Halftoning Over a Hexagonal Grid

Pierre-Marc Jodoin and Victor Ostromoukhov

Université de Montréal
Montréal, Canada

ABSTRACT

In this contribution, we present an optimal halftoning algorithm that uniformly distributes pixels over a hexagonal grid. This method is based on a slightly modified error-diffusion approach presented at SIGGRAPH 2001. [1] Our algorithm's parameters are optimized using a simplex downhill search method together with a *blue noise* based cost function. We thus present a mathematical basis needed to perform spectral and spatial calculations on a hexagonal grid. The proposed algorithm can be used in a wide variety of printing and visualization tasks. We introduce an application where our error-diffusion technique can be directly used to produce clustered screen cells.

Keywords: halftoning, error-diffusion, hexagonal grid, blue noise, Fourier transform.

1. INTRODUCTION

1.1. Motivations

Digital halftoning is a technique for rendering images with a wide dynamic range on devices having a limited number of available intensity levels. Driving color and grayscale printers is a typical application of digital halftoning. Most modern laser and inkjet printers possess a limited number of available output intensity levels, whereas the input signal may be considered as continuous-tone. In most applications, the input and output signals are sampled on a square or rectangular grid. It is for this reason that most research in digital halftoning has been focused on halftoning using square (or orthogonal) grids. Only a small amount of work has been done on hexagonal grids. [2–4]

Traditional digital halftoning using orthogonal grids has made tremendous progress during the last ten years. A number of algorithms have been proposed that considerably increase the visual quality of the produced images. [1, 5–7] At the same time, none of the algorithms mentioned above can be easily reused with satisfactory results on hexagonal grids.

In the present contribution, we will try to fill this gap. We introduce an error-diffusion algorithm with optimized coefficients that produces decent output on a hexagonal grid. We base our algorithm essentially on the work made by Ostromoukhov[1] and reusing ideas derived from other contributions. [5, 6, 8, 9]

The motivations for our work were two-fold:

– first, from a theoretical point of view, generalization of existing halftoning algorithms to a hexagonal grid will lead to a better understanding of the fundamental digital halftoning algorithm – which is far from obvious;

– second, physical devices with a hexagonal organization of visualization elements do exist (e.g. displays with hexagonal organization of RGB spots, inkjet printers with hexagonal organization of elementary ink drops). It may be more appropriate to drive such entities directly, without passing through the intermediate orthogonal structure. An example of such application will be presented in section 4.

Further author information:
Pierre-Marc Jodoin's e-mail: jodoinp@iro.umontreal.ca
Victor Ostromoukhov's e-mail: ostrom@iro.umontreal.ca

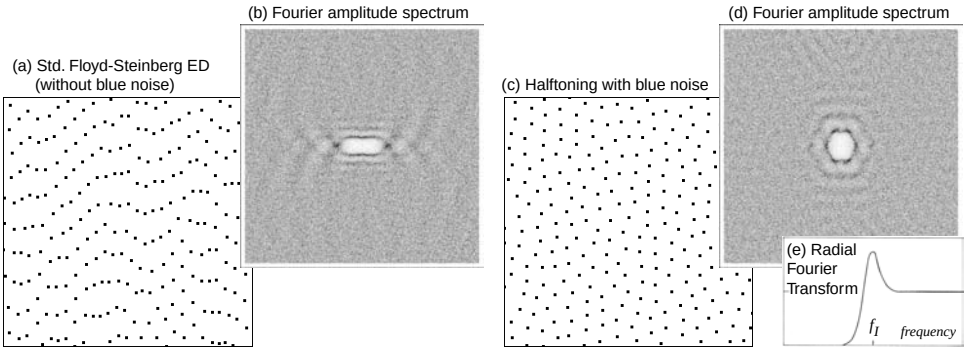

Figure 1. This figure illustrates the visual properties of blue noise: a light gray uniform patch halftoned with the scanline-path Floyd-Steinberg E-D algorithm (left) and with the algorithm proposed in[1] (right). The Radial Fourier Transform (blue noise profile) was taken from Ulichney's book[10] .

1.2. Error-Diffusion

Error-diffusion is a widespread binarization algorithm that is mostly used to render images on black-and-white devices especially in the case where the preservation of fine image detail is needed.[11] This algorithm is a fairly good compromise between speed, simplicity and visual quality.

The way error-diffusion works is simple. First, every input pixel $f(x, y)$ is selected one after the other. These pixels have a value ranging between 0 and 1 and the order in which they are processed depends on the chosen *path* (usually *serpentine* or *scanline*). When a pixel $f(x, y)$ is selected, it is compared to a fix *threshold* t that is generally set at 0.5. Whenever $f(x, y)$ is larger than t, the value "1" is assigned to the output image $g(x, y)$, otherwise it is set to "0". This output value is then compared to the input intensity level and put their difference in the error image $e(x, y)$. The error is finally "distributed" to the neighboring pixels according to a given coefficient set similar to the one presented in Figure 3(a). The basic error-diffusion algorithm using a *scanline* path and the Floyd-Steinberg (FS) coefficient set is presented in Table 1.

Table 1. Classical error-diffusion algorithm processing the pixels in a scanline order using Floyd-Steinberg (FS) coefficient set.

