A True Random Number Generator Algorithm From Digital Camera Image Noise For Varying Lighting Conditions

Rongzhong Li
Departments of Computer Science and Physics
Wake Forest University
Winston-Salem, NC 27109
Email: rzlib2l@gmail.com

***Abstract*—We present a True Random Number Generator (TRNG) using the images taken by web or mobile phone cameras. We use all three RGB color channels to obtain the random numbers whereas previous studies used only one. We investigated the physical and statistical properties of the random noise in a digital photograph obtained by a camera system, and we made several approximations to efficiently collect the best random signals from the pixels in the images to map them to random sequences. In short, the algorithm excludes each pixel's saturated values to get its unbiased bits. An additional transposing operation shuffles the raw sequence to achieve better randomness. The final sequence passes all of the NIST randomness tests. The algorithm involves very few calculations and is especially suitable for smart phones. With modern mobile cameras, it can work on the go and achieve a fast bit rate. With readily available commodity hardware with no hardware changes, we observe a random number generate rate of 60 Mbps. A minor hardware optimization can result in a rate of about 1 Gbps.**

***Keywords*—RNG; mobile camera; Gaussian; noise; bitrate**

I. Introduction

Randomness is the nature of our universe [1]. Random number generation has become the basis of modern cryptography [2] and random event simulations, such as weather forecasting [3], Monte Carlo calculations [4], quantum mechanics calculations [5], and molecular simulations [6], [7]. Those applications all rely on the quality of random sequences produced by the Random Number Generator (RNG).

The source of randomness can be physical, but they can also be computational. For example, the kernel of Linux keeps a physical entropy source pool and hash it to random numbers [8]. Many mathematicians have developed their pseudo-RNG algorithms based on complex mathematical theories [9].

With the development of digital ecosystems, apps for mobile transactions now require fast and reliable algorithms to generate random numbers. At the same time, the resolution and speed of in-phone cameras have reached a sufficiently high level that they can now perform as parallel random bits generators and reduce the use of the precious CPU resources. The idea of generating random numbers from images is not new [10]. For example, Lavarand [11] takes pictures of lava lamps and extracts random data from the floating material's pattern, but the bandwidth is low.

Previously, Sanguinetti et al. generated random numbers using an LED on a mobile phone camera's sensor [12]. The quantum nature of the light source can generate random photons, and the camera sensor accurately captures the photons' numbers to result in a set of random numbers. However, their algorithm requires an LED paired with a good camera sensor, a carefully tuned working environment, and may require a box to isolate the whole system. We also notice that they use only the green channel coupled with an LED even though red and blue channels are available. Their speed is between 100Mbps to 1Gbps. They claim their performance could be tripled by utilizing all the 3 color channels, but it is unclear whether the photons from 3 LEDs will interrupt each other when detected by sensor. A more recent study by Duping Zhang et al. uses the built-in flash lamp of a smart phone as the random source. However their method requires the users' conscious operations [13].

In our present study, we instead utilize all types of noises that exist in digital camera system. After investigating the property of those noises, we make several simple but effective approximations to fix the biased patterns in data. Then these data are fed to a direct bitwise operation to generate random 0 or 1 bits. The bits are first stored in memory and then read out in a certain order. Multiple sequences are generated under different lighting conditions, and they have passed the standard NIST Randomness Tests [14].

II. Noise Sources in Camera Sensors

A naive use of the camera data is to put the brightness at each point (pixel values) of the digital image into a sequence. If the camera is pointing at a random scene, the sequence is presumably random. However, for web cameras and especially for cellphone cameras, the scenario can vary a lot. They can be pointing to the laptop user, the still floor, or even thrown in the pocket. Before we utilize the camera's data, we must first understand what we actually obtain from an image sensor. We use an in-depth test below to show that the direct bit flow is not actually random.

The actual brightness from the sensor's pixel is:

$$b = b_{NF} + noise$$

where b is the final brightness and b_{NF} is the noise free brightness. b_{noise} represents all the additive noises.

