# Refining raw pixel values using a value error model to drive texture synthesis

***Henry Gordon Dietz; Department of Electrical and Computer Engineering, University of Kentucky; Lexington, Kentucky***

## Abstract

*The goal in photography is generally the construction of a model of scene appearance. Unfortunately, statistical variations introduced by photon shot and other noise introduce errors in the raw value reported for each pixel sample. Rather than simply accepting those values as the best raw representation, the current work treats them as initial estimates of the correct values, and computes an error model for each pixel's value. The value error models for all pixels in an image are then used to drive a type of texture synthesis which refines the pixel value estimates, potentially increasing both accuracy and precision of each value. Each refined raw pixel value is synthesized from the value estimates of a plurality of pixels with overlapping error bounds and similar context within the same image. The error modeling and texture synthesis algorithms are implemented in and evaluated using KREMY (KentuckY Raw Error Modeler, pronounced "creamy"), a free software tool created for this purpose.*

## Introduction

Most image processing is based on the assumption that a pixel has a single, accurately known, value per color channel. If error in these values is modeled, it most often is modeled for the entire capture system as a final property. Instead, consider using an error model to guide an algorithm to enhance the apparent accuracy and precision of raw values.

In *Sony ARW2 compression: Artifacts and credible repair* [1], a model for error of each lossy-compressed raw pixel value in an ARW2 file was used to drive a raw repair algorithm using texture synthesis. The method was shown to very effectively remove compression artifacts, but that processing also seemed to provide some improvement in dynamic range and signal-to-noise ratio (SNR) even where there were no obvious compression artifacts. This can be seen in Figure 1, which shows a fragment of a Sony ILCE-7RM2 raw studio shot from DPReview[2] processed normally as an ARW2 and as a raw DNG file improved by KARWY. It was this observation that inspired the current work: is it generally possible to enhance even uncompressed raw sensor data using an error model to guide a texture synthesis algorithm?


***Figure 1.*** *Does KARWY enhance ARW2 dynamic range and SNR?*

### *Raw image data*

Although most digital cameras normally save images in heavily-processed formats based on the JPEG/JFIF standards, an increasing number of cameras offer the option of saving raw image sensor data. Here, the word "raw" does not denote one particular file format. For example, at this writing, `dcraw`[3] recognizes and can decode raw images in any of 689 somewhat different raw file formats. The word "raw" really refers to data being recorded from the sensor with minimal processing. The data is generally encoded with a roughly linear gamma, and has not been demosaiced to interpolate full color information to each pixel location. The idea is to postpone processing decisions so that the image data can later be transformed as flexibly as possible and rendered without loss of quality.

Most image enhancement processing is about improving the *rendering* of a captured image. In that context, it makes sense to accept raw image data as input. For rendering, there is no reason to modify the data in the raw file. However, altering the pixel data in the raw file is exactly what this paper proposes.

Most image sensors are inherently analog devices and any processing that happens in the analog domain, before analog to digital conversion, unavoidably embeds those processing decisions in the raw data. The resulting digitized raw data also is large, so saving it directly would limit the number of photos that can be stored on a memory card, slow file write times, and eventually limit the burst rate at which images can be captured. Thus, raw sensor data is commonly compressed before writing. This digital compression may be technically lossy or lossless, but it is useful to note that even conversion from analog to digital values is an inherently lossy process – and there are various other aspects of image sensing that introduce errors in the form of noise. The goal in this paper is computational removal of these corruptions in order to recover truer, or at least *more credibly true*, raw data.

### *Improving raw image data*

The current work does not mark the first time a tool has been built to improve raw data rather than its rendering. Some

tools have been built to repair defects in raws from specific camera models. For example, DeOrbIt[4] attempts credible repair of "white orb" defects produced by the FUJIFIM X10 camera and KARWY[1] attempts to undo the corruptions due to a specific type of lossy compression used by some Sony cameras.

There also have been tools produced to generically improve the raw data quality from many different types of camera. Although the DNG file format is usually presented as a more portable file format for raw image data, and Adobe Digital Negative Converter[5] appears to merely translate raw data into that file format, the tool actually does make changes to the raw pixel data values. Even more similar to the current work is RawImageClearer[6], which grew out of work on an Avisynth plugin called RemoveGrainHD; both use a "denoising box" to implement a type of smart median filter to reduce noise in the raw data.

Unfortunately, the goal of the current work is incompatible with median filtering. Although median filtering is capable of effectively reducing noise and therefore improving SNR, it cannot produce values that do not already exist in the image data. In effect, median filtering is imposing a type of posterization that improves SNR at the expense of reducing tonal quality. KREMY seeks to enhance the tonal quality while simultaneously reducing noise.

### *Stacking, error models, and texture synthesis*

Image stacking[7] is a process by which a series of registered images of the same scene are intelligently combined. Stacking can be used to enhance any of a variety of image properties; for example, stacking images captured with various focus distances can merge the sharpest portions from each image to produce a single result image with all distances in focus. Here, the primary goal is to enhance dynamic range and SNR, which can be accomplished by stacking using simple weighted averaging. This averaging need not introduce any blur as a side-effect of the processing, and, unlike median filtering, the number of distinct pixel tone values easily can increase by a factor of as much as $n$ when $n$ images are averaged.

The problem with stacking is that the input to KREMY is just a single image. Thus, instead of stacking images, the trick is to stack pixel values sampled from various places within the single original image that look similar – implementing a type of texture synthesis. The approach is conceptually similar to block-matching and filtering methods[8], but differs in that we are working on raw data and explicitly using an error model to control the synthesis. The raw value error model is not only used to determine which portions of the image are similar enough to be stacked, but also to derive weightings for averaging and to ensure that the final raw pixel values do not wander outside the error bounds obtained from the original image. Although the improved image might not be a more accurate representation of the scene, no raw pixel value is ever changed by more than the uncertainty to which its value was known; the pixel values are simply being adjusted to more credible positions within their error bars.