Function ERROR_DIFFUSION(f[.],t): . Create output image g[.] . Create error image e[.] . **For** each pixel P of image f[.] on a scanline path **Do** . **If** f[P] $>$ Threshold **Then** . g[P] $\leftarrow$ white . **Else** . g[P] $\leftarrow$ black . e[P] $\leftarrow$ (f[P]-g[P]) . DISTRIBUTE_ERR_FS_SCANLINE(e[P],f[.],P) . **return** output image g[.]	**Function** DISTRIBUTE_ERR_FS_SCANLINE(E,f[.],P): . f[pixel at the right of P] += $E \times 7/16$. f[pixel at the bottom right of P] += $E \times 1/16$. f[pixel under P] += $E \times 5/16$. f[pixel a the bottom left of P] += $E \times 3/16$

1.3. Blue Noise

Let us consider the problem of processing with a halftoning algorithm, an input image $f(x, y)$ of size $X_s \times Y_s$ where each pixel is assigned a **constant** intensity level $f(x, y) = I \in [0, 1]$. The goal of any halftoning algorithm is to generate a bi-level output image $g(x, y)$ that *approximates* best the input image. The output must consequently contain $X_s \times Y_s \times I$

Table 2. Pseudo-code used to calculate the radial Fourier transform (RFT) of a two dimensional Fourier amplitude spectrum (G[.]).

```
Function COMPUTE_RFT(G[.]):
.   Create a one-dimensional array RFT[.]
.   (X_c, Y_c) ← the center point of G[.]
.   For each pixel P of image G[.] Do
.       (X_p, Y_p) ← the coordinates of pixel P
.       dst ← sqrt((X_p − X_c)^2 + (Y_p − Y_c)^2)
.       RFT[dst] += G[P]
.   RFT[.] ← NORMALIZE(RFT[.])
.   Return RFT[.]
```

white pixels and $X_s \times Y_s \times (1 - I)$ black pixels in order to have an average intensity level I. Each of the white pixels covers an area of $\frac{1}{I}\text{pixel}^2$ and the average distance between each other is equivalent to

$$\lambda_I = \begin{cases} \frac{1}{\sqrt{I}}, & \text{if } I \leq 0.5 \\ \frac{1}{\sqrt{1-I}}, & \text{otherwise.} \end{cases} \tag{1}$$

Since these pixels are assumed to be uniformly distributed, λ_I induces a *principal frequency* $f_I = 1/\lambda_I$[12, 13] clearly visible in the frequency domain (see Figure 1 (d) and (e)). This kind of frequency shape with a peak over the principal frequency and radial symmetry is called the **blue noise** shape. As presented in Figure 1(e), the blue noise spectrum is often plotted as a one-dimensional profile function, often referred to as *Radial Fourier Transform*(RFT). The algorithm that computes RFT from $G(\alpha, \beta)$, the Fourier transform of the bi-level output image $g(x, y)$, is given by the pseudo-code in Table 2.

For the remainder of this article, after the problem statement in the next section, we will discuss the proposed algorithm along with its mathematical background and the optimization process in section 3. Section 4 will propose some possible applications for our algorithm while section 5 will comment on some results and section 6 will draw conclusions.

2. PROBLEM STATEMENT

The method presented in this paper was built upon the generic concept of a **lattice**. We define a lattice as being a two-dimensional array of discrete points distributed over a coordinate system $(\vec{v}_1, \vec{v}_2)$ where $\vec{v}_1$ and $\vec{v}_2$ are unitary base vectors separated by a non-zero angle θ. In this contribution, we call these points **lattice points**. By their very nature, these points cover no surface area and are labeled by a coordinate pair (u, v). The Cartesian coordinates of a lattice point defined by its integer indices u and v are given by

$$\Gamma(u, v) = u\vec{v}_1 + v\vec{v}_2 \tag{2}$$

where $\Gamma(.)$ is the conversion function. By using the well known *Delaunay triangulation* algorithm, we can subdivide the lattice into a set of triangles and find its dual structure called the *Voronoi diagram*.[14] From the basic theory of computational geometry, we know that whenever all *Delaunay triangles* are equilateral, as in our case, the *Voronoi regions* are nothing other than perfect *hexagonal cells*. In the context of this paper, we give these cells the name **hexagonal pixels**. An example of this Voronoi diagram is presented in Figure 2.

Let us now consider a continuous input function $h(i, j)$ defined in the two-dimensional space R^2 where $h(i, j) \in R$. This function can be sampled over a lattice in such a way that each lattice point is associated with the function value at that point:

$$f(u, v) = h(\Gamma(u, v)) \tag{3}$$

where $f(u, v)$ is the sampled function.

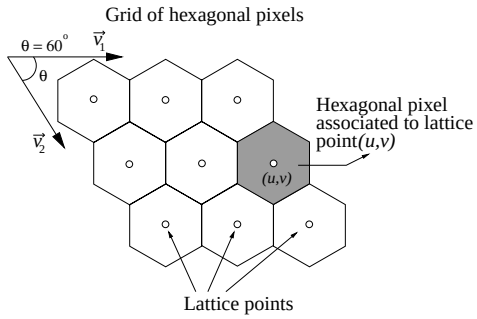

Figure 2. View of a hexagonal grid. Such a structure is a particular Voronoi diagram since each Voronoi region is a perfect hexagonal pixel.

It is well known in the halftoning community that a sampled function such as $f(u,v)$ can be *approximated* by a bi-level function $g_h(u,v)$. More precisely, a halftoning method refers to the process of representing a sampled function such as $f(u,v)$ over a limited number of values, typically set to 0 and 1. The problem we solve in this contribution can therefore be formulated this way: from a two-dimensional function $f(u,v)$ sampled over a hexagonal lattice, generate a bi-level function $g_h(u,v)$ that best approximates $f(u,v)$. We consider that $g_h(u,v)$ approximates $f(u,v)$ well, whenever the following two requirements are respected:

1. The local intensity of $g_h(u,v)$ integrated over a small region equals $f(u,v)$. In other words,

$$f(u,v) = \int_{u-\epsilon}^{u+\epsilon} \int_{v-\epsilon}^{v+\epsilon} g_h(u,v)dudv \tag{4}$$

   where ϵ is a constant as large as a few hexagonal pixels.

2. If $f(u,v) = C$ for all values of u and v where C is a constant, the binary pixels of $g_h(u,v)$ must be homogeneously distributed in space. In other words, $g_h(u,v)$ should have spectral **blue noise** characteristics as defined above.

To reach these requirements, we decided to use a modified version of the error-diffusion halftoning algorithm previously introduced.

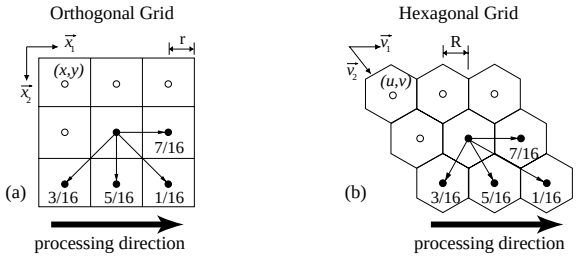

Figure 3. Example of the Floyd-Steinberg algorithm applied to an orthogonal grid (a) and to a hexagonal grid (b). The affine transformation between the orthogonal grid and the hexagonal grid is made by the bijection operator $\tau(.)$.

