A Software Platform for Manipulating the Camera Imaging Pipeline

Hakki Can Karaimer(✉) and Michael S. Brown

Department of Electrical Engineering and Computer Science,
Lassonde School of Engineering, York University, Toronto, Canada
karaimer@yorku.ca, mbrown@eecs.yorku.ca

Abstract. There are a number of processing steps applied onboard a digital camera that collectively make up the camera imaging pipeline. Unfortunately, the imaging pipeline is typically embedded in a camera's hardware making it difficult for researchers working on individual components to do so within the proper context of the full pipeline. This not only hinders research, it makes evaluating the effects from modifying an individual pipeline component on the final camera output challenging, if not impossible. This paper presents a new software platform that allows easy access to each stage of the camera imaging pipeline. The platform allows modification of the parameters for individual components as well as the ability to access and manipulate the intermediate images as they pass through different stages. We detail our platform design and demonstrate its usefulness on a number of examples.

Keywords: Camera processing pipeline · Computational photography · Color processing

1 Introduction

Digital cameras are the cornerstone for virtually all computer vision applications as they provide the image input to our algorithms. While camera images are often modeled as simple light-measuring devices that directly convert incoming radiance to numerical values, the reality is that there are a number of processing routines onboard digital cameras that are applied to obtain the final RGB output. These processing steps are generally performed in sequence and collectively make up the camera imaging pipeline. Examples of these processing steps include Bayer pattern demosaicing, white-balance, color space mapping, noise reduction, tone-mapping and color manipulation. Many of these processing steps are well-known research topics in their own right, e.g. white-balance, color space mapping (colorimetry), and noise reduction.

Although cameras are the most prominent hardware tools in computer vision, it is surprisingly difficult to get access to the underlying imaging pipeline. This is because these routines are embedded in the camera's hardware and may involve proprietary image manipulation that is unique to individual camera manufacturers. This is a significant drawback to the research community. In particular,

B. Leibe et al. (Eds.): ECCV 2016, Part I, LNCS 9905, pp. 429–444, 2016.
DOI: 10.1007/978-3-319-46448-0_26

it forces many researchers to work on topics outside the proper context of the full imaging pipeline. For example, much of the work targeting white-balance and color constancy is performed directly on the camera-specific raw images without the ability to demonstrate how it would affect the final output on the camera. Another example includes noise reduction (NR) targeting sensor noise. On a camera, NR is applied before many of the non-linear photo-finishing routines (e.g. tone-curve manipulation), however, researchers are generally forced to apply NR on the final non-linear sRGB image due to a lack of access to the camera pipeline. This presents a significant mismatch between assumptions made in the academic literature and real industry practice.

Contribution. We present a software platform to allow easy access to each stage of the imaging pipeline. Our approach operates on images saved in DNG raw image format which represents the unprocessed sensor response from the camera and the starting point for the camera processing pipeline. Our platform allows images to be opened and run through a software rendering API that parallels the onboard processing steps, including the individual processing components and their associated parameters. More specifically, our platform provides API calls that allow the modification of processing components' parameters and full access to the intermediate images at each processing stage. Such intermediate images can be modified and inserted back into the pipeline to see the effect on the final output. The proposed software platform is easily integrable with other softwares such as Matlab and provides a much needed environment for improving camera imaging, or performing experiments within the proper context of the full camera imaging pipeline.

The remainder of this paper is organized as follows: Sect. 2 discusses related work; Sect. 3 overviews our platform and the ways the pipeline can be accessed and manipulated; Sect. 4 provides a number of examples that use the platform on various routines. The paper is concluded with a short discussion and summary in Sect. 5.

2 Related Work

The basic steps comprising the camera processing pipeline are illustrated in Fig. 1 and may vary among different cameras' make and model. Full details to each component are outside the scope of this paper and readers are referred to [22]

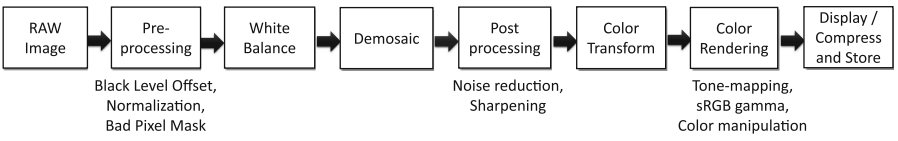

Fig. 1. This figure (adapted from [22]) overviews the common steps applied onboard a camera. Different camera hardware implementations can vary, however, most of these components will be included and in a similar processing order.

for an excellent overview. As mentioned in Sect. 1, many of the components in the pipeline (e.g. white-balance, noise reduction) are stand alone research topics in the computer vision and image processing community. Unfortunately, the hardware implementation of the camera pipeline and closed nature of proprietary cameras makes it difficult for most researchers to directly access the individual components.

To address this issue, open hardware platforms have been proposed. Early work included the CMU-camera [23] targeting robotic vision. While individual components (e.g. white-balance) were not accessible, the camera provided low-level image access and subsequent hardware releases allowed capture of the raw sensor response. A more recent and comprehensive hardware platform was the FrakenCamera introduced by Adams et al. [1]. The FrankenCamera was designed as a fully operational camera that provided full access to the underlying hardware components, including control of the flash and shutter, as well as the underlying imaging pipeline. The FrankenCamera platform targeted computational photography applications, however, the platform was suitable for modifying individual components in the imaging pipeline. While the FrankenCamera project has officially stopped, much of the platform's functionality has been incorporated into the recent Android's Camera2 API [15] that is available on devices running Android OS. The proposed work in this paper is heavily inspired by the FrankenCamera open design and aims to provide similar functionality via a software-based platform. The benefits of a software framework over a hardware solution is that it can work on images saved from a variety of different cameras and in an off-line manner. Moreover, a software platform allows greater flexibility in processing the image at intermediate stages than possible on fixed hardware implementations.

