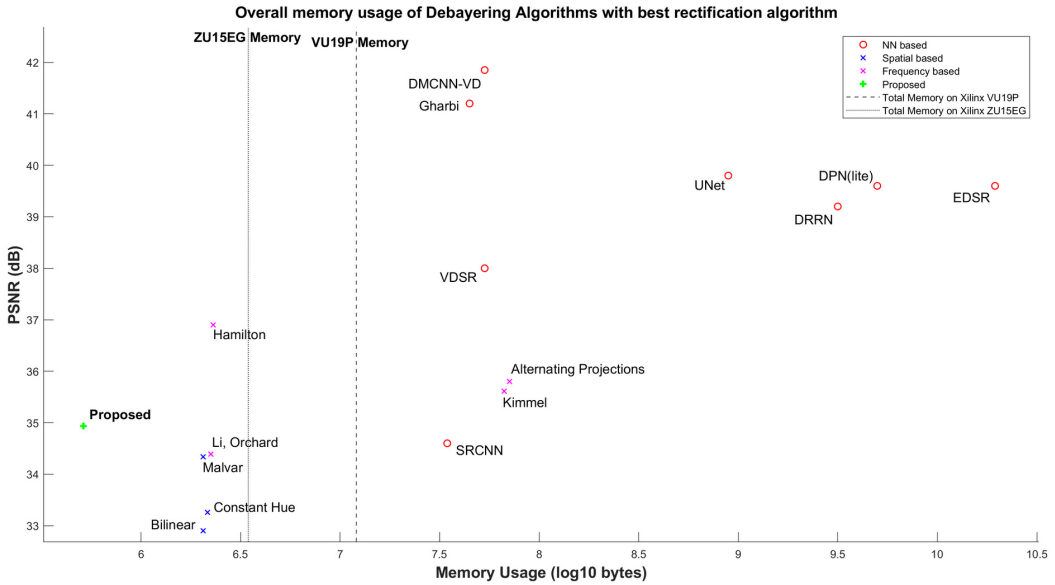

Figure 8: Memory Utilization by a theoretical debayering + rectification pipeline, constructed from various debayering algorithms and the most efficient rectification algorithm considered [34]

Rectification		Debayering	
Bwd Mapping	8100	Bilinear	**15.12**
		Malvar [31]	**15.12**
		Smooth Hue [7]	**15.12**
Oh, Kim	**2044.72**	Hamilton [23]	75
		Gunturk [14]	15693
Junger *et al.*	2120.63	Kimmel [27]	14175
		Li, Orchard [40]	35.63
Best	2044.72	Best	15.12
Rectification + Debayering Best			2059.84
Ours			**513.00**

Table 4: Theoretical estimates of memory consumption of different Debayering and Rectification algorithms, while processing a 1920x1080 pixels image, in kilobytes

5. Discussion

The results provide a valuable justification that rectification and demosaicing performance can be maintained despite the drastic gains in memory and dataflow efficiency. Firstly, we highlight that the splatting process is largely dependent on the splatting radius and decay function. Sharper fall-offs yield crisper images, but are slightly more expensive to compute in the decay function block, and larger splatting radii eliminate holes in more extreme rectification cases, but also comes at the cost of needing to splat to more pixels, that is hardware expensive. The LUT results are promising, and it can be concluded that we can achieve minimal reconstruction errors using even a low order polynomial approximation. Also, since our rectification method relies only on a general LUT to model distortions, we can incorporate more complex lens models, or refine a LUT created from simpler models to improve rectification accuracy. Future research may include color channel coupling to be adopted as a terminal stage of the pipeline and additional filtering operations to be incorporated within the decay function. While the works that outperform Splatty in terms of PSNR, all use more complex and multi-pass demosaicing, these concepts can in future also be applied to the post-splatting operations of Splatty, to similarly, improve output qualities.