3. PROPOSED ALGORITHM

3.1. Model Description

As we stated previously, with the help of an error-diffusion algorithm, our goal is to compute a halftone image $g_h(u,v)$ that approximates a function $f(u,v)$ sampled over a hexagonal grid. Unfortunately, the basic error-diffusion algorithm

presented in section 1.2 was not designed to work over hexagonal grids and thus generates disturbing artifacts. To solve that problem, we use a *variable-coefficient approach*[1,9] which proposes to use 256 coefficient sets, one for each of the 256 intensity levels. Furthermore, the serpentine path was adopted to reduce directional artifacts mainly visible in the shadows and the highlights.[15] The complete pseudo-code for this method using the notation of section 1.2 can be found in Table 3. Note that the 256 coefficient sets used in function `DISTRIBUTE_ERROR(.)` will be further optimized by minimizing a given **cost function** built upon the blue noise constraint in order to meet section 2's requirements. That cost function will be described in detail in section 3.2.

Table 3. Pseudo-code of the error-diffusion algorithm used to distribute pixels over a hexagonal grid. Note that there is one coefficient set for each of the 256 intensity levels. Since the pixel value $f[P]$ was sampled between 0 and 1, its multiplication by 255 gives an index corresponding to the appropriate coefficient set in the "Coeffs[.]" array.

Function `ED_HEXA`(f[.],Threshold,Coeffs[.]): . Create output image g[.] . Create error image e[.] . **For** all pixels P of image f[.] on a . serpentine path **Do** . **If** f[P] $>$ Threshold **Then** . g[P] $\leftarrow$ white . **Else** . g[P] $\leftarrow$ black . e[P] $\leftarrow$ (f[P] - g[P]) . `DISTRIBUTE_ERROR`(e[p],f[.],P,Coeffs[.]) . **return** output image g[.]	**Function** `DISTRIBUTE_ERROR`(Error,f[.],P,Coeffs[.]): . CS $\leftarrow$ select one coefficient set Coeffs[255*f[P]] . **If** processing direction goes from Left to Right **Then** . `DISTRIBUTE_ERROR_L_R(CS,Error,f[.],P)` . **Else** . `DISTRIBUTE_ERROR_R_L(CS,Error,f[.],P)`

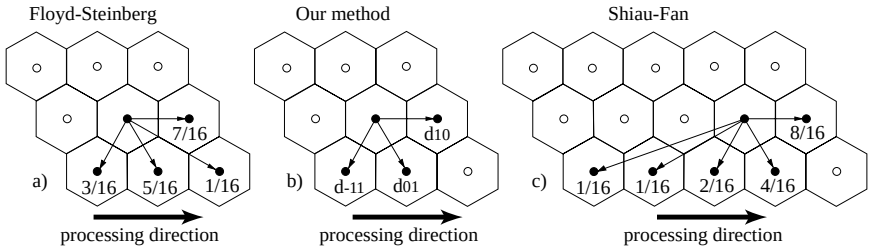

Figure 4. This figure shows how the error is distributed over the hexagonal pixels with Floyd-Steinberg's method, our method and Shiau-Fan's method.

3.2. Optimization Process

In the previous section, an error-diffusion algorithm (`ED_HEXA(.)`) was proposed to uniformly distribute pixels on a hexagonal grid. However, in order to reach the *blue noise* requirement presented in section 2, we need to optimize the 256 coefficient sets. To do so, we implemented a simplex downhill method[16] in combination with a blue noise based **cost function** and made it converge toward optimal coefficient values. This kind of method to find optimal coefficient weights has been used several times in the past.[1,6,17] As shown in Figure 5, the function `COST_FUNCTION`(I,set) computes the area between $G_h(\alpha,\beta)$'s radial Fourier transform $RFT(f)$ and a given analytical function $F(f)$. It was found that for $F(f)$, a normalized Gaussian function with standard deviation $\sigma = \frac{f_g}{10}$ was a good compromise between simplicity and result quality.[17] The cost function algorithm is presented in Table 4.

As suggested by Ostromoukhov,[1] we did not optimise every 256 coefficient sets. To preserve stability of the whole optimization process, we only optimized a few *key levels* and linearly interpolated the coefficients between these specified

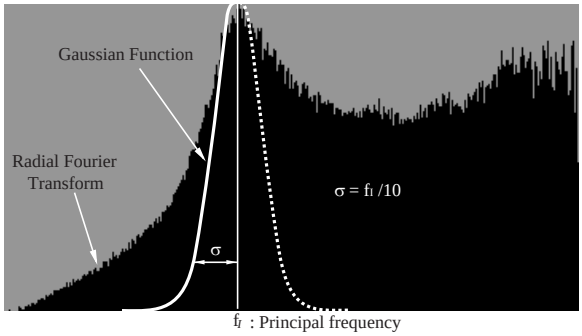

Figure 5. To calculate its error value, the function COST_FUNCTION(I, set) first computes the hexagonal halftone image $g_h(u,v)$ of average intensity "I". It then computes a radial Fourier transform $RFT(f)$ and calculates a Gaussian function $F(f)$ centered on principal frequency f_I.[8,12] $F(f)$'s standard deviation is $\sigma = \frac{f_I}{10}$ and the cost is the area between $F(f)$ and $RFT(f)$: $cost = \int_{f=0}^{f_I} ||RFT(f) - F(f)||df$

Table 4. Cost function used within the simplex downhill search method to find optimal coefficient sets