There have been a number of works that have targeted modeling the camera processing pipeline beyond simple tone-curves for radiometric calibration (e.g. [10,18–20,30]). These methods use input pairs of raw and sRGB images to derive a mapping to convert an sRGB image back to the raw linearized sensor response. In some cases, this mapping closely follows the underlying imaging pipeline [10, 18,20,30], however, in other cases the mapping is approximated by a 3D look up table [19]. Another noteworthy example is the work by Baek et al. [4] that incorporated a fast approximation of the camera pipeline for displaying raw images in the camera's view finder. While this work focused on translating sparse user interaction applied on the view finder to control the final photo-finished image, it elucidated the need to incorporate the non-linear processing steps to give a more realistic representation of the final output to the user. While these methods are useful for simulating the onboard camera imaging process, they only provide a proxy for a real camera pipeline.

The benefits of considering the full camera pipeline in various computer vision and image processing tasks have been demonstrated in prior works. For example, Bianco et al. [6,7] showed that the overall color rendition of an image can be improved by considering white-balancing and color space conversion together in the pipeline. Work by Tai et al. [25] showed that the non-linear mapping

applied onboard cameras had a significant impact on image deblurring results. Recent works by Nam et al. [21] and Wong and Milanfar [27] demonstrated improvements in image denoising when the non-linear processing steps in the camera pipeline are considered in the noise model. These prior works often have to motivate their arguments via synthetic imagery generated using relatively simple camera processing models. This lack of access to a complete software platform that is able to emulate the full camera imaging pipeline is the impetus for our work.

3 Platform Overview

Our platform uses images that are saved in the Adobe Digital Negative (DNG) format. While this format is not yet supported by many of the DSLR cameras, it is currently being supported by the newer Android phones that implement the Camera 2 API. With Android's adoption of DNG, the number of raw images captured by mobile devices are expected to increase significantly. However, in the event that images are not captured in DNG, camera-specific raw formats can be converted to DNG using the Adobe DNG conversion software tool [3]. The DNG image format not only contains the raw image data but also contains meta-data that specifies parameters (e.g. scalar values or a 1D or 3D look up table (LUT)) intended to be used by different stages in the processing pipeline.

Our platform is made possible by rewriting the interface of the open source Adobe DNG SDK software [2] that provides a full software implementation of a camera pipeline to convert the DNG raw image to its final sRGB output. While this is an engineering feat, the implementation is non-trivial. The stand alone Adobe DNG SDK is not designed to allow changes to the parameters of the individual stages, instead the SDK uses the values in the DNG files meta-data directly. Thus the processing pipeline had to be decomposed into its individual stages and API calls designed to access and modify the underlying parameters. In addition, the unmodified SDK uses a multi-threaded design that breaks the image into a number of small tiles and processes them separately. This makes it difficult to access coherent intermediate images in the pipeline using the native SDK. Our modification changes this tiling structure to allow access to the intermediate image at each stage. We have also added API calls to allow customized demosaicing and noise reduction which is not supported in the native SDK.

Figure 2-(A) overviews the processing steps that are available in the proposed camera imaging platform. The top shows the steps with the associated parameters used by each of the components while Fig. 2-(B) shows the intermediate images at each stage in the pipeline. In the following, we detail each stage and its associated parameters that can be modified. The type of parameters used by the individual stages are also discussed. In the case of a 1D LUT, the same LUT is applied to each color channel individually.

Stage 1: Reading the raw image (`Params:` *None*). The unmodified raw image is read from the DNG image file. This is the unprocessed image produced by the sensor that is still in its mosaiced Bayer pattern format.

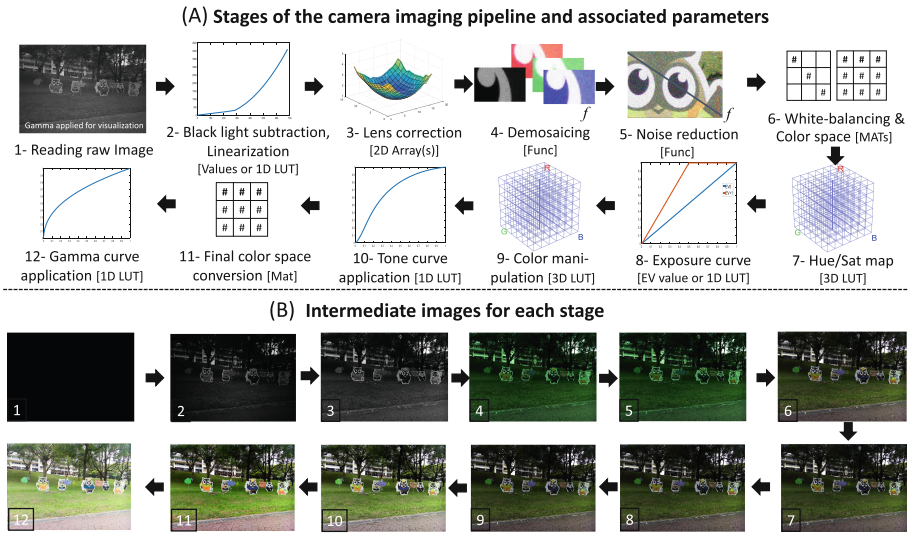

Fig. 2. The camera processing pipeline routines accessible by our software platform are shown in (A). Each component is denoted by the type of parameters it takes, e.g. scalar values, 2D Arrays, 3×3 matrices [MAT], function calls [func], or 1D or 3D Look up tables [LUT]. In addition, the software platform API supports direct access and manipulation to the intermediate images at each stage as shown in (B). (Color figure online)

