Plane-dependent Error Diffusion on a GPU

Yao Zhang[a], John Ludd Recker[b], Robert Ulichney[c], Ingeborg Tastl[b], John D. Owens[a]

[a]University of California, Davis, One Shields Avenue, Davis, CA, USA;
[b]Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA, USA;
[c]Hewlett-Packard Co., 165 Dascomb Road, Andover, MA, USA

ABSTRACT

In this paper, we study a plane-dependent technique that reduces dot-on-dot printing in color images, and apply this technique to a GPU-based error diffusion halftoning algorithm. We design image quality metrics to preserve mean color and minimize colorant overlaps. We further use randomized intra-plane error filter weights to break periodic structures. Our GPU implementation achieves a processing speed of 200 MegaPixels/second for RGB color images, and a speedup of $30 - 37\times$ over a multi-threaded implementation on a dual-core CPU. Since the GPU implementation is memory bound, we essentially get the image quality benefits for free by adding arithmetic complexities for inter-plane dependency and error filter weights randomization.

Keywords: Halftoning, Plane-dependent Error Diffusion, Parallel Processing, GPU Computing.

1. INTRODUCTION

Error diffusion converts a multi-level image to a binary image and serves as one of the most time-consuming stages in a printer imaging pipeline. The classic Floyd-Steinberg error diffusion[1] is an inherently serial process, making it unfriendly for modern parallel processors such as GPUs and multicore CPUs. Recently, programmable and high-performance GPUs have attracted several research efforts in adapting this algorithm for massively parallel processing.[2–4] We previously adopt a block-based scheme by Li et al.[5] in favor of its ample fine-grained parallelism for GPU processing.[2] Deshpande et al.[3] use a technique that processes pixels in parallel along the image diagonal in a wavefront fashion, as proposed by Metaxas.[6] In order to overcome the lack of parallelism towards image corners, they design an ingenious hybrid approach that distributes less parallel parts of the workload to the multicore CPU, and highly parallel parts of the workload to the GPU. Trager et al.[4] use a similar wave-front approach, but resort to simultaneous processing of multiple pages to saturate the parallel capacity of the GPU.

In this work, we extend our previous work on block-based error diffusion[2] to color images. While a naive extension would apply the same error diffusion method to individual color planes independently, this method tends to print multiple colorants on the same pixel locations. As a result, the perception of halftone grain is high, and the image quality suffers; the more desirable arrangement would be a homogeneous arrangement of all color pixels, not just pixels within a given color plane. In order to minimize dot-on-dot printing, a better approach considers the dependency of color planes. To this end, we incorporate such a plane-dependent approach[7] into our GPU-based error diffusion method. For best image quality, we carefully select the inter-plane error-weighting coefficients by designing and optimizing image quality metrics to preserve mean color and minimize colorant overlaps. We further use randomized intra-plane error filter weights to break periodic structures. We demonstrate that our GPU implementation achieves both high performance and good image quality for a number of experiments.

The rest of the paper is organized as follows. Section 2 reviews the plane-dependent error diffusion algorithm and the specification of the image quality metrics. Section 3 presents the GPU implementation and performance results. Section 4 discusses our experimental results on image quality. Section 5 concludes the paper and describes future work.

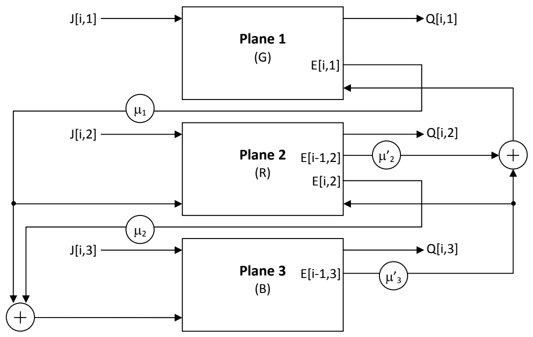


Figure 1. Plane-dependent error diffusion for a three-plane RGB image. $J(i, j)$ stands for the plane j of pixel i of the input image. $Q(i, j)$ stands for the plane j of pixel i of the output image. $E(i, j)$ stands for the quantization error for the plane j of pixel i of the input image. μ_i and μ'_i respectively stands for the forward and backward inter-plane error-weighting coefficient.

2. ALGORITHM REVIEW

2.1 Plane-dependent Error Diffusion

Plane-dependent error diffusion[7] works by feeding forward and feeding back color errors among color planes. Just like regular error diffusion, the method sequentially traverses an image pixel by pixel in a particular scan order. For each pixel, it processes all color planes successively. Besides diffusing errors to neighbor pixels as in regular error diffusion, the method feeds forward the error of each current plane to the next plane of the current pixel, and feeds back the error of each current plane to the previous plane of the next pixel, as shown in Figure 1. The feed-back and feed-forward errors are weighted by a set of inter-plane coefficients. Note that setting inter-plane coefficients to zero simply ignores inter-plane dependency and results in regular error diffusion. The plane-dependent technique is orthogonal to an error diffusion algorithm, making it easily integrated into the GPU-based pinwheel error diffusion from our previous work.[2] The pinwheel algorithm parallelizes error diffusion by dividing an image into blocks and processing them in parallel using a "pinwheel" scan order.[5] We also use stochastic perturbations on the intra-plane error filter weights[8] to significantly mitigate periodic artifacts.

