Construction, Quality Assessment, and Applications of Pixel Value Error PDF Models

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

Abstract

Increasingly sophisticated algorithms, including trained artificial intelligence methods, are now widely employed to enhance image quality. Unfortunately, these algorithms often produce somewhat hallucinatory results, showing details that do not correspond to the actual scene content. It is not possible to avoid all hallucination, but by modeling pixel value error, it becomes feasible to recognize when a potential enhancement would generate image content that is statistically inconsistent with the image as captured. An image enhancement algorithm should never give a pixel a value that is outside of the error bounds for the value obtained from the sensor. More precisely, the repaired pixel values should have a high probability of accurately reflecting the true scene content.

The current work investigates computation methods and properties of a class of pixel value error model that empirically maps a probability density function (PDF). The accuracy of maps created by various practical single-shot algorithms is compared to that obtained by analysis of many images captured under controlled circumstances. In addition to applications discussed in earlier work, the use of these PDFs to constrain AI-suggested modifications to an image is explored and evaluated.

Introduction

Although noise in images has been the subject of many research papers[1], and noise often is modeled using a probability density function (PDF), the PDF is commonly a distribution that is assumed rather than empirically measured. Two key assumptions commonly made are that:

1. The PDF is not a function of the coordinates within the image, but is consistent over the whole image
2. There is no significant correlation between the ideal pixel values and the noise components imposed upon their measured values

Neither of these assumptions is fully correct. However, making these assumptions appears to dramatically improve the practicality of computing and using a noise model. Given that modern sensors often have tens of millions of sensels, not making the first assumption is viable only for the simplest of noise models. It is less clear that the second assumption is necessary, and arguably the advantage in making that assumption is merely that it allows a relatively simple equation to be used for the model rather than empirical data.

The literature discusses a wide range of parametric PDF distributions[1]. Perhaps most commonly assumed is a Gaussian (normal) distribution, which characterizes the noise distribution by a curve with a single parameter, σ, where σ^2 is the variance. Gaussian distributions are symmetric around the ideal (mean) value, so other types of distributions are used for skewed histograms: Rayleigh, Erlang, and Exponential distributions. Each of the above distributions is believed to be a good model of noise coming from particular types of sources. Although it is unlikely to occur in real images, Uniform noise is often assumed where simplicity of the model is most important. There are also some noise sources that inject noise as Impulses (Salt and Pepper) rather than continuous functions. While it is possible to obtain a model that arbitrarily closely matches noise observed by using a combination of the above noise distributions, the current work suggests that there is no benefit in doing so, and the error introduced by the second assumption above is at least as significant.

All the above formulaic distributions are simply attempting to fit a curve to the pixel value histogram based on the assumption that the curve will be easier to apply than a PDF computed directly from a histogram. In a variety of earlier work it was found that not only was using a PDF based on a histogram computationally feasible, but it was not difficult to drop the second assumption. Thus, the problem shifts from picking parameters for a formula to efficiently creating the most accurate histogram possible.

Over a period of nearly a decade, our research group has been using empirical measurements to create a variety of types of pixel value error models. The following section briefly describes the structure of these PDFs. The catch is that it is only possible to compute a true distribution of values when the ideal value is known – and empirically, it never is *known*. Thus, for a wide variety of applications significantly different methods have been used to compute the histogram and transform it into a PDF. The primary goal of the current work is thus to obtain a more precise understanding of how accurately the results of various empirical computation methods approximate the true PDF. A total of 48 different algorithms are implemented and evaluated in the current work.

The Model

The pixel value error model investigated here is fundamentally an empirically-determined type of probability density function (PDF). The model is based on computing a histogram in which the ideal pixel values are given by the Y axis coordinate and the actual values read are given by the X axis coordinate. The biggest problem is that empirically the ideal value of any pixel is typically unknowable, and thus the algorithms used to compute the histogram, discussed later in this paper, involve various approximations and tradeoffs.

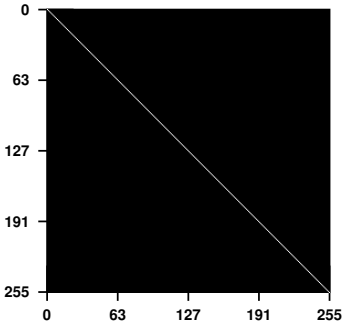

Figure 1. *Pixel value error model for a noiseless capture*

Many image formats record as many as 16 bits per sensel sampled, so one might expect a $3\times65536\times65536$ data structure would be required for modeling pixel value error in an image with three color channels. However, the uncertainty in the model generally exceeds the precision with which even a $3\times256\times256$ table can encode them, so sensel values may be scaled to fewer bits to simplify the computation and reduce the size of the resulting model. From various experiments, even the sensel value scaling method used is apparently not critical; for example, as long as the scaling is in the same space used to apply the model, even application of gamma correction has relatively little impact on the accuracy.

The computation of the model data structure begins with computing a histogram for each color channel in which the value at each point V_{read}, V_{ideal} is the measured number of times V_{read} was read when V_{ideal} was the true value for that spot in the scene. The desired structure is essentially a 2D PDF for each color channel in which the conversion from histogram to PDF is performed per row. Logically, if the true value is V_{ideal}, then the sum of probabilities of all values possibly read when V_{ideal} was the true value should sum to 1:

$$\sum_{i=0}^{255} model[i, V_{ideal}] = 1 \tag{1}$$

The histogram counts in each row of the model should thus be normalized to meet this equation. However, the most useful normalization is slightly different: make $model[V_{read}, V_{ideal}]$ be the probability that a value read as V_{read} due to noise could be representing a true value of V_{ideal}. This suggests a normalization of the probabilities not to a sum of 1, but to a maximum value of 1. The probabilities normalized in this way can be viewed as approximating the *confidence* that replacing the value V_{read} with V_{ideal} would be correcting noise rather than making a change that is inconsistent with the original scene.

The result of that normalization rule is a model that can be visualized and manipulated as a multi-channel image in which the model is computed independently for each color channel. Of course, if the model is encoded as an image using 8-bit unsigned integer values, the probabilities are scaled not from 0..1, but from 0..255.