## Creation of a raw pixel value error model

Before any of the transformations of raw pixel values can be performed, it is necessary to create a model of raw pixel value error. It does not matter what this error comes from:

- Photon shot noise
- Analog sensor noise
- Imprecision in the analog-to-digital conversion
- Digital roundoff, colorspace conversions, and black point computations, etc.
- Lossy compression

All that is needed is a computational method by which a potential change of any particular raw pixel value can be judged viable or not.

### *The error model in KARWY*

In KARWY[1], the error model is dominated by accuracy loss due to the lossy compression scheme – which is obviously not the case for KREMY typically operating on raw values that were either left uncompressed or were losslessly compressed. For each pixel site, KARWY computes a minimum and maximum viable true value using intimate knowledge of the ARW2 raw file compression algorithm. Linear 14-bit raw data is converted into 11-bit values using a five-linear-segment curve specified in the file metadata as thresholds at which the spacing between distinguished original values doubles. These 11-bit values are then lossy-compressed in 32-pixel blocks using a form of delta encoding. The deltas are encoded by storing precise 11-bit minimum and maximum values for pixels in the block and using a scaled 7-bit offset to specify the other pixel values in the block.

### *Empirically-determined error models*

In order for KREMY to be able to improve raw files coming from nearly any type of camera, the only practical approach is to create an error model directly from empirical measurements of raw image data.

There are many possible levels of detail for the error model, ranging from a single percentage noise value to calibrated per-pixel probability distributions for the true value given the raw value recorded for that pixel, the ISO setting used on the camera, temperature of the sensor, estimate of photon shot noise, etc. More detailed error models should lead to more accurate reconstructions of the true raw pixel values, but the most detailed are impractical to construct and use. KARWY was successful using an error model tracking simple min-max range bounds for the true pixel value at each pixel site in the image, so it seems reasonable for KREMY to also focus on models that use min-max bounds.

#### *Image stacking to create an error model*

The first approach tried for construction of the error model was one based on stacking of multiple raw calibration images. The error model itself takes the form of an array which is 4x65536x2, separately recording for each of 4 color channels, for each of $2^{16}$ possible 16-bit raw pixel values, the minimum and maximum true values that could have been encoded as the given raw value. The four color channels allow for different color

filter array designs, but two of the channels are G for Bayer RGB cameras. Similarly, most cameras produce raw data with fewer than 16 bits per pixel, but having the model note that some values never occur is harmless.

A C program was written to perform the stacking of two or more uncompressed DNG raw files to compute this pixel value error model, and it operates as follows:

1. Set the camera to the same exposure parameters that will be used for the images to be improved and capture multiple, perfectly aligned, raw images of a constant scene. The full range of tones should be present in the scene.
2. Using the `-u` command line option of Adobe Digital Negative Converter, each of the images is converted into an uncompressed raw DNG file. This results in the raw data being a contiguous block of 16-bit unsigned short integer values in the DNG raw file.
3. The stacking does not merge the images, but rather constructs min-max records representing the range of 16-bit values seen at each pixel location. For example, if a particular pixel has the value 1024 in the first image, 942 in the second, and 984 in the third, the record would summarize this as having the range 942..1024.
4. Each min-max record is used to "vote" for min and max values in the range it spans. For example, if the 942..1024 record was for color channel 1, then a vote for 942 as the min and a vote for 1024 as the max would be registered for every value of channel 1 between 942 and 1024. For each possible pixel value, a sorted list of possible {min, count} and {max, count} tuples is maintained and each vote simply increments the count. To keep the data structure from becoming too large, only a fixed maximum number of vote tuples are kept for each combination of channel and raw value; when that limit is exceeded, the innermost-valued tuple is discarded to make space.
5. Once all pixel locations have been processed, the tuples for each combination of channel and value are reduced to single min and max values by summing votes working inward until sufficient votes have been counted to ensure corrupted pixels are ignored. Those min and max values are recorded, along with a confidence value determined from the counts.
6. Although typical images contain far more pixel sites than there are combinations of channel number and pixel value, it is common that many combination of channel number and pixel value simply do not occur in a given image. This can cause some table entries to be uninitialized. More significantly, the method used in step 4 does not necessarily produce a smooth, nor even monotonic, sequence of min and max values as pixel value is varied within a color channel. For example, channel 1's max for 942 might be 1024, but channel 1's max for 941 might be 1030 because it came from a different set of pixel value observations. Thus, multiple passes are made over the data to interpolate the min and max values to make smooth, monotonic, sequences. The interpolation processing is linear, but is biased to favor higher confidence values.

The hope was that the very expensive processing above would produce very high quality error models, and that those could then be used to judge the quality of error models created by simpler methods. However, high numbers of apparently corrupted pixel values compromised the results. What caused so many pixel values to be corrupted? We had underestimated the difficulty in maintaining perfect pixel-level alignment with high-resolution cameras that contain moving mechanical parts, especially using certain lenses that contain optical image stabilization. Using a color-checker target (which has large areas of consistent value), and selecting the best-aligned few shots of many captured, produced more consistent results... but the complexity of this process was clearly impractical.

### *Error model based on standard deviation*

Fundamentally, uncorrelated value errors behave like noise. Thus, conventional methods for describing noise should be applicable to construction of an error model. Instead of stacking multiple images, standard deviations were computed on patches within a single image.