Stage 2: Black light subtraction and linearization (`Params:` *Level values* or *1D LUT*). The unmodified raw image is linearized such that its values range from [0-1] in the processing pipeline. Many cameras provide a *BlackLevel* parameter that represents the black level of the sensor that deviates from 0 due to sensor noise. This is often image specific and related to other camera settings, including ISO, gain, etc. An additional *WhiteLevel* (maximum value) can also be specified. If nothing is provided, the `min` and `max` value of all intensities in the image is used to normalize the image. Another alternative is to provide a 1D LUT to perform the linearization. The 1D LUT shown in the Fig. 2-(A) is from an Nikon D40.

Stage 3: Lens/Flat Field correction (`Params:` $4 \times \text{Array}_{N \times M}$). Many cameras provide a spatially varying correction that compensates for lens distortion and uneven light fall. For example, the Motorola Nexus 6 provides four (one for each Bayer pattern pixel where G is repeated twice) scene dependent 13×17 2D arrays that are used to provide this flat field correction. These arrays are scaled and bilinearly interpolated to the image size, then multiplied to the mosaiced image.

Stage 4: Demosaicing (`Params:` func). The demosaicing step converts the single channel raw image to three full-size R/G/B color channels by interpolating the missing values in the Bayer pattern. We denote this operation as an arbitrary

function, func. The default interpolation is a standard bilinear interpolation based on the Bayer pattern layout.

Stage 5: Noise reduction (`Params:` func). Similar to the demosaicing stage, noise reduction is denoted as an arbitrary function, func. This function (not provided in the Adobe SDK) has access to the intermediate image and returns back a filtered image to the pipeline.

Stage 6: White-balancing and color space conversion (`Params:`. Two 3×3 matrices). This stage performs the necessary color space conversion between the camera specific RGB color space and a standard color space (e.g. CIE XYZ or ProPhoto RGB). This colorimetric procedure involves a 3×3 white-balance matrix (generally a diagonal matrix) and a 3×3 color space transformation matrix. The default color space used by the Adobe SDK is the ProPhoto RGB, which is a wide gamut color space commonly used for photographic color manipulation.

Stage 7: Hue/Sat map application (`Params:` 3D LUT). This optional procedure is intended to be part of the color space conversion to allow a non-linear transformation to be incorporated to improve the color rendition. While this is referred to as a 'hue' and 'saturation' modification, it is implemented as a 3D LUT applied directly to the RGB values obtained in Stage 6. For example, when saving a DNG file using the X-Rite camera calibration software [28], X-Rite adds a $6 \times 6 \times 3$ LUT to the DNG meta-data. From our experience, most cameras DNG files do not include this step.

Stage 8: Exposure compensation (`Params:` *EV value*, 1D LUT). The exposure compensation is a digital exposure adjustment. While the input is given as an exposure value (EV) that is used to control shutter and aperture settings on a camera, in the digital case, this simply applies a linear gain (either up or down) to the intensities values. The EV value passed as a parameter will generate a 1D LUT with 4096 values. Alternatively, a 1D LUT can be provided directly.

Stage 9: Color manipulation (`Params:` 3D LUT). Cameras often apply their own proprietary color manipulation that is linked to different picture styles on the camera [18]. Like the Hue/Sat map, this is applied as a 3D LUT where RGB values are interpolated based on the table's entries. The size of this table can be arbitrary, for example, images saved using Nikon D40's Camera Vivid setting have a $36 \times 16 \times 16$ LUT added in the DNG meta-data.

Stage 10: Tone-curve application (`Params:` 1D LUT). A camera-specific tone-map can be specified. This is part of the photo-finishing process on board the camera. For example, the Nikon D40's Camera Vivid profile includes a LUT with 248 entries. If no tone-curve is specified, the Adobe DNG has a default tone-curve that is shown in Fig. 2).

Stage 11: Final color space conversion (`Params:` 3×3 Matrix). This color space conversion converts the internal camera working color space into the final output-referred color space. This is done using a 3×3 matrix and is assumed to be related to color space used at stage 6. The most common color space for

cameras is the standard RGB (sRGB) and Adobe RGB. In this paper, the sRGB color space is used for all examples.

Stage 12: Gamma curve application (`Params:` 1D LUT). The final stage is a gamma curve that is applied as a 1D LUT with 4096 entries. This is intended to represent the sRGB gamma correction that is part of the sRGB specification, however, it can also be used for additional color modification and photo-finishing.

These twelve steps make up the collective stages that can be controlled via API calls or direct image modification to intermediate images. Access to this suite of components provides a comprehensive means for manipulating the image from the input raw to its final sRGB output. Note that it is not necessary that all steps be applied. For example, exposure compensation, noise reduction, hue/sat map modification, etc., can be skipped as necessary.

