

Advances in Linear Pixel Shuffling¹

Peter G. Anderson
Computer Science Department
Rochester Institute of Technology,
Rochester, NY 14623-0887
`anderson.at.si.n.cs.rit.edu`

April 11, 2000

¹Presented at the Conference on Fibonacci Numbers and Their Applications, Pullman, Washington, July, 1994.

Abstract

Given an interval or a higher dimensional block of points, that may be either continuous or discrete, how can we probe that set in a smooth manner, visiting all its regions without slighting some and overprobing others? The method should be easy to program, to understand, and to run efficiently.

We investigate a method of visiting the pixels (the elements of a rectangular matrix) and the points in the real unit cube based on an arithmetic progression with wrap-around (modular arithmetic). For appropriate choices of parameters, choices that generalize Fibonacci numbers and the golden mean, we find equidistributed collections of pixels or points, respectively.

We illustrate this equidistributivity with a novel approach to progressive rendering of digital images. We also suggest several opportunities for its application to other areas of image processing and computing.

Our Main Problem

We want to be able to generate points in n -dimensional spaces, where these spaces are either continuous (the n -cube) or discrete (arrays of dimension n). We want our points to be easy to specify—preferably by a linear rule—and we want them as evenly spread as possible. In the discrete case, we want to generate a permutation of all the points in an array; in the continuous case, we want to avoid duplicates.

A particularly good solution to this problem when $n = 1$ is to select points in the unit interval by the rule that the k -th point is

$$x_k = ((k\alpha)) \tag{1}$$

where $\alpha = (1 + \sqrt{5})/2$ (i.e., the familiar golden mean) and $((x)) = x - [x]$ denotes the fractional portion of x (see [1, 16]). This rule works well—it generates points that are well dispersed—because α is hard to approximate with rationals [8]. If $x_j \approx x_k$ for $j < k$, then $x_{k-j} \approx 0$, (or $x_{k-j} \approx 1$), consequently $(k-j)\alpha \approx m$ for some integer m . This gives a rational approximation, $\alpha \approx m/(k-j)$ which should not be especially good unless m and $k-j$ are both large.

Write $a\%b$ for the value of a reduced modulo b . The corresponding good solution to the one-dimensional discrete case is given by

$$X_k = kF_{m-1}\%F_m \tag{2}$$

which is simply a discretization of the rule given above for x_k . This gives a permutation of the numbers $0, 1, \dots, F_{m-1}$. In case the sequence has a length, L , that is not a Fibonacci number, let $A \approx L/\alpha$ such that $\gcd(A, L) = 1$, and let

$$X_k = kA\%L \tag{3}$$

Generalizing Continued Fractions, Golden Means, and Fibonacci Sequences

In [4] we constructed a generalization to higher dimensions of periodic continued fractions whose period is one: $\alpha = \alpha(1, p) = [0; \bar{p}] \approx 1/p$. In the notation “ $\alpha(1, p)$ ” the first parameter denotes dimensionality, and the second parameter an arbitrary positive integer. We used the observation that $(((p+1)\alpha)) = (p+1)\alpha - 1$ cuts the interval from α to 0 in a manner geometrically similar to the way α cuts the interval from 0 to 1:

$$\frac{\alpha - 0}{1 - 0} = \frac{\alpha - (((p+1)\alpha))}{\alpha - 0} \tag{4}$$

Stated algebraically, this is : $\alpha - ((p+1)\alpha - 1) = \alpha^2$, or $\alpha(\alpha + p) = 1$.

Although scalar multiplication does not generalize nicely to vector multiplication, we found that we could generalize this notion of geometric similarity. In two dimensions, let the vector $\alpha = \alpha(2, p) = (x, y) \approx (1/p^2, 1/p)$, and let $(((p^2 + 1)\alpha)) = (p^2 + 1)\alpha - (1, p)$ sit within the rectangle with base vertex α spanned by $((1-p)x, 1 + (1-p)y)$ and $(0, 0)$, as α sits within

the unit square, i.e., the rectangle whose base vertex is 0 spanned by $(1, 0)$ and $(0, 1)$. Stated in terms of matrices and (column) vectors,

$$\begin{pmatrix} -px & -x \\ 1-px & -y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} (p^2+1)x-1 \\ (p^2+1)y-p \end{pmatrix}, \quad (5)$$

which simplifies to a pair of simultaneous quadratic equations, $x(x+p)^2 = 1$ and $x(x+p) = y$.

The scalar $\alpha(1, p)$ appears in an eigenvector-eigenvalue equation,

$$\begin{pmatrix} p & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ \alpha \end{pmatrix} = \begin{pmatrix} \alpha+p \\ 1 \end{pmatrix} = (\alpha+p) \begin{pmatrix} 1 \\ \alpha \end{pmatrix} \quad (6)$$

The vector $\alpha(2, p) = (x, y)$ similarly appears in

$$\begin{pmatrix} p & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ y \\ x \end{pmatrix} = \begin{pmatrix} x+p \\ 1 \\ y \end{pmatrix} = (x+p) \begin{pmatrix} 1 \\ y \\ x \end{pmatrix} \quad (7)$$

These observations lead to our calling vectors like (x, y) “generalizations of periodic continued fractions,” and, when $p = 1$, “generalized golden means.”

The above 2×2 and 3×3 matrices can be used to define some “Fibonacci-like” sequences, as follows. The sequence $G = G(1, p)$ is given by

$$\begin{pmatrix} p & 1 \\ 1 & 0 \end{pmatrix}^m \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} G_{m+1} \\ G_m \end{pmatrix} \quad (8)$$

In other words:

$$G_0 = 0, \quad G_1 = 1, \quad G_k = pG_{k-1} + G_{k-2} \quad (9)$$

($G = G(1, 1)$ is the familiar Fibonacci sequence.) The ratio of two successive members of the sequence is

$$\alpha + p = \lim_{k \rightarrow \infty} \frac{G_{k+1}}{G_k} \quad (10)$$

(where α is $\alpha(1, p)$; $\alpha(1, 1)$ is the familiar golden mean). Similarly, the sequence $G = G(2, p)$ is given by

$$\begin{pmatrix} p & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^m \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} G_{m+1} \\ G_m \\ G_{m-1} \end{pmatrix} \quad (11)$$

In other words:

$$G_0 = 0, \quad G_1 = 1, \quad G_2 = 1, \quad G_k = pG_{k-1} + G_{k-3}, \quad \text{for } k > 2 \quad (12)$$

and then

$$x + p = \lim_{k \rightarrow \infty} \frac{G_{k+1}}{G_k} \quad (13)$$

Let n and p be positive integers. Let $\alpha = \alpha(p, n)$ be a point in the n -cube given by,

$$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n) \in [0, 1]^n \subset \mathbf{R}^n \quad (14)$$

where the first component, $\alpha_1^{(n)}$, is the unique, positive real solution of the equation $x(x + p)^n = 1$. (The solution is unique, because $p > 0$ implies that the left-hand side is monotonically increasing for $x > 0$.) The other components are given by $\alpha_k = \alpha_1(\alpha_1 + p)^{k-1}$ for $k = 2, \dots, n$. For the scalar case, $n = 1$,

$$\alpha = \frac{1}{2}(\sqrt{p^2 + 4} - p) \quad (15)$$

When $p = 1$, this is the familiar golden mean, 0.618..., and the limit of the ratio of two successive Fibonacci numbers.

