Local Linear Approximation for Camera Image Processing Pipelines

Haomiao Jiang[a], Qiyuan Tian[a], Joyce Farrell[a], Brian Wandell[b]
[a] Department of Electrical Engineering, Stanford University
[b] Psychology Department, Stanford University

Abstract

Modern digital cameras include an image processing pipeline that converts raw sensor data to a rendered RGB image. Several key steps in the pipeline operate on spatially localized data (demosaicking, noise reduction, color conversion). We show how to derive a collection of local, adaptive linear filters (kernels) that can be applied to each pixel and its neighborhood; the adaptive linear calculation approximates the performance of the modules in the conventional image processing pipeline. We also derive a set of kernels from images rendered by expert photographers. In both cases, we evaluate the accuracy of the approximation by calculating the difference between the images rendered by the camera pipeline with the images rendered by the local, linear approximation. The local, linear and learned (L^3) kernels approximate the camera and expert processing pipelines with a mean S-CIELAB error of $\Delta E < 2$. A value of the local and linear architecture is that the parallel application of a large number of linear kernels works well on modern hardware configurations and can be implemented efficiently with respect to power.

1. Introduction

The image processing pipeline in a modern camera is composed of serially aligned modules, including dead pixel removal, demosaicing, sensor color conversion, denoising, illuminant correction and other components (e.g., sharpening or hue enhancement). To optimize the rendered image, researchers designed and optimized the algorithms for each module and added new modules to handle different corner cases. The majority of commercial camera image processing pipelines consist of a collection of these specialized modules that are optimized for one color filter array design - Bayer pattern (one red, one blue and two green pixels in one repeating pattern).

New capabilities in optics and CMOS sensors have make it possible to design novel sensor architectures that promise to offer features that extend the original Bayer RGB sensor design. For example, recent years have produced a new generation of architectures to increase spatial resolution {Foveon; Langfelder}, control depth of field through light field camera designs {Lytro, Pelican; Light.co}, extend dynamic range and sensitivity by the use of novel arrangements of color filters {RGBW references} and mixed pixel architectures {Shree Nayar}. There is a need to define an efficient process for building image rendering pipelines that can be applied to each of the new designs.

In 2011, Lansel et al. proposed an image processing pipeline that efficiently combines several key modules into one computational step, and whose parameters can be optimized using automated learning methods [1,2,3]. This pipeline maps raw sensor values into display values using a set of local, linear and learned filters, and thus we refer to it as the L^3 method. The kernels for the L^3 pipeline can be optimized using simple statistical methods. The L^3 algorithm automates the design of key modules in the imaging pipeline for a given sensor and optics. The learning method can be applied to both Bayer and non-Bayer color filter arrays and to systems that use a variety of optics. We illustrated the method using both simulations [4] and real experimental data from a five-band camera prototype [5]. Computationally, the L^3 algorithm relies mainly on a large set of inner products, which can be efficient and low power. [http://on-demand.gputechconf.com/gtc/2015/video/S5251.html].

The L^3 algorithm is part of a broader literature that explores how to incorporate new optimization methods into the image processing pipeline. For example, Stork and Robinson developed a method for jointly designing the optics, sensor and image processing pipeline for an imaging system [2008, OSA Theoretical foundations for joint digital–optical analysis of electro-optical imaging systems]. Their optimization focused on the design parameters of the lens and sensor. Khabashi et al. [] propose using simulation methods and Regression Tree Fields [] to design critical portions of the image processing pipeline. Heide et al. [] have proposed that the image processing pipeline should be conceived of as a single, integrated computation that can be solved using modern optimization methods as an inverse problem. Instead of applying different heuristics for the separate stages of the traditional pipeline (demosaicing, denoising, color conversion), they rely on image priors and regularizers. Heide and colleagues [FlexISP; CVPR] use modern optimization methods and convolutional sparse coding to develop image pipelines as well as to address the more general image processing techniques, such as inpainting.

Here we identify two new applications of the L^3 pipeline. First, we show that the L^3 pipeline can learn to approximate other highly optimized image processing pipelines. We demonstrate this by comparing the L^3 pipeline with the rendering from a very high quality digital camera. Second, we show that the method can learn a pipeline that is created as the personal preferences of individual users. We demonstrate this by arranging for the L^3 pipeline to learn the transformations applied by a highly skilled photographer.