If there is no noise, V_{read} is always equal to V_{ideal} and the result is the model shown in Figure 1. Elsewhere, these model images are shown without axis labeling because the model is really defined over the interval from 0 to the maximum sensel value and the 0..255 labeling merely shows the model quantization. Generally, the diagonal line should have the maximum value in each color channel even with noise. Any noise distribution appears as a horizontal spreading of that diagonal line. Empirically, the shape of the noise distribution is usually not a close approximation to Gaussian nor any other simple parametric shape, but a somewhat messy function of both the color channel and V_{ideal}. For example, it is very common (and unsurprising) that the measured distribution often skews to the right at low V_{ideal} values and to the left at high V_{ideal} values. The different sensitivities of different color channels also tend to slightly separate error distributions by color. Given these irregularities, representation of the pixel value error model as a multi-channel image is relatively compact, easy to examine and manipulate, and fast to apply.

Why Model Pixel Value Error?

Given an ideal image and a corrupted version, a noise model is easily constructed. However, when imaging a scene, except under exceptionally carefully controlled circumstances, the ideal image is not knowable. Thus, it has long been standard practice in the image-processing field to *start with an assumed ideal, noiseless, image and algorithmically add noise to produce one or more corrupted versions*. This is a fundamentally incorrect approach. For example, training an AI to denoise images by reducing algorithmically-imposed noise is dangerous because training is a somewhat opaque process and tiny artifacts can dramatically alter the learned behavior. For example, an AI that perfectly removes perfectly-imposed Gaussian noise might be much less effective removing the noise that actually occurs in captured images, missing opportunities to remove noise in some areas while making hallucinatory changes in others.

Additionally, in a very literal sense the thing that must be modeled is usually not noise, but the error distribution in pixel values as recorded from the sensor. This difference is not as subtle as it might seem. There are many different mechanisms imposing noise: photon shot noise, leakage current, read noise, low-level digital processing and encoding, etc. These mechanisms interact and also are significantly affected by transient properties such as the shutter speed (integration interval) and ISO (gain) used and the temperature of the sensor. Accurately modeling noise thus tends to result in a very complex model with many parameters, so often a distribution is assumed and accuracy suffers. In contrast, for processing a particular image, all that is typically needed is a model quantitatively describing the statistical error in reported pixel values for that image – the model is essentially a lookup table that gives probabilities for specific errors in values reported.

There are four main reasons that a pixel value error model of the form discussed here can be useful: (1) to evaluate noise quality of images and regions within images, (2) to recognize when scene content has or has not changed in a time series of captures, (3) to distinguish noise from actual scene content, or (4) to constrain image enhancements (even AI modifications) to be consistent with knowable properties of the scene photographed. The following subsections briefly overview these types of uses.

Evaluating Noise Quality Of An Image

Evaluating noise quality of a capture using a pixel value error model PDF expressed as an image is intuitive and straightforward. Visual examination of the PDF can be done using any image editor or viewer, and the horizontal spreading of the diagonal line (seen in the noiseless PDF of Figure 1) directly shows the distribution of value errors for all ideal values and on each color channel. Differencing two PDF images in an image editor trivially reveals how the distributions differ. Quantitative evaluations can be done by various simple computations over the PDF – and methods discussed later in this paper allow efficiently creating the PDF from a single image. For example, the average error in pixel values (scaled to 0..255) in a color channel could be computed as:

$$\frac{\sum_{j=0}^{255}\sum_{i=0}^{255}(abs(i-j)\times model[i,j])}{\sum_{j=0}^{255}\sum_{i=0}^{255}model[i,j]} \tag{2}$$

Similarly, the signed average bias of the distribution (skew) can be computed as:

$$\frac{\sum_{j=0}^{255}\sum_{i=0}^{255}((i-j)\times model[i,j])}{\sum_{j=0}^{255}\sum_{i=0}^{255}model[i,j]} \tag{3}$$

Recognizing Scene Content Changes

Scene content can change over a time series of captures due to motion of scene elements, changes in lighting, or changes in the camera viewpoint. However, in most sequences a majority of the scene content does not change from one frame to the next. This principle figures prominently in most video compression schemes. For example, MPEG-4 encoding efficiency depends on effective motion estimation, which is accomplished using any of several algorithms to match scene objects in pairs of frames[2]. Of course, the same object not only can change position from one frame to the next, but even in stable lighting the pixel values within the object can change by noise.

There is a fundamental ambiguity in seeing a pixel value change over time: is the pixel value different due to scene change or noise? This question generally cannot be answered with complete certainty, but is approachable statistically using a model of pixel value error. The question can be restated as what is the probability that a pixel read as V_0 in one frame is read as V_1 in a subsequent frame due to noise? The pixel value error model described here can be used to answer this question by treating either V_0 or V_1 as the V_{ideal} and the other as V_{read}.

For example, in a 2016 paper[3], this analysis was applied to determine how much additional information is captured when the framerate is increased to infinity. The key observation is that with a finite number of photons lighting the subject over a given interval, increasing framerate surprisingly quickly approaches an information content limit dominated by photon shot noise. Thus, very high framerates do not imply proportionately higher data rates or file sizes.