We also work with Fibonacci-like number sequences, $G = G(n, p)$, defined as follows. Fix $p, n > 0$.

$$\begin{aligned} G_0 &= 0 \\ G_k &= 1, \text{ for } 1 \leq k \leq n \\ G_k &= pG_{k-1} + G_{k-n-1}, \text{ for } k > n. \end{aligned} \quad (16)$$

$G(1, 1)$ is the familiar Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... The points α and the sequences $G(p, n)$ are related by

$$\lim_{k \rightarrow \infty} \frac{G_{k+1}}{G_k} = \alpha_1 + p = \frac{1}{\alpha_n}. \quad (17)$$

Continuous Linear Pixel Shuffling

When $p = 1$, the unit interval $[0, 1)$ is very smoothly probed by the fractional parts of integer multiples of the golden mean, $\alpha = \alpha(1, 1)$: $x_k = ((k\alpha))$. The ‘‘smoothness’’ degrades as large values of p are chosen for $\alpha = \alpha(p, 1)$, which become successively easier to approximate with rationals [8]. The unit cube $[0, 1)^n \subset \mathbf{R}^n$ is probed by the sequence of points

$$\bar{x}_j = ((j\alpha)) = (((j\alpha_1)), ((j\alpha_2)), \dots, ((j\alpha_n))). \quad (18)$$

Discrete LPS, Method 1: $Z_A \times Z_B = Z_{AB}$

Let $p = n = 1$. An interval of numbers is probed (or shuffled) by the rule $X_j = jF_{m-1} \% F_m$ (Eq. (2)). Here, we are generating a sequence of numbers between 0 and $F_m - 1$. If $j = 0, 1, \dots, F_m - 1$, the sequence is a permutation, since pairs of adjacent Fibonacci numbers are always relatively prime. Call this the ‘‘Fibonacci shuffle’’.

Similarly, using $G = G(2, 1)$, we probe the discrete two-dimensional $G_{m-1} \times G_m$ -rectangle by the rule

$$\bar{X}_j = (jG_{m-2} \% G_{m-1}, jG_{m-2} \% G_m) \quad (19)$$

To guarantee that all $G_{m-1} \times G_m$ points are visited before any point is visited twice, we must verify the following three relative primality constraints:

$$\gcd(G_{m-2}, G_{m-1}) = \gcd(G_{m-1}, G_m) = \gcd(G_{m-2}, G_m) = 1 \quad (20)$$

When $p = 1$, $m = 0.25$, the G sequence is: 0, 1, 1, 1, 2, 3, 4, 6, 9, 13, 19, 28, 41, 60, 88, 129, 189, 277, 406, 595, 872, 1278, 1873, 2745, 4023, 5896. Notice that there are several primes in this sequence; in particular, 1873 is prime, and the pair (1278, 1873) is particularly useful as approximate coordinates of a computer graphics screen. Higher-dimensional probes work similarly.

To shuffle (i.e., visit in a jumbled order) the pixels in an $A \times B$ rectangle we first ensure that A and B are relatively prime, so that the group $Z_A \times Z_B$ is isomorphic to the cyclic group $Z_{A \times B}$. Then, choose a generator $(x, y) \in Z_{A \times B}$ so that $x/A \approx \alpha_1$ and $y/B \approx \alpha_2$, where $\alpha(2, 1) = (\alpha_1, \alpha_2)$.

Discrete LPS, Method 2: Change of Basis

Our experience has been a little frustrating trying to locate good parameters as outlined in Method 1, above. However, for $n = 2$ and $p = 1$, consider the $G_m \times G_m$ table, T , whose entry at position (i, j) is

$$T_{i,j} = (iG_{m-2} + jG_{m-1})\%G_m \quad (21)$$

This table, which we exploit below for image digital halftone masks, has numerically close values geometrically spread far apart. On the other hand, each number in $(0, \dots, G_m - 1)$ occurs G_m times, not just once. (Each value will occur G_m times because each triple (G_{m-2}, G_{m-1}, G_m) has no nontrivial common factor.) As a consequence, we cannot straightforwardly visit the pixels in the order specified. Rather, we must develop a double loop rule which allows us to visit the pixels labeled 0, then the pixels labeled 1, and so on.

Table 1 shows the mask tile corresponding to $G_{10} = 19$:

Notice that the eight immediate neighbors of 0 in this table do not include the values close to 0: 1, 2, 17, or 18. Furthermore the positions of the 0's are well dispersed in the table.

This distribution of the values $(0, 1, \dots, G_m - 1)$ suggests a second technique for enumerating all G_m^2 points ("pixels") in a $G_m \times G_m$ square: visit all the points labeled 0 in a good order (the meaning will be made clear below) followed by visiting all the 1's, then the 2's, and so on. The following two lemmas will allow us to establish the visiting algorithm.

Lemma 1. $T_{i_1, i_2} = 0$ if and only if $i_2 = k(G_m - G_{m-2})\%G_m$ and $i_1 = kG_{m-1}\%G_m$, for $0 \leq k < G_m - 1$.

Proof. "If:" take all arithmetic modulo G_m , and apply the subscript rule given by Eq. 21.

$$\begin{aligned} T_{i_1, i_2} &= T_{kG_{m-1}, (-kG_{m-2})} \\ &= (kG_{m-1})G_{m-2} + (-kG_{m-2})G_{m-1} \\ &= 0 \end{aligned} \quad (22)$$

"Only if:" the $G_m \times G_m$ array whose entries are given by Eq. 21 contains exactly G_m zeros, since the numbers G_{m-2} , G_{m-1} , and G_m are relatively prime (i.e., no integer larger than unity divides all three). The same reasoning gives us that the G_m array entries, corresponding to $k = 0, \dots, G_m - 1$, are all different. **Q.E.D.**

0	13	7	1	14	8	2	15	9	3	16	10	4	17	11	5	18	12	6
9	3	16	10	4	17	11	5	18	12	6	0	13	7	1	14	8	2	15
18	12	6	0	13	7	1	14	8	2	15	9	3	16	10	4	17	11	5
8	2	15	9	3	16	10	4	17	11	5	18	12	6	0	13	7	1	14
17	11	5	18	12	6	0	13	7	1	14	8	2	15	9	3	16	10	4
7	1	14	8	2	15	9	3	16	10	4	17	11	5	18	12	6	0	13
16	10	4	17	11	5	18	12	6	0	13	7	1	14	8	2	15	9	3
6	0	13	7	1	14	8	2	15	9	3	16	10	4	17	11	5	18	12
15	9	3	16	10	4	17	11	5	18	12	6	0	13	7	1	14	8	2
5	18	12	6	0	13	7	1	14	8	2	15	9	3	16	10	4	17	11
14	8	2	15	9	3	16	10	4	17	11	5	18	12	6	0	13	7	1
4	17	11	5	18	12	6	0	13	7	1	14	8	2	15	9	3	16	10
13	7	1	14	8	2	15	9	3	16	10	4	17	11	5	18	12	6	0
3	16	10	4	17	11	5	18	12	6	0	13	7	1	14	8	2	15	9
12	6	0	13	7	1	14	8	2	15	9	3	16	10	4	17	11	5	18
2	15	9	3	16	10	4	17	11	5	18	12	6	0	13	7	1	14	8
11	5	18	12	6	0	13	7	1	14	8	2	15	9	3	16	10	4	17
1	14	8	2	15	9	3	16	10	4	17	11	5	18	12	6	0	13	7
10	4	17	11	5	18	12	6	0	13	7	1	14	8	2	15	9	3	16

Table 1: The dither mask corresponding to $G_{10} = 19$; i.e., $T_{ij} = (9i + 13j)\%19$. The box around the 0 at location (13,10) is referred to in the discussion following Lemma 1.

The subscript position $(i_1, i_2) = (G_{m-1}, G_m - G_{m-2})$ within the $G_m \times G_m$ array approximates the position of $(\alpha_1, 1 - \alpha_2)$ within the unit square (where $\alpha(2, 1) = (\alpha_1, \alpha_2)$); consequently, the multiples of (i_1, i_2) are smoothly distributed. The entry $(13, 10) = (13, 19-9)$ is indicated by a box in Table 1 (that table is shown transposed from the rule of the present discussion).

Next, we locate the position of a 1 within T .

Lemma 2. $T_{j_1, j_2} = 1$, if $j_1 = G_{m-1}^2 + G_{m-2}^2$ and $j_2 = G_{m-1}^2$.