2. Proposed Method: Local Linear and Learned

In our previous work, we used image systems simulation to design a pipeline for novel camera architectures [4,5]. We created synthetic scenes and camera simulations to create sensor responses and the ideal rendered images. We used these matched pairs to define sensor response classes where the transformation from the sensor response to the desired rendered image could be well-approximated by an affine transformation. The L^3 parameters define the classes, C$_i$, and the transformations from the sensor data to the rendered output for each class, T$_i$.

We use the same L^3 principles to design an algorithm that learns the linear filters for each class from an existing pipeline. This application does not require camera simulations; instead, we can directly learn the L^3 parameters using the sensor output and corresponding rendered images. The rendered images can be those produced by the camera vendor, or they can be images generated by the user.

The proposed method consists of two independent modules: 1) learning local linear kernels from raw image and corresponding rendered RGB image 2) rendering new raw images into desired RGB output. The learning phase is conducted once for one camera model, and the kernels are stored for future rendering. The rendering process is efficient as it involves loading the class definitions and kernels and applying them to generate the output images.

2.1. Kernel Learning

In general, our task is to find for each class a $Px3$ linear transformation (kernel), T_i such that

$$argmin_{T_i} \sum_{j \in C_i} L(y_j, X_j T_i)$$

Here, X_j, y_j are the jth example data set from the RAW sensor data and the rendered RGB image values for class i. The function L specifies the loss function (visual error). In commercial imaging applications, the visual difference measure in CIE ΔE_{ab} can be a good choice for the loss function. In image processing applications, the transformation from sensor to rendered data is globally non-linear. But, as we show here the global transformation can be well approximated as an affine transform for appropriately defined classes C_i.

When the classes C_i are determined, the transforms can be solved for each class independently. The problem can be expressed in the form of ordinary least-squares. To avoid noise magnification in low light situations, we use ridge regression and regularize the kernel coefficients. That is

$$T_i = argmin \, ||\tilde{y} - XT_i||^2 + \lambda ||T_i||^2$$

Here, λ is the regularization parameter, and $\tilde{y}$ is the output in the target color space as a $Nx3$ matrix. The sensor data in each local patch is re-organized as rows in X. There are P columns, corresponding to the number of pixels in the sensor patch. The closed-form solution for this problem is given as

$$T_i = (X^T X + \lambda I)^{-1} X^T \tilde{y}$$

The computation of T_i can be further optimized by using singular vector decomposition (SVD) of X. That is, if we decompose $X = UDV^T$, we have

$$T_i = V * diag(\frac{D_j}{D_j+\lambda})U^T\tilde{y}$$

The regularization parameter (λ) is chosen to minimize the generalized cross-validation (GCV) error {http://www.stat.wisc.edu/~wahba/ftp1/oldie/golub.heath.wahba.pdf}. We performed these calculations using several different target color spaces, including both the CIELAB and sRGB representations.

2.2. Patch Classification

To solve the transforms T_i, the C_i classes must be defined. The essential requirement for choosing classes is that the sensor data in the class can be accurately transformed to the response space. This can always be achieved by increasing the number of classes (i.e., shrinking the size of the class). In our experience, it is possible to achieve good local linearity by defining classes according to their mean response level, contrast, and saturation. Mean channel response estimates the illuminance at the sensor and codes the noise. Contrast measures the local spatial variation, reflecting flat/texture property of the scene. Finally, saturation type checks for the case in which some of the channels no longer provide useful information. It is particularly important to separate classes with channel saturation.

2.3. Image Rendering

The L^3 rendering process is shown in Fig 1. Each pixel in the sensor image is classified using the same criteria as in the training module. We then apply the appropriate linear transformation, T_i, to the data in the P pixels in the patch surrounding the pixel. This linear transform computes the rendered pixel value. Hence, the rendered values are a weighted sum of the sensor pixel and its neighbors. The kernel coefficients differ between classes.