4 Results

We have developed a fully functioning software platform for use on a Windows-based PC. The software framework is developed in C++, however, it has also been modified such that API calls can be made directly from Matlab. In this section, we demonstrate several examples that serve to illustrate the various benefits our platform. The examples are divided loosely into three categories: (1) basic functionality; (2) evaluating stages at certain points in the pipeline; and (3) evaluating stages within the proper context of the full pipeline. Specifically, Sect. 4.1 demonstrates several examples that show the basic ability to manipulate the pipeline components (e.g. EV levels, tone-curve modification, and demosaicing). Section 4.2 provides an example to evaluate the color conversion stage, a task that is currently difficult to do with existing tools. Section 4.3 provides examples targeting white-balance, image denoising and image blurring that show the benefits of considering these tasks within the full pipeline.

4.1 Basic Processing

Exposure Compensation and Tone-mapping. Figure 3 starts with a simple example showing the effects of manipulating parameters for the exposure compensation and tone-mapping stages. Figure 3-(A) shows a number of EV values that are passed directly to our platform's API which generates the 1D LUT shown. In the case of the tone-mapping, the 1D LUTs are directly passed to the API as shown in Fig. 3-(B). The images shown represent the final sRGB output obtained using these parameters in the full camera pipeline.

Demosaicing. Figure 4-(A, B) demonstrates examples of two different demosaicing procedures applied to an image. In particular, we use the default bilinear interpolation and the work by Gunturk et al. [16] that uses alternating projections. Interestingly, in the work by [16], the results were demonstrated by simulating a mosaiced image by using an sRGB image and arranging its colors into a Bayer pattern structure. In our example, their approach is applied directly to a real mosaiced raw image and then returned back to the pipeline to produce a realistic result.

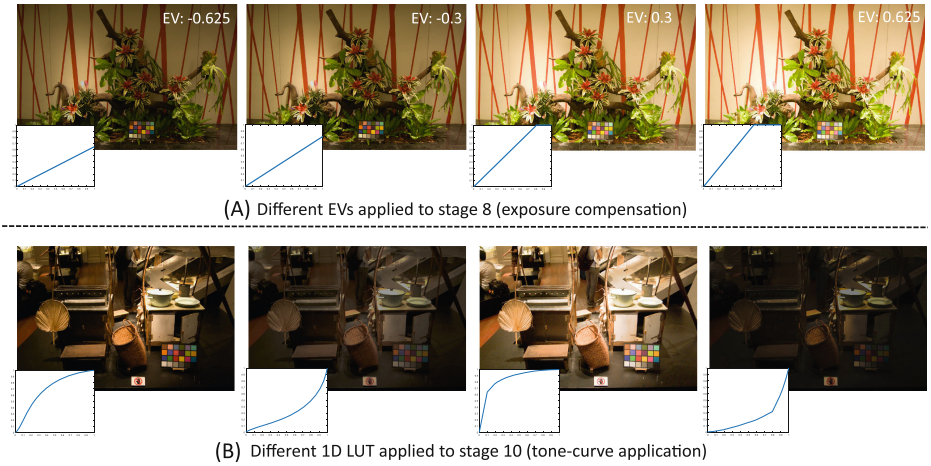

Fig. 3. (A) Examples of applying different EV and 1D LUTs for exposure compensation (stage 8) and (B) tone-mapping (stage 10).

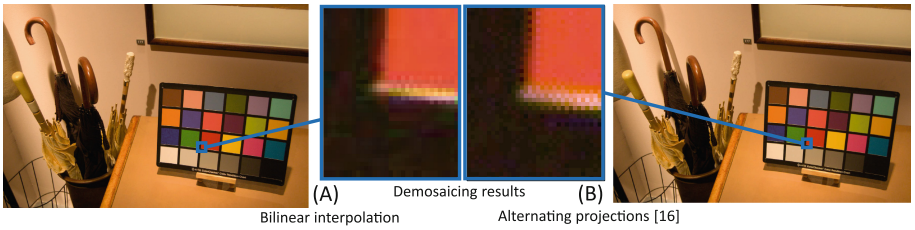

Fig. 4. Demonstrating the results of two different demosaicing algorithms, in particular (A) bilinear interpolation and (B) [16]. In this example, the intermediate image passed to stage 4 (demosaicing) is modified using [16] and inserted back into the imaging pipeline to obtain the final sRGB output. (Color figure online)

4.2 Evaluation of Components

Colorimetry Example. One challenge for existing computer vision and image processing research is the ability to obtain intermediate images in the camera pipeline to evaluate the effectiveness at individual stages. An excellent example of this is the color conversion component (stage 6). This stage is crucial in making sure that different camera-specific color spaces align to the same canonical color space after color conversion. Examining this stage in the camera pipeline is essentially evaluating the quality of the colorimetric calibration of the camera.

To demonstrate our platforms ability to assist with this task, we captured standard color rendition charts with four different mobile cameras (LG-G4, Motorola Nexus 6, Samsung S6-Edge, and an HTC One M9) under different illuminations. These mobile device cameras all support the DNG file format and have embedded in their DNG meta-data the camera's onboard parameters for

this color conversion. This allows us to compare the results of the native camera's colorimetric ability with two other approaches: (1) the widely used X-Rite calibration software [28], and (2) a recent method by Bastani and Funt [5].