2.2 Image Quality Metrics

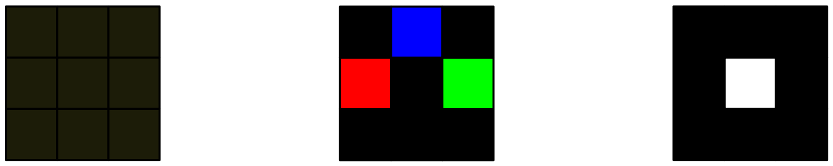
While it is not mentioned in the previous work[7] how to set inter-plane coefficients, we find it to be a non-trivial problem and the key to best image quality. We choose to set these coefficients to maximize image quality, which we quantify with two metrics: Bias and Grain.

An important property of any halftoning method is that it is mean preserving. The Bias metric specifies the degree to which the output deviates from the intended mean value. Specifically, we define Bias as the percentage of output color pixels that are short or in excess of the mean color of the input image,

$$Bias = \frac{|number_{red} - mean_{red}| + |number_{green} - mean_{green}| + |number_{blue} - mean_{blue}|}{total\ number\ of\ pixels},$$

in which $number_{color}$ stands for the number of quantized output pixels that have a component of this color, and $mean_{color}$ stands for the number of such pixels needed in the output image to preserve the mean of this color component in the input image.

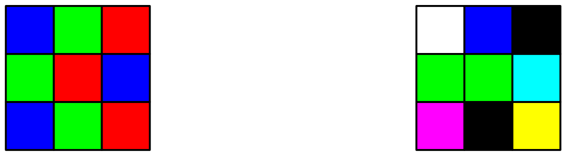
Another desirable halftoning property is perceived homogeneity, not just for pixels of a given color plane, but also for the arrangement of pixels collectively from all planes. The term "grain" as an image quality degradation historically referred to the appearance of silver halide crystals in analog photographs adding a texture that is not present in the image but in the rendering process. In dispersed-dot color halftoning methods, such as error-diffusion, there are many ways to arrange the binary pixels to render a target color; arrangements where the texture is more visible are described as having higher grain. In this paper, we quantify grain as the percent of



(a) Input image of 9 pixels with a gray value of $\frac{1.0}{9} = 0.11$, or $(R, G, B) = (0.11, 0.11, 0.11)$.

(b) Output image with zero Bias and zero Grain. $(R_{avg}, G_{avg}, B_{avg}) = (\frac{1.0}{9}, \frac{1.0}{9}, \frac{1.0}{9}) = (0.11, 0.11, 0.11)$.

(c) Output image with zero Bias and $\frac{1}{9} = 11\%$ Grain. $(R_{avg}, G_{avg}, B_{avg}) = (\frac{1.0}{9}, \frac{1.0}{9}, \frac{1.0}{9}) = (0.11, 0.11, 0.11)$.



(d) Output image with $\frac{|3-1|+|3-1|+|3-1|}{9} = 67\%$ Bias and zero Grain. $(R_{avg}, G_{avg}, B_{avg}) = (\frac{1.0\cdot3}{9}, \frac{1.0\cdot3}{9}, \frac{1.0\cdot3}{9}) = (0.33, 0.33, 0.33)$.

(e) Output image with $\frac{|3-1|+|5-1|+|4-1|}{9} = 100\%$ Bias and $\frac{4}{9} = 44\%$ Grain. $(R_{avg}, G_{avg}, B_{avg}) = (\frac{1.0\cdot3}{9}, \frac{1.0\cdot5}{9}, \frac{1.0\cdot4}{9}) = (0.33, 0.56, 0.44)$.

Figure 2. The Bias and Grain metrics applied to a 3×3 input image of RGB three planes.

pixels where unwanted plane overlap occurs, or more specifically, pixels of colors of yellow, cyan, magenta, and white for RGB images,

$$Grain = \frac{number_{yellow} + number_{cyan} + number_{magenta} + number_{white}}{total\ number\ of\ pixels}.$$

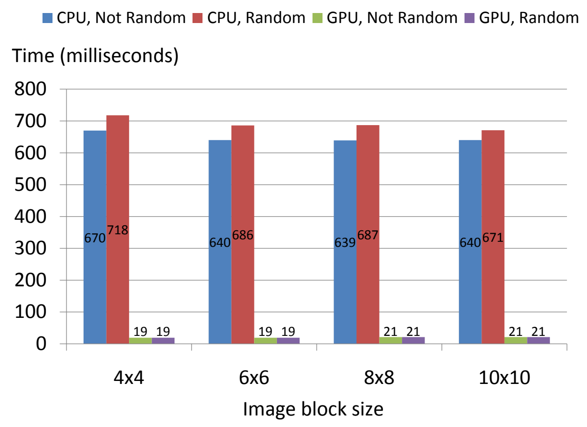
As a limitation, this Grain metric is only applicable to relatively dark gray values, because it does not distinguish between the intended and the unnecessary colorant overlaps for light gray values.