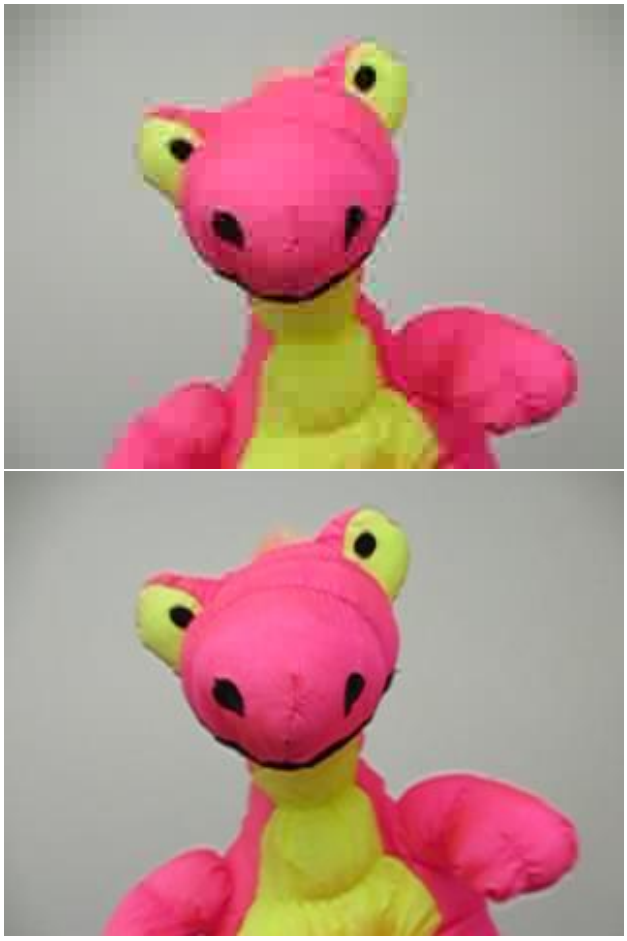

Figure 2. *Video frame and frame rendered by TIK*

If it can be determined that a pixel value has changed by more than is likely to be noise, that new sample provides information about the new scene content. Conversely, if the pixel value change over time is explainable as noise, then each sample is potentially providing additional information about the unchanging scene content in that position. The time sequence of values of a pixel over a time interval in which all value changes were explainable by noise can be averaged to provide a more accurate value for the pixel over that entire interval. In the most fundamental sense, brightness is nothing more than average photon arrival rate over time, and the average photon arrival rate can be known more precisely over an interval in which more photon arrivals are counted. This principle is the core concept underlying Time Domain Continuous Imaging (TDCI)[6] and `tik`[7]. For example, Figure 2 shows two renderings of the same scene over the same time interval. The top image is a heavily-artifacted original frame from a video captured at 240FPS. The relatively artifact-free bottom image was rendered by examining that frame and, for each pixel, replacing the frame's pixel value with the average

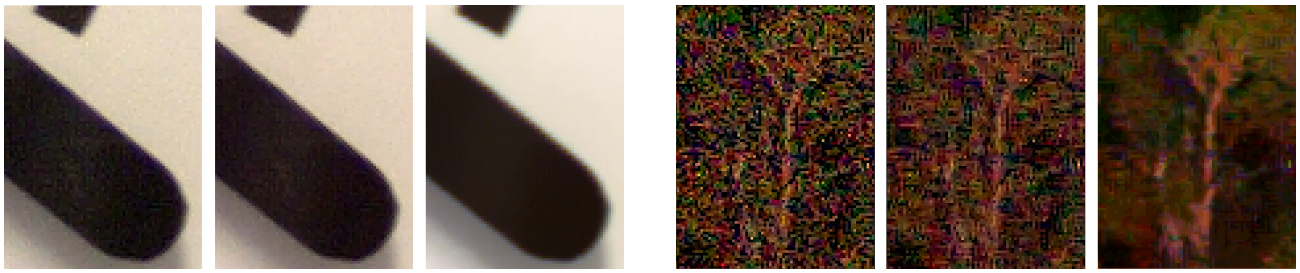

Figure 3. *Image enhancements: raw, `kremy'17`[4], and `kremy'22`[5] renderings of two crops*

value over the multi-frame interval (extending both forwards and backwards) for which that pixel had the same value within noise bounds.

The error model in `tik` is computed by stacking two or more DNG raws or JPEG images extracted from a video. Pairwise votes are summed and the resulting histogram is smoothed and processed to make it monotonic. The algorithm is a more complex approximation to the methods described in a later section here as raw-pair-box or JPEG-pair-box.

Distinguishing Noise From Scene Content

Image denoising essentially depends on distinguishing between noise and actual scene content: noise features should be eliminated while scene features should be preserved or even enhanced. Using a pixel value error model provides an efficient probabilistic discriminator. For example, KentuckY Raw Error Modeler (`kremy`)[4][5] creates and uses a pixel value error model to denoise (and slightly increase the resolution of) a raw image. As can be seen in Figure 3, the processed images show substantially reduced noise and a modest increase in visible scene detail.

For the original version of `kremy`[4], several methods were tried to compute a $4\times65536\times2$ error model – unlike the error models used here, this model simply gives minimum and maximum bounds on what each 16-bit ideal pixel value could be read as for each of four color channels (the two green channels in a Bayer filter pattern were treated separately). Initially, a separate program was written that used a rather complex voting algorithm to compute this model from a sequence of two or more 16-bit DNG raw files captured of the same scene. That computation was then replaced with a computationally cheaper and much more traditional noise estimation algorithm computing standard deviations for consistently-shaded patches (and ignoring other patches) within a single image. However, the final 2017 version used a simpler filter instead of standard deviations to construct the model from a single image. The denoising it implemented was adjusting original pixel values *within their error bounds* to be more consistent with similar textures found elsewhere in the image.

The dramatic improvement in the 2022 version of `kremy`[5], as seen in Figure 3, is primarily due to use of an error model of the form described in this paper. The error model is computed from a single raw image using a slight variation on what the current work describes as a raw-one-box approach, but still using four color channels. Different precisions for the error model were also investigated and it was concluded that scaling-up from a