The big problem with computing noise using standard deviations is that they should only be computed on regions that have a consistent shading, but it is unclear what such regions are in a noisy image. It is reasonable to assume that, with the exception of stars in night sky images, a true pixel value generally will be similar to the true value of at least one of its neighbors. Indeed, the anti-aliasing filters used in most digital cameras ensure a certain level of similarity independent of the scene. Thus, the idea is to use how the standard deviation changes as patch size is changed to determine the appropriate patch size.

C code was written that determined bounds for each individual pixel site by computing the standard deviation on a block of 9 pixels and then incrementally removing the most outlying pixel from that block until only a few pixels remained or the outlying pixel was within 2 standard deviations. Unfortunately, the ranges set by this approach, and by other methods based on standard deviations, produced inconsistent results.

### *Error model based on similarity ranges*

A very simple way to define error is that it is the difference between things that were supposed to have the same value. Of course, merely examining a single image, it is impossible to know which pixel sites were supposed to have the same value. However, most scenes have various regions in which the scene appearance is constant – evenly-shaded patches. Within each such patch, pixels that were supposed to have the same value should have *similar* values even in the presence of error – error should be relatively small. Further, measuring that small difference should provide an estimate of the error.

Of course, large portions of most images are not evenly shaded. Regions of an image that have relatively large differences between same-channel nearby pixel values tell us nothing about the error because the true pixel values probably differed by an unknown and significant amount. These regions are simply ignored when constructing the raw value error model.

Clearly, an error model produced by examining only the properties of contiguous blocks of pixels that have similar values cannot produce a different model for each pixel site. Thus,

the error model produced is essentially of the same form as was produced by image stacking: an array which is 4x65536x2, separately recording for each of 4 color channels, for each of $2^{16}$ possible 16-bit raw pixel values, the minimum and maximum true values that could have been encoded as the given raw value.

Crude filtering (not a full standard deviation computation) was used to identify regions that were likely to have resulted from an evenly-shaded portion of the scene. Each such region was used to create a min-max range estimate, which was then processed very much like the min-max records obtained using image stacking, but with a simpler voting procedure. Values missing min-max data were given interpolated min-max values and two additional passes forced the min and max values to be monotonic. This simple approach was the most effective one tried, and is the current implementation in the C-coded KREMY.

## The complete enhancement algorithm

The first step in using KREMY is to use logic taken from dcraw to open an uncompressed raw DNG file and obtain a pointer to the raw image data so that it can be examined and revised. However, that presupposes that the raw image is provided in DNG format – an option few cameras provide. Thus, Adobe Digital Negative Converter[5] is used to convert a native raw into an uncompressed DNG with appropriate metadata.

Once the raw data from the DNG is accessible, similarity range analysis is applied to create the error model. The resulting model provides minimum and maximum bounds for each possible 16-bit value.

### *Smoothing*

To reduce the impact of specific types of visual artifacts, KARWY performs two types of stochastic smoothing operations on the the pixel estimates. The first is optional, with adjustable strength. It attempts to identify pixels that had initial values that are almost certainly far from their correct value, and for each substitutes an initial value that would yield smoother local shading. This substitution does not necessarily blur details; it basically biases the texture synthesis to favor a smoother tonal transition. A second type of smoothing is always performed by KARWY. It attempts to recognize and break-up "Blondie" parallel-line artifacts which are specific to Sony's lossy ARW2 compression. Neither of these smoothing methods are used in KREMY; in fact, no smoothing is applied per se.

### *Texture Synthesis*

Texture synthesis is the process of creating a texture that "looks like it belongs" and inserting it in an image. Normally, the unit of synthesis is one or more pixels, as was the case in the texture synthesis we implemented in DeOrbIt[4]. In contrast, here, as in KARWY[1], we only seek to synthesize refinements of pixel values. Thus, the goal is to synthesize the most credible texture while keeping all pixel values within their computed error bounds.

It is important to note that texture synthesis is not smoothing nor is it blurring the image. By definition, texture synthesis is producing patterns – logically the opposite of smoothing. In fact, KREMY's texture synthesis may enhance edges at the same time as reducing noise (increasing SNR). The reason is simple: texture synthesis is replacing random noise with more highly correlated patterns. Both sharp edges and smoothly-shaded regions are highly correlated patterns of pixel values.

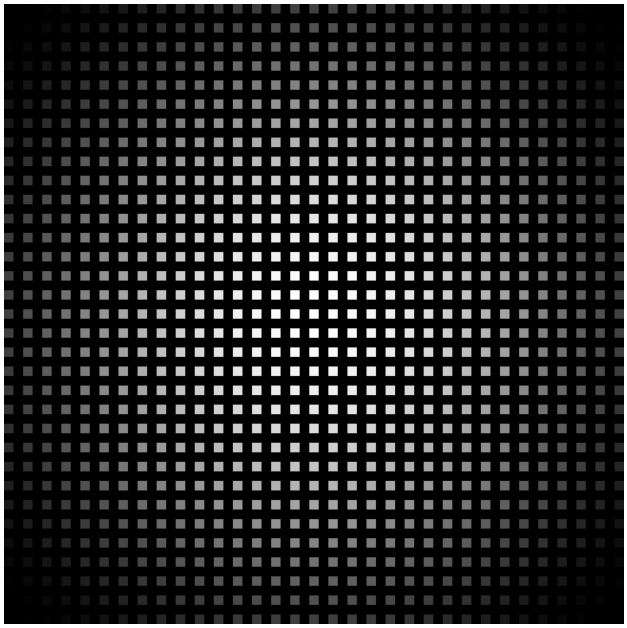

***Figure 2.*** *Spiral weighting pattern for texture synthesis*

The texture synthesis algorithm used in KREMY is very closely related to that used in KARWY[1]. Although many texture synthesis methods merely copy pixel values, the methods used in both KARWY and KREMY compute improved raw pixel values using weighted averages of values of other pixels that appear in a similar context within the image. Both also determine similarity by applying an error model. As was suggested earlier in this paper, the primary difference between the two tools is in the error model used.