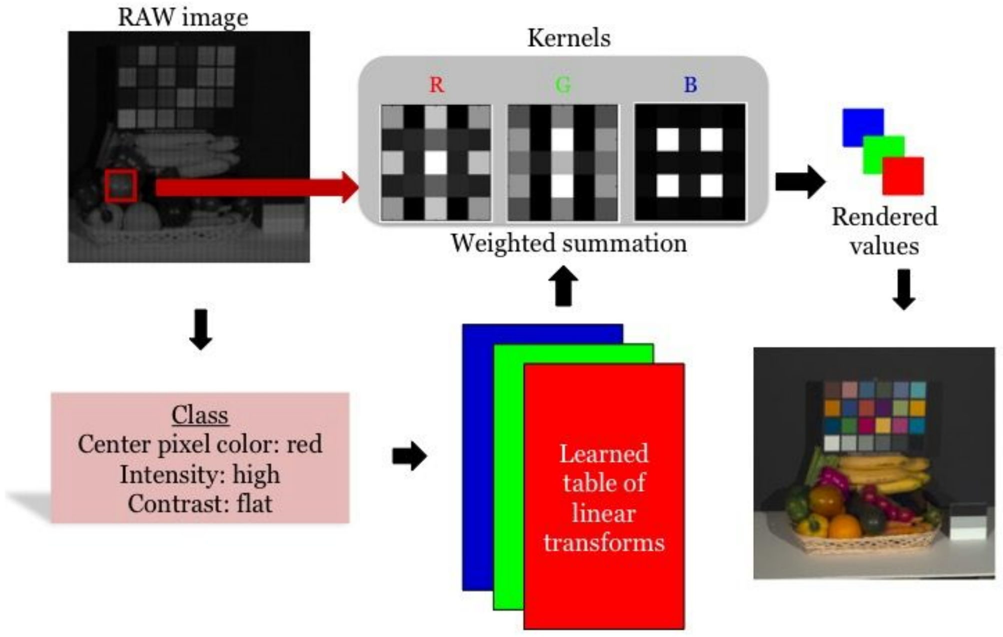

Fig 1. Overview of the L^3 processing pipeline. Class specific linear transforms are precomputed and stored in a table. Each captured sensor pixel is classified into one of many possible classes, and the appropriate linear transform is applied to the pixel and its neighborhood to render the data

This rendering process can be parallelized pixel-wise and performed relatively quickly. By using hundreds of processing units simultaneously, the rendering speed can be substantially accelerated (by orders of magnitude) compared to serial CPU computing. Fast rendering is important for applications that utilize unconventional CFAs, such as rendering high dynamic range videos captured in a single shot using novel CFAs.

3. Results and Discussion

3.1. Learning the kernels of an existing camera

We show how to learn and evaluate the kernels, T_i, of any camera that provides both Raw and rendered image data. Specifically, we solve for a set of L^3 kernels that approximate the rendering pipeline implemented by a camera vendor.

In one experiment, we use an image dataset from a Nikon D200 camera. The set includes 22 corresponding sensor and JPEG images of a variety of natural images. To perform the analysis, we first found the proper spatial alignment between the raw sensor data and the target output. The local linear kernels were estimated using data from 11 randomly selected images and then tested on the other half. Figure 2 (left) shows two rendered images, one produced by the camera image processing pipeline (top) and the other produced by an L^3 image processing pipeline (bottom). The L^3 pipeline used 200 classes and 5x5 kernels (P=25).

We assessed the accuracy of the color and spatial reproduction by calculating the S-CIELAB visual difference between the rendered images. To calculate the S-CIELAB errors we assumed the images are rendered and viewed on an LCD monitor that we calibrated. The ΔE_{ab} error image (right) is typical of all the ones in our set: the mean S-CIELAB ΔE_{ab} value is 1.59, indicating that the general visual difference is very small for human observers. Thus, L^3 parameters can be found that approximates most locations in the image for this Nikon D200 camera.