Function COST_FUNCTION(I,set):
. $f[x,y] \leftarrow$ Create a flat image of intensity I
. $g_o[x,y] \leftarrow$ ED_HEXA(f[.],0.5,set)
. $G_o[\gamma,\delta] \leftarrow$ FOURIER_TRANSFORM($g_o[x,y]$).
. $G_h[\alpha,\beta] \leftarrow \frac{\sqrt{3}R^2}{r} G_o[T(\gamma,\delta)]$
. $RFT(f) \leftarrow$ COMPUTE_RFT($G_h[\alpha,\beta]$)
. **If** $I < 0.5$ **Do**
. $\quad \lambda_I \leftarrow \frac{1}{\sqrt{I}}$
. **Else**
. $\quad \lambda_I \leftarrow \frac{1}{\sqrt{1-I}}$
. $\sigma \leftarrow \frac{\lambda_I}{10}$
. $f_I \leftarrow \frac{1}{\lambda_I}$
. $F[f] \leftarrow$ NORM_GAUSSIAN_FUNCTION(f_I,σ)
. $C[f] \leftarrow ||RFT[f] - F[f]||$
. cost $\leftarrow \int_o^{f_I} C[f]df$
. **return** cost

levels. Furthermore, because of error-diffusion's symmetry propriety above and under 0.5, we only optimized the first 128 coefficents and transposed it to the 128 others. The 128 optimized coefficient sets are presented in Table 6.

One aspect of the problem that was not tackled yet concerns the number of coefficients the coefficient sets must have. After trying many different configurations, we realized that the simple case with 3 coefficients (d_{10},d_{01} and d_{-11} as shown in Figure 4(b)) produces fairly good results. Furthermore, in contrast with the cases where more than 3 coefficients are used, the optimization process converges more rapidly and is less prone to fall into local minimas.

3.3. Mathematical Context

As presented in Figure 3(a), the error-diffusion algorithm was originally made to work over an **orthogonal lattice**. Also, it is more convenient to implement an orthogonal lattice in a computer program than a hexagonal lattice. For these reasons, we decided not to use ED_HEXA(.) during the optimization process to create the hexagonal halftone result $g_h(u,v)$. Instead, we used this algorithm on an orthogonal function $f_o(x,y)$ to create a bi-level output $g_o(x,y)$ also represented over an orthogonal grid. Once this last result is computed, its orthogonal lattice points are projected over

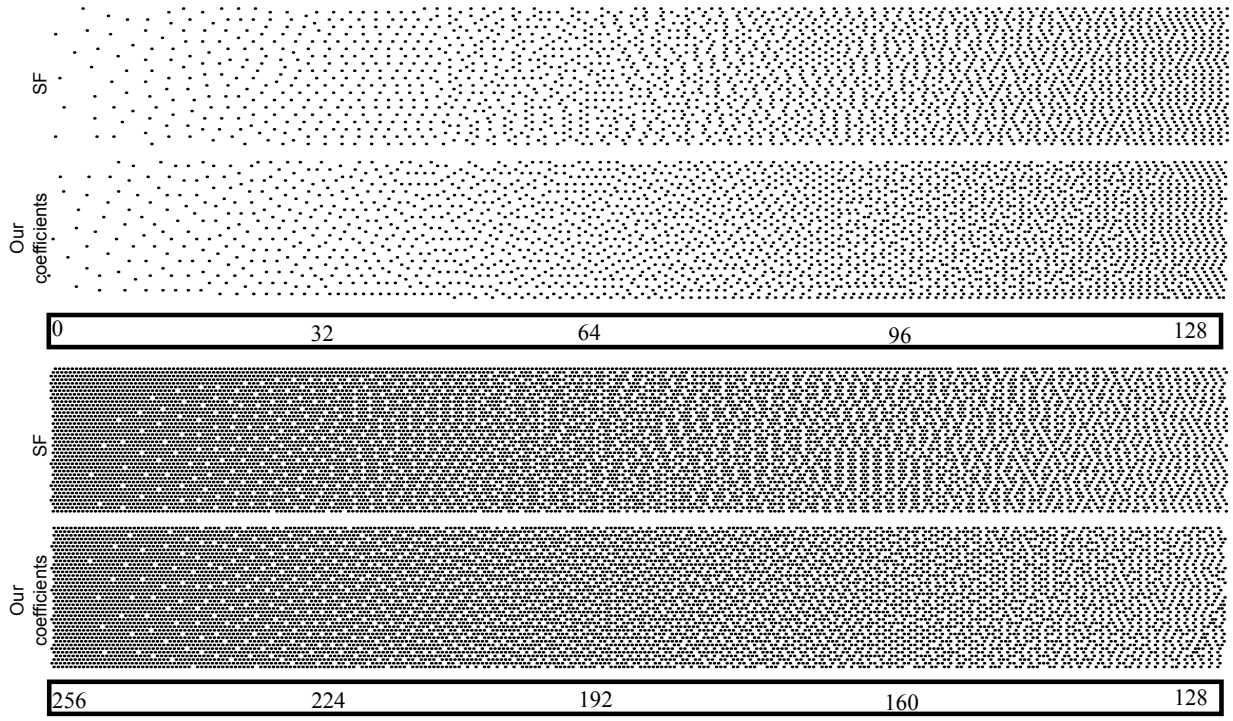

Figure 6. This figure represents a gray scale ramp sampled over a hexagonal lattice where each black dot stands for a lattice point. On top is the result of the `ERROR_DIFFUSION(.)` algorithm using a serpentine path and the well know *Shiau-Fan*(SF) coefficient set.[5] At the bottom is the result of the `ED_HEXA(.)` algorithm using our optimized coefficients (see Table 6). There is a significant difference between these two approaches especially around levels 64, 96 160, 192 and 224