### *Color filter array issues*

There is a second significant difference between the texture synthesis models used in KARWY and KREMY: all Sony cameras using the ARW2 raw format that KARWY processes incorporate very similar Bayer color filter arrays. While most digital cameras use color filter arrays, they do not all have the same colors or pattern.

Classic Bayer patterns repeat a 2x2 block pattern of Red, Green, Blue, and Green. However, the Green pixels in Red/Green rows are often slightly different from those in Blue/Green rows. In dcraw[3], this mismatch is managed using a command-line option to force separate interpretation of two Green channels. Several 2x2 patterns explicitly using four-color filters also have been proposed:

- Cyan, Yellow, Green, Magenta – Canon PowerShot G1
- Red, Green, Blue, and Emerald (Cyan) – Sony F828
- Red, Green, Blue, and White (Clear)

KREMY supports all 2x2 patterns by always treating each

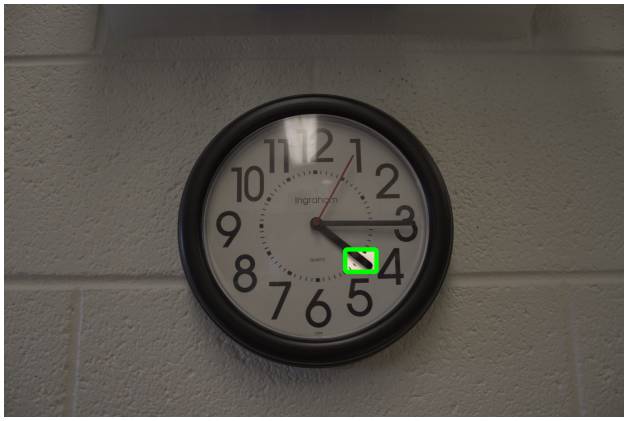

***Figure 3.*** *Canon EOS Digital Rebel XT @ ISO 100 showing crop*

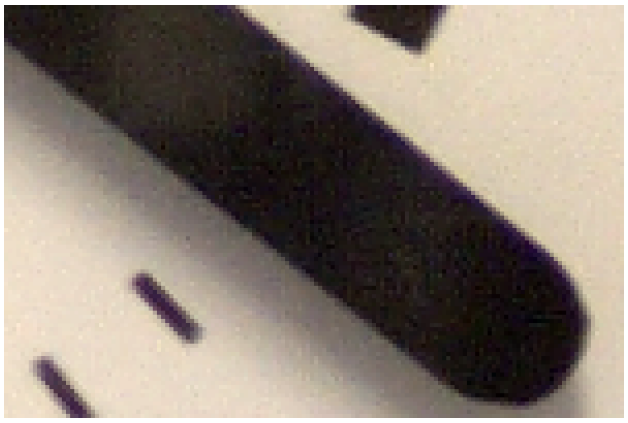

***Figure 4.*** *150x100 pixel crop, normally processed raw*

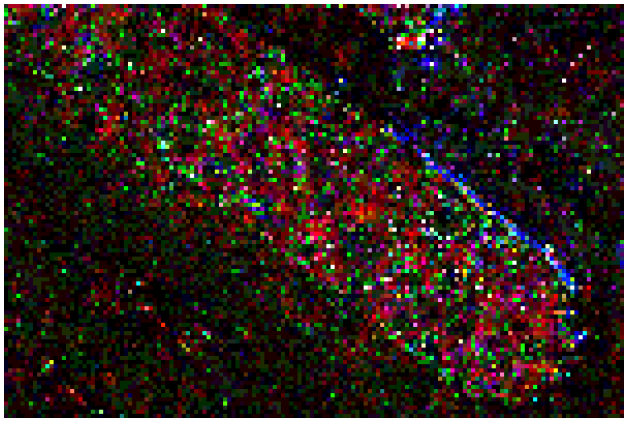

***Figure 5.*** *150x100 pixel crop, enhanced KREMY vs. raw*

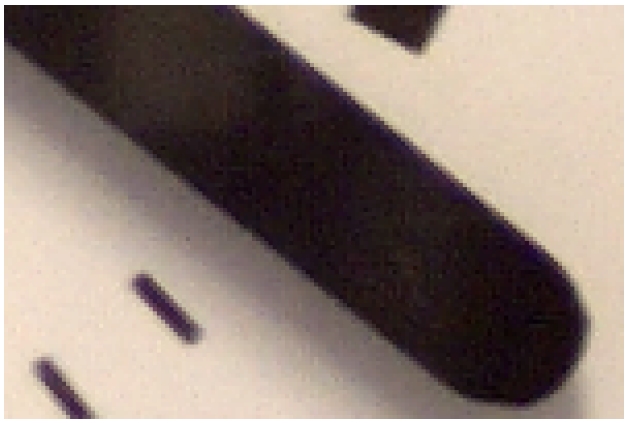

***Figure 6.*** *150x100 pixel crop, KREMY-improved raw*

position in the 2x2 pattern as a separate color channel. This potentially misses opportunities for improving Green data, but makes it unnecessary for KREMY to be told what colors are in each position. This "colorblind" four-channel treatment is not sufficient to handle the few cameras that use larger repeating blocks (e.g., the Fujifilm X-Pro1) nor in systems that layer colors (e.g., the Foveon X3 sensor in the Sigma DP2), but many of those cameras are also problematic for DNG raw encoding.

### *Spiral search for similar pixels*

In KREMY, as in KARWY, the texture synthesis is based on searching for similar pixels in a 1089-position spiralling-out sequence centered at the pixel in question, with positions outside the image boundaries ignored. However, KARWY does this search only for pixels with ambiguous values, while KREMY tries to improve all pixel values.