We use the 24 patch color rendition chart to calibrate the color space conversion parameters using the X-Rite software and the method by [5]. In this case, the color space conversion is computed from the camera raw color space to the ProPhoto color space. For all methods, including the camera native, the white-balance matrix is estimated using the neutral colors on the color chart under a specific illumination. In order to compare these three methods, we need to apply the 3×3 color conversion matrices to stage 6. In the case of X-Rite, we also apply the additional hue/sat map (stage 7) that is used by X-Rite to provide a further non-linear correction for the color space mapping. The color pipeline is stopped at the appropriate location for each method and the intermediate image is obtained and the color patches' average chromaticity values are compared.

The results are shown in Fig. 5-(A) for color rendition charts captured under two different illuminations. The plots show the average chromaticity of the 24

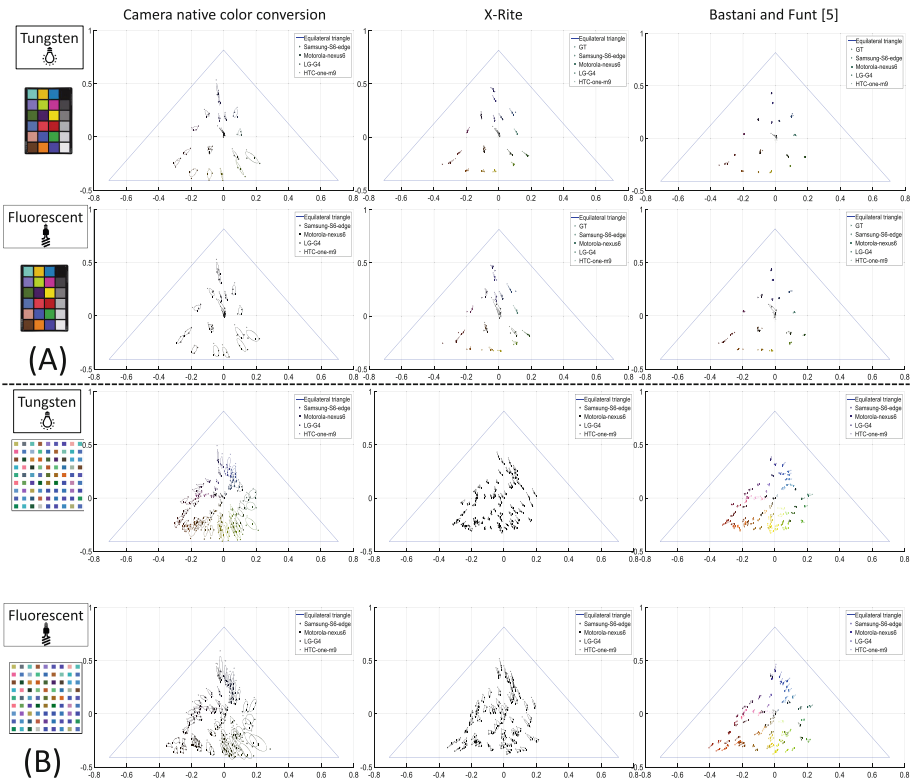

Fig. 5. This figure demonstrates the ability to evaluate different color space conversion (stage 6) methods applied to four cameras. See Sect. 4.2 for details. (Color figure online)

Table 1. The table shows the comparisons of error between native cameras, X-Rite's, and [5] color calibration (CC: color chart, AM: additional materials).

Error	Incandescent			Fluorescent			Outdoor		
	CC	AM#1	AM#2	CC	AM#1	AM#2	CC	AM#1	AM#2
Camera	0.84	1.67	3.43	0.97	2.37	4.49	0.68	1.80	3.01
X-Rite	0.33	1.06	1.57	0.48	1.68	2.72	0.15	0.74	0.95
Bastani and Funt [5]	**0.14**	**0.75**	**1.07**	**0.25**	**1.16**	**1.38**	**0.12**	**0.70**	**0.70**

color patches from the four different cameras. Under ideal colorimetric mapping, the chromaticity values should all lie at the exact same location in the chromaticity plot, however, due to errors in the color conversion matrices, they are not the same. To help with the visualization, we fit a Gaussian ellipsoid to show the spread for each color patch among the four cameras. Our experiment shows that the method by Bastani and Funt [5] provides the most consistent color space mapping.

For the example in Fig. 5-(A), X-Rite and [5] have an unfair advantage as they were calibrated using the same 24 color rendition chart that is used to show the results. To test these methods ability on additional materials, we use their estimated color space conversion parameters on a new set of color patches consisting of 81 different types of materials. This is shown in Fig. 5-(B), where again we see the work by Bastani and Funt [5] obtain the best results.

We further evaluated these three methods by computing quantitative errors with respect to the ground truth color values of the color rendition chart [29] in the ProPhoto RGB color space. In this case, we follow the procedure common in color research and consider the angular error $\epsilon_{angle}(e_{color})$ of a color e_{color} from the ground truth color e_{gt} is computed as follows:

$$\epsilon_{angle}(e_{color}) = cos^{-1}\left(\frac{e_{color} \cdot e_{gt}}{||e_{color}||\ ||e_{gt}||}\right). \tag{1}$$

Table 1 shows the angular errors for the color chart (CC) and the additional 81 materials (AM). As demonstrated in the plots in Fig. 5-(A, B), the method by Bastani and Funt [5] provides the best results. This type of analysis is challenging without the support of our platform.