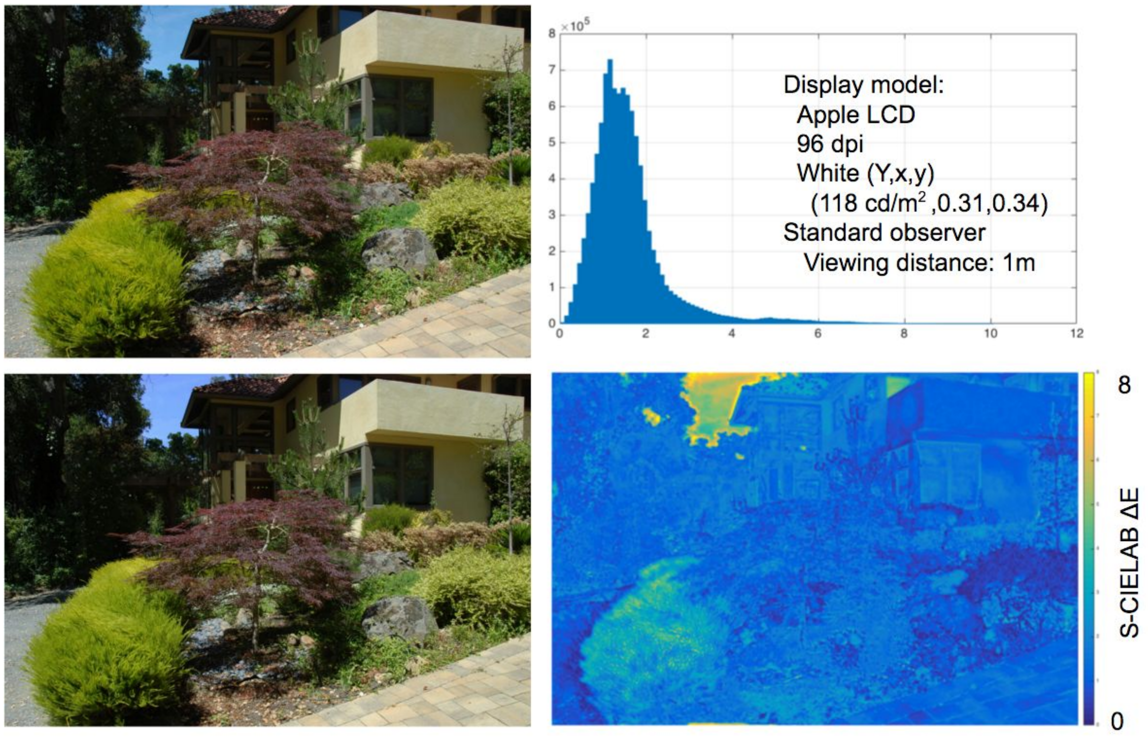

Fig 2. Comparison between camera RGB (upper left) and L^3 RGB rendered with local linear filters (lower left). The image at the lower right shows the S-CIELAB ΔE_{ab} values for each pixel. The histogram of errors is shown on the upper right. The mean error is 1.59, the peak error is near 8, and the standard deviation of the ΔE_{ab} values is 0.9. These errors are typical for the 11 images in the independent test set. The full resolution images for the figures in this paper can be found at [URL].

There are some regions of the image where the camera pipeline and L^3 pipeline differ. In this image the locations with the largest visual differences are the blue sky and the bush in the lower left. The approximation becomes more precise as we include more training images and more classes.

3.2. Selecting the classes

When there is enough training data, the accuracy of the L^3 kernels can be improved by adding more classes. However, adding more classes increases the total size of stored kernels. Also, there is room for innovation in the class definitions, and different choices can have various impacts.