Although we illustrate the Bias and Grain metrics for three-plane RGB images, we should note that these two metrics are applicable to any number of planes as well. Figure 2 illustrates the Bias and Grain metrics applied to the 3×3 input RGB image in Figure 2(a). Figure 2(b) represents an ideal output that preserves the mean color and minimizes the overlap of color dots, while Figures 2(c)–2(e) produce worse image quality with either a high Bias or a high Grain.

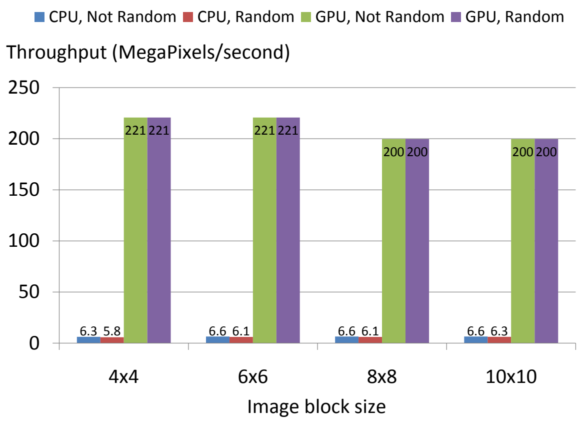
3. PERFORMANCE RESULTS

Our test platform uses a 3.2 GHz Intel Core i5 dual-core CPU, a GTX 560 graphics card with 1 GB video memory, CUDA 4.0, and the Windows 7 operating system. In our previous work,[2] we implemented the block-based pinwheel error diffusion for single-plane (black and white) images using CUDA,[9] where we map each image block to one CUDA thread, and thus a number of image blocks to one CUDA block. We integrate the plane-dependent technique into our previous single-plane GPU implementation in a straightforward way by generalizing a grayscale value to a multi-plane pixel object using C++ classes. To obscure the visual artifacts of repeating patterns, we use randomized error filter weights generated by a very simple multiply-with-carry[10] random number generator.

Figure 3 shows the GPU and CPU performance comparison in terms of timing and pixel processing throughput for various image block sizes. The CPU implementation runs 4 threads on a dual-core processor (2 threads per core) using OpenMP, which we find to be faster than running 1 thread per core. In all cases, our GPU implementation gets a speedup of $30 - 37\times$ over the CPU. Compared with our previous GPU-based single-plane implementation,[2] the RGB multi-plane implementation takes about $3\times$ longer, which is reasonable, as the

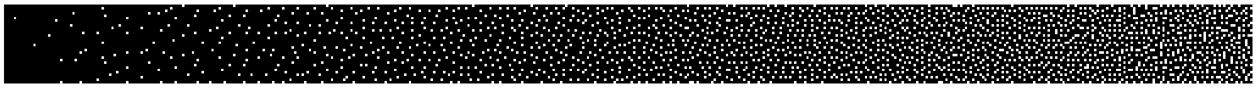


(a) Timings.

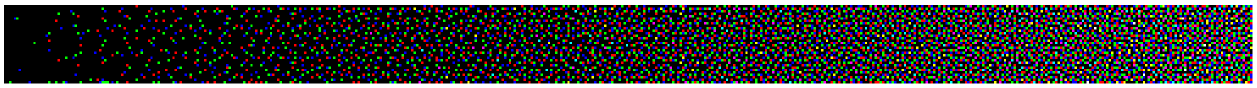


(b) Throughput.

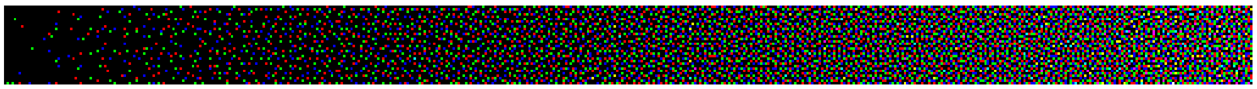
Figure 3. Performance comparison between the GPU and CPU for a 2048×2048 input image with various image block sizes. Random/Not Random indicates if error filter weights are randomized.



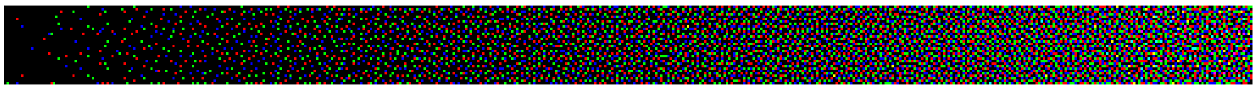
(a) Set all inter-plane coefficients to 0. $Bias = 0.2\%$. $Grain = 10.4\%$.