4.3 Evaluating Tasks Within the Full Pipeline

In the following, we show several tasks that benefit from having access to the full processing pipeline.

White-balancing/color constancy. One of the key processing steps applied to virtually all images is white-balancing. This procedure falls into the larger research of color constancy that mimics our human perceptual ability to perceive materials under different illuminations as the same color. White-balance approximates this ability by attempting to ensure that at least the neutral scene

Fig. 6. (A) White-balancing algorithms applied on raw images. (B) sRGB output of white-balanced images.

materials appear achromatic in the camera color space. White-balance is applied directly to the camera raw image while the image values are still in the camera's RGB color space.

A number of research papers on this topic (e.g. [9,12]) provide subjective results of their white-balance result directly on the camera raw images. Such results, however, have little visual meaning as they provide a visual comparison in an non-standard camera-specific color space. A more appropriate way to subjectively evaluate the results would be to run the white-balanced result through the full camera pipeline to produce a realistic output that would be produced by the camera. Figure 6 shows the difference in these two approaches. In particular, two well known white-balance methods, Grey world [8] and Grey edge [26] are applied to an input image. Figure 6-(A) shows the results on the raw image and Fig. 6-(B) shows the sRGB outputs. The sRGB output provides a much more realistic comparisons of the two algorithms.

Noise Reduction. Similar to white-balancing, another research area that is at a disadvantage by not having access to the full camera pipeline is image denoising. Noise reduction is a well-suited research topic and interested readers are referred to [11] for an excellent overview. One of the major sources of image noise is what is collectively referred to as sensor noise and is attributed to underlying imaging sensor (CMOS or CCD). Because this noise is present on the sensor, it is present in the raw image at the start of the pipeline. As such, noise reduction is often applied before the non-linear stages in the camera pipeline. However, since few researchers have access to the camera pipeline, noise reduction methods, e.g. the popular BM3D method by Dabov et al. [13], are typically applied and evaluated on the sRGB output. Figure 7 demonstrates the disadvantages of applying image denoising outside the proper context of the full imaging pipeline. Our example works from a synthetic image to provide a ground truth input to compute the peak signal to noise ratio (PNSR). Figure 7-(A) shows a raw image that has been corrupted with zero-mean Gaussian noise. The noise profile for three different homogenous patches with increasing intensity values are shown. We can see that this Gaussian noise profile appears uniform over the different patches in the raw image, with only a shift by the mean intensity. Below this, we show the corresponding noisy raw image that has been processed through the whole imaging pipeline, including the non-linear stages (e.g. stage 9, 10, 12). We can

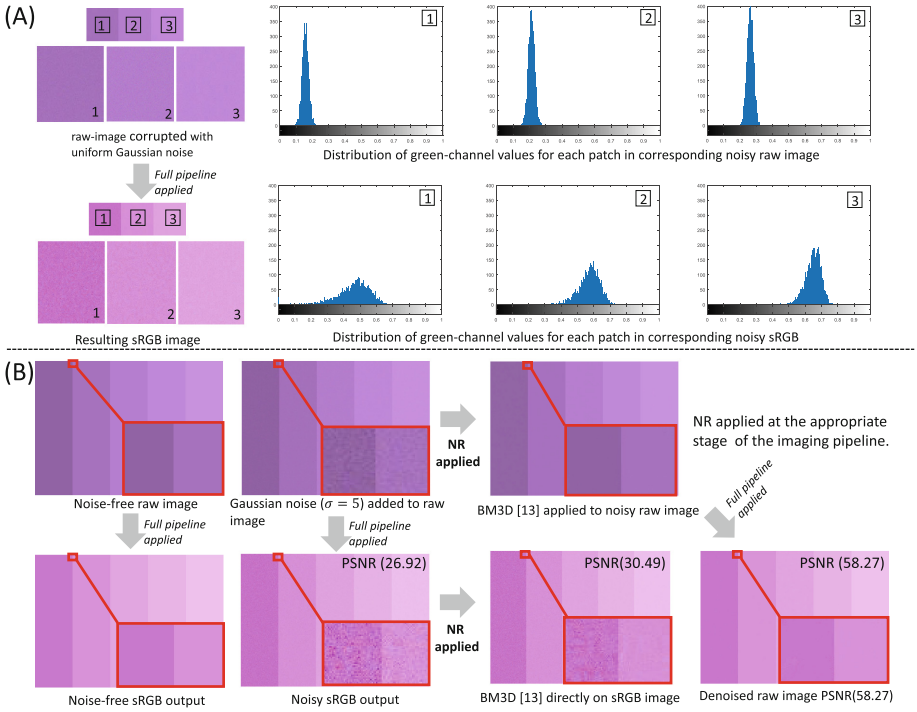

Fig. 7. (A) Noise profiles for three color patches for a raw image and corresponding sRGB image. (B) PSNR comparison of application of BM3D [13] on the raw image at stage 5 and on the final sRGB output. (Color figure online)

see that the noise distribution for different patch is significantly affected by the non-linear processing stages. In fact, it is not possible to make a uniform noise assumption for the sRGB image as the non-linear process has changed this property of the noise.

