An improved raw image enhancement algorithm using a statistical model for pixel value error

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

Abstract

When an image is captured using an electronic sensor, statistical variations introduced by photon shot and other noise introduce errors in the raw value reported for each pixel sample. Earlier work found that modest improvements in raw image data quality reliably could be obtained by using empirically- determined pixel value error bounds to constrain texture synthesis. However, the prototype software implementation, KREMY (KentuckY Raw Error Modeler, pronounced "creamy"), was not effective in processing very noisy images. In comparison, the current work has reimplemented KREMY to make it capable of credibly improving far noisier raw DNG images. The key is a new approach that uses a simpler, but statistical, model for pixel value errors rather than simple bounds constraints.

Introduction

In normal use, the purpose of a camera is to create an accurate model of scene appearance – which is primarily determined by the physical properties of the objects in the scene. However, a camera is limited to using light from the scene to sample the appearance. Thus, it seems natural to think of accurately measuring photons as the key to producing a higher quality model of scene appearance. The catch is that photon emission from any light source is subject to statistical variations over short periods of time: photon shot noise. Even if the sensor perfectly records every photon from the scene, that data does not accurately model the true "average" appearance of the scene, which is what human observers perceive.

Photon shot noise is not the only way in which the pixel values can diverge from the ideal. There are a variety of potential sources of noise within the camera and its electronics. Precisely characterizing each of the separate noise sources is generally difficult and often impractical for commercially-available cameras. The complexity is multiplied by the fact that noise depends on diverse factors including the sensitivity to light selected (the ISO setting) and the current operating temperature of the camera. Thus, the current work takes a pragmatic approach: measuring the combined effect and generating a simplified probability-based pixel value error model, as detailed in Section .

Earlier work [1][2] has demonstrated that using an appropriate model of noise to drive "credible repair" of a raw image can be very effective. Texture synthesis was the mechanism used to refine raw pixel values within computed error bounds. While results were excellent for tight bounds, for noisier images there are more instances of dramatically wrong pixel values. Bounds that cover those outliers cause texture synthesis to fail, but using tighter bounds causes wrong pixel values to be passed unchanged. In contrast, the current work does not refine pixel values within bounds, but replaces each pixel value with a new value determined using probabilistic texture matches. The new values are usually close to the originals, but can be arbitrarily different, so noisier images can be more effectively improved.

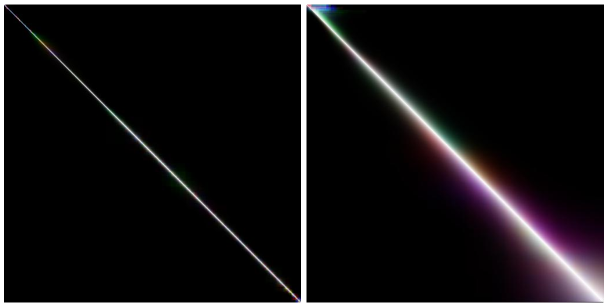
Figure 1. Sample pixel value error models, low and high noise

The following section describes the pixel value error model implemented in the new version of KREMY (KentuckY Raw Error Modeler). Section discusses texture synthesis using that error model. The improved raw images are compared to those produced by the original KREMY in Section .

Pixel Value Error Models

In 2015, texture synthesis driven by a pixel value error model was employed in KARWY[1] to credibly repair compression artifacts associated with the lossy compression scheme used in Sony ARW raw image files. KARWY separately computed error bounds for each individual pixel by essentially reversing the compression arithmetic to compute the range of true values that could result in the observed value. For lossy-compressed ARW files, the bounds are not only due to an approximation to log compression (which is commonly used in various camera raw formats), but also due to a clever encoding scheme that scales values based on the set of all pixel values within the same 32-pixel block. The result is bounds that very precisely model errors due to ARW compression, but not other noise sources, such as photon shot noise. Thus, KARWY's model is both specific to a particular raw format and insufficient as a model of total noise.