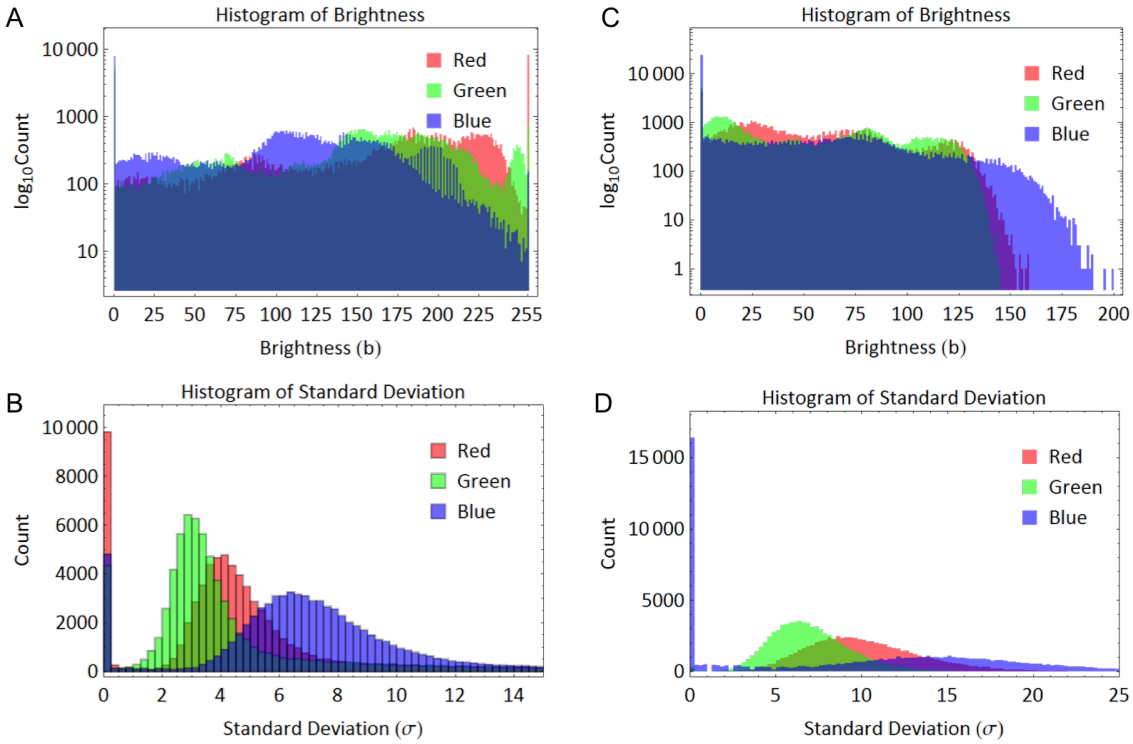

Fig. 1. Brightness and standard deviation histograms of digital photographs. The statistics on the left column are from pictures taken in a bright condition and the ones on the right column are from pictures taken in a dark condition.

Shot noise (photon noise) comes from the quantum nature of light and can be described by Poisson statistics. If the incident light onto the camera sensor has a photon flux I, the total photon received by sensor during time t would be It, while the noise signal will have a quantity of $\sqrt{It}$ [15].

Dark noise N_d comes from the self-generated thermal electrons within the sensor's silicon layer. It also follows Poisson statistics. The total noise during time t would be $\sqrt{N_d t}$.

Read noise N_r comes from the chip's reading process. It's a one-time noise only introduced during reading.

The total noise can be represented by $\sqrt{It + N_d t + N_r^2}$, while the signal is It. So the Signal to Noise Ratio (SNR) is given by $\frac{It}{\sqrt{It+N_d t+N_r^2}}$ [16].

Camera manufacturers and photographers aim to increase the SNR, but noises always exist to some degree. In good lighting conditions, the noises are less noticeable, but even covering the lens with fingers can show that the pixels are fluctuating a lot. These noises, while not optimal for great photographs, are actually welcomed in our algorithm because they act as our physical entropy pool.

III. Noise Model

The following image data and tests are based on a ThinkPad T410 integrated camera, as well as an iPhone 5s back camera. We expect similar results with any modern digital cameras.

As shown in Fig. 1, the image histogram shows the distribution of brightness in a single snapshot. As a simple integrated camera, its aperture and frame rate are fixed. To get a correctly exposed image, there is a built-in mechanism to increase the sensitivity (ISO) under dark conditions and decrease sensitivity under bright light. As a result, a typical photo's histogram should spread over shadow, middle, and highlight regions. However, we may still get overexposure or underexposure areas when shooting photos with both bright and dark components. In this case, a brightness histogram will have peaks at the extremes, and the corresponding pixels are said to be "saturated". Notice the upper histograms in Fig. 1 are plotted by $\log_{10} Count$, the saturated values are highly populated.

Compared with brightness, the standard deviation of each pixel has a smaller range. In a dark condition, the sensor will increase sensitivity and lead to larger deviations, which creates more uncertainty on each pixel. The majority of the standard deviation ranges from 2 to 25. A few large standard deviations may come from the subject's movement before the camera. A significant number of 0 standard deviations may result from either dead pixels or saturated pixels. If we sum up the counts at brightness 0 and 255, the total count approximately equals the count of the 0 standard deviation cases.

The brightness probability distributions of representative pixels in 600 frames are shown in Fig. 2 for both bright and dark conditions. These sample pixels are randomly chosen at prime numbers. They can be approximated by their corresponding normal distribution probability density function (PDF) with parameters (μ, σ). However, the fitting will fail

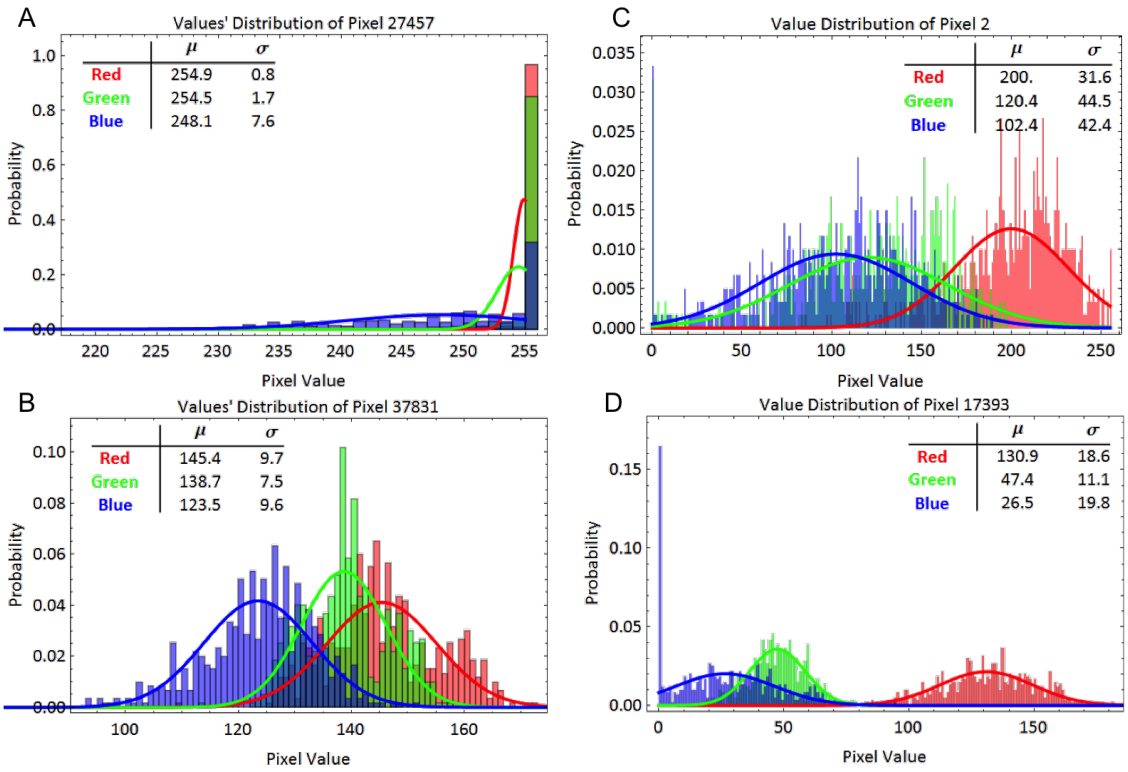

Fig. 2. Brightness probability distributions for 600 frames at representative pixels. The images used to generated the probability distributions on the left column are taken in a bright condition and the ones used for the probability distributions on the right column are taken in a dark condition.