$g_h(u,v)$'s hexagonal lattice points using a **bijection** operator $\tau(.)$ (see Figure 3 and 7). This operator has the general form $\tau(x,y) = (u,v)$. By using the coordinates of Figure 3(a) and (b), we find that for a given point (x,y) defined on an orthonormal basis, its projection to a hexagonal basis is:

$$\tau(x,y) = \left(\frac{R}{r}x + Ry, \sqrt{3}Ry\right) \tag{5}$$

where R is the hexagonal pixel half-size and r is the orthogonal pixel half-size as shown in Figure 3. Images $g_o(x,y)$ and $g_h(u,v)$ can be linked together by equation (5) in order to get

$$g_h(\tau(x,y)) = g_o(x,y). \tag{6}$$

In summary, during the optimization process, the error-diffusion method creates $g_o(x,y)$ and the bijection operator $\tau(.)$ maps this orthogonal grid to $g_h(u,v)$. To objectively optimize the method's parameters, we compute $g_h(u,v)$'s Fourier transform ($G_h(\alpha,\beta)$) and see how close it is to the blue noise shape. For a matter of convenience, even if $G_h(\alpha,\beta)$ could be computed by directly applying the Fourier transform to $g_h(u,v)$, we made the decision to convert the affine transformation $\tau(.)$ to the Fourier domain. As shown in [18] , the spatial affine transformation of equation (6) can be directly transposed to the frequency domain as follows:

$$G_h(\frac{r}{R}\gamma, \frac{\delta - \gamma r}{\sqrt{3}R}) = \frac{\sqrt{3}R^2}{r}G_o(\gamma,\delta) \tag{7}$$

where $G_o(\gamma,\delta)$ is the Fourier transform of $g_o(x,y)$. The spectral bijection **T**(.) can thus be represented by the following

equation:

$$\mathbf{T}(\gamma, \delta) = (\frac{r}{R}\gamma, \frac{\delta - \gamma r}{\sqrt{3}R}).$$

The relation between the two bijection operators τ and **T** is illustrated in Figure 7.

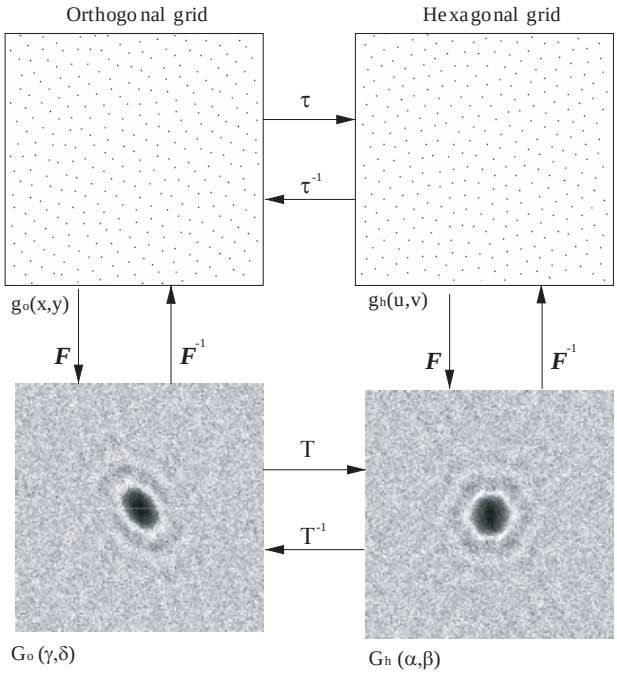

Figure 7. The following illustrates the process pipeline where the affine bijection $\tau(.)$ converts the orthogonal image $g_o(x, y)$ into the hexagonal one $g_h(u, v)$ and the spectral bijection **T**(.) converts the Fourier transform $G_o(\gamma, \delta)$ into $G_h(\alpha, \beta)$. **F**(.) is the Fourier transform function.

4. POSSIBLE APPLICATIONS

By its very nature, our method preserves small image details better than most concurrent approaches. [19] However, even if such a method is known for generating some of the most pleasing visual results, it is not suitable for a large variety of high resolution laser printers.[19,20] Mostly because of the **dot gain** effect, the individual dots dispersed over the output image by the error-diffusion algorithm are hardly printable. For that reason, laser printers are driven most of the time by *clustered-dot* halftone methods. For example, to work around the dot gain effect, many printer drivers take advantage of a **threshold matrix** with a clustered organization of its dots, hence the generic name *clustered-dot* for this kind of matrix (see Figure 8(a)).[15] Even if such a threshold algorithm is at the same time, fast, easy to implement and well suited for most laser printing devices, it experiences some serious limitations. First, in low resolution, it generates poor results since it filters all high frequency details.[19] Second, it can only generates a fix number of gray levels which can produce visible steps in some cases. For example, the matrix shown in Figure 8(a) can only generate 12 different intensity levels and thus induces unacceptable false contours such as those in Figure 10(a).

One solution to these limitations, is to join together the clustered-dot algorithm with the error-diffusion method. That way, we can expect that the advantages of one will minimize the drawbacks of the other. Even if similar ideas were already proposed in the past,[2,3,21] none of these approaches offers optimal coefficient sets. In this contribution, we apply our `ED_HEXA(.)` algorithm (in addition with the optimized coefficient sets) over a **quasi-hexagonal** grid such as the one in Figure 8(b). As Fan proposed it in[3] , instead of processing the individual pixels with the error-diffusion method, we

process the individual threshold clusters. We thus sequentially compute the local error cumulated over each cluster and redistribute it to its neighbors (see Figure 8(b)). To do so, we slightly modified the `ED_HEXA(.)` procedure to fit the quasi-hexagonal grid and called the new version `ED_QUASI_HEXA(.)` as presented in Table 5.

Table 5. Modified error-diffusion algorithm used over a quasi-hexagonal grid such as the one in Figure 8(b).

Function `ED_QUASI_HEXA`(f[.],Coeffs[.]):	**Function** `APPLY_THRESHOLD`(f[.],g[.],Cluster[.],error):
. Create output image g[.]	. totalError←0
. Create 2D Clustered Dot Threshold table Clt[.]	. **For** all pixels P in f[.] covered by Cluster[.] **Do**
. Create error buffer e[.]	. **If** f[P] + error > Cluster[P] **Then**
. **For** all cluster C of table Clt[.] on a	. g[p]← white
. serpentine path **Do**	. **Else**
. err = `APPLY_THRESHOLD`(f[.],g[.],Clt[C],e[C])	. g[p]← Black
. `DISTRIBUTE_ERROR`(err,e[.],f[.],Coeffs[.])	. totalError += f[p]-g[p]
. **return** output image g[.]	. **return** totalError/number of pixels visited

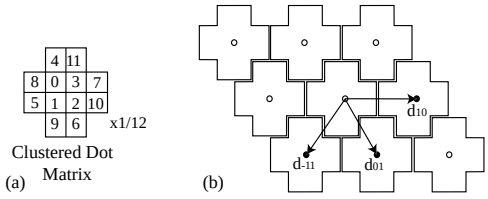

Figure 8. (a) A 12 level Clustered-dot threshold matrix and (b) the way error is distributed over a *quasi-hexagonal grid*.