(b) Set all inter-plane coefficients to 0.1. $Bias = 0.07\%$. $Grain = 2.4\%$.



(c) Set all inter-plane coefficients to 0.2. $Bias = 0.03\%$. $Grain = 1.1\%$.



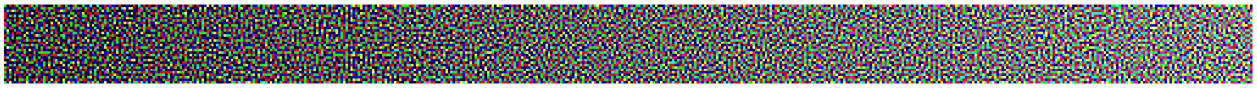
(d) Set all inter-plane coefficients to 0.3. $Bias = 0.05\%$. $Grain = 0.5\%$.

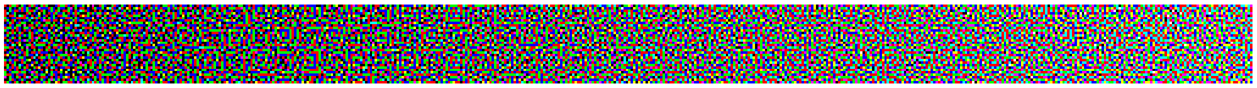
Figure 4. Larger inter-plane error-weighting coefficients help reduce dot-on-dot printing on a gray ramp in range $[0, 0.25]$. Image size: 512×32 pixels. The pinwheel image block size: 10×10. Without randomized intra-plane error filter weights.

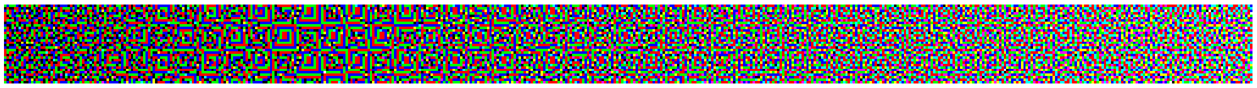
memory footprint increases by 3×, and the performance is memory-bound according to our previous analysis.[2] A memory-bound program also gives us opportunities to get the image quality benefits for free by adding arithmetic complexities for inter-plane dependency and randomization, noting that the randomization does not affect the GPU performance at all, as shown in Figure 3. In contrast to our previous results on single-plane error diffusion,[2] multi-plane error diffusion shows similar performance for varied image block sizes. This is because multi-plane pixels require strided memory access and are now the dominant source of memory uncoalescing, which diminishes the impact of block sizes on memory efficiency.

4. DISCUSSION ON IMAGE QUALITY

We carry out a number of experiments to study plane-dependent error diffusion, as well as the randomization on error filter weights. Figure 4 shows an example of applying the plane-dependent technique to a gray ramp in range $[0, 0.25]$ with varied inter-plane error-weighting coefficients. Figure 4(a) is generated using zero inter-plane coefficients, which is equivalent to independently applying error diffusion to three planes, and thus leads to the worst-case scenario where all red, blue, and green dots are overlapped. By increasing the inter-plane coefficients,


(a) Set all inter-plane coefficients to 0.2.


(b) Set all inter-plane coefficients to 0.3.


(c) Set all inter-plane coefficients to 0.4.

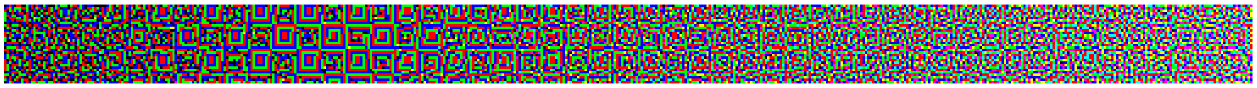
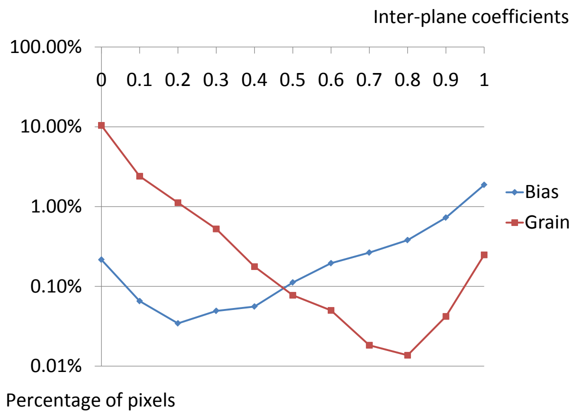
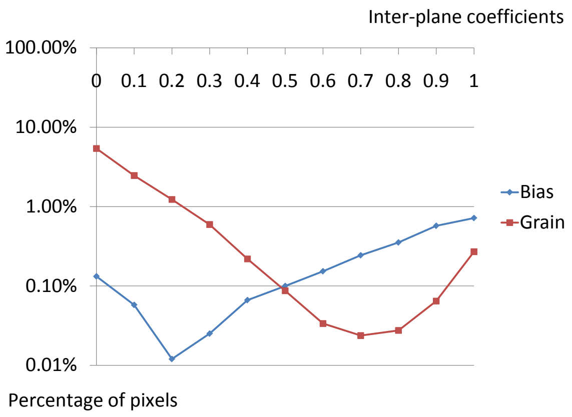

(d) Set all inter-plane coefficients to 0.5.