The most direct way to measure noise from all sources is to physically fix the camera and capture a sequence of exposures of the exact same test scene under essentially identical exposure settings and lighting conditions. For each pixel, histogramming the values obtained over the sequence of c captures defines a probability density function (PDF) for the value of that pixel. This approach was first used in TIK[3] for the very different purpose of converting time-sequenced images into time domain continuous per-pixel waveforms. TIK also encoded the error model in much the same way used here, although the captures for TIK were not raw images but a sequence of full color frames from a video of a completely static test scene. This type of analysis does not unambiguously determine the true value a pixel should have, but it is a reasonable and common assumption that, given c sample readings r_i, the average value approaches the true value v:

$$v = \sum_{i=1}^{c} \frac{r_i}{c}$$

Thus, bounds for each ideal value v are simply the minimum and maximum pixel values r_i that were summed to produce each v.

A variety of methods for constructing a pixel value error model were considered in developing the original version of KREMY[2], all still targeting computation of bounds on the true value of image pixels. The final method implemented was based on identifying evenly-shaded patches. Variation of pixel values within each such patch was used to define minimum and maximum bounds on the true value v. Portions of the image not identified as evenly-shaded were ignored in creating the model, and there is no guarantee that error bounds will be computed for all pixel values occurring in the image. Thus, pixel values without computed bounds have their minimum and maximum bounds interpolated from bounds on other values using two smoothing passes to ensure bounds change monotonically with changes in the observed pixel value. Empirically, this type of bounds-per-measured-value model becomes significantly less effective as image noise increases.

The key to good handling of high noise levels seems to be directly addressing the statistical nature of noise. As an extreme example, the probability that a pixel that should be white will be seen as black due to photon shot noise is not high, but neither is it zero. Put more philosophically, extreme alterations of observed pixel values should be possible, but should require extreme evidence that the change is appropriate. A probabilistic model can implement this type of weighting. The current work thus employs a pixel value error model that associates probabilities with each possible pair of observed vs. true pixel values.

A Probabilistic Error Model

The error model construction used in the new KREMY is based on computing a *probability density function* (PDF). If all r_i involved in producing each value v are histogrammed, simply dividing by the number of samples yields a PDF for v.

Although the computed v values may be of arbitrary precision, it is reasonable to map v values into integers of precision no greater than the precision of a pixel sample r_i. If both the pixel values and probabilities are encoded with b-bit precision, the complete set of PDFs for all v can be expressed as a matrix of $2^b \times 2^b$ entries. This probability density map becomes an error model expressed as a simple square image in which position x, y encodes the probability that a value $v = y$ was read as x. To maintain accuracy, the values for each y are treated separately, so the error model values are scaled so that the maximum probability x for each y has the full scale value. If the images contain Red, Green, and Blue color channels, a color square image can represent the superposition of the three separate error models, one for each color.

```
let m = maximum pixel value, (1<<b)-1;
let f = filter threshold for maximum noise;
for (c = each color channel)
  map[c][0..m][0..m] = 0; // clear color c map

  // accumulate histogram of neighbor ranges
  for (i = each image pixel of color c)
    let ri = value of pixel i scaled 0..m;
    let rj = value of most similar neighbor;
    let d = abs(ri - rj);
    // increment square region of map
    if (d < f) map[c][ri..rj][ri..rj] += 1;

  // normalize to make probabilities
  for (y = each row in map) {
    // assume diagonal is max probability
    let scale = m / map[c][y][y];
    for (x = each column in map)
      map[c][y][x] *= scale;
```

Figure 2. *Histogram-like algorithm to make probabilistic error model*

Figure 1 shows two such error models computed by the new KREMY. The first is from a very low noise capture, a crop of which is shown in Figure 4; the second is from the much noisier capture that produced the crops shown in Figures 13 and 14.

Probabilistic Model from a Single raw Capture

The problem with this type of error model is primarily that KREMY is intended to reduce noise in a single image without requiring a training image data set – nor a detailed model of how parameters like ISO setting and sensor temperature change the noise. In the current work, the error model is created from the single capture we are trying to enhance by applying two insights.