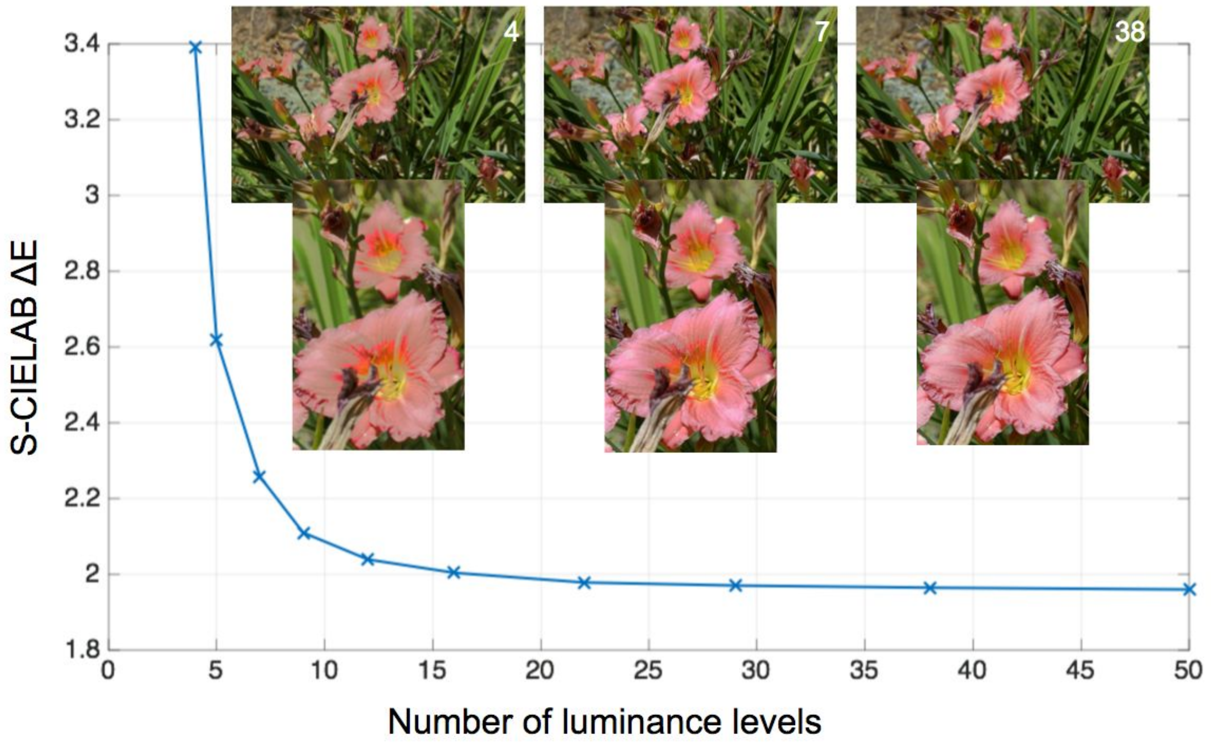

Fig 3. The selection of classes can have a large impact on the quality of rendered images. This graph shows images rendered with a small, medium, and large number of response level classes. In all cases the response levels are separated logarithmically. The flower (zoomed view) changes substantially as the number of levels increases, and the mean rendering error declines significantly as the number of classes increases from 4 to 15.

An important decision is to select the classes based on the sensor response levels. The noise characteristics of the pixel responses differ significantly at low and high sensor irradiance levels. The kernel solutions differ substantially as the sensor response level changes, and the rate of change is fastest at low sensor response levels. When the number of classes based on levels is small (4-5), the image is rendered incorrectly and there is frequently color banding(Figure 3). These effects gradually disappear as the number of classes based on response levels increases. In our experience, 15-20 luminance levels per channel is sufficient to reach a high quality rendering. Figure 3 quantifies this effect, showing that as the number of classes increases beyond 15, for this image the rendering does not significantly improve. We also find that it is efficient to use a logarithmic spacing of the luminance levels, so that there are many more levels at the low response levels than the high response levels.

For the Nikon D200 data, increasing the patch size does not improve performance. The mean S-CIELAB ΔE_{ab} value is 1.4611 when using 5x5 patches, and the mean ΔE_{ab} value is 1.4482 using 7x7 patches. Note that 7x7 almost doubles the computing complexity so that a small patch (5x5) is preferred.

We expect that the specific parameter values may differ for different optics and sensor combinations.

3.3. Learning individual preferences

We train and test on 26 pairs of raw camera images and RGB images created by our colleague David Cardinal, a professional photographer [http://www.cardinalphoto.com/] who rendered each image using his personal preferences (camera settings and post-capture rendering) . The images shown in Figure 4 were captured with a Nikon D600 camera. The collection of images includes several types of cameras and the content spans different types of natural scenes, human portraits, and scenic vistas.

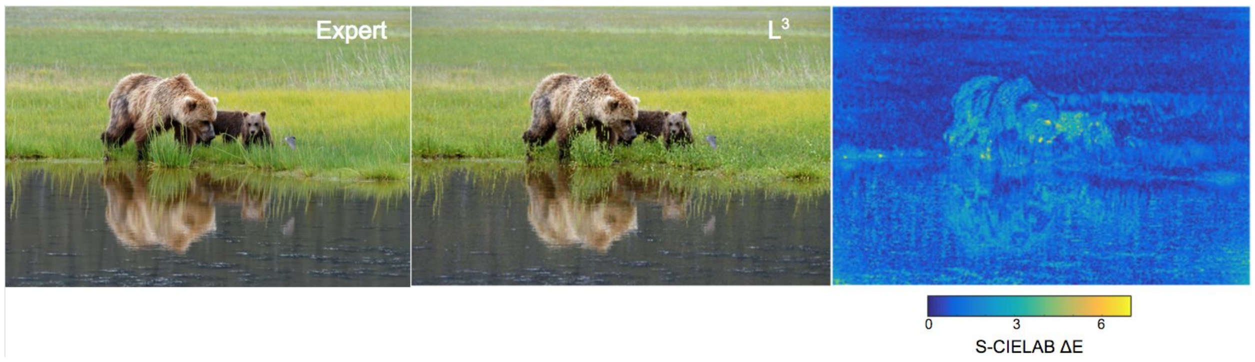

Fig 4. Left: images rendered from the raw sensor data by an expert photography. Middle: Rendering using local linear filters that approximate the rendering by the expert for this image. Right: The S-CIELAB ΔE_{ab} value for each pixel in the image. The full resolution images for the figures in this paper can be found at [URL]. See text for details.