Proof. Recall,

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^m \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} G_{m+1} \\ G_m \\ G_{m-1} \end{pmatrix} \quad (23)$$

The 3×3 matrix in Eq. 23 has unit determinant, and so do all of its powers:

$$\begin{vmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix}^{m+1} = \begin{vmatrix} G_{m+2} & G_m & G_{m+1} \\ G_{m+1} & G_{m-1} & G_m \\ G_m & G_{m-2} & G_{m-1} \end{vmatrix} = 1 \quad (24)$$

Using $G_m = G_{m-1} + G_{m-3}$ and $G_m = 0$ (arithmetic modulo G_m), we have

$$\begin{aligned} 1 &= \begin{vmatrix} G_{m+2} & G_m & G_{m+1} \\ G_{m+1} & G_{m-1} & G_m \\ G_m & G_{m-2} & G_{m-1} \end{vmatrix} \\ &= \begin{vmatrix} G_{m-2} + G_{m-1} & 0 & G_{m-2} \\ G_{m-2} & G_{m-1} & 0 \\ 0 & G_{m-2} & G_{m-1} \end{vmatrix} \\ &= G_{m-1}^3 + G_{m-1}^2 G_{m-2} + G_{m-2}^3 \\ &= (G_{m-1}^2 + G_{m-2}^2) G_{m-2} + (G_{m-1}^2) G_{m-1} \end{aligned} \quad (25)$$

The final line of the above computation puts the expression in the form of Eq. (21). **Q.E.D.**

So, the algorithm we were seeking, to enumerate the points of the $G_m \times G_m$ square is a “change of basis.” Simply use the basis vectors

$$I = (G_{m-1}, G_m - G_{m-2}), \quad J = (G_{m-2}^2 + G_{m-1}^2, G_{m-1}^2) \quad (26)$$

instead of $(1, 0)$ and $(0, 1)$, as shown in the following program, in which X_MAX and Y_MAX are both G_m .

```

I = (1,0); J = (0,1) /* basis vectors */
for( a = 0; a < X_MAX; a++ ) {
    for( b = 0; b < Y_MAX; b++ ) {
        visit point a*I + b*J
    }
}

```

The new pixel ordering rule is given by:

$$\overline{X}_k = a_k(G_{m-1}, G_m - G_{m-2}) + b_k(G_{m-2}^2 + G_{m-1}^2, G_{m-1}^2) \quad (27)$$

where $a_k = k \% G_m$ and $b_k = [k/G_m]$, and $k = 0, 1, \dots, G_m^2 - 1$. The variables a_k and b_k correspond to the for-loop induction variables **a** and **b** in the above program.

Continuous Progressive Rendering

A common form for computer graphics programs to take is shown in the following pseudocode fragment:

```
for every pixel (x,y)
    compute c = color_function(x,y)
    paint the color c at pixel (x,y)
```

These programs do not necessarily need to paint those pixels in “row order,” saving the most interesting pixel locations for many minutes into the plotting procedure. The scan lines can be painted in the Fibonacci shuffle order, or the pixels can be painted in the two-dimensional $G = G(2, p)$ shuffling order ($n = 2$), as given by Eq. (19) or (27). This alternative pixel ordering can provide early indication that the image being plotted is the image desired, and it serves as an efficient speedup of graphics software development, parameter space experimentation, or image data bank browsing.

Because the shuffling is by a simple linear rule in the method given in Eq. (19), the shortest distance from the currently plotted pixel and any previously plotted pixel is easy to determine: it is simply the minimum distance *any pixel* has ever been to $\bar{0}$. Consequently, the image can be rendered early with “fat pixels” to illuminate the entire screen. As the population of rendered pixels gets larger, the size of the fat pixels can be gradually reduced, always assuring that no new fat pixel ever overwrites a previously correctly drawn pixel. Figures 1 through 4 show one of our favorite pictures at several levels of rendering. (See [3] and [2] for more details.)

Image Compression

The speed with which images become recognizable, and the redundancy of the painting of most of the pixels late in the rendering process, suggests a compression algorithm for binary images; namely, only describe the painting of new fat pixels at locations where the color needs to be changed. This has the novel effect that the truncation of such a compressed description to a prefix is a lossy compression, not a description of lossless compression of a fraction of the image.

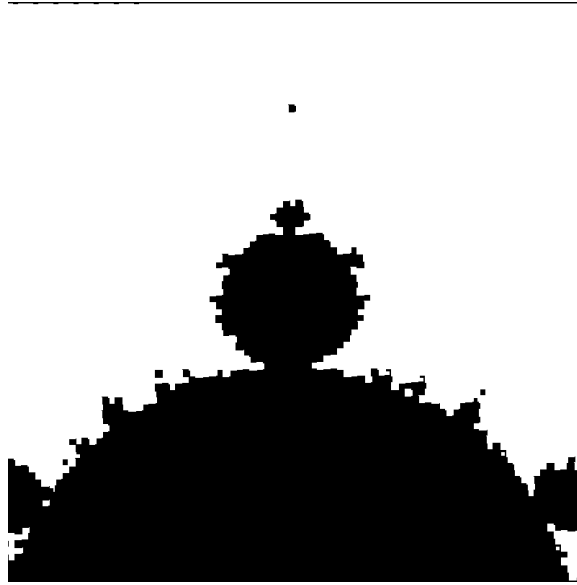


Figure 1: A portion of the Mandelbrot set—rendered 1%.

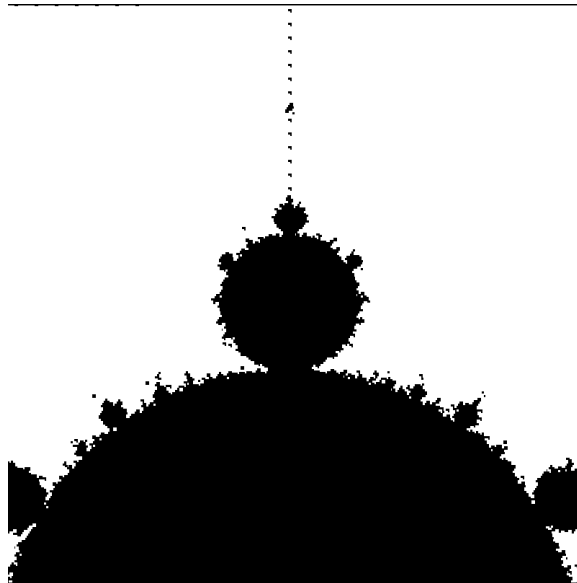


Figure 2: A portion of the Mandelbrot set—rendered 4%.

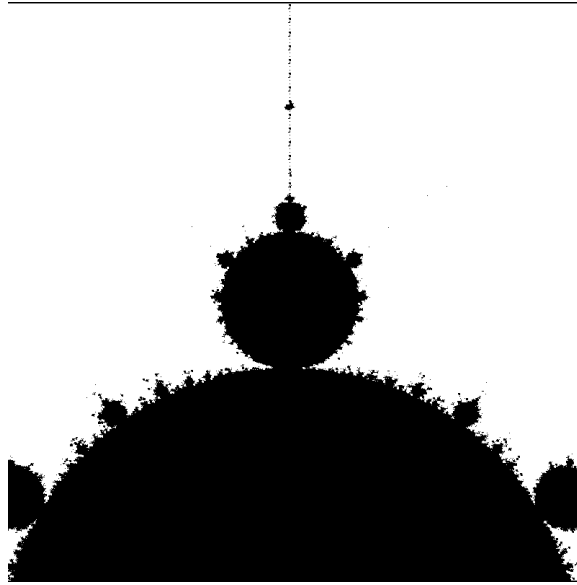


Figure 3: A portion of the Mandelbrot set—rendered 25%.