The first insight is that a pixel value error model describes the probability that a given value is observed for a specific true value. However, this also can be viewed as **describing the probability with which two different pixel samples, r_i and r_j, actually represent the same** v. This different perspective is significant because it removes the need to know the true value v – which cannot be directly computed using a single capture of an unknown scene.

The second insight is that, **under most circumstances, at least one same-color pixel near this pixel will differ from this pixel's value by little more than noise**. This observation does not really provide a way to estimate v, but it does give a pair of readings, r_i and r_j, that are fairly likely to represent the same v. Of course, this method tends to overestimate noise, but large errors only come from pixels with no similar values nearby – such as distant stars in a night sky – and a simple filtering process can reject recording a pair of readings when the difference is clearly too large to be caused by noise. This leaves only relatively subtle gradients to be confused with noise, and the impact of being one pixel off in a subtle gradient is rarely significant.

Computation of the error model is thus based on analysis of all pairs of most-similar neighbor values, r_i and r_j, to construct the three-dimensional probability density function `map`; the basic algorithm is shown in Figure 2. The histogram-like construction is unusual in that a conventional histogram would increment a single entry, whereas here if $\delta =$`abs(ri-rj)`, then $(\delta+1)^2$ `map` entries are incremented representing that fact that values in that range might really represent the same true value. Also note that the normalized probabilities for a row in the final `map` do not sum to 1; the probability ratios are preserved, but the highest probability is encoded as $(1 << b) - 1$ so the `map` can have the highest accuracy possible while being stored as a small integer lookup table – or as a PGM image per color channel.

Use of Raw Image Data

Higher-end consumer cameras typically not only provide for obtaining images in common export formats, particularly JPEG, but also in a proprietary raw format. These raw formats generally wrap the unprocessed sensor data in a Tag Image File Format (TIFF or TIF) wrapper. For example, Sony ARW, Nikon NEF, Canon CRW, and Adobe DNG, are all TIFF wrappers around raw sensor data. The main differences are in the compression methods used and how auxiliary data (image dimensions, exposure parameters, color balance data, etc.) are encoded.

Most image export formats use only $b = 8$ data, so a 256×256 model should suffice. The raw sensor data is generally the result of linear 12-16 bit analog to digital conversion, which suggests that an error model should have $2^{16} \times 2^{16}$ entries. Experimentally, the noise levels are usually such that moving from a 256×256 to 512×512 or larger model increased execution time with the higher-precision probabilities merely making noise reduction slightly less aggressive.

More complex adjustments are needed because each raw pixel typically samples only one color channel. This is because the sensels themselves are sensitive to a broad spectrum, but do not distinguish colors. To distinguish colors, a *color filter array* (*CFA*) pattern is imposed on the sensor. Most commonly, the CFA is a repeating 2×2 pattern of red and green over green and blue (RG/GB), also known as a *Bayer filter*, but a wide variety of other patterns have been used. Most alternative patterns still use red, green, and blue, but many cell phones and Fujifilm cameras use larger repeating patterns. Some alternative color sets aim at improving low-light performance; for example, some older cameras (e.g., Canon G1) used cyan, magenta, yellow, and green to lets twice as much light pass for 3/4 of the colors, and patterns using red, green, blue, unfiltered (often called white) lets about three times as much light pass to the unfiltered pixels. It also is common that two slightly different green filters are used in a Bayer pattern either deliberately or as a side effect of the manufacturing process.