Figure 5. Too-large inter-plane coefficients leads to "square" artifacts in the gray ramp in range $[0.25, 0.5]$. Image size: 512×32 pixels. The pinwheel image block size: 10×10. Without randomized intra-plane error filter weights.



(a) Without randomized intra-plane error filter weights.



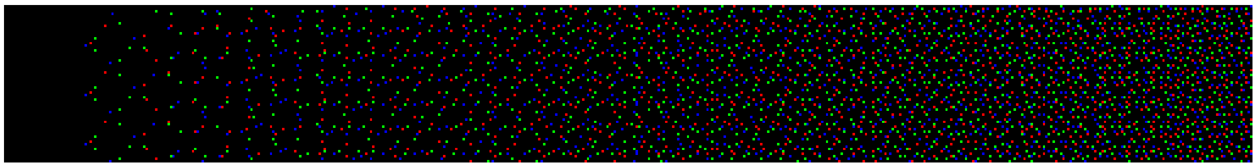
(b) With randomized intra-plane error filter weights..

Figure 6. Image quality metrics with varied inter-plane coefficients with and without randomized intra-plane error filter weights. Input image is a gray ramp in range $[0, 0.25]$. Image size: 512×32 pixels. The pinwheel image block size: 10×10.

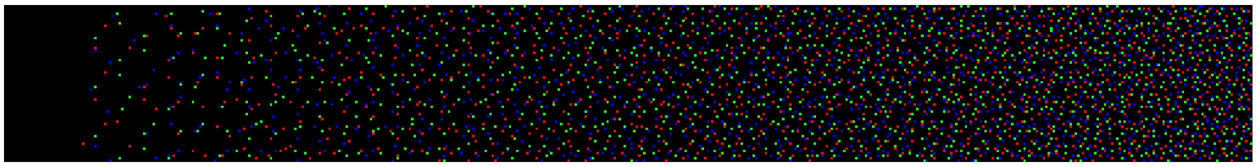
Figure 4(b)–4(d) gradually reduce the overlap of color dots, namely the colors of white, yellow, magenta, and cyan.

We also quantitatively count the overlap of color dots and evaluate the mean color of output pixels in terms of the Bias and Grain metrics. As indicated in Figure 6, a coefficient of 0.2 significantly reduces colorant overlap from 10% to 1%. Although even larger coefficients continue to improve Grain, the output images become less attractive because of increasing Bias. Moreover, as shown in Figure 5, large coefficients introduce "square" visual artifacts in the brighter ramp range $[0.25, 0.5]$, probably due to the out-of-control error propagation along the "pinwheel" scan path.

We use randomized error filter weights to break the periodic patterns, which are largely artifacts of the pinwheel algorithm and accentuated by the small pinwheel image block size. Figure 7 illustrates the effects of this technique on a larger 128×512 gray ramp in range $[0, 0.0625]$. Figure 8 and Figure 9 show the images generated using zero inter-plane coefficients without and with randomized error filter weights. Although the randomization we employ is designed to affect only dot placement, not colorant overlap, it has limited ability to reduce colorant overlap, exemplified in the ramp region in Figure 9 compared to that in Figure 8. After visually



(a) Without randomized intra-plane error filter weights.



(b) With randomized intra-plane error filter weights.

Figure 7. Randomized intra-plane error filter weights help break periodic structures in a 512×64 halftone ramp in range $[0, 0.0625]$. Inter-plane coefficients value: 0.2. The pinwheel image block size: 10×10.

comparing images generated with varied inter-plane coefficients, we select Figure 10, which represents the best visual image quality using a coefficient value of 0.2 and randomized error filter weights.

5. CONCLUSION

We have demonstrated a plane-dependent technique applied to GPU-based error diffusion for color images. To minimize the overlap of colorants, we carefully select inter-plane error-weighting coefficients by visual image appearance, as well as by using the quantitative metrics we designed. We further improve the image quality using randomized error filter weights.

We see several directions for future work: (1) in addition to RGB images, apply the plane-dependent technique to CMYK images; (2) define a metric to measure colorant overlap for general images, given that our current Grain metric is only applicable to a relatively dark gray ramp; (3) develop tone-dependent inter-plane coefficients to further improve image quality.