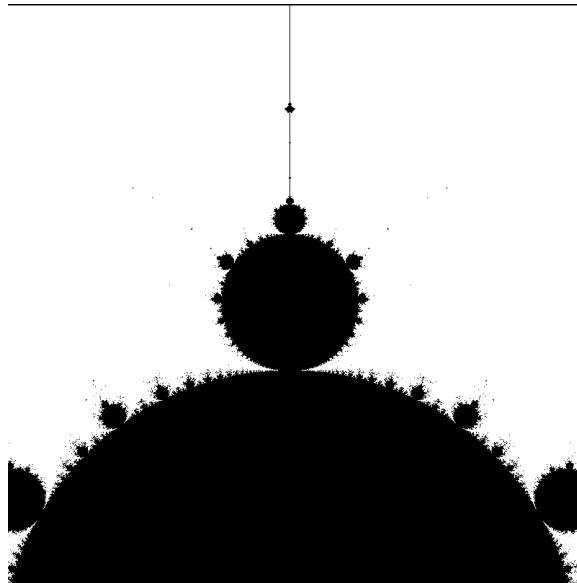


Figure 4: A portion of the Mandelbrot set—rendered 100%.

```

set out_image to all white;
initialize (x,y) ;
initialize fatness width;
for( k = 0; k < pixel_count; k++ ) {
    update (x,y) and width;
    if( out_image(x,y) != in_image(x,y) )
        paint fat pixel of "width"
        in out_image at (x,y);
    report k;
}

```

Figure 5: The compression algorithm.

```

set out_image to all white;
for each k
    determine (x,y) and width;
    paint fat pixel of "width"
    in out_image at (x,y);

```

Figure 6: The decompression algorithm.

The compression algorithm (see Figure 5) works with two images, the input image, `in_image`, and a copy of that image, `out_image`, which is initialized to all white. It sets about creating the copy by plotting fat pixels in the same order that the rendering algorithm discussed above operates, except that in case the pixel in `out_image` is already the correct color (black or white), it does not modify `out_image`. When it does modify `out_image`, it simply writes to the description report the pixel's identification number.

The decompression algorithm (see Figure 6) operates by reading the reported pixel identification numbers written by the compression algorithm, and interpreting each of them as a command to draw a fat pixel in its own version of `out_image`.

Early experiments of this are promising [14]. The compression ratios—in the mid 80% range—are competitive with run length encoding, and the images are quite recognizable when only the first 12.5% of the encoded description is used to form the image.

Sampling

In the field of medical imaging, it is often necessary to rescale the brightness mapping in order to enhance the contrast of the portions of diagnostic interest within an image. For example,

X-rays should optimize the tone scale towards displaying bones as clearly as possible. X-rays can be very large, so it would be time-consuming to try to find all the bone pixels, determine their brightness histogram, and then rescale. The image can be subsampled at the sequence of pixels labeled \bar{X}_j (as given in Eqs. (19) and (27)), and when enough bone pixels have been located (a few thousand rather than a million), their data can be used to represent that of the whole image. It is clearly necessary that the sampled image portions be very smoothly distributed, so pseudorandom sampling is ruled out.

Searching Spaces

It is often necessary to search a large image for one or more copies of a subimage, such as a highlighted “region of interest,” a person’s face, or registration marks on forms used in optical mark recognition and optical character recognition systems.

In order to search for the subimage, one places the subimage over a subwindow of the large image and computes the distance or correlation between the subimage and the subwindow contents. By moving the subimage all over the large image, searching for a large correlation, one will eventually locate the subimages of interest. A row-order motion of the subimage over the image is clearly painfully slow. Moving the subimage to successive points \bar{X}_j (Eqs. (19) and (27)), appears to be significantly faster than the commonly used Fourier “FFT” methods [23].

Dithering in Two and Three Dimensions

Image Halftoning

A region of black-and-white pixels on a piece of paper or a computer monitor can give the impression of, say, 25% gray if 25% of the pixels are black and those black pixels are smoothly distributed throughout the region. One way to achieve such a smooth distribution is to use an LPS rule: generate the first 25% of the pixels in the region of interest and color them black. A halftoning dispersed dot dithering mask, $M[A][B]$, in the sense of Bayer [5] can be constructed by the same rule we used above to visit the pixels in an $A \times B$ rectangle. We work with $G = G(2, p)$ for small values of p (generally $p = 1$). Let $A = G_m$ and $B = G_{m+1}$. Then, $M[i][j] = k$, where i and j are the coordinates of \bar{X}_k . This M has the form of an addition table, so we only need the two values $M[1][0]$ and $M[0][1]$ to generate any value: $M[i][j] = (iM[1][0] + jM[0][1])\%AB$.

The above construction corresponds to our LPS Method 1, given in Eq. (19). An alternative, and simpler, construction, based on Method 2, Eq. (27), is to specify the mask as

$$M[i][j] = (iG_n + jG_{n+1})\%G_{n+2} \quad (28)$$

This mask is definitely meant to be developed “on the fly”; that is, the values $M[i][j]$ can be computed as needed rather than pre-computed and stored; the computing effort is no

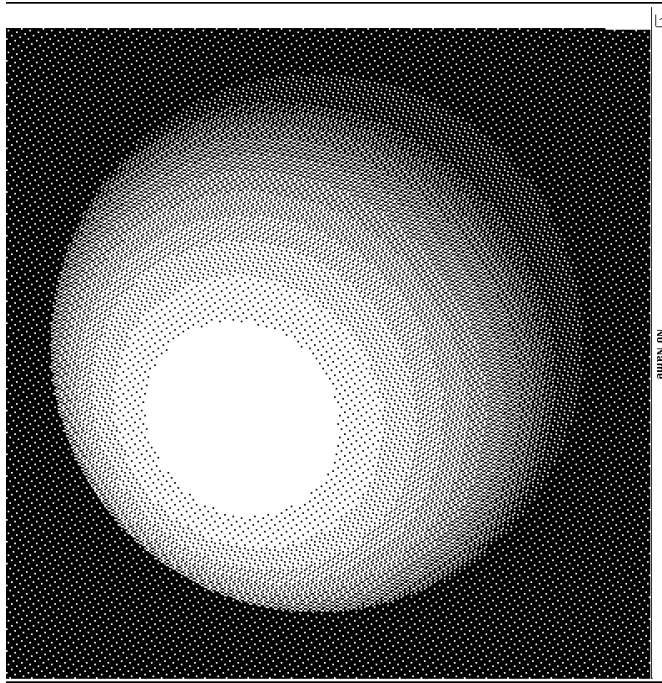


Figure 7: Shaded ball using the $G_9 = 13$ mask.

more than the cost of subscribing. See Table 1 for the 19-level mask. The halftoned images Figs. 7–12 were generated using this Method 2.

Whichever method is used to form a dithering mask, we recommend the parameters, $G_{19} = 595$, $G_{21} = 1278$, and $G_{22} = 1873$. These parameters produce excellent halftoning results [26] as shown in Figures 10 and 11. (The two synthetic figures of a shaded ball and a ramp were chosen for their reproduction abilities in a medium over which we have little control. These images are rendered in a 600×600 array of pixels. The “Lenna” images [20] are 200×200 pixels; these sizes were chosen to allow the dot structure to show clearly as well as being less subject to corruption by the various steps in the printing processes for this report.) For contrast, we also display Figs. 7–9, which show the effect of the mask based on $G_9 = 13$, which only provides thirteen gray levels. The dither mask that we would use to produce 19 levels is shown in Table 1.

We also show the effect of “dithering with white noise”, i.e., using the results of `drand48()` for a dither mask; see Figs. 13–15. White noise, by definition, has equal power at all frequencies. Visually, this means that there are disturbing (muddy) artifacts which correspond to the low frequency components. The human visual system is insensitive to very high spatial frequency information—such patterns are integrated to give the appearance of gray. Good dithering techniques have a “blue noise” aspect (so called because white noise is the uncorrelated random signal with roughly equal power at all frequencies; the power in blue noise is concentrated at the higher frequencies); see [27].