at the two extremes when their counts all accumulate at the boundary values.

The overall behavior of the noises is usually modeled as Additive Zero Mean Gaussian Noise (AWGN) [17]:

$$\text{PDF}(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

where x is the observable, μ is the mean value, and σ is the standard deviation of the distribution.

The mean value μ can be treated as the noise free brightness b_{NF}, while the noise is added to the base brightness. So the PDF$(noise, 0, \sigma)$ becomes PDF(b, b_{NF}, σ), which inherits the noise's statistical properties.

We can derive random bits from the values of a distribution as shown in Fig. 3. If the random bit depth is 3, we can get the last 3 binary digits from the brightness. The mathematical expectation of the last bit's value is:

$$\text{E}(bit) = \sum_{b=\mu-4\sigma}^{\mu+4\sigma} \text{PDF}(b, \mu, \sigma)(b \bmod 2)$$

Assuming a Gaussian PDF, 99% of the samples should be in the range $(\mu-3\sigma, \mu+3\sigma)$. When calculating the summation, the range from $\mu - 4\sigma$ to $\mu + 4\sigma$ is sufficient to obtain the characteristic "bell" shape. For small σ, the summation is replaced by integration during numerical evaluation.

Ideally, all of the pixels should generate unbiased random bits. We would expect $\mu = 0.5$ from a perfect normal

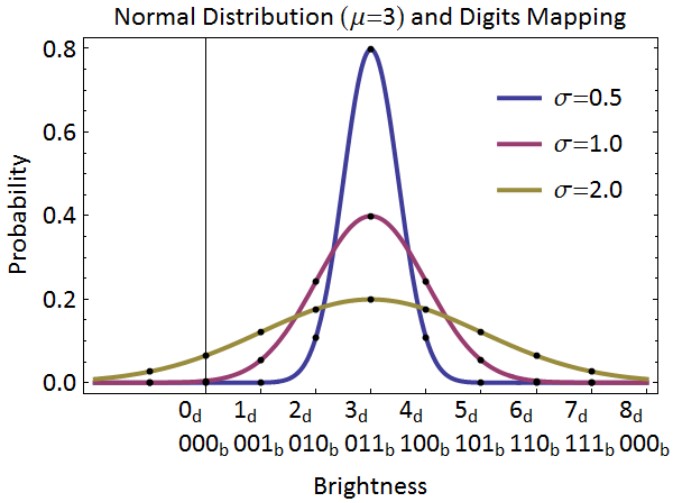

Fig. 3. A normal Gaussian distribution mapped to binary digits. The Gaussian PDFs are shown for $\mu = 3, \sigma = 0.5, 1.0, 2.0$. The last binary bits of the brightness are shown below the decimal numbers. The black dots represent each standard deviation, σ

distribution with symmetric range. But it is not always the case in digital cameras with limited dynamic range. Values outside the range are clipped to the boundary values. When calculating the summation, the clipped brightness $b_{clipped}$ takes the form:

$$b_{clipped}(b) = \begin{cases} b & 0 \le b \le 255 \\ 0 & b < 0 \\ 255 & b > 255 \end{cases}$$

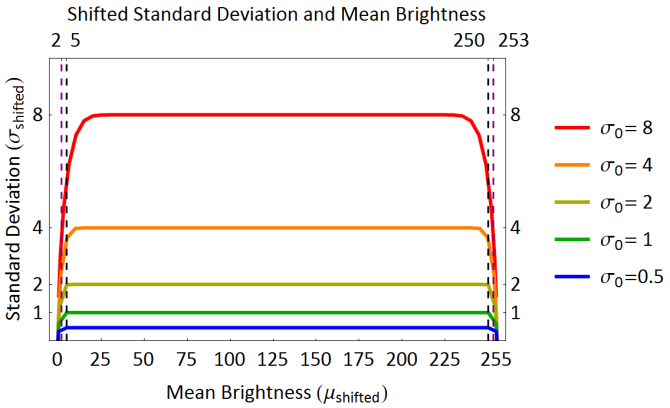

Fig. 4. The shifted mean brightness and standard deviations from clipped brightness. μ ranges from -10 to 265. The vertical axis is plotted by $\sqrt{\sigma}$, while $\sigma = 0.5, 1, 2, 3, 4, 6, 8$. These ideal Gaussian distribution's (μ, σ) are shifted to the observed values we obtain from the digital camera images.

The shifted expectation value then becomes:

$$\mathrm{E}_{shifted}(bit) = \sum_{b=\mu-4\sigma}^{\mu+4\sigma} \mathrm{PDF}(b, \mu, \sigma)(b_{clipped}(b) \bmod 2)$$

Due to the clipping, all the excessive probabilities are added up to the boundary value. The actual mean brightness and standard deviation are shifted from the ideal Gaussian distribution's (μ, σ). We can calculate the $(\mu_{shifted}, \sigma_{shifted})$:

$$\mu_{shifted} = \sum_{b=\mu-4\sigma}^{\mu+4\sigma} b_{clipped}(b)\mathrm{PDF}(b, \mu, \sigma)$$

$$\sigma_{shifted} = \sqrt{\sum_{b=\mu-4\sigma}^{\mu+4\sigma} (b_{clipped}(b) - \mu_{shifted})^2 \mathrm{PDF}(b, \mu, \sigma)}$$

The $(\mu_{shifted}, \sigma_{shifted})$ of an ideal $\mathrm{PDF}(b, \mu, \sigma)$ are plotted in Fig. 4.