5. RESULTS

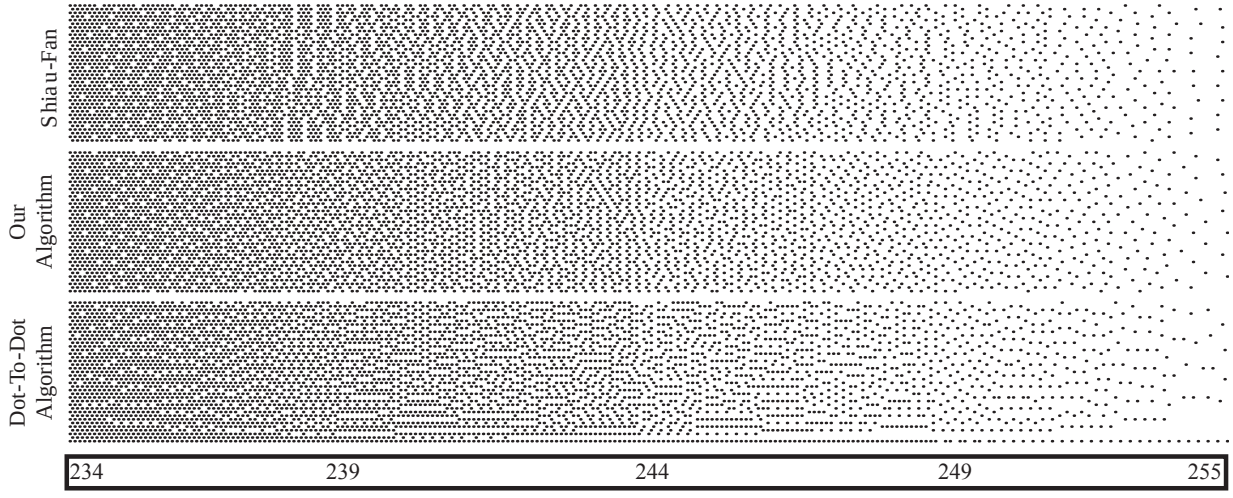

Figure 9. Result of 3 different methods applied on a *quasi-hexagonal* grid.

The final algorithm is fast, conceptually simple and memory efficient since it requires only a few operations per pixel. All computations needed for optimization of the ED coefficients are done once. These optimized coefficients are hardcoded into the final algorithm.

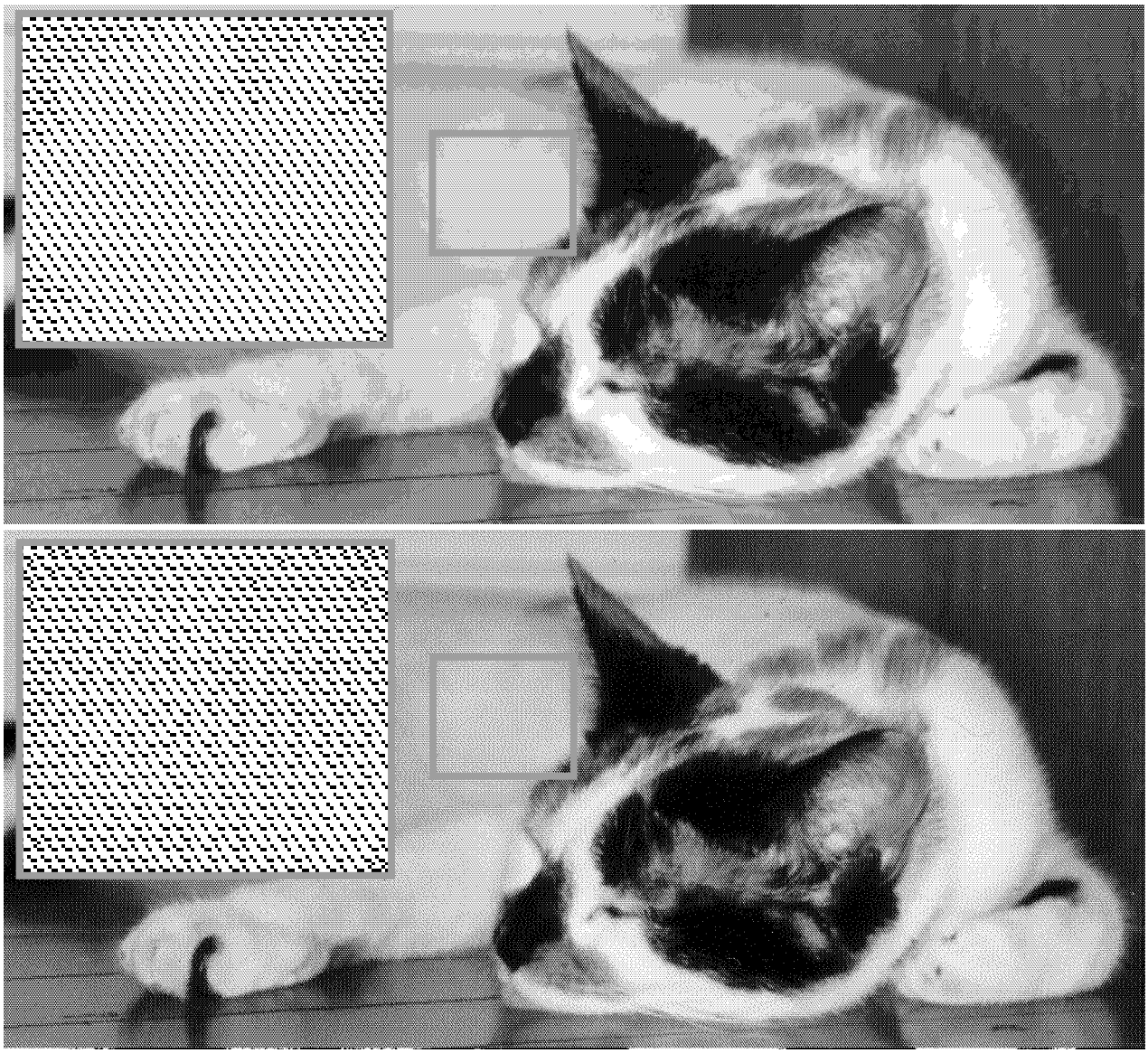

Figure 10. The first result was rendered by simply applying a 12 level *clustered-dot* matrix over the input image. The second one was computed with the combination of `ED_HEXA(.)` and the *clustered-dot* matrix. Note that the micro-structure is the same in both images.