In the interest of simplicity, the new KREMY represents the error model as a set of four PGM images, one for each of the four color channels assumed to be repeating in a 2×2 pattern. Thus, there are four nearest same-color pixels, which for the pixel at (x,y) would be found at $(x,y-2)$, $(x-2,y)$, $(x+2,y)$, and $(x,y+2)$. A CFA pattern that uses two identical green filters is accepted and treated as having two different green filters. The same simplified model of CFA patterns also is used in the synthesis of textures. However, when the html-interfaced version of KREMY is used, the error model is returned as a single, more browser-friendly, color JPEG file in which the two green channels are averaged together – as shown in Figure 1.

```
let r = texture synthesis radius;
for (i = each image pixel)
  let val = 0, sum = 0;
  for (j = each color(i) pixel within r of i)
    for (p = each pixel offset in a patch)
      let w = map[color(j+p)][*(i+p)][*(j+p)]/m;
      w = pow(w, 1/strong) * distweight(i, j);
      sum += w;
      val += (w * *j);

  new *i = (val / sum);
```

Figure 3. *Simplified probability-weighted texture synthesis algorithm*

Texture Synthesis

Normally, enhancement of image quality for noisy images is modeled as a combination of smoothing and sharpening algorithms. However, earlier work[1][2] found that texture synthesis[4] can effectively enhance both smoothness and sharpness simultaneously, and the current approach uses it even more aggressively in this way.

Texture synthesis normally is used for inpainting[5]: the process of creating credible replacements for missing, damaged, or unwanted regions of an image. In KARWY[1] and the original KREMY[2], texture synthesis is instead used to adjust the value of each pixel within its computed bounds. For each pixel, the texture synthesis processing examined up to 1089 nearby same-color pixels in a spiral order, creating a weighted average value as a function of:

- The bounded range for the pixel value.
- The bound-overlap similarity of each of the nine pixels in the patch centered at the pixel to the patch centered at the pixel to be improved.
- The base distance weighting for that position in the spiral, which smoothly decreases as the distance from the pixel being adjusted increases.

A confidence metric was computed as each position in the spiral is evaluated, and the spiral evaluation is terminated early if sufficient confidence is achieved. Random noise bounded by half the remaining uncertainty in the value was added to the weighted average value to further discourage posterization, and the pixel value was adjusted toward this weighted sum.

This approach worked extremely well for KARWY's repair of ARW2 compression artifacts. With the error model computed somewhat differently (as discussed earlier) and addition of two final extra steps, it also was effective for improving moderately noisy raw images in the original KREMY. The first extra step attempts to keep the overall brightness of the image from dramatically changing, and the second attempts to keep local contrast similar to the original. Based on ability to remove deliberately-added noise, noise was reduced by up to approximately 3EV without any smoothing... but higher noise levels obtained little improvement.

The main difference in the new version of KREMY is that synthesis is not constrained by bounds, but by probabilities. All similar pixel within a fixed radius are always incorporated in the probabilistically-weighted average value (which is computed faster due to allowing a better memory access ordering than the spiral), no pixel value bounds are enforced, and the two additional steps are removed. A simplified approximation to the new texture synthesis algorithm is given in Figure 3.

In effect, the new KREMY is **individually removing every pixel and replacing it with a texture-synthesized replacement**. Except for the fact that the original pixel value is included in computing patch match weights, this is literally using inpainting to replace each pixel individually. The `strong` parameter is used to control the sharpness of the probability decay, with larger values decaying slower because `w` values are probabilities between 0 and 1. The `distweight()` function smoothly decreases in value as the distance between points `i` and `j` increases, giving slight emphasis to nearer patch matches. Thus, arbitrarily large changes can be made to individual pixel values – and the results generally are better, especially for very noisy images.

Results

The new KREMY was implemented in C, with the core logic taking approximately 800 lines of code. There are two main versions that function nearly identically. The original KREMY was implemented inside `dcraw`[6], and one new version does the same, in-place editing the raw image data within an uncompressed DNG. However, the most recent version of DCRAW was released in 2018. The version using `raw2dng`[7] is self-contained and directly creates a DNG output file from any of the raw input formats it understands, but it does not understand as many raw formats as some DNG converters. The HTML form interface is another 1100 lines of C code, using Adobe Digital Negative Converter[8] to prepare raw input files for the `dcraw`-based version.