The improved value of each pixel is a weighted average of the values of similar pixels found. The similarity weighting for a match is the product of:

- the similarity of a 3x3 block of pixels around the search candidate to those around the pixel to be improved. Similarity is zero if any of the nine corresponding pixels has a value which is not within the computed value error bounds. Otherwise, similarity of each pixel is scored as 1 minus the square of the fraction of the error bound that the pixel is distant from the reference pixel value. Weighting by the product of the eight neighbors is the primary reason that credible textures are synthesized. Note that the only one of the nine pixels coming from the same color channel as the pixel being improved is the one in the center.
- the base distance weighting for that position in the spiral plus 1. The base distance weightings in the spiral decrease with distance as shown in Figure 2, which is the same pattern used by KARWY[1].

Each similar pixel's value is multiplied by the similarity and added to a sum, which is finally divided by the sum of the similarities to get the new pixel value. To ensure stable behavior when no similar pixels are found, the sum is initialized with the original pixel value. To speed the processing, the search may also be truncated after the similarity sum has reached a value (currently 64) indicating high confidence in the improved value.

### *Final processing*

After the pixel values have been updated by texture synthesis, the data may be saved back into the DNG file. However, there are two types of minor flaws that can be introduced by the fact

***Figure 7.*** *Canon PowerShot S70 @ ISO 50 showing crop*

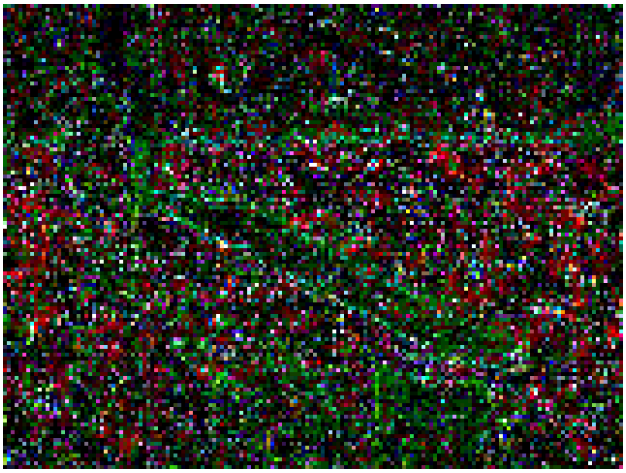

***Figure 9.*** *160x120 pixel crop, enhanced KREMY vs. raw*

***Figure 8.*** *160x120 pixel crop, normally processed raw*

***Figure 10.*** *160x120 pixel crop, KREMY-improved raw*

that each update of a pixel can skew the properties of the image.

The first correction involves the average brightness of the image. Changes may have made the image brighter or darker than it was before. A simple pass is made over the image comparing brightness of each spot to the original brightness in that spot. If necessary, the pixel's value is bumped up or down to keep the overall brightness from changing dramatically.

The second correction involves local contrast. If a pixel value change increased local contrast, this single pass attempts to reduce contrast back to the original levels.

These final processing steps help avoid introducing blotchy patterns, but they can be omitted with minimal harm.

## Results

To experimentally evaluate KREMY, raw files from a wide range of cameras were collected and processed with the tool. In all cases, KREMY was run without any parameters specified; the enhancement was determined completely automatically based on KREMY's analysis of the uncompressed DNG input.

Of course, raw files need to be rendered to be shown in this paper. For that purpose, dcraw or rawtherapy (set to "neutral") were used to render the images. After cropping (and scaling by converting each pixel into an 8x8 block so that JPEG rendering within a PDF would not alter it), some of the images were given simple level adjustments to make differences more visible.

### *Base ISO APS-C Bayer DSLR example*

Figure 3 shows a raw photo captured using an APS-C sensor format Bayer-filtered DSLR, the Canon EOS Digital Rebel XT, and the crop area marked in green is shown in Figure 4. This image was taken in August 2006 and we no longer have this camera, so it would not be possible to capture additional shots to create an error model by stacking. Further, the image was captured at ISO 100, the very low-noise base ISO of the camera. Yet, as the crop from the KREMY-improved raw in Figure 6 clearly shows, noise is reduced without introducing any obvious artifacts. Note that the images presented in this paper have all been rendered with just 8 bits per color channel, so it should be a little surprising that noise is visible, and visibly reduced by KREMY, for a camera that used a 12-bit analog-to-digital converter. The difference be-

***Figure 11.*** *Sony NEX-7 @ ISO 1600 showing two crops*

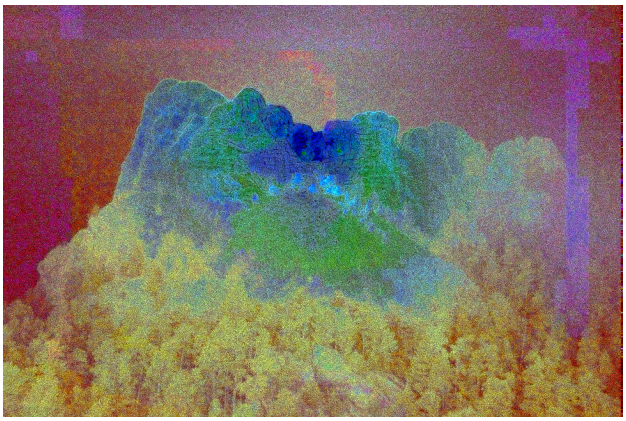

***Figure 14.*** *Full image, enhanced KREMY vs. raw*

***Figure 12.*** *150x100 pixel crop, normally processed raw*

***Figure 15.*** *150x100 pixel crop, KREMY-improved raw*

***Figure 13.*** *150x100 pixel crop, normally processed raw*

***Figure 16.*** *150x100 pixel crop, KREMY-improved raw*

tween the original raw and KREMY-improved raw, auto-leveled to make the differences more visible, is shown in Figure 5.