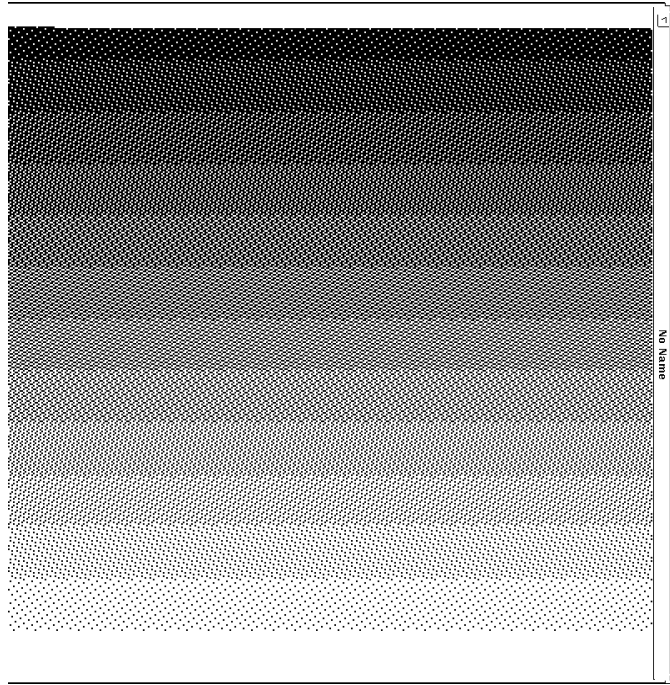


Figure 8: Ramp using the $G_9 = 13$ mask.

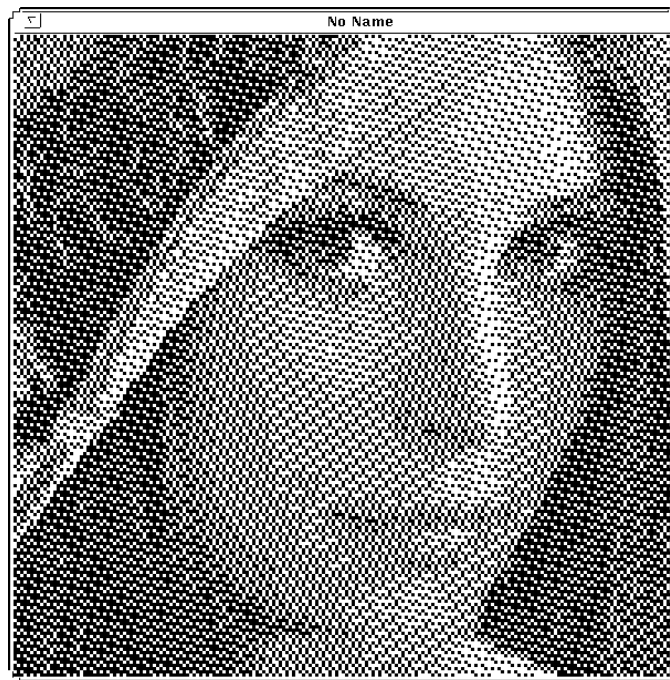


Figure 9: Lenna using the $G_9 = 13$ mask.

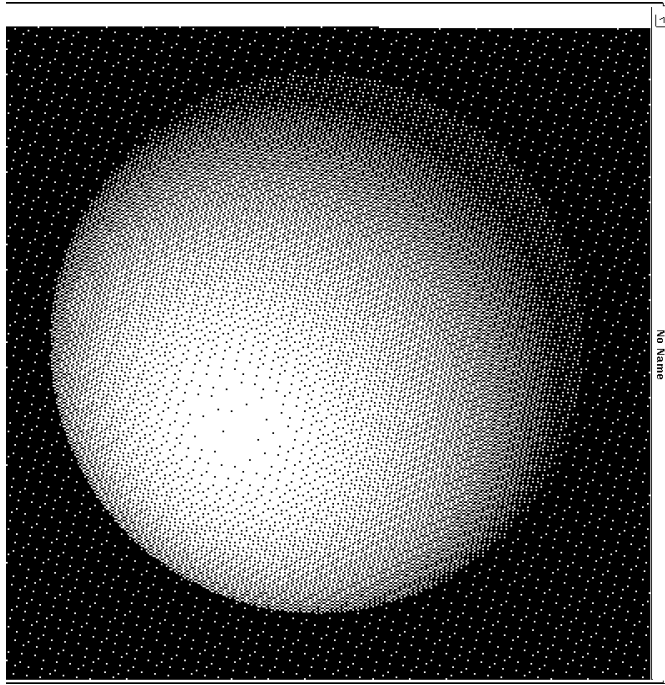


Figure 10: Shaded ball using the $G_{22} = 1873$ mask.

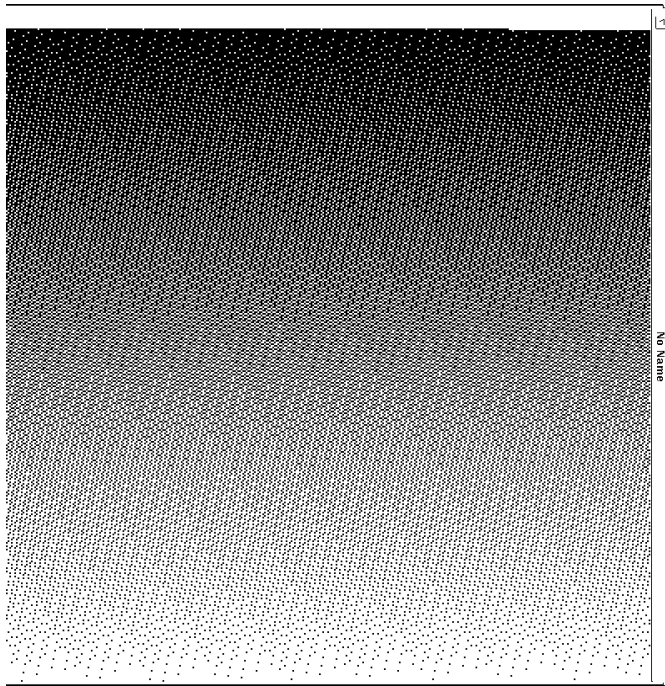


Figure 11: Ramp using the $G_{22} = 1873$ mask.



Figure 12: Lenna using the $G_{22} = 1873$ mask.

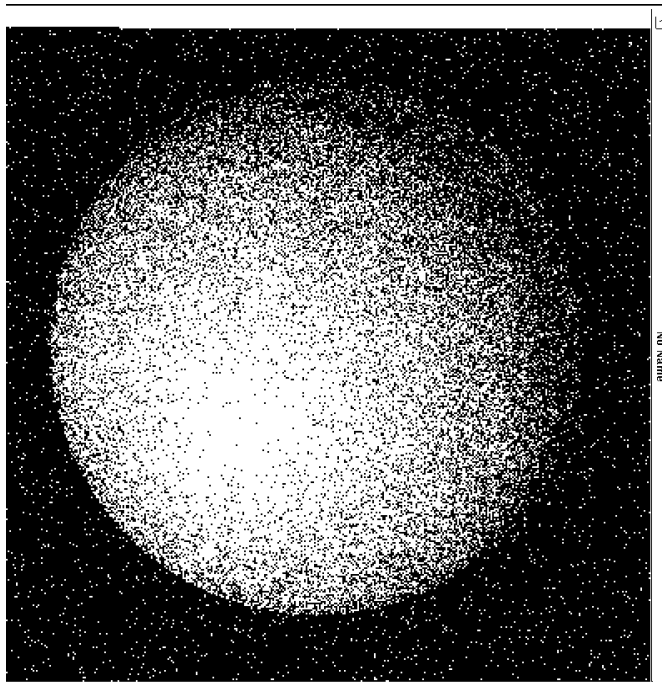


Figure 13: Shaded ball dithered with white noise.