The work here improves raw data – which is not directly observable. To avoid applying other improvements during raw conversion, `dcraw` version 9.28 [6] was used to produce `PPM` files, which were then scaled so that each pixel became an 8×8 pixel block to help individual pixels remain distinguishable in the published paper. Note that the captures in Figures 6, 9, 10, 12, 13, and 14 are from the DPReview Studio scene[9]; all other exposures were made by the author.

Performance on KREMY 2017 Test Images

Given that the new KREMY is intended to be an improvement over the original KREMY, it is important to confirm that it does not produce poorer results for any of the test cases used with the original. There were no cases found for which the new KREMY did worse than the 2017 version, although the new version could be made excessively aggressive by poor choice of parameters.

Figures 4 and 5 show how base-ISO images from two Bayer CFA cameras, a Canon Digital Rebel XT and a Canon PowerShot S70, are handled; in each figure, corresponding crops are shown

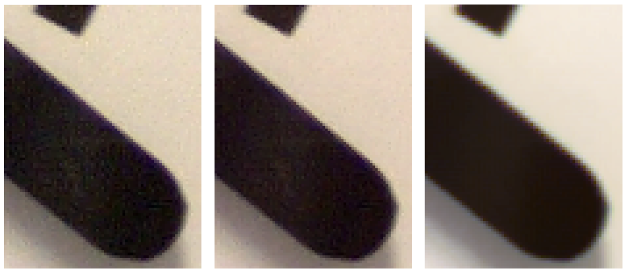
Figure 4. *Canon Digital Rebel XT @100; raw, KREMY 2017, new(3,12)*

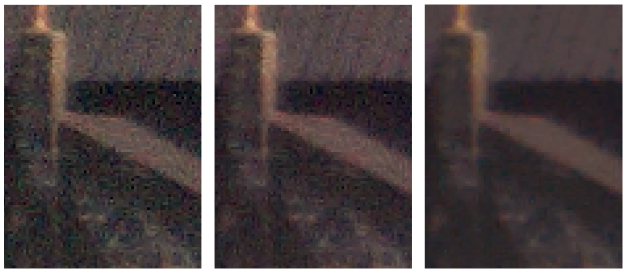
Figure 5. *Canon PowerShot S70 @50; raw, KREMY 2017, new(3,12)*

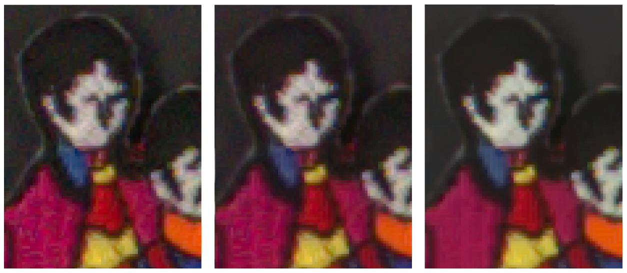
Figure 6. *Olympus E-M1 Mark II @400; raw, KREMY 2017, new(3,12)*

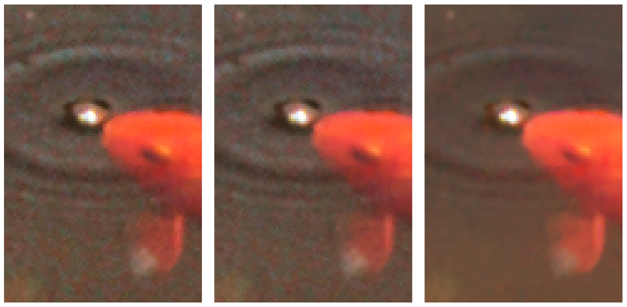
Figure 7. *Canon PowerShot G1 @50; raw, KREMY 2017, new(3,12)*

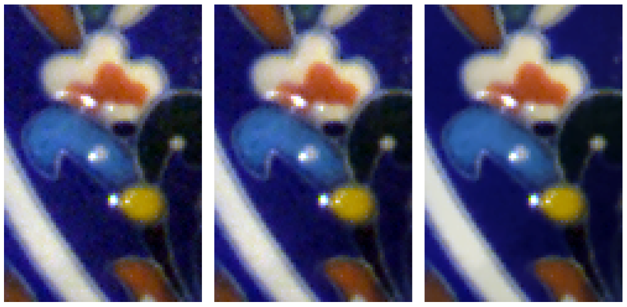
Figure 8. *Sony DSC F828 @64; raw, KREMY 2017, new(3,12)*