We can now calculate the $E_{shifted}$ and $(\mu_{shifted}, \sigma_{shifted})$ of an ideal distribution. Their relationship is shown in Fig. 5. The expectation between 0.4995 and 0.5005 are marked gray, which allows a bias of $\frac{0.0005}{0.5} = \frac{1}{1000}$. In general, the gray dots form an unbiased trapezoid region (UTR) as a function of (μ, σ), surrounded by the biased bottom region (BBR) and biased side region (BSR).

BBR comes from extremely small σ (caused by low ISO for bright conditions) that can't provide enough random information on the required bit depth. With current error allowance of $\frac{1}{1000}$, the bottom border can be approximated by:

$$\sigma_{min} \geq \frac{2^{depth}}{1.4}$$

BSR comes from broken symmetry and excessive probabilities on the saturated pixel values. With current error allowance of $\frac{1}{1000}$, the side borders can be approximated by:

$$\sigma_{side}(\mu) = \begin{cases} \frac{\mu}{3} & 0 \leq \mu < 127.5 \\ \frac{255-\mu}{3} & 127.5 \leq \mu \leq 255 \end{cases}$$

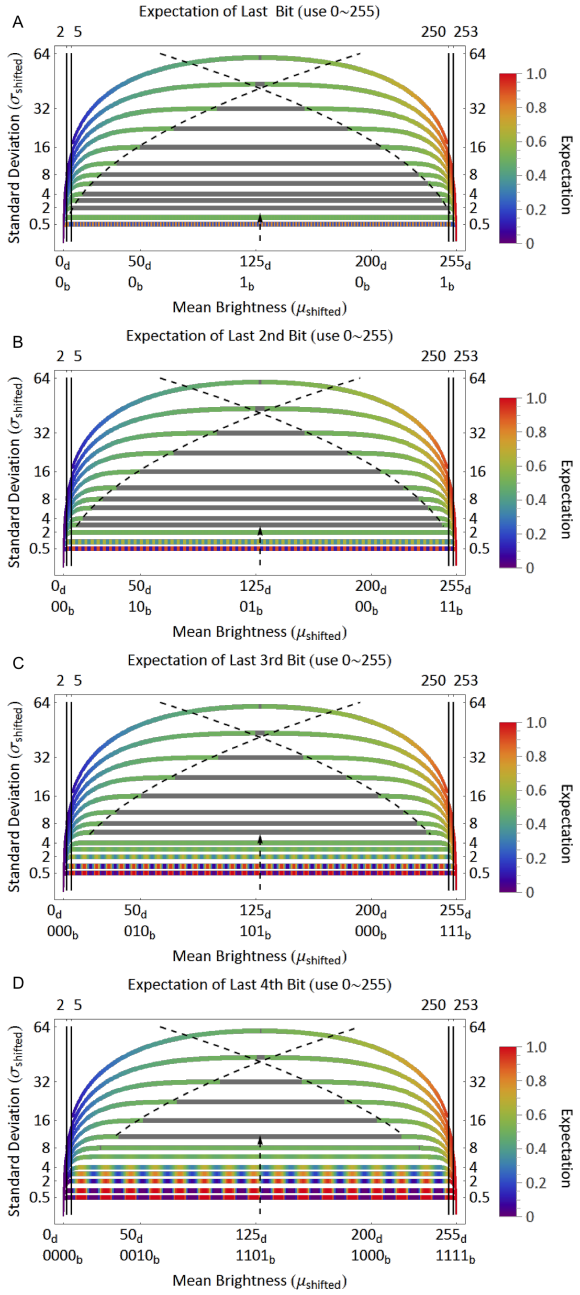

Fig. 5. The relationships between the raw expectation, shifted mean brightness, and shifted standard deviations. μ ranges from -65 to 320. The vertical axis is plotted using $\sqrt{\sigma}$. The border of the center region should be straight in the linear axis. Values between 0.4995 and 0.5005 are marked gray. Dashed lines show the fitted borders for normal σ.

To obtain unbiased data, we have to pick out the pixels belonging to the unbiased region.

IV. Approximations for Efficiently Selecting Unbiased Pixels

Since the theoretical unbiased region has been defined, in principle, we can pick out the pixels by their (μ, σ) value. However, both μ and σ are statistical properties that require data from multiple frames. These values require a significant amount of time to collect, which can impede performance,

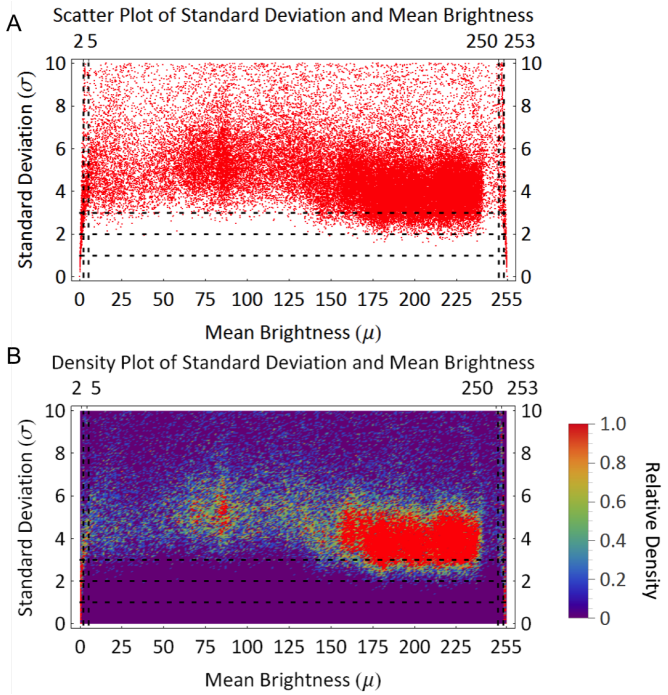

Fig. 6. Scatter plot (top) with corresponding density plot (bottom) of standard deviation of mean brightness of pixels from 20 frames . The dashed lines mark the extreme σ and μ values.

and the camera may also move and result in the change of its pixels' (μ, σ) values. Also, the Point-In-Polygon (PIP) decision is another expensive calculation [18].