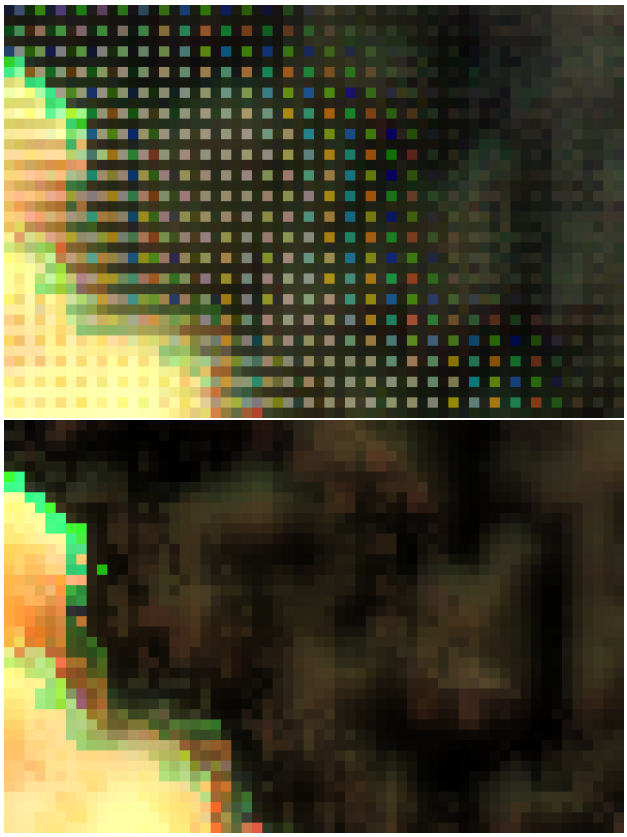

Figure 4. *Magnified crop using `PixelShift2DNG`[8] and `parsek`[9]*

256×256 to 512×512 or larger model "increased execution time with the higher-precision probabilities merely making noise reduction slightly less aggressive"[5]. Instead of tweaking pixel values within a likely minimum..maximum error range, the latest `kremy` uses the error PDF to *replace every pixel value with one created by probability-guided texture synthesis*.

Constraining Enhancements

Although it could be argued that `kremy` is also an example of the use of a noise model to constrain enhancements, this use is even more direct in Probabilistic Alignment Raw Stitcher Experiment from Kentucky (`parsek`)[9]. This tool was created to perform multi-shot super-resolution processing, especially on pixel-shift capture sequences. Pixel-shift is a technique in which

the camera's sensor is shifted by small, precise, amounts between exposures in order to obtain a higher-resolution sampling of the scene. Although computational alignment showed that the actual shifts were not as expected due to tiny amounts of camera movement, the primary defect in pixel-shift super-resolution is that movement of scene objects can cause different color channels to be sampling different scene content in the exposure sequence. A yellow flower swaying in the breeze caused the artifacting seen in the top image of Figure 4, whereas `parsek` recognized that the errant pixel values are highly improbable and suppressed their contribution in stitching. In fact, the entire stitching process in `parsek` uses a confidence computation that is strongly driven by the pixel value error model.

The error model used in `parsek` is is computed by a method described here as raw-pair-box. However, because image pairs are known to be slightly misaligned, `parsek` also applies confidences that are computed as a function of the alignment error of the pixels. For example, if the red pixel in one image that is closest to a given red pixel in another image is sampling a scene position 2/3 of pixel distant, then that pair is weighted less than pairing with a red pixel in another image that is only 1/4 of a pixel distant. This approach works extremely well in part because object movement in a scene frequently will change the object's apparent shape (e.g., the flower in Figure 4 twists as a gust hits it), so MPEG-4-style motion estimation is less effective. Not coincidentally, the error model also decreases noise for the super-resolution image because, unlike traditional pixel-shift super-resolution, multiple samples that are the same within noise are able to be averaged for each position.

Algorithms To Create The Model

The primary contribution of the current work is the definition, implementation, and evaluation of 48 different algorithms for computing a pixel value error model. This included the creation of an open-source implementation called `errpdf`. The results of all methods are intended to converge on identical models for identical inputs, although there are significant differences in the computational cost and in the quality of the model produced. All these algorithms also share the property that they assume no a priori knowledge of an ideal, noiseless, reference image.

The 48 different algorithms represent all combinations of options for three phases of processing. There are three options for the input handling, four basic approaches for computing the histogram from that image data, and four algorithms used to convert the histogram into the pixel value error PDF. These are overviewed in each of the following subsections.

Handling Of Input Images

There are literally hundreds of different image formats in common use and `errpdf` can process most of them. However, independent of input file format, the processing within `errpdf` largely assumes that the image data can be processed using 8 bits per color channel with three color channels: Red, Green, and Blue.

The OpenCV library[10] directly gives `errpdf` the ability to read files in a wide variety of image formats that are largely intended for image viewing. Windows bitmaps (BMP), JPEG and JPEG 2000, Portable Network Graphics (PNG), WebP, Portable Pixel Map (PPM), and Tag Image File Format (TIFF or TIF) are among the most commonly used. It is important to note that TIFF is not really an image format, but a way to structure the contents of a file as a collection of tagged fields; what OpenCV supports is the "Baseline TIFF" specification, which encodes RGB images either without compression or compressed using one of two different Run Length Encoding (RLE) schemes. All of these file formats share the basic property that each pixel has data for all three color channels.

Thus, the first algorithm choice is to process the image data as an array in which each pixel site has an 8-bit unsigned value for each color channel. While this seems an obvious choice, most image capture devices do not capture all three colors at each pixel location. Typically, a Color Filter Array (CFA) is used to pass only one color channel at each sensel location. The result is that for any given pixel, two of the three color channel values are not directly detected by the corresponding sensel, but are interpolated from neighboring sensels that had the desired filter color. This interpolation process, also known as demosaicing, may be as simple as averaging the nearest same-color sensel values or it can be as complex as applying a trained AI to compute the missing color values. There are two main problems with this:

1. Interpolating values generally involves combining values from multiple sensels, and any such procedure inherently tends to reduce noise. The result is that computing a pixel value error model from, for example, a JPEG image will tend to significantly underestimate the noise in the sensel measurements. It is worth noting that JPEG in particular uses lossy compression in a YUV colorspace, and those transformations also tend to smooth the image data, reducing the apparent noise. JPEG images produced by many cameras also apply various transformations to repair lens defects ranging from correcting lens pincushion or barrel distortion to color-dependent scaling to correct transverse (also called lateral) chromatic aberration – other forms of interpolation that also tend to reduce apparent noise.
2. The two interpolated color channels for each pixel multiply the computational effort spent in performing any analysis. In effect, the analysis is done over three times as many values as there were sensels. Thus, processing time tends to be about 3× as long as necessary.