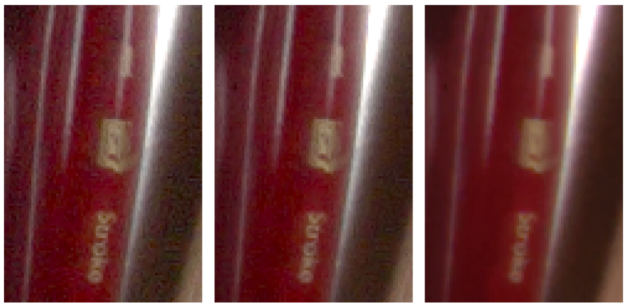

Figure 9. Apple iPhone 7; raw, KREMY 2017, new(3,12)

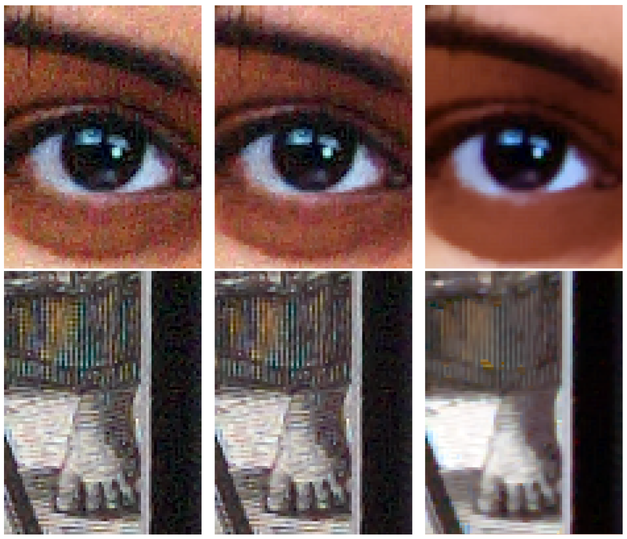

Figure 10. Nikon D810 @1600; raw, KREMY 2017, new(3,12)

from the original raw, the raw improved by KREMY 2017, and the raw improved using the new version of KREMY. The notation *new(3,12)* indicates that the images were processed by the new algorithm with a strength setting of 3 and a texture search radius of 12 pixels, which also sufficed for a medium-ISO image from a Bayer CFA Micro Four Thirds mirrorless Olympus E-M1, shown in Figure 6.

Both versions of KREMY also work with non-Bayer CFAs, provided that the pattern is a 2×2 repeat. Figure 7 shows a base ISO capture taken using a Canon PowerShot G1, which uses a cyan, magenta, yellow, green CFA. A red, green, emerald, blue CFA is used in the Sony DSC F828, with the result shown in Figure 8. There does not seem to be any difficulty associated with non-Bayer patterns; the enhancement is as effective using the same parameters applied to Bayer CFA cameras.

Figure 9 shows that small sensors, such as the Apple iPhone 7, can produce significant noise even at base ISO, but can be repaired well. In comparison, the full-frame Nikon D810 (Figure 10) is only a little noisier at the relatively high ISO of 1600 – and becomes disturbingly noiseless using the same *new(3,12)* setting used for the other examples, also partially repairing moiré. Slightly stronger settings are useful for very noisy images. An underexposed high-ISO image from a Bayer CFA APS-C mirrorless Sony NEX-7 is shown in Figure 11. The cropped area (trees