Some empirical approximations can be made by observing the property of the (μ, σ) scatter plot in 20 frames from a real world camera, as shown in Fig. 6. All of the pixels from the sensor from the red channel are plotted by their (μ, σ) coordinates. The corresponding density plot shows the relative probability of each conformation. The green and blue channels shows similar features that lead to the same conclusions.

Both small and large σ are quite rare. Large σ can happen at any μ, while extremely small σ only happens at the boundary values. This is because any fluctuation outside the range will be counted as the saturated boundary values. Recall the relationship between μ and σ in Fig. 4, we can merge it with Fig. 6. The result in Fig. 7 shows that extremely small σ can be excluded by reducing the range of μ we use.

However, discarding the boundary values will result in their probability of visit becoming 0, and the total probability is no longer 1. Therefore, we must renormalize the expectation value, which becomes the ensemble average, such that the clipped probability density function takes the form:

$$\text{PDF}_{clipped}(b, \mu, \sigma) = \begin{cases} \text{PDF}(b, \mu, \sigma) & min \leq b \leq max \\ 0 & b < min \\ 0 & b > max \end{cases}$$

where min can be some small values (e.g., $1 \sim 5$) and $max = 255 - min$, so that the range is symmetric.

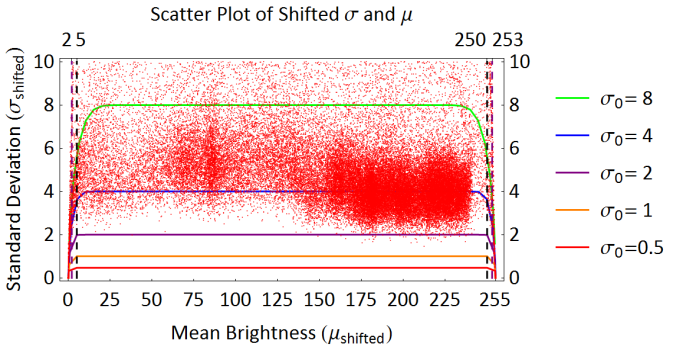

Fig. 7. The scatter plot with shifted (μ, σ) superimposed with Fig. 4. The (μ, σ) in the earlier scatter plot are actually shifted. The extremely low σ from the signal source are quite rare, and most of them are near the boundary and come from the clipping effect. See text for additional details.

The ensemble average of the last digit is calculated by:

$$< bit >= \frac{\sum_{b=\mu-4\sigma}^{\mu+4\sigma} \text{PDF}_{clipped}(b, \mu, \sigma)(b_{clipped}(b) \bmod 2)}{\sum_{b=\mu-4\sigma}^{\mu+4\sigma} \text{PDF}_{clipped}(b, \mu, \sigma)}$$

The recalculated ensemble averages of some last digits are shown in Fig. 8. Note that the shifted μ and σ plays a role as the index of pixels, so they are not recalculated for the new figure. A more complete combination set can be found in supplementary materials.

Though the main purpose of last operation is to avoid the BBRs near boundaries, the new BSRs are also significantly weakened compared to the original BSRs. The largest biased area used to be at the top left/top right corner (Fig. 5), but now they are shifted to the tiny bottom corners(Fig. 8). The bottom border is not affected, while the new side boarder of BSR can be approximated by:

$$\sigma_{side}(\mu) = \begin{cases} \frac{\mu - min}{2} & 0 \leq \mu < 127.5 \\ \frac{255 - \mu - min}{2} & 127.5 \leq \mu \leq 255 \end{cases}$$

Further reducing the brightness range does not significantly exclude the BSR, and doing so discards too many usable pixels. We therefore chose the range between 2 and 253 in our subsequent analyses and in our algorithm. Fig. 9 shows the mean value of the last bit in 20 sample frames, and we observe that the bias is reduced significantly by discarding the boundary values. To decide how many random bits to get from one pixel, we can use the derived relationship:

$$depth \leq 1.4 \log_2 \sigma_{min}$$

and Table I shows the minimum σ required for certain depth of unbiased random bits.

As discussed before, the standard deviation depends on sensitivity, which is adjusted to the lighting condition by the camera's driver. The driver has a mechanism to tune the sensitivity with brightness, i.e. change σ with average μ of the whole image. A versatile choice should adapt to different lighting conditions by pre-calculating the average μ of the image.

However, since a camera can always move, we may have to average the image's brightness from time to time. A