6. Conclusions

We present a novel forward mapping algorithm that merges two common image preprocessing steps - demosaicing and rectification, for improved streaming feasibility. Embedded camera systems often are limited by their throughput capabilities due to hardware resource limitations, namely memory and computational blocks. We have shown that it is possible to maintain state of the art- demosaicing and rectification results by reducing the memory footprint to $O(min(m,n))$ from $O(mn)$ by merging these steps. We have also validated a polynomial rectification LUT that maintains remapping accuracies to 1E-10 RMSE. We hope that enabling efficient preprocessing through unified splatting will allow future systems to increase resolution and frame-rate operations while reducing hardware resources.

References

[1] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, and François Berry. Accelerating cnn inference on fpgas: A survey. *arXiv preprint arXiv:1806.01683*, 2018.

[2] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch. Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga-implementation. In *2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, pages 93–101, July 2010.

[3] Chaim Baskin, Natan Liss, Evgenii Zheltonozhskii, Alex M Bronstein, and Avi Mendelson. Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 162–169. IEEE, 2018.

[4] Rhys Buggy, Marco Forte, and François Pitié. Neural net architectures for image demosaicing. In *Applications of Digital Image Processing XLI*, volume 10752, page 1075218. International Society for Optics and Photonics, 2018.

[5] Baoquan Chen, Frank Dachille, and Arie Kaufman. Forward image mapping. In *Proceedings Visualization'99 (Cat. No. 99CB37067)*, pages 89–514. IEEE, 1999.

[6] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288, 1993.

[7] David R. Cok. Signal processing method and apparatus for producing interpolated chrominance values in a sampled color image signal.

[8] Eastman Kodak Company. *Kodak LossLess True Color Image Suite*, 1999(accessed 23 June 2020).

[9] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*, pages 184–199. Springer, 2014.

[10] Eric Dubois. Frequency-domain methods for demosaicking of bayer-sampled color images. *IEEE Signal Processing Letters*, 12(12):847–850, 2005.

[11] Johannes Fuertler, Konrad J Mayer, Michael Rubik, Harald Penz, Joerg Brodersen, Gemeiner Christian, Christian Eckel, and Herbert Nachtnebel. Streaming warper with cubic spline interpolation for rectification of distorted images on fpgas. In *Real-Time Image Processing 2008*, volume 6811, page 68110H. International Society for Optics and Photonics, 2008.

[12] Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. Deep joint demosaicking and denoising. *ACM Transactions on Graphics (TOG)*, 35(6):1–12, 2016.

[13] Pierre Greisen, Simon Heinzle, Markus Gross, and Andreas P Burg. An fpga-based processing pipeline for high-definition stereo video. *EURASIP Journal on Image and Video Processing*, 2011(1):18, 2011.

[14] Bahadir K Gunturk, Yucel Altunbasak, and Russell M Mersereau. Color plane interpolation using alternating projections. *IEEE transactions on image processing*, 11(9):997–1013, 2002.

[15] Hachem Halawana. *Partial demosaicing of CFA images for stereo matching*. PhD thesis, Ph. D. dissertation, Université de Lille 1, 2010.

[16] Richard I Hartley. Theory and practice of projective rectification. *International Journal of Computer Vision*, 35(2):115–127, 1999.

[17] Felix Heide, Markus Steinberger, Yun-Ta Tsai, Mushfiqur Rouf, Dawid PajÄ…k, Dikpal Reddy, Orazio Gallo, Jing Liu abd Wolfgang Heidrich, Karen Egiazarian, Jan Kautz, and Kari Pulli. Flexisp: A flexible camera image processing framework. *ACM Transactions on Graphics (Proceedings SIGGRAPH Asia 2014)*, 33(6), December 2014.

[18] Simon Heinzle, Pierre Greisen, David Gallup, Christine Chen, Daniel Saner, Aljoscha Smolic, Andreas Burg, Wojciech Matusik, and Markus Gross. Computational stereo camera system with programmable control loop. *ACM Transactions on Graphics (TOG)*, 30(4):1–10, 2011.