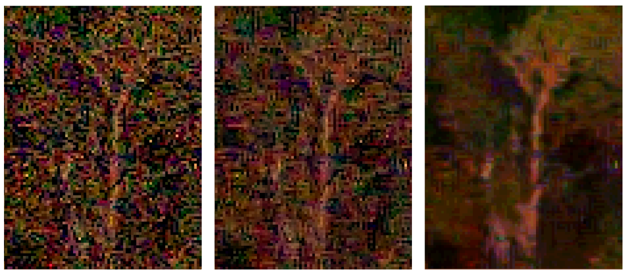

Figure 11. Sony NEX-7 @1600; raw, KREMY 2017, new(5,12)

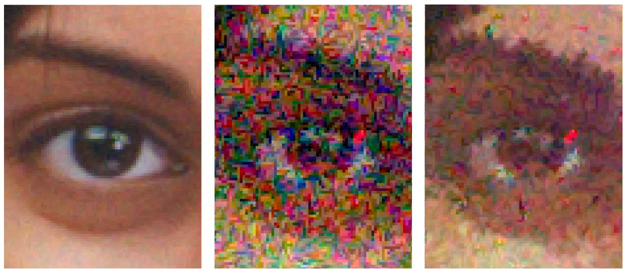

Figure 12. Sony A7C @1600 raw; @204800 raw, new(5,24)

below Mount Rushmore at night) is also significantly underexposed, compounding the noise problem. However, *new(5,12)* produces a far better result than KREMY 2017.

Community Evaluation

One of the best ways to evaluate this type of tool is to have a large user community compare results with other tools using their own photos. Although a WWW browser-interfaced version was posted and announced in DPReview forums, and feedback was positive, insufficient reviews of the tool were recorded to draw any conclusions.

Very High ISO Tests

Although the NEX-7 image in Figure 11 certainly had a lot of noise, since 2017, many cameras have added support for much higher ISOs. These ISOs are commonly seen as choices to be used only in desperation, and are handled very poorly by most raw processing pipelines – although some cameras are surprisingly effective producing JPEGs at these ISOs.

Sensors in high-end cameras have been switching from front to back illumination (BSI), which tends to improve SNR. For example, the Sony A7C's 24MP BSI full-frame sensor produces visibly cleaner output at ISO 1600 (leftmost image in Figure 12) than the 36MP FSI sensor in Nikon D810 (upper left image in Figure 10). Thus, the A7C allows pushing the ISO two stops higher than the maximum of the D810, to 204800. A similar level of improvement can be seen in the Canon R5, which Figure 13 shows is quite clean at ISO 1600, and reaches a maximum of 102400. The new algorithm is quite effective at reducing noise at these high maximum ISO settings; noise is comparable to a raw shot at roughly 3-4EV lower ISO, but image sharpness remains significantly poorer than at lower ISOs.

Although tools directly improving raw images, rather than improving images in the process of producing a rendered JPEG,