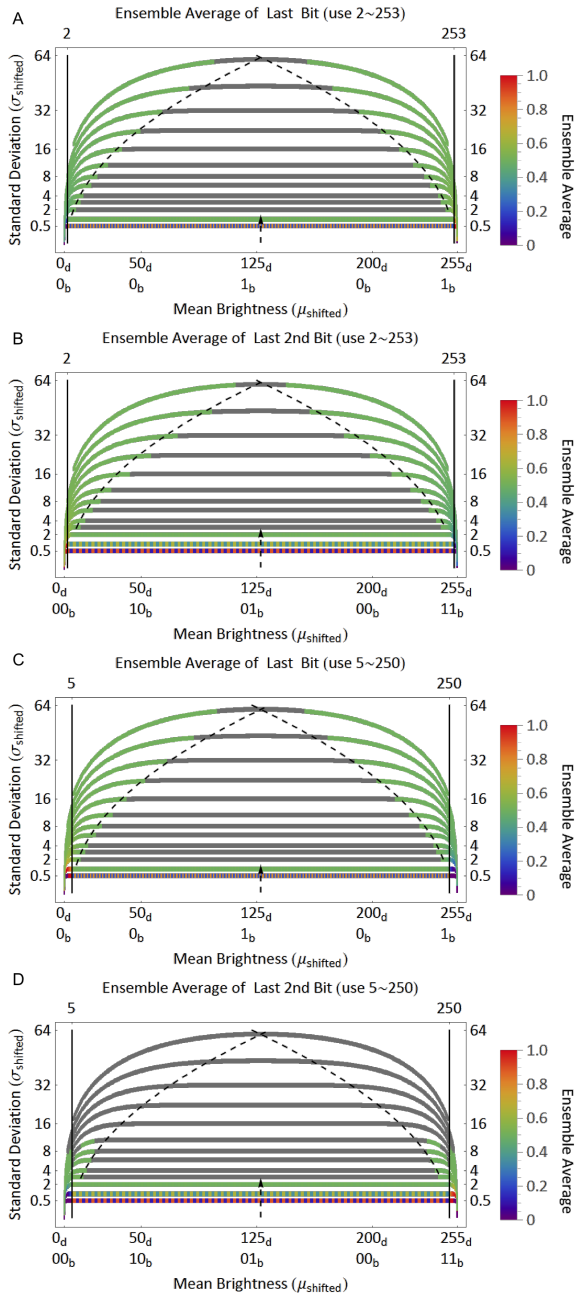

Fig. 8. The relationships between the new expectation, shifted mean brightness, and shifted standard deviations after discarding the different sets of boundary values. μ ranges from -65 to 320. The vertical axis is plotted using $\sqrt{\sigma}$. Values between 0.4995 and 0.5005 are marked gray. Dashed lines show the fitted borders for normal σ.

TABLE I. RESTRICTION TABLE

$depth$	1	2	3	4	5
σ_{min}	1.43	2.86	5.71	11.43	22.86

conservative approach is simply using only the last bit to meet all the situations to obtain values from most of the pixels. A more liberal approach is to obtain values from the last 3 or 4 bits, but the standard deviation has to be large and is only practical in dark conditions. Considering the environment of the camera may be unpredictable and change significantly, we only use the last bit in our algorithm to ensure the highest

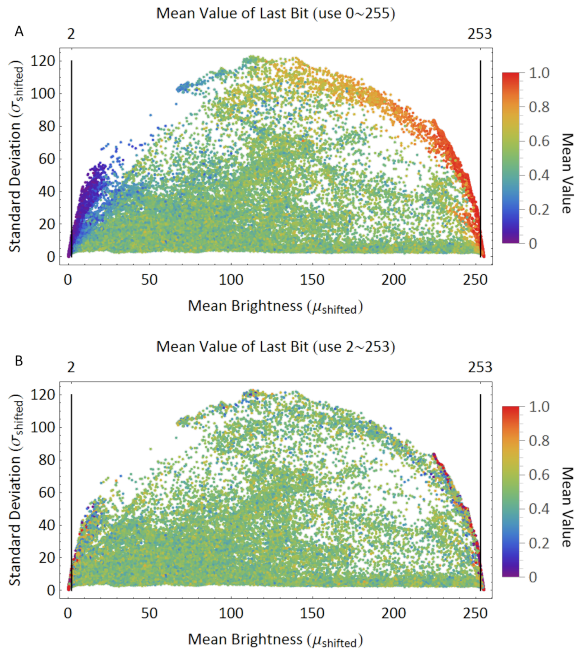

Fig. 9. The mean value of last bit in 20 sample frames. After discarding boundary brightness, the bias is reduced significantly. Note the shifted μ and σ plays a role as the index of pixels, so they are not recalculated for the second graph.

quality of randomness of our values in any lighting conditions even though we are intentionally limiting the number of bits we could obtain.

V. ALGORITHM PSEUDO CODE

We now present our general algorithm for obtaining random numbers from images obtained by a digital camera that utilizes all three RGB channels. We use a digital camera to obtain multiple images. For each image, we select out the pixels with brightness ranging from 2 to 253 (excluding the boundaries). We continue this loop for each image until the desired number of random numbers are obtained. All of the bits from one frame become a sublist of the total sequence.

However the raw sequence is not perfectly random. There exists a BSR for each frame. When the image is not well exposed, the bias can be large in each frame. Also, some blocking features still remained even if the pixels' positions are shifted after picking. An assistant randomness extractor may solve these problems, but it's not rooted in the algorithm itself and requires extra calculations. After several trial and errors, we found a neat solution to achieve randomness within the algorithm.

Since the algorithm is supposed to continuously take pictures within short time intervals, we can flip the bits every two frames. To spread the fix evenly over the whole sequence, we shuffle the sequence by writing in row-major order then reading in column-major order as a square matrix. The matrix dimension is $\lfloor\sqrt{RequiredLength}\rfloor$. Excessive bits from the last frame are appended to the tail to hide the dimension information. Because the usable pixels always change in each frame, this simple operation introduces extra randomness.

Below is the pseudo-code of the algorithm:

Algorithm 1 A pseudo-code for the RNG

Initialize the camera and set variables
Allocate square array
Timing start;
Generation:
while (*NumSoFar* $<$ *NumNeeded*) **do**
  Take one snapshot; ▷ C1
  Pick out the brightness $\in [2, 253]$; ▷ $\mathcal{O}(N)$
  Take the last bits as a *SubList*; ▷ $\mathcal{O}(N)$
  If (*Frame* is even) flip the bits in *SubList*; ▷ $\mathcal{O}(N)$
  Add *SubList* to *FinalList* in row-major order; ▷ C2
  NumSoFar = *NumSoFar* + *SubList.Length*; ▷ C3
end while
Output:
Print in column-major order; ▷ $\mathcal{O}(N)$
Extra bits are appended to the tail of the sequence; ▷ C4
Timing end;

The overall time complexity of the algorithm is in $\mathcal{O}(N)$. We note that since the operation within each frame can be done in parallel, if implemented properly on a parallel architecture, such as a GPU, the performance can be reduced to between $\mathcal{O}(\sqrt{N})$ and $\mathcal{O}(N)$.