Each of the individual images can be well-approximated by the L^3 method. Figure 4 shows a typical example of the expert's rendered RGB, the rendered RGB image with local linear filters, and the visual difference for each pixel. The mean S-CIELAB ΔE_{ab} value for this image is 1.458, and peak error is about 7, and the overall quality is similar to what we achieved for standard pipeline approximation for Bayer pattern.

As we analyzed the collection of images, from different cameras and different types of scenes, the cross-image validation does not always accurately capture the rendering. The expert's choice of rendering varies significantly as the scene types change, with some types of scenes giving rise to more choices for sharpening and others for a softer focus. Hence, there is no single set of kernels that summarizes the expert. Summarizing the performance of an expert would require capturing a number of different styles and then deciding which style would be best for an individual image. In this case, the value of the method is the ability to store and operate on the linear kernels to obtain different effects. Used in this way, the L^3 method can be designed to learn to approximate an individual user's preference in different contexts, say for outdoor scenes, indoor, portraits, and so forth.

4. Conclusion

The L^3 rendering pipeline is valuable in part because it is simple and compatible with the limited energy budget on low power devices. In some of these applications, it may be desirable to substitute complex image processing algorithms by a simple algorithm based on data classification and a table of local linear transformations. The simplicity arises from the reliance on tables of kernels that are learned in advance and the use of efficient and local linear transforms. The locality of the method, which is a form of kernel regression, does not require optimization algorithms or searches which can require extensive computation [Heide].

Simplicity is valuable, but the method must also be able to produce high quality renderings. Here, we demonstrate that the simple L3 method can closely approximate image processing pipeline of a high quality Nikon D200 camera with a Bayer CFA (Figure 2). In this case the L3 kernels that are estimated for specific camera settings generalize across images. Our analysis of the cross-validation error shows that the L^3 the kernels that are learned from examples of raw data and rendered images can be reused, though there are important considerations concerning the design of the classes and the ability to generalize (Figure 3).

We also find that L^3 can reproduce the transformation from raw sensor data to rendered RGB for individual pictures produced by a photographic expert (Figure 4). In this case, however, there is no clear way to generalize between different types of images and contexts (portraits, outdoor scenes, indoor scenes). We are exploring whether it is possible to find automatic ways to group images into categories and then apply the same kernels within these broader categories.

References

[1] Lansel, SP. and Wandell, BA., "Local linear learned image processing pipeline," Imaging Systems and Applications, Optical Society of America (2011).

[2] Lansel, SP., "Local Linear Learned Method for Image and Reflectance Estimation", PhD thesis, Stanford University, 2011.

[3] Lansel, SP. et al., "Learning of image processing pipeline for digital imaging devices," WO Patent 2,012,166,840, issued December 7, 2012.

[4] Tian, Q. et al., "Automating the design of image processing pipelines for novel color filter arrays: local, linear, learned (L^3) method," IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, 2014.

[5] Tian, Q., et al., "Automatically designing an image processing pipeline for a five-band camera prototype using the Local, Linear, Learned (L3) method", IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, 2015.

Additional references to include

Joint Demosaicing and Denoising via Learned Nonparametric Random Fields
Daniel Khashabi, Student Member, IEEE, Sebastian Nowozin, Jeremy Jancsary, Member, IEEE,
and Andrew W. Fitzgibbon, Senior Member, IEEE

GENERALIZED ASSORTED PIXEL CAMERA
SYSTEMS AND METHODS
Applicants
The Trustees of Columbia University
in the City of New York, New York, NY
(US); Sony Corporation, Tokyo (JP)
Inventors: Shree K. Nayar, New York, NY (US);
Fumihito Yasuma, Tokyo (JP); Tomoo
Mitsunaga, Kawasaki (JP)

TODO:
Evaluate of the accuracy of the linear fits within each class.
We should probably for ourselves calculate the PSNR just so we have a goofy number compare with other goofy people.
Make point in another paper about simulation solving the problem for the Purdue paper.

At the level of a high spatial resolution sensor (e.g., 1.2 um) with a typical (f#2.4 lens), do we ever get an edge or corner? Or is it the case that even at edges and corners in the original image, the optics spreads the light so that the sensor array sees something that is nearly planar (i.e. a slanted gradient of intensity). HJ points out that if you have a big patch (e.g., 50x50), sure you will have an edge. But if your demosaicking algorithm is applied to relatively small regions of the RAW image, say 5x5, then locally it will always seem like a smoothly changing field. **