Simply processing only the color information that came from a sensel can avoid these disadvantages despite using a viewing-oriented file format like JPEG. However, this approach only can be fully effective where the encoding is not too lossy and no significant corrections have been applied. It is also necessary that the CFA pattern be known so that `errpdf` knows which color channel to trust in each pixel. Typical CFA patterns consist of a repeating 2×2 block with two Green filters, one Red, and one Blue. For example, the Canon PowerShot SX530HS camera used for some tests has the repeating pattern  , which can be specified on `errpdf`'s command line as `-c rggb`.

The third input option for `errpdf` is raw image files. The word "raw" is not a file format, but a descriptive word imply-

ing that the sensor data are not "cooked" but are available essentially as they came from the sensels in the encoding used. For example, there typically are no or very few corrections applied and the data is not interpolated, but one color channel per sensel. Most raw formats are encoded as TIFF files, but with a variety of manufacturer-specific fields and encoding schemes. For example, ARW files are produced by Sony cameras, CR2 by Canons, NEF by Nikons, etc. Adobe has been pushing a particular TIFF variant, DIgital Negative (DNG), as a standard raw representation, but the transformation from many camera raw formats to DNG is not lossless.

Unfortunately, OpenCV does not directly understand raw formats. Thus, `errpdf` uses either `dcraw`[11] or `unprocessed_raw`[12] as a helper application. The output from either of those tools is a file in a monochrome Portable Gray Map (PGM)[13] with an unsigned 16 bit value per pixel. OpenCV can read a PGM, but the PGM is not interpreted as a color image. Thus, it is again necessary to use a command line argument to specify the CFA pattern as well as `-r` to specify use of a raw decoder. The command line option `-d` specifies that `dcraw` should be used for raw decoding instead of the default `unprocessed_raw`; `dcraw` is no longer maintained, but supports some older raw formats that `unprocessed_raw` does not. The raw format decoding step adds a little execution time, but the processing of raws is nearly as fast as the processing of CFA-indexed JPEGs.

Histogram Generation

Empirical creation of a histogram recording differences from V_{ideal} is complicated by the fact that V_{ideal} is fundamentally unknowable. Thus, the question is how to obtain a reasonable approximation.

Although it is not guaranteed, it is reasonable to assume that if we capture the exact same scene with the same camera settings multiple times to compute an average value for each pixel location, those average values should approximate V_{ideal}[14]. The capture of such an image sequence is significantly more difficult than one might assume. Not only must the scene and camera settings be completely controlled, but earlier work[9] revealed that, under typical conditions, a camera mounted on a solid tripod often does not maintain sensel-level alignment across a series of exposures. Even camera-generated movement must be minimized by actions such as placing an SLR's movable mirror in the up position, locking the focus and anti-shake mechanisms, disabling the mechanical shutter, and using a lens that does not automatically stop-down the aperture for each exposure.

Some types of noise, such as that imposing a fixed pattern across the sensor, are not removed by averaging multiple captures – but we are explicitly not creating a model that is a function of the coordinates within the image. It is also possible to correct fixed pattern noise before the images are input to this algorithm, and most cameras will at least partially correct even the "raw" image data using a "bad pixel" list and subtracting a dark exposure if the shutter speed is long. More fundamental is the question of the correctness of averaging as the computation. The averaging done in `errpdf` is simply finding the arithmetic mean, which has also been used by others[14], but can be significantly impacted by outliers in the distribution of values. Using the median or mode would reduce the influence of outliers, and it also is possible to directly remove outliers meeting a specific condition from the mean computation, but all of those methods would make the computation of an "average" image significantly more complex. Despite these potential flaws, the current work treats the average image as the best practically knowable approximation to the ideal image: the reference truth against which the quality of other methods can be judged.

The computation of the histogram using the average of a series of captures not only requires the controlled sequence of captures, but two passes over all images. The first pass computes the average image while the second records the histogram by incrementing $model[V_{read}, V_{average}]$ for each pixel location. If the CFA pattern is specified, only the color channel of the corresponding sensel is processed for each pixel; otherwise, all three color channels are used from each pixel location. Use of this algorithm is specified by `a` for averaging in `errpdf`.

A much cheaper alternative is the pair differencing algorithm specified by `p`. This uses a controlled sequence of as few as two images without computing an average. Instead, the first image is treated as the source of V_{ideal} values and the second as V_{read} values. While this is clearly not as good an approximation as the averaging approach using many captures, treating the data from just two images in this way is less likely to underestimate the noise than comparing to the average of the two images. However, the main reason for including the algorithm here is that this type of histogramming has been used in earlier software[7][9]. The primary advantage of this method is that, when applied to each pair of temporally adjacent images in a longer sequence, the histogram computation can be done in a single pass over the image sequence.

The last two histogram computation algorithms compute the histogram from a single image. Relatively few alternative handlings of noise operate on one capture without a "ground truth" noiseless reference[15], but this ability is a huge benefit in that it avoids the problematic capture of a controlled sequence of images and also can model noise issues that are unique to a particular capture. For example, even in a tightly-controlled capture sequence, it is likely that the sensor temperature will increase with capture sequence number; deriving the model from a single capture has the potential to more accurately account for the exact conditions that existed for that capture. The problem is that, as for the pair computation method, V_{ideal} is not known. In both these single-image histogram computations, the histogram is driven by examining neighboring pixels; note that if a CFA is specified, only sensels sampling the same color channel are considered when identifying a pixel's neighbors.

The first method for computing a histogram from one image is specified by `o`. The algorithm is based on the idea that there is a very high probability that V_{ideal} for a given pixel and at least one of its neighbors are essentially the same. This assumption is obviously false in at least some cases, such as imaging stars in a dark night sky. However, the occurrence of high-contrast detail that covers just a single pixel is not common for most scenes captured using high-pixel-count sensors and typical lenses. Most CFA-based cameras even incorporate an Anti-Alias (AA) filter

that forces adjacent sensel sites to see similar light. Thus, it is highly likely that the most similar of the eight neighboring pixel values is within the noise bounds of the current pixel's value. This algorithm simply treats the current pixel value as V_{ideal} and the nearest-valued neighbor as V_{read}. The catch is that more different values, which may simply have more noise, are never selected as nearest-valued to their neighbors; however, they are still incorporated in the histogram because they will be the V_{ideal} when finding their nearest-valued neighbor. The histogram computation using this method is fast, but can be expected to underestimate noise.