The optimization process using both the simplex downhill search method and the function `COST_FUNCTION(.)` converged in less than two minutes on a 1.4 GHz Athlon processor for each *key levels*. We compared these results with others obtained with a *simulated annealing algorithm*[22] and saw that the results were very much similar at every key levels. That observation made us believe that even if the simplex method can theoretically fall into local minimas, in the present case it converges near the global minima. We thus made the decision to keep working with the simplex method, mainly because of its speed.

Concerning the `ED_HEXA(.)` procedure applied to a perfect hexagonal grid, results are shown in Figure 6. The reader can see what function `ERROR-DIFFUSION(.)` using *Shiau-Fan* coefficient set[5] and a serpentine path gives by opposition to `ED_HEXA(.)` using the optimized coefficient sets. We can see that the latter produces globally less artifacts

especially around levels 64, 96, 160, 192 and 224.

Finally, results of ED_HEXA(.) applied over a *quasi-hexagonal* grid (such as the one in Figure 8) are presented in Figure 9 and 10. Figure 9 shows the difference between ED_HEXA(.) using Shiau-Fan's coefficient set(SF), our approach and Fan's dot-to-dot algorithm.[3] We decided to put a grayscale ramp going from $\frac{232}{255}$ to $\frac{255}{255}$ because the clustered-dot matrix with 12 thresholds divides the $0-1$ scale into 12 sections of equal length ($\frac{21.3}{256}$) all having a similar configuration. For this reason, the artifacts shown between $\frac{232}{255}$ and $\frac{232}{255}$ are exactly the same between $\frac{211}{255}$ and $\frac{232}{255}$, $\frac{190}{255}$ and $\frac{211}{255}$, $\frac{158}{255}$ and $\frac{190}{255}$ and so on down to zero.

Figure 10 shows the striking difference between our approach and a straight 12 levels *clustered-dot* threshold. Among other things, we can see that our approach minimizes the false contours while preserving the fundamental regular clustered structure of the matrix. The two images were printed in 300 dpi.

6. CONCLUSION

In this paper, an optimal error-diffusion technique was introduced. This simple and fast method uniformly distributes pixels over a hexagonal grid. Instead of using the classical error-diffusion algorithm (ERROR-DIFFUSION(.)) we implemented a slightly modified version of that algorithm (ED_HEXA(.)). This method takes advantage of 255 coefficient sets, each being optimized in order to minimize artifacts around their corresponding intensity level.

The optimization process used a simplex downhill search method together with the blue noise based cost function COST_FUNCTION(.). During the optimization process, for a matter of simplicity, we made the decision not to use the hexagonal grid $g_h(u,v)$ directly. Instead, an orthogonal grid $g_o(x,y)$ was generated and converted to $g_h(u,v)$ using a *bijection* operator τ. By making use of the spectral *bijection* operator **T**, the function COST_FUNCTION(.) calculates the hexagonal grid Radial Fourier Transform (RFT) and measures how close it is to the blue noise profile. After many tests made over a large variety of images, it was found that a configuration with 3 coefficients was a good choice regarding simplicity, speed and better optimization efficiency.

By its conceptual simplicity and speed, our algorithm can be used to work over a variety of printing and visualization devices as illustrated in figure 10. It is thus possible to drive physical hexagonal entities directly, without having to pass through an intermediate orthogonal structure. We have shown that by mixing together clustered-dot matrices and ED_HEXA(.), the classical problems of dispersed dots can be considerably improved while reducing the false contours.

In the future, we expect to reduce the artifacts around level $\frac{1}{2}$ and $\frac{1}{3}$ and address the question of the uniform areas located around the 12 intensity levels : $21.3, 42.6, 63.9, 85.2$,...,234.6. We think that this last challenge could be addressed like a multilevel contouring artifact.[23,24]

ACKNOWLEDGMENTS

The authors would like to thank Pierre Poulin and Justin Bur for their very precious help.

REFERENCES

1. V. Ostromoukhov, "A simple and efficient error-diffusion algorithm," in *SIGGRAPH 2001, Computer Graphics Proceedings, Annual Conference Series*, pp. 567–572, 2001.
2. G. Goertzel and G. Thompson, "Digital halftoning on the IBM 4250 printer," *IBM Journal of Research and Development* **31**, pp. 2–15, Jan. 1987.
3. Z. Fan, "Dot-to-dot error diffusion," *Journal of Electronic lmaging* , Jan.
4. R. Stevenson and G. Arce, "Binary display of hexagonally sampled continuous-tone images," *Journal of the Optical Society of America* **2**, pp. 1009–1013, July 1985.
5. J. Shiau and Z. Fan, "A set of easily implementable coefficients in error diffusion with reduced worm artifacts," in *SPIE*, **2658**, pp. 222–225, 1996.
6. P. Li and J. Allebach, "Tone dependent error diffusion," in *Proceedings SPIE Conf. Electronic Imaging*, pp. 293–301, (San Jose, Ca), 2002.
7. G. Marcu, "An error diffusion algorithm with output position constraints for homogeneous highlights and shadow dot distribution," *Journal of Electronic Imaging* **9**(1), pp. 46–51, 2000.
8. K. Knox, "Error image in error diffusion," in *Proceedings SPIE Conf. Electronic Imaging*, pp. 9–14, (San Jose, CA), Feb. 1992.
9. R. Eschbach, "Reduction of artifacts in error diffusion by means of input-dependent weights," *Journal of Electronic Imaging* **2**, pp. 352–358, October 1993.
10. R. Ulichney, *Digital Halftoning*, MIT Press, Cambridge, MA, USA, 1987.
11. R. Floyd and L. Steinberg, "An adaptive algorithm for spatial gray scale," in *SID 75, Int. Symp. Dig. Tech. Papers*, p. 36, 1975.
12. R. Ulichney, "Dithering with blue noise," *Proceedings of the IEEE* **76**, pp. 56–79, 1988.
13. K. Knox, "Evolution of error diffusion," *Electronic Imaging* **8(4)**, pp. 422–429, 1999.
14. F.P. Preparata and M.I. Shamos, *Computational Geometry : An Introduction*, Springer-Verlag, 1985.
15. R. Ulichney, "A review of halftoning techniques," in *Proceedings SPIE Conf., Proc. SPIE* **3963**, pp. 378–391, 2000.