### *Base ISO Compact Bayer example*

Raw image capture is not just available in DSLRs (digital single-lens reflex cameras), but has long been available in more compact cameras – especially those intended as more portable "second cameras" for serious photographers. Figure 7 shows a raw photo captured using the Bayer-filtered 1/1.8" CCD of a Canon PowerShot S70 in August 2005, and the crop area marked in green is shown in Figure 8. Again, this image was captured at the camera's lowest available ISO 50, but there is still significant noise visible. Figure 10 shows the dramatic improvement made by using KREMY to improve the raw. The auto-leveled

***Figure 17.*** *Olympus E-M1 Mark II @ ISO 400 crop, original raw*

***Figure 18.*** *Olympus E-M1 Mark II @ ISO 400 crop, KREMY-improved raw*

difference image in Figure 9 makes it clear that the transformation performed in texture synthesis is quite complex, with some obvious correlations, but overall there was much more random noise in this image than in the one from the Canon DSLR. A higher noise level is expected given the much smaller sensor, and correspondingly smaller pixels, used in the S70; however, the crop is also taken from a darker portion of the image, which increases the impact of photon shot noise and dark noise. Note that the edges in the scene (even the diffuse lines on the roof) are at least as well defined as they were in the original raw despite the smoother tonality of the KREMY-processed raw.

### *High ISO APS-C Bayer Mirrorless example*

With more modern higher-end cameras, the noise at low ISOs is generally not strong enough to visibly survive the processing required to format this paper for publication. However, at high ISOs in lighting requiring longer shutter speeds, noise still becomes apparent and even overpowering. Figure 11 shows an image of Mount Rushmore National Memorial taken just as the park was closing on a cloudy (dark and starless) night in June

***Figure 19.*** *Nikon D810 @ ISO 1600 eye crop: left raw, right KREMY*

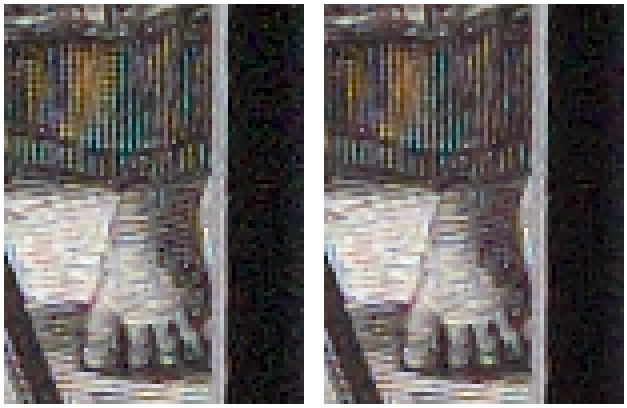

***Figure 20.*** *Nikon D810 @ ISO 1600 foot crop: left raw, right KREMY*

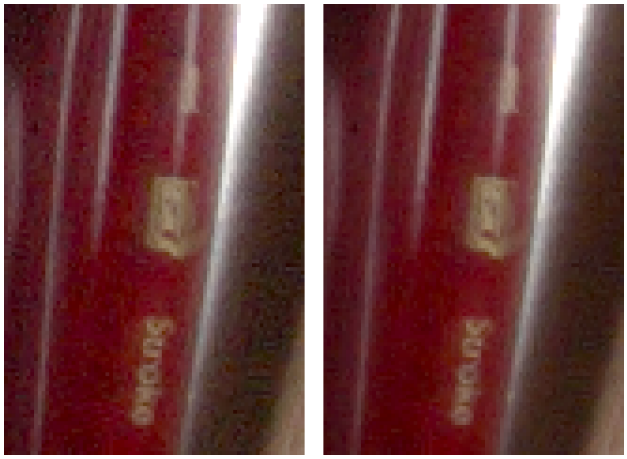

***Figure 21.*** *Apple iPhone 7 crop: left raw, right KREMY*

2012; the minimal lighting required the Sony NEX-7 to use a 0.3s exposure at ISO 1600 to capture detail in the carved faces. Both crops of this image are severely underexposed, yet KREMY provided an obvious improvement in both SNR and clarity of the shapes of objects in the scene. KREMY's transformation of the crop in Figure 12 to that in Figure 15 is quite effective, but the transformation of Figure 13 into Figure 16 is far more dramatic.

***Figure 22.*** *Canon PowerShot G1 @ ISO 50 crop: left raw, right KREMY*

***Figure 23.*** *Sony DSC-F828 @ ISO 64 crop: left raw, right KREMY*

The most disturbing noise in Figure 16 has a structure that is related to a flaw in Sony's lossy raw compression algorithm – the "Blondie" parallel line artifacts described in the paper about KARWY[1]. KARWY directly uses the ARW compression data to recognize and repair these artifacts, while KREMY works off an uncompressed DNG. KREMY never sees the ARW compression data, so it does not recognize these structures as artifacts: they are not part of the error model – although the texture synthesis still manages to improve many of them. For example, the mild "color blocking" seen in Figure 14 is correcting blocking artifacts created by Sony's block compression algorithm despite KREMY having no direct knowledge that such a problem existed.

### *Medium ISO Micro Four Thirds Bayer example*

Micro Four Thirds and 1" sensor cameras are popular for packing lots of features into a smaller system, but they do suffer considerably more noise than larger sensors. The Micro Four Thirds Olympus E-M1 Mark II produces high quality images, but even at ISO 400 there is plenty of room for improvement by KREMY. Figures 17 and 18 clearly demonstrate this with a 160x120 pixel crop from a raw studio shot posted by DPReview[2].

