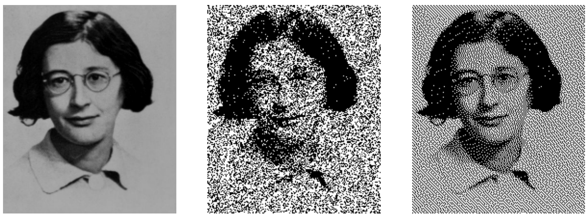
How to hide data in dithering patterns

In this note we describe a simple method for encoding arbitrary data in dithered binary images. The density is about 0.25 bits per pixel in non-saturated regions, and zero bits in saturated regions. Unless the encoded data has some pattern, the encoding is not visible.

1 Description of the method

Sometimes you need to represent gray-scale data by black and white pixels. The simplest technique is *random dithering*, where you throw a random binary pixel with the probability of being white determined by the gray level. Random dithering is trivial to implement, but it loses a lot of resolution. A better technique is *error diffusion*, where you traverse the pixels in a certain order order and select the black or white value that minimizes the ongoing average error. Notice that this depends on the order of traversal. For uniform regions it tends to produce visible patterns, and this can be avoided by traversing the pixels in a more or less irregular way (for example, a Hilbert curve is often used).



gray-scale random error diffusion

Since there is a lot of choice when dithering an image, we can encode a lot of information in these choices. Assuming that we will be able to recover the binary image exactly, the simplest way to encode the data is to have a **table of patterns** such as this:

pattern																
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
intensity	0	1	1	2	1	2	2	3	1	2	2	3	2	3	3	4
group	–	a_0	a_1	–	b_0	c_0	c_1	d_0	b_1	e_0	e_1	d_1	–	f_0	f_1	–

A binary image is thus divided in 2×2 cells, and each cell is identified with one of the patterns of the table (cells marked with "–" are not used). Then the pairs of patterns x_0 and x_1, which have always the same intensity, are considered equivalent and each of them is used to encode a bit of information, losing the original pattern.

The *carrying bit content* of a binary image is defined as the number of 2×2 cells that match a valid pattern in this table. Notice that saturated regions (either black or white) can not encode any information, so that it is better to avoid them as much as possible. They can be avoided, for example, by applying a retinex-like transform in the input image, before dithering.

	original	$\gamma = 0.5$	$\gamma = 2$	retinex
gray				
binary				
bytes	987.25	232	343.5	881.875

The following figure shows the effect of the actual encoding. We encode a stream of random bits, and a stream of zero bits. Notice that the stream of zeros introduces a visible pattern in the image. To avoid these patterns, the data to be encoded must have a uniform distribution (for example, by compressing it).

			(click on each image to see it bigger)
input	random	zeros	

2 Implementation

A C implementation of this technique is available in imscript, as the program `mdither`. All the experiments described in this page have been created automatically by extracting the comments in the source (see the HTML source to view them).

2.1 Floyd-Sternberg dithering

To binarize a gray-scale image by Floyd-Sternberg dithering you can use the program “`dither`”

```
dither i/weil.png weil-dit.png
```

weil.png

weil-dit.png

2.2 Counting the carrying capacity of an image

The program “`mdither count`” prints the number of bits, bytes, kilobites and megabytes that can be potentially encoded on a given image

```
mdither count weil-dit.png > weil-capacity.txt
```

```
15218 bits 1902.25 bytes 1.85767 k 0.00181413 M
```

2.3 Encoding bits into a carrier image

The program “`mdither encode`” encodes a stream of bytes into a carrier image. In the following example we encode a random stream of bits and a stream of zeros in the same carrier image.

```
mdither encode weil-dit.png weil-random.png < /dev/urandom
mdither encode weil-dit.png weil-zeros.png  < /dev/zero
```

weil-random.png

weil-zeros.png

2.4 Decoding bits from an image

And this information can be extracted by the program "`mdither decode`":

```
mdither decode weil-random.png | hexdump -vn 128 > weil-random.txt
mdither decode weil-zeros.png  | hexdump -vn 128 > weil-zeros.txt
```