16. W. V. et al., *Numerical Recipes: Example Book (C)*, Cambridge University Press, 1998.
17. P. Jodoin and V. Ostromoukhov, "Error-diffusion with blue-noise properties for midtones," in *Proceedings SPIE Conf. Electronic Imaging*, pp. 293–301, (San Jose, Ca), 2002.
18. J. Gaskill, *Linear Systems, Fourier Transforms and Optics*, John Willey & Sons, New York, 1978.
19. K. Knox, "Introduction to digital halftones," in *IS&T's 47th Annual Conference, ICPS'94: The Physics and Chemistry of Imaging Systems*, **2**, pp. 456–459, May 1994.
20. K.T.Knox, "Evolution of error diffusion," *Journal of Electrical Imaging* **8**(4), pp. 422 – 429, 1999.
21. R. Eschbach, "Pixel-based error-diffusion algorithm for producing clustered halftone dots," *Journal of Electronic Imaging* **3**, pp. 198–202, April 1994.
22. L. Ingber, "Simulated annealing: Practice versus theory," *Mathematical and Computer Modeling* **18**, pp. 29–57, 1993,http://www.ingber.com/.
23. G. Lin and J. Allebach, "Multilevel screen design using direct binary search," in *Proceedings SPIE Conf. Electronic Imaging*, pp. 264–277, (San Jose, CA), Feb. 2002.
24. K. S. Q. Yu, K. Parker and R. Miller, "Improved digital multitoning with overmodulation scheme," in *Proceedings SPIE Conf. Electronic Imaging*, pp. 264–277, (San Jose, CA), Feb. 1998.

Table 6. List of the coefficient sets optimized by our method. The *key-levels* are those highlighted on the left.

	d10	d-11	d01
0	6691	0	3309
1	6691	0	3309
2	6576	316	3108
3	6462	629	2909
4	6348	940	2711
5	6236	1248	2516
6	6124	1554	2322
7	6014	1857	2129
8	5904	2157	1938
9	5795	2456	1749
10	5688	2751	1561
11	5581	3044	1375
12	5474	3335	1190
13	5369	3624	1007
14	5265	3910	825
15	5161	4194	645
16	4682	4237	1081
17	4303	4272	1425
18	3997	4300	1704
19	3743	4323	1934
20	3530	4342	2128
21	3900	4165	1935
22	4516	3871	1613
23	4375	3722	1904
24	4214	3551	2236
25	4027	3354	2619
26	4000	3779	2221
27	3972	4224	1804
28	3943	4689	1368
29	3912	5177	911
30	3879	5690	431
31	3785	5701	514
32	3693	5712	595
33	3603	5722	675
34	3514	5733	753
35	3509	5694	798
36	3504	5655	841
37	3499	5618	883
38	3494	5581	925
39	3489	5545	965
40	3485	5510	1005
41	3480	5476	1044
42	3476	5442	1082
43	3471	5409	1120
44	3399	5139	1462
45	3333	4891	1776
46	3272	4664	2064
47	3216	4454	2330
48	3164	4260	2576
49	3116	4080	2804
50	3071	3912	3017
51	3029	3756	3215
52	2990	3610	3400
53	2954	3473	3574
54	2919	3344	3737
55	2887	3223	3890
56	2856	3109	4034
57	2827	3002	4171
58	2800	2900	4300
59	2774	2804	4422
60	3134	3401	3466
61	3460	3942	2598
62	3757	4435	1808
63	4029	4886	1086
64	4278	5300	422
65	4249	5324	427
66	4220	5347	432
67	4192	5371	437
68	4163	5395	442
69	4134	5418	447
70	4106	5442	452
71	4077	5465	457
72	4049	5489	462
73	4020	5512	467
74	3992	5536	472
75	3964	5559	477
76	3936	5582	482
77	3907	5605	487
78	3879	5628	492
79	3851	5652	497
80	3823	5675	502
81	3795	5698	507
82	3768	5721	512
83	3740	5744	517
84	3712	5767	521
85	3684	5789	526
86	3743	5747	510
87	3802	5705	493
88	3860	5663	477
89	3918	5622	461
90	3975	5580	444
91	4032	5539	428
92	4089	5498	412
93	4146	5458	396
94	4202	5417	381
95	4258	5377	365
96	4313	5337	349
97	4369	5298	334
98	4424	5258	318
99	4478	5219	303
100	4532	5180	288
101	4586	5141	273
102	4640	5103	258
103	4693	5064	243
104	4746	5026	228
105	4799	4988	213
106	4851	4950	198
107	4904	4913	183
108	4955	4876	169
109	5007	4839	154
110	5058	4802	140
111	5109	4765	126
112	5160	4729	111
113	5210	4693	97
114	5260	4657	83
115	5310	4621	69
116	5360	4585	55
117	5409	4550	41
118	5458	4514	27
119	5507	4479	14
120	5556	4444	0
121	5506	4403	91
122	5448	4356	196
123	5380	4299	321
124	5299	4232	469
125	5200	4150	650
126	5077	4048	875
127	4920	3918	1162