The second method for computing a histogram from a single image is specified by `r` for region. Various methods for computing noise in an image[1] suggest that only variations within "similar patches" should be histogrammed, but how can one define similarity without assuming at least an approximate model for noise? The `o` method above asserts that the value of the current pixel, V_{here}, and the most similar valued neighbor, $V_{similar}$, probably represent the same V_{ideal}. If that is true, we argue that other nearby pixels with values between V_{here} and $V_{similar}$ are likely part of the same region. Thus, for each of the eight compass directions, the value of the pixel two away is checked to see if it is between the values V_{here} and $V_{similar}$. If it is, given that the value of the intervening pixel V_{mid}, $model[V_{mid}, V_{here}]$ is incremented. Compared to the `o` computation, this computation is somewhat more expensive, but has the desirable attribute that while doing a simple patch filtering, it is less likely than the `o` approach to underestimate the noise.

Histogram To PDF Conversion

Given a histogram computed by one of the four approaches described above, four different algorithms then can be used to encode it as a pixel value error PDF were explored.

The first of these is designated `n`, for the normalization algorithm. This is the PDF normalization discussed earlier, normalizing the probabilities so that the maximum is 1.0, which gets encoded as the unsigned byte value 255. All four alternative algorithms for converting the histogram into a PDF end with this same normalization step.

Typically, $\forall i$, $model[i,i]$ will have a normalized probability of 1.0 using the `n` algorithm. Not having that property would have the seemingly nonsensical implication that V_{ideal} could be outside of the noise bounds for the value V_{ideal}. However, a histogram can embody any distribution – even ones that appear very strange. In addition to the diagonal having the highest probability, intuitively, the probability in each row should monotonically approach 1.0 for entries closer to the diagonal. This property is forced if `m` is specified to make the output monotonic. The method use to force the property processes each row of the histogram separately. Starting at the end of each row and working toward the diagonal, if the current histogram bucket holds a lower count than the previous, the previous bucket's count is copied into the next bucket. The same processing is then repeated working from the minimum value to the diagonal.

An alternative way to force monotonicity can be specified as `b`, the box algorithm. Since none of the four methods for building the histogram can be certain about the ideal value for each pixel position, it can be argued that incrementing $model[V_a, V_b]$ is just as appropriate as incrementing $model[V_b, V_a]$. Logically, all that is known is that V_a and V_b can represent the same V_{ideal}. Thus, the box algorithm generates a new histogram in which, if the original histogram had a count of K in $model[V_a, V_b]$, then K is added to all positions in the new histogram in the box with corners V_a, V_b and V_b, V_a. This box histogramming is relatively expensive computationally, but not prohibitive when the histogram has just 256×256 buckets. The new histogram is inherently monotonic toward the diagonal in each row, but also enforces some continuity along the diagonal – which is important because the histogram made from a single scene (as all these methods construct it) is almost always a very sparse sampling of all possible tonal transitions.

The fourth method for converting a histogram into a PDF is specified by `s`, for smoothing – although it might be more correct to call it a diffusion process. It replaces the original histogram with a new histogram having a more smooth distribution. However, this algorithm works by iteratively diffusing the distribution along the diagonal. The smoothing carries fractional counts from each row to its neighboring rows, shifting the propagated fraction of the distribution along the diagonal so that the contribution from $model[i, j]$ is shared with $model[i+1, j+1]$ and $model[i-1, j-1]$. This is done using many passes (currently, 128) running in alternating up and down directions. This approach is even more effective than the box algorithm in filling-in sparse histograms, but also is computationally expensive. This method has the interesting property that it does not force monotonicity where the original histogram was dramatically non-monotonic.

Results

As was stated earlier, all 48 algorithms discussed here have been implemented in a single software tool called `errpdf`. This open source program is written in C++ using the OpenCV library. Where appropriate, OpenMP[17] directives are used to specify parallel execution to reduce runtime. The program input can be a single image or a time sequence of two or more images of a static scene.

There were a total of 1124 images in 34 different time sequences of test images processed. These included cases testing all 48 algorithms: raw, CFA JPEG, and JPEG input processing as well as different histogram methods and algorithms for conversion into PDFs. The images were shot using three different cameras selected to represent a wide range of noise behaviors:

- **Sony A7RV**: BSI CMOS 61MP full-frame mirrorless
- **Canon PowerShot SX530HS**: FSI CMOS 16MP superzoom, somewhat similar to a smartphone sensor but with completely uncooked raws via CHDK[16]
- **Canon PowerShot Elph160**: CCD 20MP compact

Each capture sequence included between 25 and 64 captures and each image was recorded as both raw and JPEG. The Sony captures used electronic shutter and a fully manual lens to minimize potential movement during a capture sequence; the raw images were encoded in the default ARW format. CHDK[16] was used to obtain DNG raws from the Canon PowerShots, which normally would produce only JPEG files. In all tests using raw input,

Table 1: Relative Execution Cost

	n ormal	m onotonic	b ox	s mooth
a verage	24.3	24.3	24.3	24.4
p air	reference	1	1.03	1.17
o ne	1.45	1.45	1.46	1.59
r egion	3.31	3.31	3.34	3.48

Table 2: Average Error In Error Estimates

	n ormal	m onotonic	b ox	s mooth
a verage	+16%	+27%	-3%	reference
p air	-16%	+56%	+18%	+18%
o ne	-50%	-16%	-13%	-9%
r region	+50%	+149%	+30%	+35%

the `unprocessed_raw` program was used as a helper to decode the raw formats. Different sequences were captured at base ISO and at the highest "recommended" ISO of that camera model. Several test scenes were used, all lit using the well-controlled lighting of a commercial 2×2-foot light box.