Acknowledgments

We wish to thank Jan Allebach for helping investigate the tone-dependent algorithm, Jason Mak for carefully proofreading and editing the paper, and Yangzihao Wang for computer support on performance testing. Thanks also to our funding agencies, the HP Labs Innovation Research Program, the National Science Foundation (Award 0541448), and the SciDAC Institute for Ultrascale Visualization, and to NVIDIA for equipment donations.

REFERENCES

1. Floyd, R. and Steinberg, L., "An adaptive algorithm for spatial grey scale," in [*Digest of the Society of Information Display*], 36–37 (1976).
2. Zhang, Y., Recker, J. L., Ulichney, R., Beretta, G. B., Tastl, I., Lin, I.-J., and Owens, J. D., "A parallel error diffusion implementation on a GPU," in [*Proceedings of SPIE: IS&T/SPIE Electronic Imaging 2011 / Parallel Processing for Imaging Applications*], **7872**, 78720K:1–9 (Jan. 2011).
3. Deshpande, A., Misra, I., and Narayanan, P. J., "Hybrid implementation of error diffusion dithering," in [*International Conference on High Performance Computing*], (2011).
4. Trager, B., Wu, C. W., Stanich, M., and Chandu, K., "GPU-enabled parallel processing for image halftoning applications," in [*2011 IEEE International Symposium on Circuits and Systems (ISCAS)*], 1528–1531 (May 2011).
5. Li, P. and Allebach, J. P., "Block interlaced pinwheel error diffusion," *Journal of Electronic Imaging* **14**(2), 1–13 (2005).
6. Metaxas, P. T., "Optimal parallel error-diffusion dithering," in [*Proceedings of SPIE*], **3648**, 485–494 (1999).