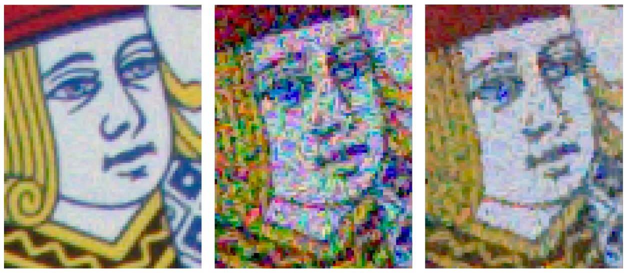

Figure 13. *Canon R5 @1600 raw; @102400 raw, new(5,48)*

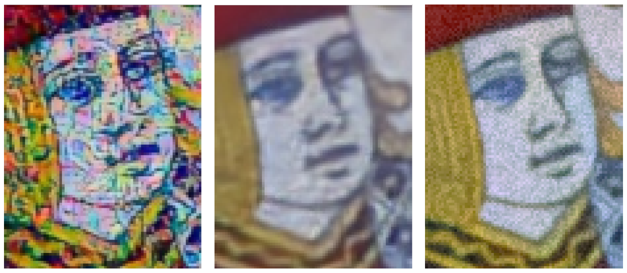

Figure 14. *Remini Professional, AI Image Denoiser, and DxO PureRAW*

are not common, there are many denoising algorithms and tools[10] – most now are based on neural networks. A representative sampling of how three of them handle the Canon R5 ISO 102400 image is given in Figure 14. Most denoising tools do almost nothing with images this noisy. A better-than-average example is Remini Professional[11], which only seems to accomplish a modest increase in sharpness and contrast. AI Image Denoiser[12] is much more aggressive, significantly enhancing details, but also applying heavy smoothing. DxO PureRAW[13], which directly improves the raw image using deep learning trained on "millions of images analyzed by DxO over 15 years," was easily the most effective of the many denoisers tested. Compared to the method presented in this paper, PureRAW was arguably better overall, producing purer colors and more contiguous edges and lines. However, it smeared some fine details affected by moiré, slightly decreasing resolution obtained from test chart images, and GPU acceleration was required to give comparable execution time to that of our approach running on a single processor core.

Conclusion

Although the algorithms used here to make the new version of KREMY are extremely simple, the effectiveness in simultaneously reducing noise and retaining or improving sharpness was consistently better than the original version of KREMY and comparable to the best software currently available. This performance was achieved not through use of machine learning with large training datasets, but by very straightforward computation creating and using a pixel value probability density function to guide texture synthesis.

The serial implementation here is not fast, but it is easily parallelized, and it also could be re-cast in a form suitable for execution as an optional step in a camera's raw processing pipeline. Combining this type of pixel value error model with camera-model-specific tweaks or AI methods will surely produce better results. It also is significant that the output here is an improved raw file, and additional improvements can be applied in demosaicing and rendering final images in formats like JPG, PNG, HEIF, etc.

Additional information, and a link to the live WWW form version, are available at `http://aggregate.org/DIT/KREMY`.

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 (2/14/2016); DOI: 10.2352/ISSN.2470-1173.2016.2.VIPC-227

[2] Henry Gordon Dietz and Paul Selegue Eberhart, "Refining raw pixel values using a value error model to drive texture synthesis," Electronic Imaging 2017, Image Processing: Algorithms and Systems XV, pp. 56-66(11) Visual Information Processing and Communication VII, pp. 1-10 (1/29/2017); DOI: 10.2352/ISSN.2470-1173.2017.13.IPAS-084

[3] Henry Dietz, Paul Eberhart, John Fike, Katie Long, Clark Demaree, and Jong Wu. "TIK: a time domain continuous imaging testbed using conventional still images and video," Electronic Imaging 2017, Digital Photography and Mobile Imaging XIII, pp. 58-65(8) (1/29/2017); DOI: 10.2352/ISSN.2470-1173.2017.15.DPMI-081

[4] A. A. Efros and T. K. Leung. "Texture synthesis by non-parametric sampling," Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999, pp. 1033-1038 vol.2, DOI: 10.1109/ICCV.1999.790383

[5] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. "Image Inpainting," SIGGRAPH 00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, July 2000, pp. 417-424, DOI: 10.1145/344779.344972

[6] Dave Coffin, "Decoding raw digital photos in Linux," *https://www.dechifro.org/dcraw/* (accessed 1/30/2022)

[7] *https://github.com/Fimagena/raw2dng* (accessed 1/30/2022)

[8] *https://helpx.adobe.com/camera-raw/using/adobe-dng-converter.html* (accessed 1/30/2022)

[9] Digital Photography Review (DPReview) Studio shot comparison, *https://www.dpreview.com/reviews/image-comparison* (accessed 1/30/2022)

[10] Linwei Fan, Fan Zhang, Hui Fan, and Caiming Zhang. "Brief review of image denoising techniques," Vis. Comput. Ind. Biomed. Art 2, 7 (2019) DOI: 10.1186/s42492-019-0016-7

[11] Remini AI Photo Enhancer, *https://remini.ai/* (accessed 1/28/2022)

[12] AI Image Denoiser, *https://imglarger.com/Denoiser* (accessed 1/28/2022)

[13] DxO PureRAW, *https://www.dxo.com/dxo-pureraw/* (accessed 1/28/2022)