### *High ISO Full-Frame Bayer DSLR example*

The Nikon D810 offers image quality among the best of any full-frame Bayer DSLR. Figures 19 and 20 are 75x100 crops from an ISO 1600 raw studio shot posted by DPReview[2]. While both of these crops show significant reduction in noise with no loss is apparent sharpness, Figure 20 shows something unexpected: KREMY has partially repaired moire' that Bayer interpolation created due to below-Nyquist sampling of the vertical line texture. The repair is not always this effective, but at least partial repair is a natural consequence of texture synthesis because the texture contrast is enhanced, making smart Bayer interpolation algorithms less likely to enhance the wrong gradient. Note that the foot crop should really be a monochromatic image, but the false color cast comes from frequencies of the texture and pixel placement beating against each other – an artifact more appropriately repaired in rendering than by changing the raw data.

### *Base ISO Cell Phone example*

Over the last few years, it has become common for cell phones to offer raw capture. DPReview is now testing cell phone cameras in much the same ways that they test other cameras. Figure 21 shows a crop of a portion of a paintbrush from a DPReview raw studio shot[2] captured with an Apple iPhone 7 at base ISO. The tiny sensor produces significant noise, but KREMY's processing is fairly effective.

### *Base ISO Non-Bayer examples*

The Canon PowerShot G1 was one of the first compact "prosumer" cameras, and in 2000 it was one of the first compact cameras to support raw capture, but it did not use a Bayer filter. Instead, it used a color filter array based on the more transparent subtractive primaries: Cyan, Magenta, Green, and Yellow. Even so, the camera's base sensitivity is just ISO 50. Figure 22 shows an 80x120 crop of a goldfish surfacing. KREMY does improve it, but the fact that each raw value is stored in just 10 bits limits how smooth tonal transitions can be made.

The Sony DSC-F828 is a very unusual camera. Although it is not small, it uses a small 2/3" CCD to enable the equivalent of a 28-200mm f/2-2.8 zoom lens, lets the body freely tilt relative to the lens, provides the "NightShot" ability to remove the NIR-blocking filter, and uses a Red, Green, Blue, and Emerald color filter array instead of the usual Bayer pattern. As Figure 23, an 80x120 crop of the decoration on a Turkish plate, shows, KREMY works just as well to improve this non-Bayer raw data as it does to improve raw data from Bayer sensors.

### *Quantifying improvement via random noise*

Several different methods were used to try to determine how much the SNR and dynamic range were being improved. The obvious method, direct measurement of SNR, did not correlate well with human perception – probably because the texture synthesis process actually enhances textural edges while it is reducing noise, leaving very little change in the contrast that SNR measures. Thus, instead of numerically measuring SNR, known amounts of noise were added to an image and quality of the various raw and KREMY-improved raws were compared.