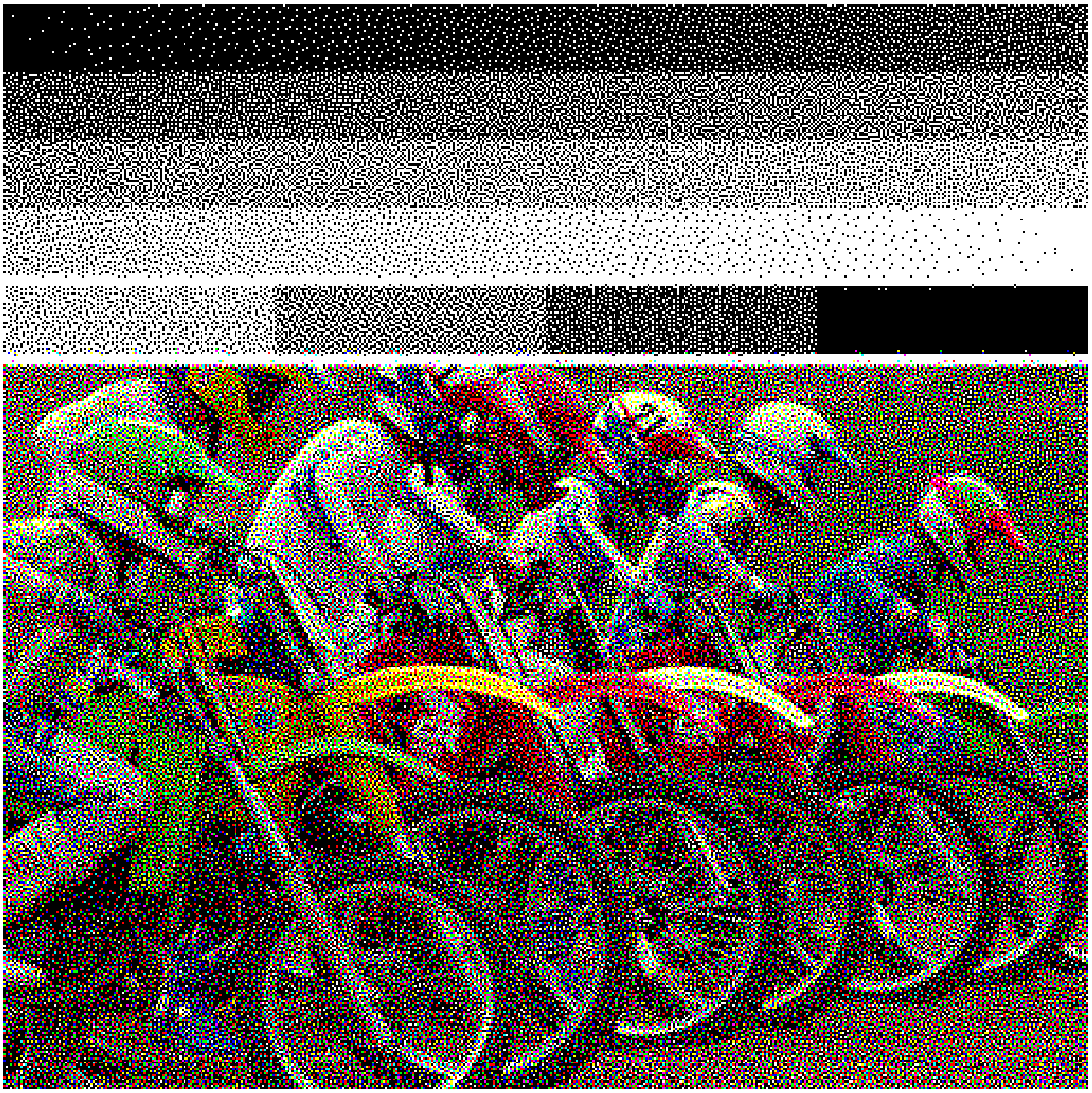


Figure 8. A 512×512 halftone image generated using a inter-plane coefficient value of 0. Without randomized intra-plane error filter weights. The pinwheel image block size: 10×10. Note the high grain due to plane-on-plane overlap, particularly in the gray ramp, and the periodic structures that follow the image block size.

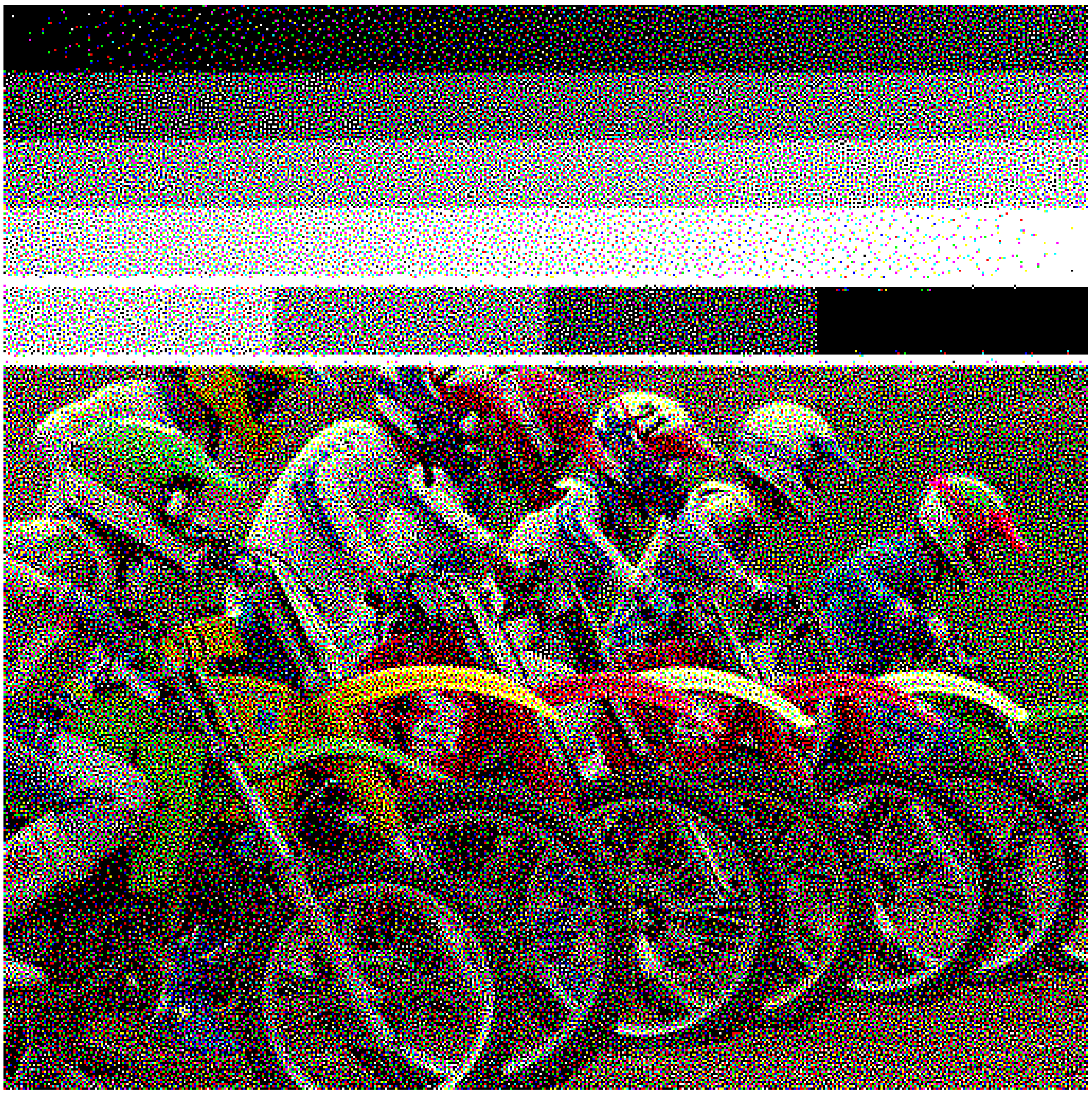


Figure 9. A 512 × 512 halftone image generated using a inter-plane coefficient value of 0. With randomized intra-plane error filter weights. The pinwheel image block size: 10 × 10. The randomization reduces block-shape periodic patterns, but a high grain is still visible.