Contents of file `weil-random.txt`:

```
0000000 b91e 61c6 31b3 a802 0c14 b3fa 2bb3 5388
0000010 ce24 35c8 131a bbd7 4670 4a06 b72d 5b28
0000020 f981 8c55 8936 b8a5 7465 cf99 d7aa 1649
0000030 0253 1fac 0324 11dc 91b6 8f75 866e 6588
0000040 b3af aceb b254 f20e 354f 5184 9a3e 30c6
0000050 b34b 6b03 12ed 1c58 6905 a494 6df7 0327
0000060 8e3c 41df 6602 c68f deef d8c2 1576 a15c
0000070 bf60 fa8d 3970 9594 d62b 5a5e 214b 0369
0000080
```

Contents of file `weil-zeros.txt`:

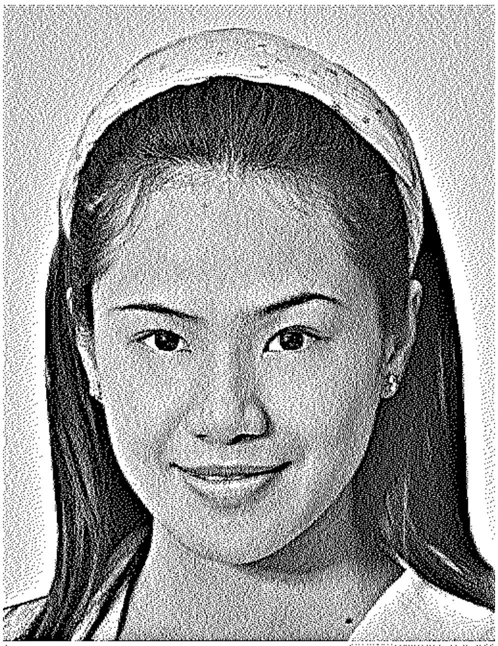
```
0000000 0000 0000 0000 0000 0000 0000 0000 0000
0000010 0000 0000 0000 0000 0000 0000 0000 0000
0000020 0000 0000 0000 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0000 0000 0000 0000 0000
0000040 0000 0000 0000 0000 0000 0000 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
0000060 0000 0000 0000 0000 0000 0000 0000 0000
0000070 0000 0000 0000 0000 0000 0000 0000 0000
0000080
```

3 Examples

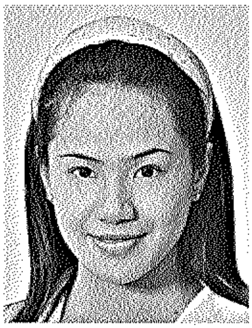
Here we show examples of random bits encoded into the example images of this project, using different resolutions. En each case, we show the binary image along the number of bytes of encoded information it contains.

In all cases, the images were pre-processed by a linear retinex filter and a contrast change that forces the background to be a light-gray (in order to maximize the available space for encoding the information).

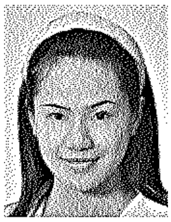
3.1 Test image "photo 1"