Depending on different architecture, the slowest operation should be put into the interval between each frame. The order of transposing (row-major order to column-major) operation can be switched for the best performance. Fig. 10 illustrates our algorithm generating a set of random numbers using all 3 RGB channels.

VI. Implementation and Practical Performance

The algorithm is implemented both in Wolfram Mathematica 9.0 [19] and C++, using different camera models. The source codes are available at request.

The Mathematica code is run on a ThinkPad T410 laptop with an integrated camera and Windows 7 OS. The resolution is 640×480 with 3 RGB channels. The color depth is 8, which provides a dynamic range from 0 to 255. The ISO is controlled by the manufacturer's driver and can adjust to environment's brightness. The default frame rate is 12. All other parameters are kept at their default values.

The C++ code is run on an Ubuntu 12.10 Desktop. Images are taken by an iPhone5S in burst mode. The resolution is 3264×2448 with 3 RGB channels. The color depth is 8, and the ISO ranges from 500 to 2500. Frame rate of burst mode is 10. All other parameters are kept at their default values.

The figures appeared in modeling section were generated using the Thinkpad's integrated camera at resolution 320×240. *Bright conditions* are under daylight, fluorescent lamp, or incandescent lamp. *Dark conditions* are in a dark room with a distant lamp, the reflected screen backlight, or simply covering the camera in bright conditions.

We generated 20×10^8 bits sequences with a total size of 2G bits. The sequences are fed to standard NIST Randomness Test [14], which divides each sequence into 1000 subsequences then run 15 independent tests. Our sequences passed all the tests.

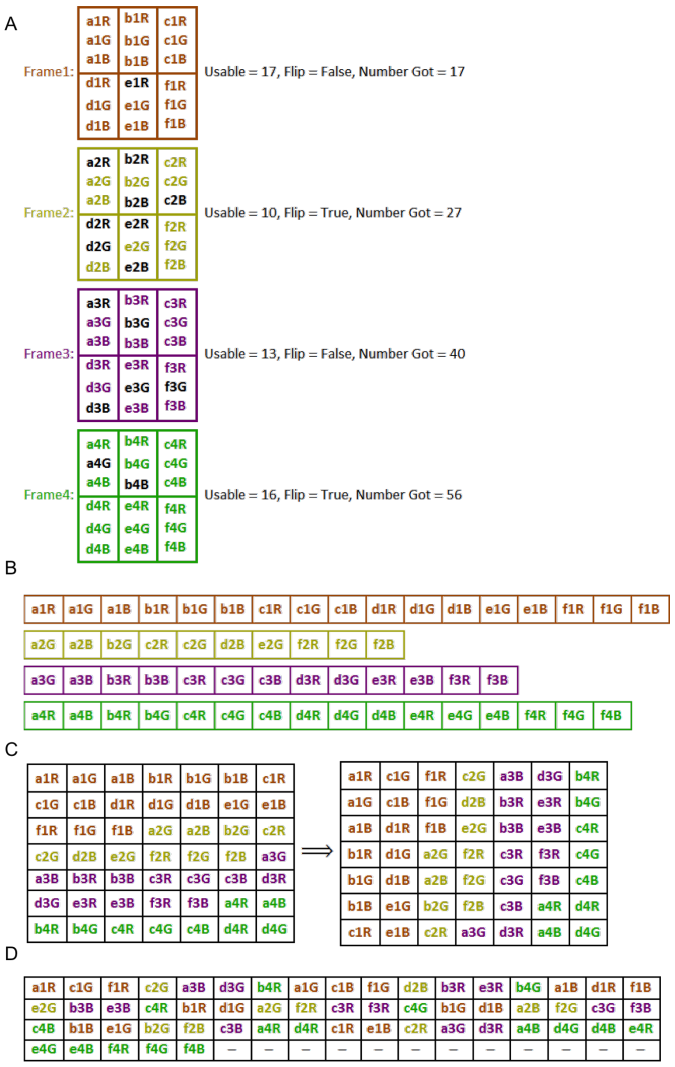

Fig. 10. Illustration of RNG algorithm. 50 random bits are requested. The sensor size is 3×2 with all 3 RGB channels. Black pixels indicate that those are boundary values and therefore are discarded.

The speed of the algorithm can be written as:

$$speed = \frac{width \times height \times channels \times depth}{max(T_{processing}, \frac{1}{framerate})}$$

where the depth is the longest unbiased digits we can obtain per pixel. If working on a smart phone's 8 MP camera (3264×2448) with 3 color channels, taking just 1 bit from each channel, setting the frame rate to be 10, with all pixels are exposed properly, and assuming that the calculation can finish within two frames, the expected speed will be $\frac{3264 \times 2448 \times 3 \times 1}{\frac{1}{10}} \frac{1}{10^6} = 240 Mbps$. In practice, the 0.3 MP web camera has a speed of 1.6 Mbps, and the iPhone5S's 8 MP camera has a speed of 60 Mbps.

We next attempted to analyze the source of the discrepancy between the theoretical and empirical rates. Unfortunately, the Mathematica code uses some built-in function calls, and it is unclear how Mathematica actually implements those functions and likely has significant unrelated overhead issues. However, the C++ implementation pre-initializes a pointer array W in

row-major order. Each W's element points to an array R's element in column-major order. The last bits are acquired by BitAnd operation. The data can be written to W during each frame interval, so it can be read from R directly in the final step. According to the C++ code, most of the operations can finish between the frame intervals except the row-major order to column-major order transform operation. If we consider the operation as a part of I/O, the random number generation speed will reach 200 Mbps, which approaches the theoretical speed.