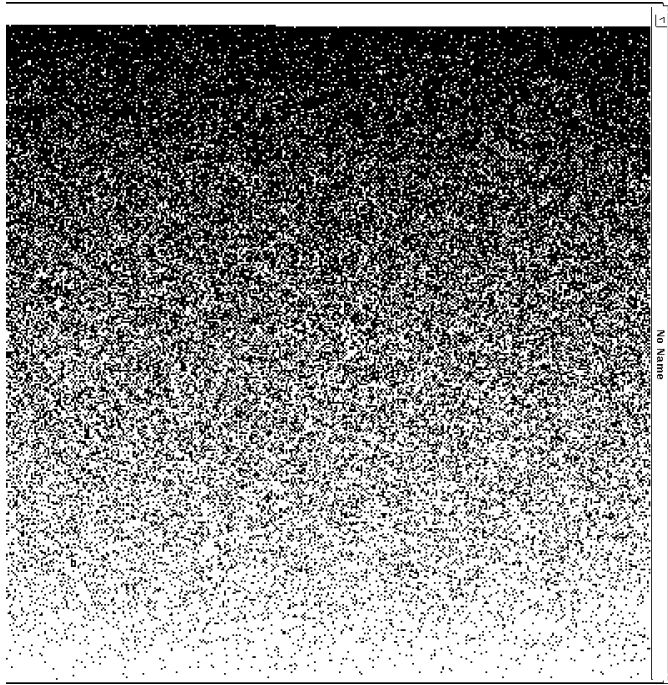


Figure 14: Ramp dithered with white noise.



Figure 15: Lenna dithered with white noise.

Animation Halftoning

To construct halftoning masks in three dimensions, we work with $G = G(3, p)$ for small values of p . We construct a halftoning mask for three dimensions:

$$M[i][j][t] = (iG_n + jG_{n+1} + tG_{n+2})\%G_{n+3} \quad (29)$$

(Note that the sequence $G(3, p)$ has the property that G_n , G_{n+1} , G_{n+2} , and G_{n+3} are always relatively prime.) Such a mask makes the dithering rule correlated from frame to frame in an animation sequence [7]. Alternatives, such as 3D error diffusion or random dithering, often introduce visually disturbing artifacts, such as “flies” and “boiling” [10]. The artifacts we have seen in some initial experiments using this scheme are more like smooth waves.

One Dimensional Dither

Finally, one-dimensional signals, can be digitized (e.g., for audio CD recordings) with dithering by a quasi-random generator, $((t\tau))$ at time t . That is, if the input, analog signal at time t is a_t , then the output, digital signal is $[a_t + ((t\tau)) - 0.5]$. Cf. [19, 28]. This is, of course, dither produced using the nonperiodic, one dimensional dither mask given by Eq. (1).

One further experiment shows the results of image dithering using a one dimensional mask of the form

$$M[i][j] = (iF_n + jF_{n+1})\%F_{n+2} \quad (30)$$

$$M[i][j] = (jF_n + iF_{n+1})\%F_{n+2} \quad (31)$$

This mask employs the usual Fibonacci sequence, $F = G(1, 1)$. The Lenna image is shown dithered by these rules in Figs. 16 and 17.

Monte Carlo Integration

The integral of a function on a unit volume domain can be approximated by averaging the value of the function at several sample points in that domain. Because the behavior of the function may be unknown, one may not be able to tell exactly how many points to sample before the samples are actually taken, so one continues to sample until the integral estimate appears to converge. If the samples are taken from a uniform grid, the number of points may increase by powers of $2n$ in dimension n (“the curse of dimensionality”). Consequently, one often resorts to “white noise” such as that generated by `drand48()` in the standard C math library. Unfortunately, this generator makes clumps and gaps in the domain, because each point is chosen independently of the others (this is usually considered to be a desired property of a pseudorandom number generator). One thousand pseudorandom points are shown in Figure 18. The error associated with an integral estimate using N white noise points is $O(N^{-1/2})$ (see [18]).

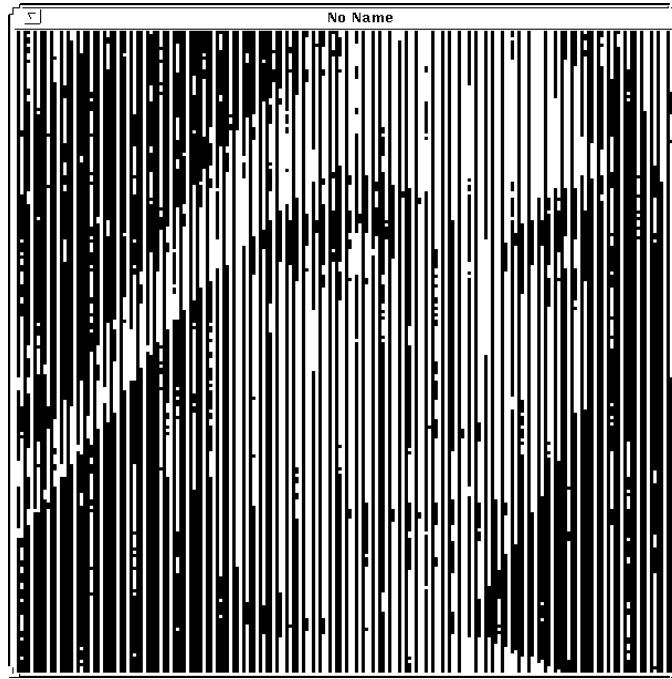


Figure 16: Lenna dithered with a one-dimensional, Fibonacci dither mask.



Figure 17: Lenna dithered with a one-dimensional, Fibonacci dither mask.

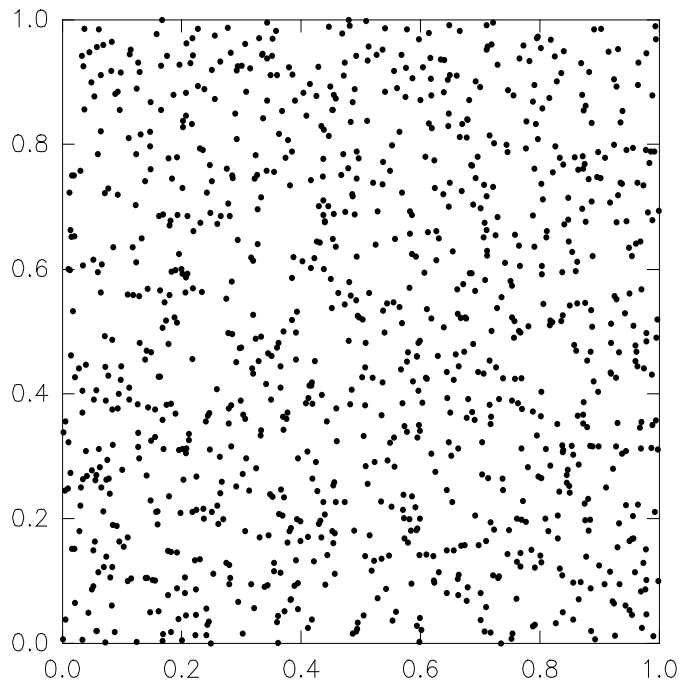


Figure 18: One thousand pseudorandom dots.

The infinite version of LPS comes to the rescue to generate sample points in the unit cube in n -space (see Figure 19). Use the points \bar{x}_j of Eq. (18) to estimate the integral by

$$\int_{I^n} f \approx \frac{1}{N} \sum_{j=1}^N f(\bar{x}_j) \quad (32)$$

The proposed sequence is very evenly distributed, and our experimental results indicate that that it may produce estimates of integrals using N points with an error of $O(N^{-1})$, i.e., twice as many good digits for N probes as white noise gives [22].

As an experiment, we selected a function $f(x, y)$ whose integral over the unit square was known to be zero, since f is the product of two functions of one variable each of which is the derivative of a function which vanishes at 0 and 1:

$$f(x, y) = x^{m-1} \cos(2\pi x^m) y^{m-1} \cos(2\pi y^m) \quad (33)$$

One such function is shown in Figure 20.