It is not practical to present all the data produced in this evaluation within this paper. However a very concise summary is easily given.

For the 48 algorithms used, the first choice is of the three types of input: raw, CFA JPEG, or JPEG. Although the resulting models are not directly comparable because of the different spaces in which pixel values are coded, raws produce the most accurate models and processing JPEGs using knowledge of the CFA pattern is slightly more accurate than trusting all color information in a JPEG equally. Execution cost of the processing is also approximately 3× higher when the CFA pattern is not used. For the remaining 4×4=16 choices of histogram method and algorithm for conversion into a PDF, two tables provide a fairly clear understanding of the behavior.

Table 1 gives the relative execution cost of each method over all 34 different time sequences of input images. It is not surprising that constructing the histogram using averaging to compute ideal pixel values is the most expensive, but the cost was only 24× that of the fastest method despite the average test case needing to process 33 image files. Note also that the fastest was the pair processing, which had to process two images while the remaining two algorithms only read a single image. In this light, the region processing stands out as relatively inefficient.

It is difficult to fully characterize how the qualities of two PDF models differ, but Table 2 gives good insight by simply examining the differences in the average pixel value error predicted across all test cases. There is good reason to believe that the `as` model is approximately as close to ideal as is empirically knowable, so summarizing differences with that provides a good quality metric. Although the closest to the `as` model was the `ab` model, the `os` model was shockingly close, predicting just 9% less error in pixel values. The performance of the region-based analysis is equally surprising because it is far poorer than expected, especially for `rm`. Considering both tables together makes a strong case for using `as` where possible (e.g., to perform calibration for shooting many images in similar studio conditions) and `os` when the model must be computed from a single image captured in the field.

To give a better feel for what real PDFs look like, Figure 5 shows a group of raw PDFs. Within that figure, the top left image is an ISO 100 (base ISO) image of the scene averaged from 64 exposures and the PDF next to it is the resulting `as` PDF. The image at the top right is a similar image created as the average of 64 exposures at ISO 12800. The sixteen PDFs that follow are computed from that ISO 12800 sequence and are presented in the same grid order used for Tables 1 and 2. Although the differences in the PDFs appear substantial, similar features appear in most and the rightmost column's smoothed PDFs are actually quite similar statistically. Note that the noise difference between ISO 100 and ISO 12800 is easily visible in the PDFs although the ISO 12800 images from this Sony are not much noisier than either of the other cameras at their base ISOs.

It is useful to note that different scene content does significantly alter the PDF computed, but primarily because the pixel value errors are grossly under sampled with some scenes. For example, for TIK[7], the primary test scene used was dominated by a GretagMacbeth ColorChecker Color Rendition Chart[18]. When a similar scene was used here a very splotchy histogram was produced. The relatively large areas of constant color and brightness led to severe under sampling. Of course, this under sampling is harmless in the sense that the portions of the PDF that would be used in applying the PDF to enhance that image are well sampled. The scene dependence simply means that to use the PDF to fully characterize a camera's noise behavior requires either a carefully-selected scene or combining of PDFs computed from multiple scenes.

Conclusion

For nearly a decade, our research group has computed PDF pixel value error models in various ways in support of a wide variety of purposes. With the completion of the current work, the quality and performance differences between different algorithms are much better understood. The open source C++ program `errpdf`, available from `https://aggregate.org/DIT/ERRPDF`, makes it unnecessary to invent and code a new algorithm for each application.

Perhaps most significantly, the difference between error models computed from a 64-exposure sequence and from one image averaged just 9% using the best choice of algorithms. The best such algorithm, `os`, is also computationally reasonably efficient. This is an important discovery because it is well known that difficult-to-model dynamic factors like sensor temperature can significantly alter the noise in a capture, but a model created from each capture trivially accounts for such noise factors.

Thus, a PDF computed from one captured image can effectively be used to filter proposed changes to that image so that any alterations made are unlikely to change the knowable attributes of the scene. For example, AI hallucinations can be blocked at the pixel level by recognizing that they are unlikely content according to the noise model computed for the original capture.