Figure 7-(B) demonstrates BM3D applied at two different places in the pipeline. The top shows BM3D applied to the raw image. The filtered result is then processed through the remaining pipeline. This represents the proper application of NR. The bottom shows BM3D applied directly to the noisy sRGB image as done in most academic literature. The PSNR for both results in the sRGB final output is computed against the ground truth (noise free) sRGB image. The PSNRs are drastically different, where the denoising applied at the right place in the pipeline is 58.27, while the application on the final non-linear sRGB stage is only 30.49.

This example serves to motivate the need for having access to the full camera pipeline when examining image denoising. Even in the inevitable case that noise reduction must be applied to the non-linear sRGB image, our software platform provides an excellent means to study sensor noise and how it is affected by the camera processing steps as done in Fig. 7-(A).

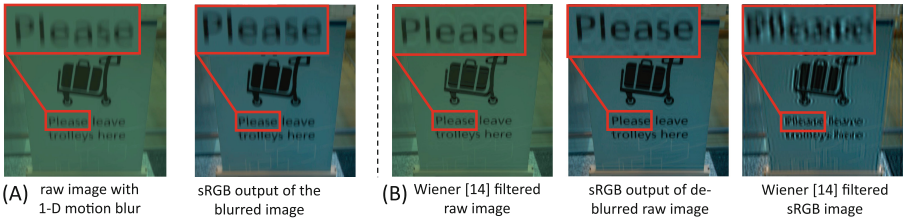

Fig. 8. (A) Motion blur is applied on the raw image and then run through the full pipeline to obtain the sRGB output. (B) Deblurring applied to the raw image and then its output using the camera pipeline, and the results obtained by directly deblurring the blurred sRGB image.

Deblurring. We conclude with a final example that was mentioned in Sect. 2 as a motivating factor for developing this software platform. In particular, the work by Tai et al. [25] demonstrated that the non-linear processes on the camera have a negative impact on image deblurring. Similar to the denoising example, [25] showed that the non-linear tone-mapping function (stage 10 and/or 12) changes the distortion profile, in this case the blur's point-spread-function (PSF), such that the PSF was no longer spatially uniform over the image. Tai et al. used this to argue that deblurring should be applied either directly to the raw image, or that care must be taken to undo any non-linear processing applied to the image before deblurring is applied. In that work, Tai et al. used a very simple camera model that applied a single 1D LUT to the raw images. This is equivalent to only applying stage 1 and 10 in our software platform.

With our platform, we are able to provide a much more realistic demonstration of Tai et al. [25] argument. Figure 8-(A) shows a raw image that has been blurred with a PSF modeling linear motion across 50 pixels and its corresponding sRGB output. This is applied directly to a demosaiced raw image (available at stage 4 in our pipeline). We then apply deconvolution via Wiener filtering [14] in two manners as shown in Fig. 8-(B). The first, as advocated by [25], is on the raw image that is then processed through the full pipeline. In the second example, we apply Wiener filter directly to the sRGB image. As expected, the results applied on the raw image are significantly better than those applied on the sRGB image. As with the prior noise reduction experiment, this example demonstrates the benefit of being able to perform various computer vision and image processing tasks within the appropriate context of the camera pipeline.

5 Discussion and Summary

This paper has presented a new software platform that allows low-level access to the individual components in the camera imaging pipeline. Specifically, our platform leverages the Adobe Digital Negative (DNG) image file format and makes the necessary modifications to the available DNG SDK to provide an extensive API for modifying the parameters of the pipeline, as well as allowing

access and modification to intermediate images that can then be inserted back into the pipeline to compute the final output that would be obtained on a camera.

The usefulness of this platform has been demonstrated on a number of examples, including white-balance, noise reduction, and colorimetry. While this work is engineering in nature, we believe this platform provides a much needed mechanism for researchers to modify individual components in the pipeline and demonstrate their results within the appropriate context of the full camera imaging pipeline. Furthermore, with the adoption of the DNG raw image format by the Android OS via the Camera 2 API, the availability of DNG files is going to significantly increase, further adding to the timeliness of this platform into the computer vision community.

One limitation of our approach is that it can only operate from the captured DNG image saved by the camera. This means camera parameters such as ISO settings that directly affect analog amplification on the sensor hardware or the image's exposure at capture time cannot be modified with our platform. This can impact work targeting tasks such as high dynamic imaging (e.g. [24]). For such cases, it will be necessary to capture a number of DNG images with varying ISO or exposure settings to simulate the manipulation on the camera.

We also note that this paper represents the current camera architecture where each stage in the pipeline is self-contained. Recent work by Heide et al. [17] demonstrated the benefits of considering a re-engineered onboard camera processing system that provides a holistic consideration to image formation that is more readily able to incorporate known priors about nature images. We also envision that in the coming years the traditional pipeline described in this paper will likely see significant changes. Once again, it will be important for researchers to have access to a software platform that allow research to be performed in the proper context of the onboard imaging system.

Acknowledgments. This study was funded in part by a Google Faculty Research Award. We would also like to thank Eric Chan from Adobe Research for his discussions on the Adobe DNG SDK.