To magnify the error behavior of two different techniques for sampling points to evaluate, we neglected the division by N , and evaluated

$$\sum_{k=1}^N f(x_k, y_k) \quad (34)$$

for many values of N . Figure 21 shows the results of four experiments, three pseudorandom estimates of N times the integral, using three different seeds, and one LPS-based estimate. The error for the LPS estimate is shown, expanded, in Figure 22; the LPS error in these experiments is about 1% of the pseudorandom method's error.

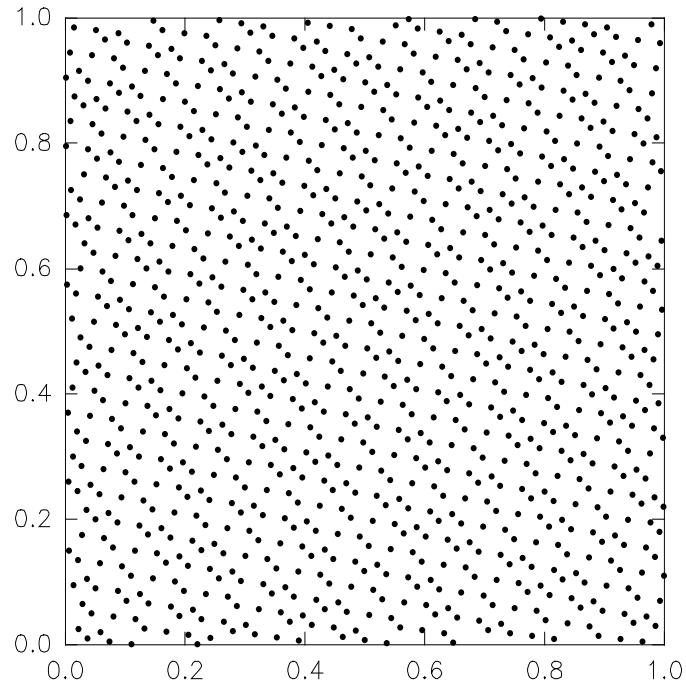


Figure 19: One thousand LPS dots.

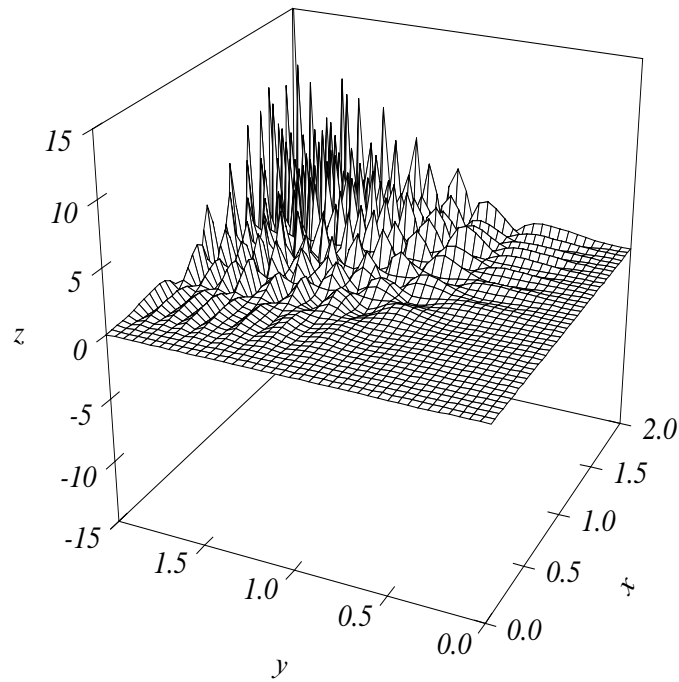


Figure 20: A function whose integral is zero.

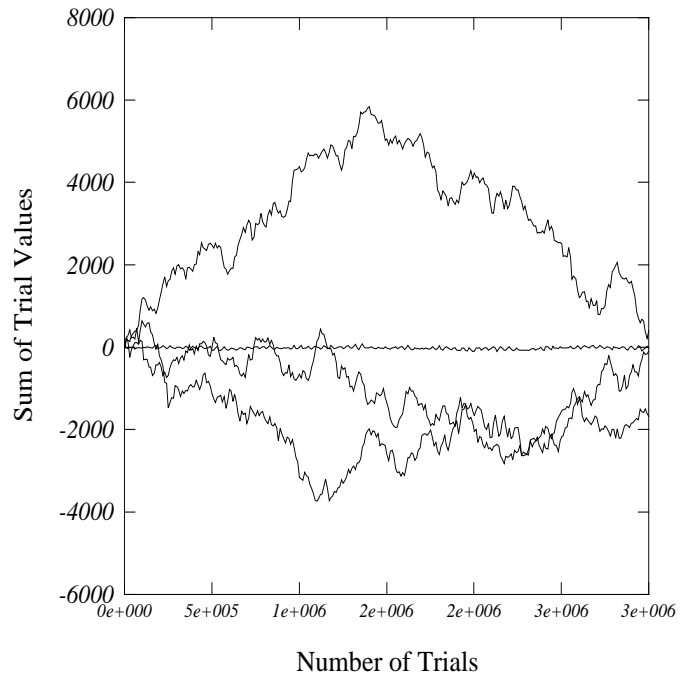


Figure 21: The error associated with three pseudorandom integral estimates and the LPS estimate. The error associated with the LPS scheme is shown hugging the abscissa.

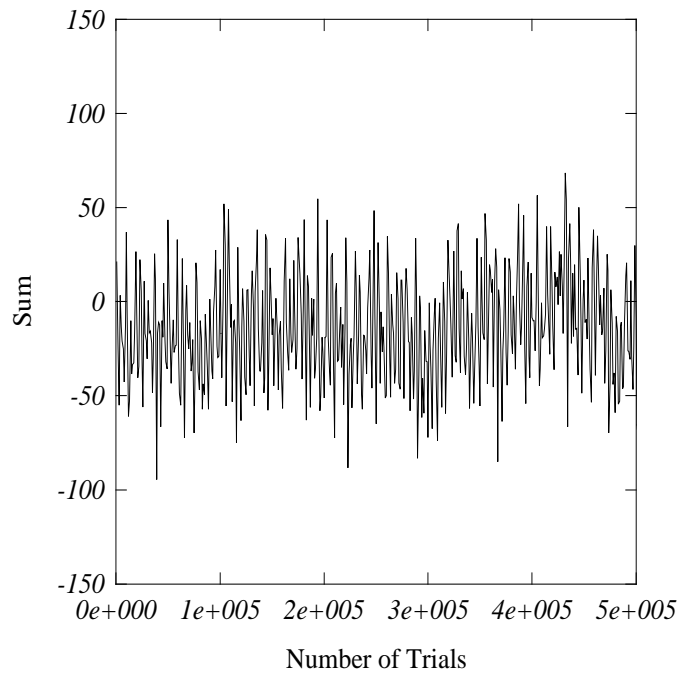


Figure 22: The error associated with the LPS estimate.

Color Palette Determination

Most current color workstations support 256^3 different colors, but they only allow a palette of 256 of these to be used for any particular image. Given a natural or synthetic image, how can the appropriate palette be chosen? A very good but very time-consuming method performs a statistical clustering analysis of the actual color usage. A quick and dirty approach is to start with an empty palette, and then render the image, pixel by pixel, using an existing palette color if the right color is already on the palette, add the new desired color to the palette if the palette is not yet full, and use the closest palette color when the palette is full. If the pixel rendering is by row order, a terrible picture will result. The bottom halves of such images look like they were painted “by number”, because colors that appear in the bottom half but did not appear in the top half often fail to have good approximations on the palette.

By traversing the pixels of the image in the linear pixel shuffling order, relatively good images result. Our initial results with some synthetic ray-traced images are visually competitive with the expensive statistical clustering approach [6].