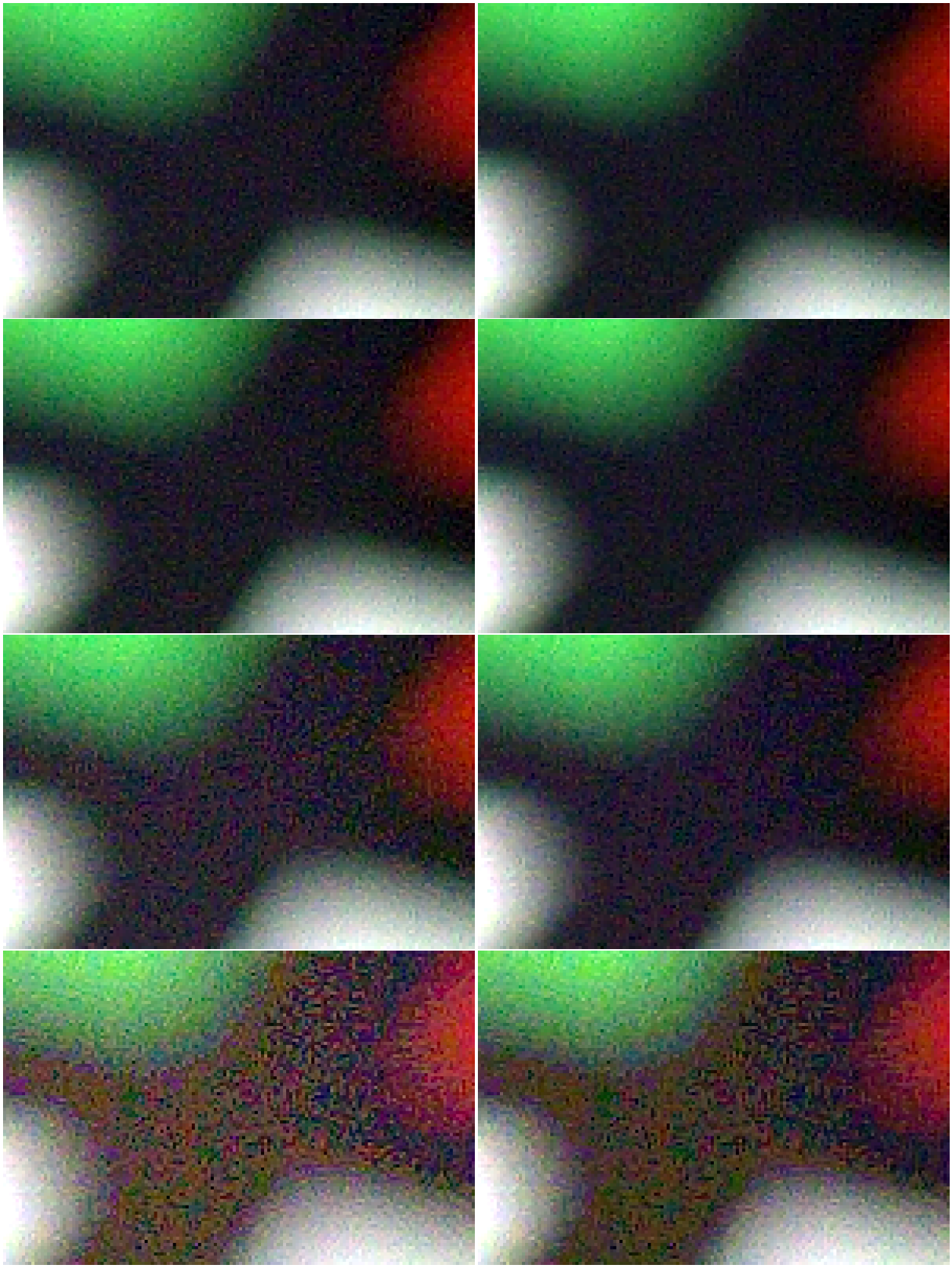

***Figure 24.*** *150x100 pixel crops from top to bottom with 0, 4, 5, and 6 LSBs randomized; left original, right by KREMY*

***Figure 25.*** *Canon 5D Mark IV @ ISO 250 showing crop region*

The original reference image, shown in Figure 25 (with a 150x100 pixel crop area marked in green), was captured using a Canon 5D Mark IV at ISO 250 – an ISO at which there is a very small level of noise present. Noise was added to the original raw by replacing the least-significant bits (LSBs) of each pixel value with a random bit pattern. The test images created this way had from 0 to 8 LSBs destroyed. The improvement in SNR and dynamic range can thus be determined by selecting which KREMY-improved images looked at least as good as the original.

Figure 24 shows raw crops of the original on the left and the corresponding KREMY-improved crop on the right. For space reasons, only the versions with 0, 4, 5, and 6 LSBs randomized are shown in the figure.

It is clear that the right side images are generally better than the left side ones, but the surprising fact is that randomizing up to four bits was not quite sufficient to make the KREMY image appear noisier. That would suggest that SNR and dynamic range are being improved by at least several stops. However, also note that there is no magic in KREMY; some image files simply do not have enough data allow recovery, and it is fairly clear that randomizing 6 bits in each 14-bit pixel reading is removing too much data.

## Conclusion

A year ago, KARWY proved that the unfortunate artifacts caused by Sony's ARW2 lossy compression could be very credibly repaired in raw image data by a novel type of computational texture synthesis. This texture synthesis was novel primarily in that it did not seek to create new pixel values, but to incrementally enhance the accuracy of pixel values while keeping them within error bounds computed based on how the lossy compression may have caused pixels to diverge from their true values.

The work described in the current paper, KREMY, extends this concept to attempt to incrementally enhance all pixel values for a much wider range of cameras. It is much more difficult to create a good error model for KREMY because, unlike analysis of lossy-compressed ARW2 files, the source of the error is not known when examining only an uncompressed raw DNG file. However, viable methods for empirical creation of an error model were devised and implemented.

With relatively straightforward modifications, a subset of the algorithms used in KARWY were able to apply the error model to synthesize credible enhancements for uncompressed DNG raws from a wide range of cameras. An improvement in dynamic range and SNR is apparent, however it was difficult to measure because the processing generaly enhances correlations: evenly-shaded areas become more even, but other structures also become more defined. Replacing the low bits of raw data with random noise revealed that an improved image was comparable to an original with 4 more bits in each pixel value – a shockingly large improvement.

Despite being written in C, KREMY is currently too slow for use via a WWW form (as was done for KARWY) or to be applied to all images as part of the default raw processing pipeline. Future work centers on increasing KREMY's speed and adding a mechanism allowing user control of how aggressive the enhancement processing should be.

## Acknowledgments

This work is supported in part under NSF Award #1422811, *CSR: Small: Computational Support for Time Domain Continuous Imaging.*

## References

[1] Henry Gordon Dietz and Paul Selegue Eberhart, *Sony ARW2 Compression: Artifacts And Credible Repair*, Electronic Imaging 2016, Visual Information Processing and Communication VII, pp. 1-10 (February 14, 2016); doi:10.2352/ISSN.2470-1173.2016.2.VIPC-227

[2] Digital Photography Review (DPReview) Studio shot comparison, *https://www.dpreview.com/reviews/image-comparison* (2017).

[3] Dave Coffin, Decoding raw digital photos in Linux, *http://www.cybercom.net/d̃coffin/dcraw/* (2016).

[4] Henry Gordon Dietz, *FUJIFILM X10 white orbs and DeOrbIt*, Proc. SPIE Electronic Imaging 2013, Digital Photography IX, 866005 (February 4, 2013); doi:10.1117/12.2004411 (2013).

[5] Adobe Digital Negative Converter 9.x, *https://helpx.adobe.com/photoshop/using/adobe-dng-converter.html* (2016).

[6] Rainer Wittmann, *A non-blurring denoiser of raw digital images*, (March 15, 2009); http://home.arcor.de/kassandro/RawImageClearer/ (accessed 2016).

[7] Deep Sky Stacker, *http://deepskystacker.free.fr/* (accessed November 26, 2016).

[8] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian, *Image denoising with block-matching and 3D filtering*, Proc. SPIE 6064, Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning, 606414 (February 17, 2006); doi:10.1117/12.643267

## Author Biography

*Henry (Hank) Dietz earned his PhD at Polytechnic University and joined the faculty at Purdue University's School of Electrical and Computer Engineering in 1986. Since 1999, he has been a Professor and Hardymon Chair at the University of Kentucky. Although best known for his work in compilers and parallel supercomputing using Linux PCs, his current focus is on improving cameras as computing systems.*