References

1. Adams, A., Talvala, E.V., Park, S.H., Jacobs, D.E., Ajdin, B., Gelfand, N., Dolson, J., Vaquero, D., Baek, J., Tico, M., et al.: The FrankenCamera: an experimental platform for computational photography. ACM Trans. Graph. **29**(4), 29:1–29:12 (2010)
2. Adobe, Sys., Inc.: DNG Software Development Kit (SDK). https://www.adobe.com/support/downloads/detail.jsp?ftpID=5475. Accessed 16 July 2015
3. Adobe, Sys., Inc.: Adobe Camera Raw and DNG Converter for Windows. https://www.adobe.com/support/downloads/product.jsp?product=106&platform=Windows. Accessed 16 July 2016
4. Baek, J., Pajak, D., Kim, K., Pulli, K., Levoy, M.: WYSIWYG computational photography via viewfinder editing. ACM Trans. Graph. **32**(6), 198:1–198:10 (2013)
5. Bastani, P., Funt, B.: Simplifying irradiance independent color calibration. In: Color Imaging XIX: Displaying, Processing, Hardcopy, and Applications (2014)

6. Bianco, S., Bruna, A., Naccari, F., Schettini, R.: Color space transformations for digital photography exploiting information about the illuminant estimation process. J. Opt. Soc. Am. A **29**(3), 374–384 (2012)
7. Bianco, S., Bruna, A.R., Naccari, F., Schettini, R.: Color correction pipeline optimization for digital cameras. J. Electron. Imaging **22**(2), 023014:1–023014:10 (2013)
8. Buchsbaum, G.: A spatial processor model for object colour perception. J. Frankl. Inst. **310**(1), 1–26 (1980)
9. Chakrabarti, A., Hirakawa, K., Zickler, T.: Color constancy with spatio-spectral statistics. IEEE Trans. Pattern Anal. Mach. Intell. **34**(8), 1509–1519 (2012)
10. Chakrabarti, A., Scharstein, D., Zickler, T.: An empirical camera model for internet color vision. In: BMVC (2009)
11. Chatterjee, P., Milanfar, P.: Is denoising dead? IEEE Trans. Image Process. **19**(4), 895–911 (2010)
12. Cheng, D., Price, B., Cohen, S., Brown, M.S.: Effective learning-based illuminant estimation using simple features. In: CVPR (2015)
13. Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K.: Image denoising by sparse 3-D transform-domain collaborative filtering. IEEE Trans. Image Process. **16**(8), 2080–2095 (2007)
14. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 3rd edn. Prentice-Hall Inc., Upper Saddle River (2006)
15. Google, Inc.: Camera2 API Package Summary. http://developer.android.com/reference/android/hardware/camera2/package-summary.html. Accessed 16 July 2016
16. Gunturk, B.K., Altunbasak, Y., Mersereau, R.M.: Color plane interpolation using alternating projections. IEEE Trans. Image Process. **11**(9), 997–1013 (2002)
17. Heide, F., Steinberger, M., Tsai, Y.T., Rouf, M., Pajak, D., Reddy, D., Gallo, O., Liu, J., Heidrich, W., Egiazarian, K., Kautz, J., Pulli, K.: FlexISP: a flexible camera image processing framework. ACM Trans. Graph. **33**(6), 231:1–231:13 (2014)
18. Kim, S.J., Lin, H.T., Lu, Z., Susstrunk, S., Lin, S., Brown, M.S.: A new in-camera imaging model for color computer vision and its application. IEEE Trans. Pattern Anal. Mach. Intell. **34**(12), 2289–2302 (2012)
19. Lin, H.T., Lu, Z., Kim, S.J., Brown, M.S.: Nonuniform lattice regression for modeling the camera imaging pipeline. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012. LNCS, vol. 7572, pp. 556–568. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33718-5_40
20. Lin, H., Kim, S.J., Süsstrunk, S., Brown, M.S.: Revisiting radiometric calibration for color computer vision. In: ICCV (2011)
21. Nam, S., Hwang, Y., Matsushita, Y., Kim, S.J.: A holistic approach to cross-channel image noise modeling and its application to image denoising. In: CVPR (2016)
22. Ramanath, R., Snyder, W.E., Yoo, Y., Drew, M.S.: Color image processing pipeline. IEEE Signal Process. Mag. **22**(1), 34–43 (2005)
23. Rowe, A., Goode, A., Goel, D., Nourbakhsh, I.: CMUcam3: an open programmable embedded vision sensor. Technical report CMU-RI-TR-07-13 (2007)
24. Serrano, A., Heide, F., Gutierrez, D., Wetzstein, G., Masia, B.: Convolutional sparse coding for high dynamic range imaging. Comput. Graph. Forum **35**(2), 153–163 (2016)

25. Tai, Y.W., Chen, X., Kim, S., Kim, S.J., Li, F., Yang, J., Yu, J., Matsushita, Y., Brown, M.S.: Nonlinear camera response functions and image deblurring: theoretical analysis and practice. IEEE Trans. Pattern Anal. Mach. Intell. **35**(10), 2498–2512 (2013)
26. van de Weijer, J., Gevers, T., Gijsenij, A.: Edge-based color constancy. IEEE Trans. Image Process. **16**(9), 2207–2214 (2007)
27. Wong, T.S., Milanfar, P.: Turbo denoising for mobile photographic applications. In: ICIP (2016)
28. X-Rite, Inc.: X-Rite ColorChecker Camera Calibration software. http://xritephoto.com/ph_product_overview.aspx?ID=1257&Action=Support&SoftwareID=986&catid=28. Accessed 16 July 2016
29. X-Rite, Inc.: X-Rite ColorChecker Chart. http://xritephoto.com/colorchecker-classic. Accessed 16 July 2016
30. Xiong, Y., Saenko, K., Darrell, T., Zickler, T.: From pixels to physics: probabilistic color de-rendering. In: CVPR (2012)