Another approach to color palette selection is to train a Kohonen neural network [11] to mimic the distribution of the colors actually used in an image with a $32 \times 32 \times 8$ “cortex”. However, if the training is done by selecting the colors from the image in row or column order, very slow training is almost assured. If the pixels are chosen according to linear pixel shuffling order, the training can be substantially faster, and the resulting images are of the highest visual (subjective) quality of the methods we have tried [6].

Neural Network Weights

One application of neural networks is to perform nonlinear two-class discrimination of points in \mathbf{R}^n . A typical structure of such a network [11] is

$$y = \text{sign}(V\sigma(W\bar{x})) \quad (35)$$

$\bar{x} \in \mathbf{R}^n$ is the point to be classified. W and V are matrices of dimensions $m \times n$ and $1 \times m$, respectively, where m is the number of “hidden units” (see Fig. 23). W is the first layer of synaptic weights, and V is the second layer. σ is a nonlinear scalar function, such as \tanh , and $y = \pm 1$ is the classification. The matrices V and W are determined by a training process to produce a low error on a set of labeled training exemplars, (\bar{x}, y) (meaning: when the network’s input is \bar{x} , the desired output is y). If W is an $m \times n$ matrix, the m -vector $\sigma(W\bar{x})$ is referred to as the feature vector for the point \bar{x} . If the features are evolved well, then the matrix V solves a relatively simple linear discrimination problem in \mathbf{R}^m .

Figure 23 shows a net with structure 3–8–1 which can be used to classify points in 2-space (one of the input nodes is generally constantly set to unity to provide a bias or threshold for the nodes in the next layer).

We have some encouraging results that for some problems we can use the points \bar{x}_j , for $j = 1, \dots, m$ as the rows of W and then solve for V using an iterated least-squares method

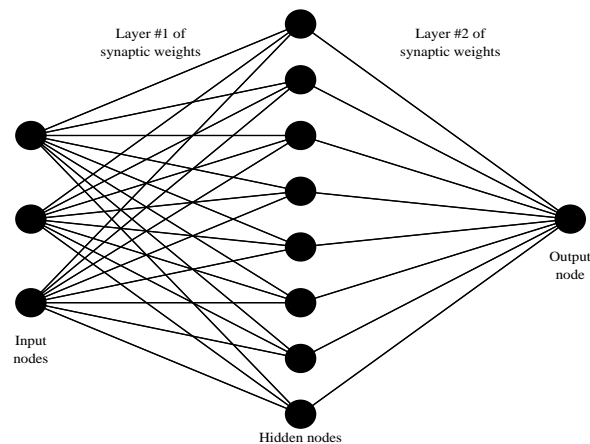


Figure 23: A 3–8–1 feed-forward neural network.

[13]. Of course, m must be large for this to work. After V is determined, we must extract a submatrix of W for the final network. This submatrix may be found by a genetic algorithm as in [15].

Other Applications for LPS

As we see from the foregoing, the LPS method has a wide variety of applications, many of which have been identified, and several initial studies indicate the value of this approach to the generation of points in space and the points within a discrete rectangular array (with the space and the array of arbitrary positive integral dimension).

Other applications that we have not developed in the present paper, but that are the subject of present investigation, include the speedup of estimations of image morphology, training data generation for neural networks, higher dimensional and gray-scale compression techniques, and normal distribution generation.

The author invites correspondence with researchers who can add to these lists of experiences and application areas.

Acknowledgements

The greatest level of thanks go to my many students who have participated in the development of these ideas in seminars on computer graphics and neural networks, and especially those who have delved deeply into the techniques in their Projects and Theses (see the bibliography).

Thanks are also due to my colleagues at Kodak Health Imaging Systems, Inc., and the Rochester Institute of Technology who provided invaluable feedback—especially Staszek Radziszowski and Frank Bernhart who reviewed an early draft of this paper for me. Any

many thanks to the anonymous referee from the Fibonacci Association who provided excellent constructive feedback.

References

- [1] P. G. Anderson, "A Fibonacci-Based Pseudo-Random Number Generator," *Proc. 1990 Conf. on Fibonacci Numbers & Their Applications.*, G. E. Bergum, A. N. Phillipou, and A. F. Horodam (eds.), pp. 1-8. Kluwer Academic Publishers, Boston, MA, 1990.
- [2] Peter G. Anderson, "Fast rendering," *Computer Language*, Feb. 1993, pp. 41-48.
- [3] Peter G. Anderson, Linear pixel shuffling for image processing: an introduction, *Electron. Imag.* **2**: 147-154(1993).
- [4] Peter G. Anderson, Multiple dimensional golden means, *Applications of Fibonacci Numbers*, Vol. 5, G. Bergum, N. A. Phillipou, and A. F. Horodam, Eds., Kluwer, Boston, 1993, pp. 1-10.
- [5] B. E. Bayer, An optimum method for two-level rendition of continuous-tone pictures, *IEEE Int. Conf. Commun.* **1**: 11-15(1973).
- [6] William Bond, RIT Master's Project, 1994.
- [7] Richard Bryant, RIT Master's Project, 1994.
- [8] J. W. S. Cassels, *An Introduction to Diophantine Approximation*, Cambridge Tracts in Mathematics and Mathematical Physics, No. 45, Cambridge University Press, Cambridge (1957).
- [9] Donald E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1975.
- [10] Reiner Eschbach, Xerox Corporation, Personal communication, 1993.
- [11] Laurene Faucett, *Neural Network Architectures*, Prentice Hall, Englewood Cliffs, NJ, 1993.
- [12] James D. Foley, et al., *Computer Graphics: Principals and Practices*, Addison-Wesley, Reading, MA, 1990.
- [13] Ming Ge, RIT Master's Project, 1994.
- [14] Edward Klehr, RIT Master's Project, 1994.
- [15] Roger S. Gaborski, Peter G. Anderson, Chistopher Asbury, and David Tilly, Genetic algorithm selection of features for hand-printed character identification, *Artificial Neural Networks and Genetic Algorithms, Proceedings of the International Conference in Innsbruck, Austria, 1993*, R. F. Albrecht, C. R. Reves, and N. C. Steele, Eds., Springer-Verlag, pp. 101-106.

- [16] U. Grenendar and W. Freiberger, *A Short Course in Computational Probability and Statistics*, Applied Mathematical Sciences 6, Springer Verlag, New York, NY (1971).
- [17] Marvin L. Minsky and Seymore A. Papert, *Perceptrons, Expanded Edition*, The MIT Press, 1988.
- [18] H. Niederreiter, Quasi Monte Carlo methods and pseudo-random numbers, *Bull. AMS*, **84**(6): 957-1041(1978).
- [19] Ken C. Pohlmann, *The Compact Disk: A Handbook of Theory and Use*, A-R Editions, Inc., Madison, WI, 1989.
- [20] Lenna Sjööblom, "Swedish accent," *Playboy*, **19**(11): 135–141(1972).
- [21] William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, *Numerical Recipes in C.*, Cambridge University Press, 1988.
- [22] Will Snyder, Ph.D. Thesis, RIT, 1994.
- [23] Adam Stein, RIT Master's Project, 1994.
- [24] G. W. Stewart, *Introduction to Matrix Computation*, Academic Press, 1973.
- [25] Uma Srinivasan, "Polynomial discriminant method for handwritten digit recognition," *SUNY Buffalo Technical Report*, December 14, 1989.
- [26] Alan Swires, RIT Master's Project, 1992.
- [27] Robert Ulichney, *Digital Halftoning*, MIT Press, Cambridge, MA, 1987.
- [28] John Vanderkooy and Stanley Lipshitz, "Resolution below the least significant bit in digital audio systems with dither," *Journal of the Audio Engineering Society* **32**(3):106–13, March, 1984. Correction, *Journal of the Audio Engineering Society* **32**(11):889, November, 1984.