The major bottleneck of the algorithm is the small σ existing in the signal source that limits the number of unbiased random digits per pixel. If a *perfectly random* number is not necessary, 3 bits per pixel can be used with relatively small reduction in the quality of the random numbers while tripling the speed. In the end, it is a balance between the speed and quality of the random numbers generated by our algorithm.

The algorithm looks optimal under dark conditions, for the majority of σ will increase (higher ISO). The quality of randomness will drop if the driver decides to use very low ISO for bright conditions. If we directly optimize the driver to always use high ISO for the RNG, we would get more than 1 bit per pixel. The shutter time can also drop to maintain the optimal $ISO \times ShutterTime$, which allows for a faster frame rate. In our present implementation, according to Table. I, if we tune the $\sigma_{min} > 2.86$, we can obtain two bits per pixel, while the frame rate can be doubled. In the end we can obtain $4\times$ speedup and reach a rate of 1 Gbps.

However, the shutter time cannot be too short or the shot noise and dark noise will not accumulate enough, and only the read noise will dominate the signal. The sustainable brightness level varies on different cameras. The setting configuration of (Aperture, ISO, ShutterTime) is the main factor to affect an image's exposure. A bright condition requires less exposure while a dark condition requires more exposure. The algorithm will fail in both quality and speed if the configuration $(\text{Aperture}_{\min}, \text{ISO}_{\min}, \text{ShutterTime}_{\min})$ or $(\text{Aperture}_{\max}, \text{ISO}_{\max}, \text{ShutterTime}_{\max})$ returns a mostly bright or mostly dark image, respectively. An ideal working condition is pointing the camera towards a perfectly random photon source, and tuning the brightness within the camera's dynamic range. The algorithm then basically converges to the previously discussed Bruno's method [12].

VII. Conclusions

The disadvantage of digital camera as a source of random numbers lies in the biased patterns in images and its blocked sensor units. A raw sequence without further processing may have a fixed-pattern noise. In our present study, we introduce an algorithm that utilizes pixels from all three RGB color channels to obtain a set of random numbers. The algorithm obtains the last few bits affected by noises, and they are essentially free from the scenery bias. The blocked features from the sensor are fixed by bit flipping and transposing the data as a matrix. Because the usable pixel length always changes for each frame, this method provides a shuffling effect that results in a better quality of randomness.

There are several benefits from this algorithm. First, the algorithm relies on the Gaussian property of noise signals rather than the model of camera. It can work on most existing devices with a digital camera. Second, there's no special requirement for accessories or working conditions. Third, the algorithm generates true random numbers directly, while keeps the calculation neat and easy to implement both in software and hardware. Smart phone app developers can readily adopt this algorithm.

Acknowledgment

The author would like to thank Prof. Sam Cho and Prof. David John for their valuable suggestions and careful revisions. The author also appreciates the initial stimulation and encouragement from Suxuan Wang to pursue this study.

References

[1] Hector Zenil. *Randomness Through Computation: Some Answers, More Questions*. World Scientific, 2011.

[2] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[3] David B Stephenson, Valentina Pavan, and Roxana Bojariu. Is the north atlantic oscillation a random walk? *International Journal of Climatology*, 20(1):1–18, 2000.

[4] Reuven Y Rubinstein and Dirk P Kroese. *Simulation and the Monte Carlo method*, volume 707. John Wiley & Sons, 2011.

[5] Julia Kempe. Quantum random walks: an introductory overview. *Contemporary Physics*, 44(4):307–327, 2003.

[6] Martin Karplus and J Andrew McCammon. Molecular dynamics simulations of biomolecules. *Nature Structural and Molecular Biology*, 9(9):646–652, 2002.

[7] Jessica D Leuchter, Adam T Green, Julian Gilyard, Cecilia G Rambarat, and Samuel S Cho. Coarse-grained and atomistic md simulations of rna and dna folding. *Israel Journal of Chemistry*, 2014.

[8] Z. Gutterman, B. Pinkas, and T. Reinman. Analysis of the linux random number generator. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15 pp.–385.

[9] George Marsaglia. Random number generators. *Journal of Modern Applied Statistical Methods*, 2(1), 2003.

[10] Anthony J Martino and G Michael Morris. Optical random number generator based on photoevent locations. *Applied optics*, 30(8):981–989, 1991.

[11] L.C. Noll, R.G. Mende, and S. Sisodiya. Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system, 1998.

[12] Bruno Sanguinetti, Anthony Martin, Hugo Zbinden, and Nicolas Gisin. Quantum random number generation on a mobile phone. *arXiv preprint arXiv:1405.0435*, 2014.

[13] Xuping Zhang, Li Qi, Zhiqiang Tang, and Yixin Zhang. Portable true random number generator for personal encryption application based on smartphone camera. *Electronics Letters*, 50(24):1841–1843, 2014.

[14] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document, 2001.

[15] Y Reibel, M Jung, M Bouhifd, B Cunin, and C Draman. Ccd or cmos camera noise characterisation. *The European Physical Journal Applied Physics*, 21(01):75–80, 2003.

[16] David Dussault and Paul Hoess. Noise performance comparison of iccd with ccd and emccd cameras. In *Optical Science and Technology, the SPIE 49th Annual Meeting*, pages 195–204. International Society for Optics and Photonics.

[17] Angelo Bosco, Arcangelo Bruna, Giuseppe Messina, and Giuseppe Spampinato. Fast method for noise level estimation and integrated noise reduction. *Consumer Electronics, IEEE Transactions on*, 51(3):1028–1033, 2005.

[18] Kai Hormann and Alexander Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20(3):131–144, 2001.

[19] S Wolfram. Mathematica edition: Version 9.0. 2012.