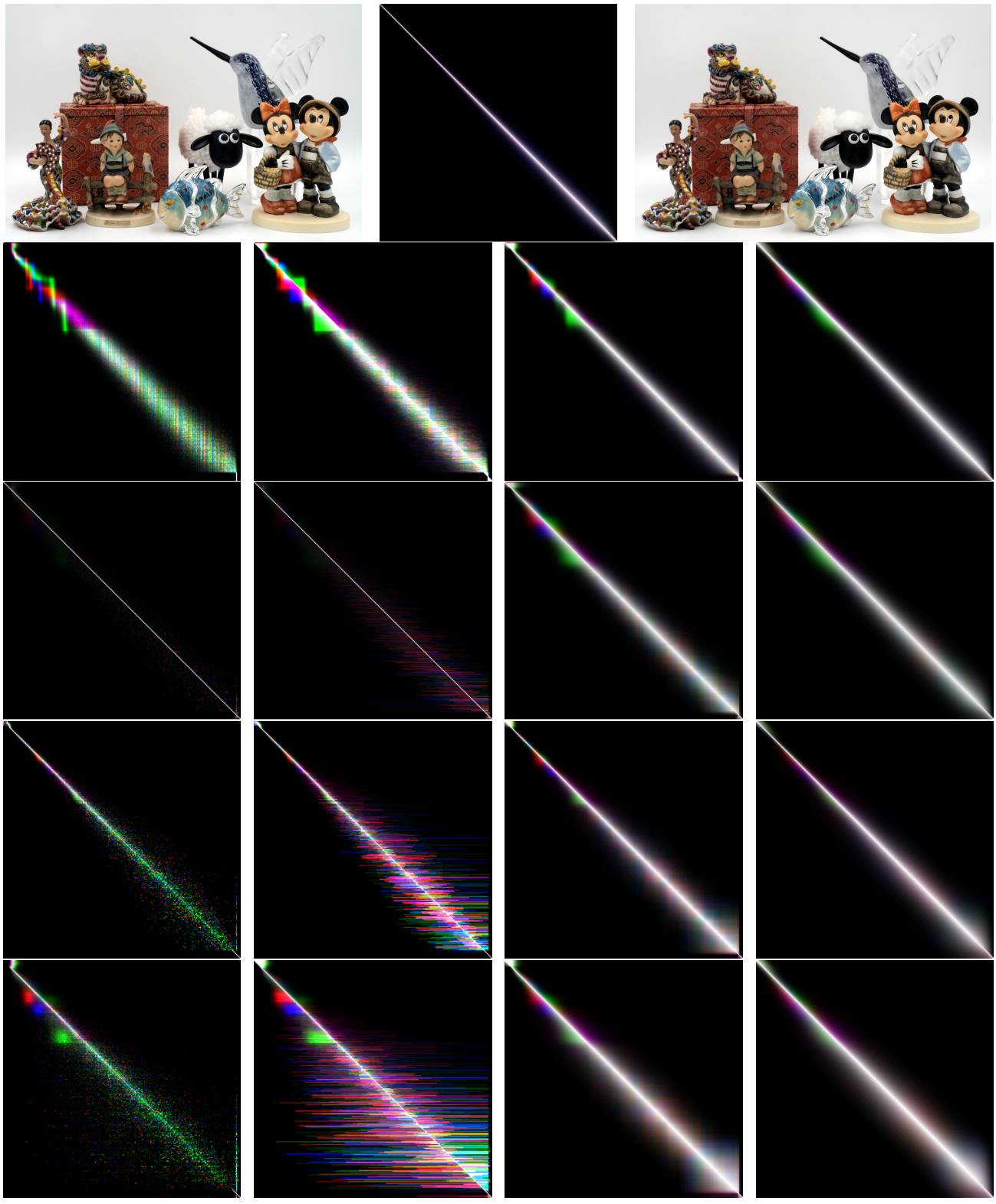

Figure 5. *A7RV shot @ 100,* `as` *PDF; shot @ 12800, complete grid of raw PDFs:* `an am ab as pn pm pb ps on om ob os rn rm rb rs`

References

[1] Ajay Kumar Boyat and Brijendra Kumar Joshi, "A Review Paper: Noise Models in Digital Image Processing," Signal & Image Processing : An International Journal (SIPIJ) Vol.6, No.2, April 2015, doi: 10.48550/arXiv:1505.03489

[2] Peter Kuhn, "Algorithms, complexity analysis and VLSI architectures for MPEG-4 motion estimation," Springer Science & Business Media, 2013, doi: 10.1007/978-1-4757-4474-3

[3] Henry Dietz, Zachary Snyder, John Fike, Pablo Quevedo;, "Scene Appearance Change As Framerate Approaches Infinity" in Proc. IS&T Int'l. Symp. on Electronic Imaging: Digital Photography and Mobile Imaging XII, 2016, doi: 10.2352/ISSN.2470-1173.2016.18.DPMI-259

[4] Henry Gordon Dietz, "Refining raw pixel values using a value error model to drive texture synthesis" in Proc. IS&T Int'l. Symp. on Electronic Imaging: Image Processing: Algorithms and Systems XV, 2017, pp 56 - 66, doi: 10.2352/ISSN.2470-1173.2017.13.IPAS-084

[5] Henry Dietz, "An improved raw image enhancement algorithm using a statistical model for pixel value error" in Proc. IS&T Int'l. Symp. on Electronic Imaging: Computational Imaging, 2022, pp 151-1 - 151-6, doi: 10.2352/EI.2022.34.14.COIMG-151

[6] Henry Dietz, Paul Eberhart, John Fike, Katie Long, Clark Demaree, "Temporal super-resolution for time domain continuous imaging" in Proc. IS&T Int'l. Symp. on Electronic Imaging: Computational Imaging XV, 2017, pp 87 - 93, doi: 10.2352/ISSN.2470-1173.2017.17.COIMG-430

[7] Henry Dietz, Paul Eberhart, John Fike, Katie Long, Clark Demaree, Jong Wu, "TIK: a time domain continuous imaging testbed using conventional still images and video" in Proc. IS&T Int'l. Symp. on Electronic Imaging: Digital Photography and Mobile Imaging XIII, 2017, pp 58 - 65, doi: 10.2352/ISSN.2470-1173.2017.15.DPMI-081

[8] LibRaw LLC, "PixelShift2DNG: Convert Sony and Pentax Pixel Shift Files to DNG," `https://www.fastrawviewer.com/PixelShift2DNG`, accessed 2/19/2025

[9] Henry Dietz, "Leveraging Pixel Value Certainty in Pixel-Shift and Other Multi-Shot Super-Resolution Processing," in Electronic Imaging, 2024, pp 142-1 - 142-7, doi: 10.2352/EI.2024.36.15.COIMG-142

[10] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000

[11] Dave Coffin, "Decoding raw digital photos in Linux," `https://www.dechifro.org/dcraw/`, accessed 2/19/2025

[12] LibRaw LLC, "LibRaw raw image decoder," `https://www.libraw.org/`, accessed 2/19/2025

[13] Jef Poskanzer, "NETPBM: Extended portable bitmap toolkit," 1993, `https://netpbm.sourceforge.net/`, accessed 2/18/2025

[14] Jun Xu, Hui Li, Zhetong Liang, David Zhang, and Lei Zhang, "Real-world noisy image denoising: A new benchmark." arXiv preprint arXiv:1804.02603 (2018)

[15] A. Krull, T-O. Buchholz and F. Jug, "Noise2Void - Learning Denoising From Single Noisy Images," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 2019, pp. 2124-2132, doi: 10.1109/CVPR.2019.00223.

[16] Canon Hack Development Kit Homepage, `https://chdk.fandom.com/wiki/CHDK`, accessed 2/20/2025

[17] OpenMP Architecture Review Board, "OpenMP Application Programming Interface," Version 6.0, November 2024, `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-6-0.pdf`

[18] Danny Pascale, "RGB coordinates of the Macbeth ColorChecker," The BabelColor Company 6 (2006): 6.