[19] Daniel Hernandez-Juarez, Alejandro Chacón, Antonio Espinosa, David Vázquez, Juan Carlos Moure, and Antonio M López. Embedded real-time stereo estimation via semi-global matching on the gpu. *Procedia Computer Science*, 80:143–153, 2016.

[20] Ástvaldur Hjartarson and Klas Nordmark. Implementing stereoscopic video processing on fpga. Master's thesis, Chalmers University of Technology / Department of Computer Science and Engineering (Chalmers), 2015.

[21] Andrew Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3946–3952. IEEE, 2008.

[22] Andrey Ignatov, Luc Van Gool, and Radu Timofte. Replacing mobile camera isp with a single deep learning model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 536–537, 2020.

[23] James E. Adams Jr. John F. Hamilton Jr. Adaptive color plane interpolation in single sensor color electronic camera.

[24] Christina Junger, Albrecht HeA, Maik Rosenberger, and Gunther Notni. FPGA-based lens undistortion and image rectification for stereo vision applications. In Maik Rosenberger, Paul-Gerald Dittrich, and Bernhard Zagar, editors, *Photonics and Education in Measurement Science 2019*, volume 11144, pages 284 – 291. International Society for Optics and Photonics, SPIE, 2019.

[25] Irina Kim, Seongwook Song, Soonkeun Chang, Sukhwan Lim, and Kai Guo. Deep image demosaicing for submicron image sensors. *Journal of Imaging Science and Technology*, 63(6):60410–1, 2019.

[26] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654, 2016.

[27] Ron Kimmel. Demosaicing: image reconstruction from color ccd samples. *IEEE Transactions on image processing*, 8(9):1221–1228, 1999.

[28] Nai-Xiang Lian, Lanlan Chang, Yap-Peng Tan, and Vitali Zagorodnov. Adaptive filtering for color filter array

demosaicking. *IEEE Transactions on Image Processing*, 16(10):2515–2525, 2007.

[29] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[30] Olivier Losson, Ludovic Macaire, and Yanqin Yang. Comparison of color demosaicing methods. In *Advances in Imaging and Electron Physics*, volume 162, pages 173–265. Elsevier, 2010.

[31] H. S. Malvar, Li-wei He, and R. Cutler. High-quality linear interpolation for demosaicing of bayer-patterned color images. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages iii–485, May 2004.

[32] Martin Matousek, Radim Sara, and Václav Hlavác. Data-optimal rectification for fast and accurate stereovision. In *Third International Conference on Image and Graphics (ICIG'04)*, pages 212–215. IEEE, 2004.

[33] DE Meyer, H Wang, D Sandin, C McFarland, E Lo, G Dawe, J Dai, T Nguyen, H Baker, MD Brown, et al. Starcam-a 16k stereo panoramic video camera with a novel parallel interleaved arrangement of sensors. *Electronic Imaging*, 2019(3):646–1, 2019.

[34] Sungchan Oh and Gyeonghwan Kim. An architecture for on-the-fly correction of radial distortion using fpga. *Proceedings of SPIE - The International Society for Optical Engineering*, 03 2008.

[35] Eli Schwartz, Raja Giryes, and Alex M Bronstein. Deepisp: Toward learning an end-to-end image processing pipeline. *IEEE Transactions on Image Processing*, 28(2):912–923, 2018.

[36] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, pages 517–524, New York, NY, USA, 1968. Association for Computing Machinery.

[37] Roman A Solovyev, Alexandr A Kalinin, Alexander G Kustov, Dmitry V Telpukhov, and Vladimir S Ruhlov. Fpga implementation of convolutional neural networks with fixed-point calculations. *arXiv preprint arXiv:1808.09945*, 2018.

[38] Nai-Sheng Syu, Yu-Sheng Chen, and Yung-Yu Chuang. Learning deep convolutional networks for demosaicing. *arXiv preprint arXiv:1802.03769*, 2018.

[39] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3147–3155, 2017.

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

[41] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[42] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, 2001.