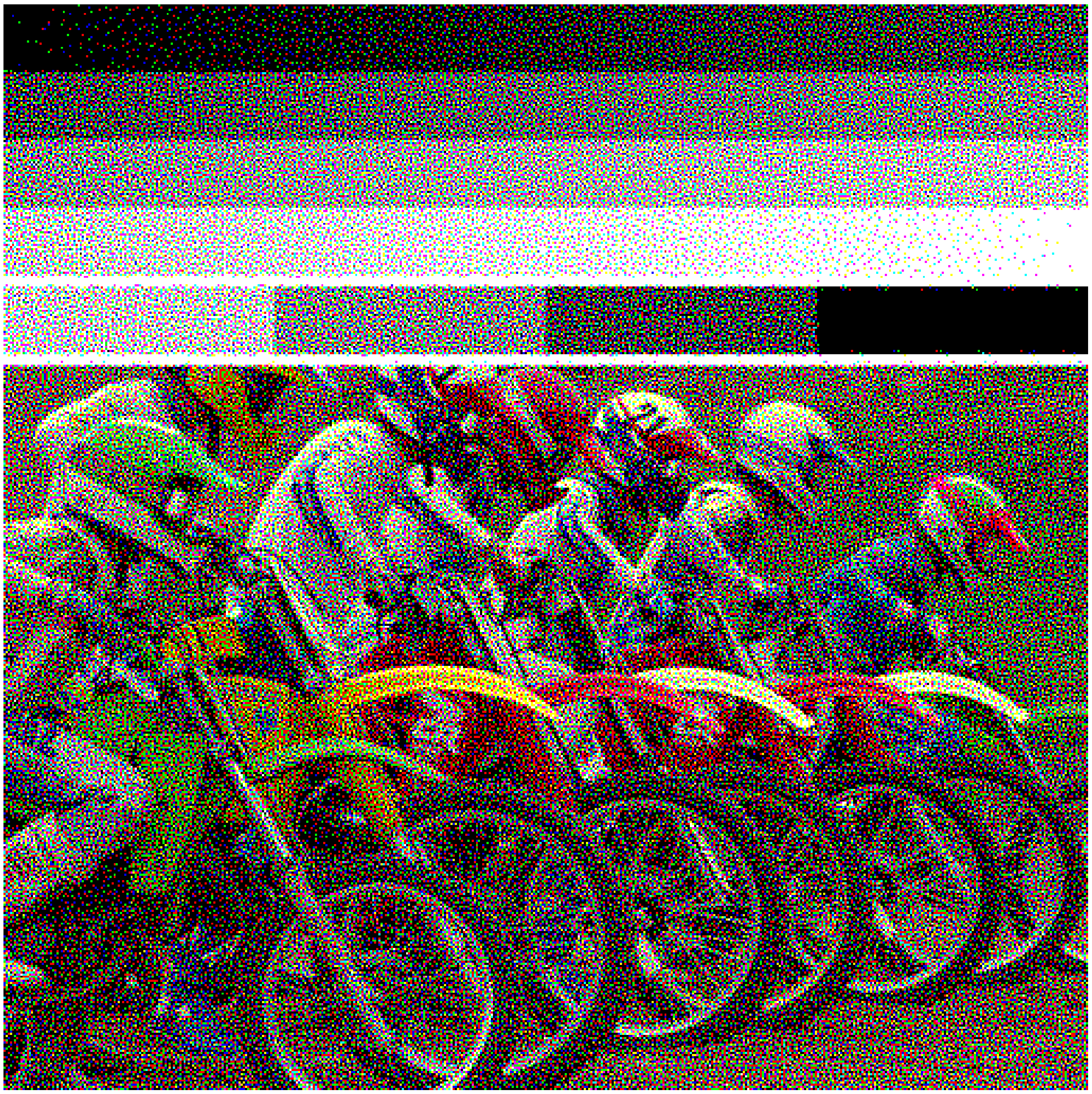


Figure 10. A 512×512 halftone image generated using a inter-plane coefficient value of 0.2. With randomized intra-plane error filter weights. The pinwheel image block size: 10×10. Both grain and block-shape periodic patterns are mitigated.

7. Lee, J.-H., Schramm, M. T., and Quintana, J. M., "Sequential color error diffusion with forward and backward exchange of information between color planes," (10 2008). US Patent 7,440,140.
8. Ulichney, R., "Dithering with blue noise," *Proceedings of the IEEE* **76**(1), 56–79 (1998).
9. NVIDIA Corporation, "NVIDIA CUDA compute unified device architecture, programming guide," (2011). `http://developer.nvidia.com/`.
10. Marsaglia, G. and Zaman, A., "A new class of random number generators," *The Annals of Applied Probability* **1**(3), 462–480 (1991).