7214.5



1694.38



742.25

411.625

252.5

3.2 Test image "photo 2"

8466.88

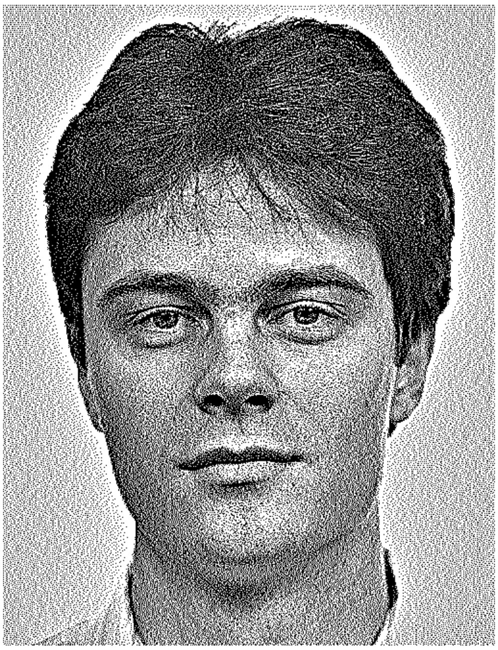
1903.38

815.75

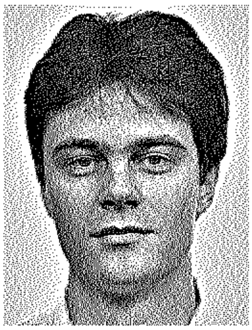
452.875

282

3.3 Test image "photo 3"



7955.5



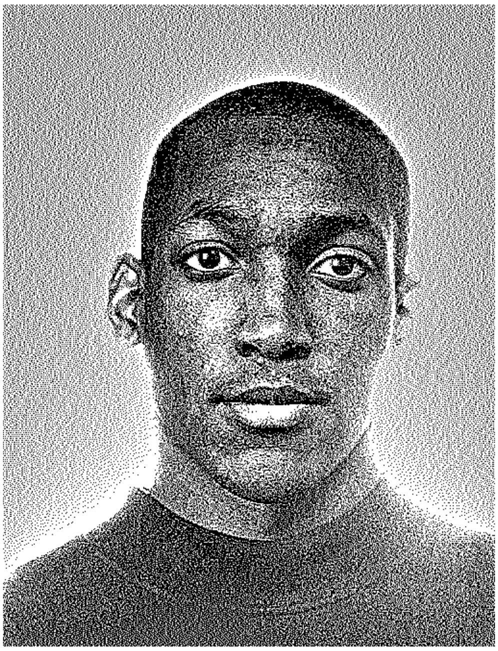
1794.75

749.125

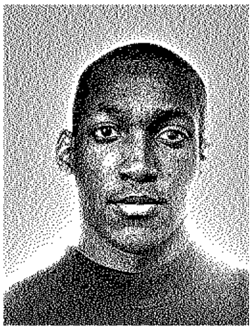
413

257.25

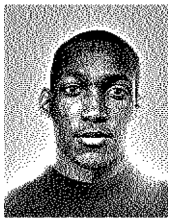
3.4 Test image "photo 4"



8424.38



2004.38



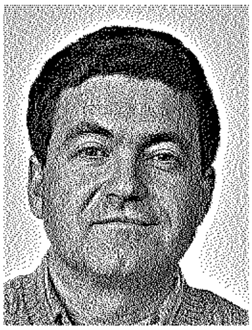
854.875

474.75

298.25

3.5 Test image "photo 5"

8019.5



1849.62

803.375

453.875

284.75

4 Conclusion and ongoing work

This note shows that a simple criterion suffices to encode a *linear amount of bits* into dithered binary images *without noticeable loss in visual quality.* The method achieves an average 25% efficiency (1/4 bits per pixel) for images without saturated regions.

Most of the improvements can be obtained by changing the table of patterns.

Here are some possible improvements:

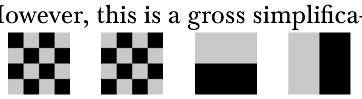
1. Improve the efficiency to 50%. This is a very low-hanging fruit. It suffices to change slightly the table of patterns so that the three groups of patterns of the same intensity belong to the same group, then we can encode 2 bits on each, effectively duplicating the available information content.

2. Improve the efficiency to near 100%. Of course full-efficiency is impossible because all the information would be used for the encoding and none would be available for the carrier image. But we can easily have a much higher rate by using a table of patterns with larger cells. For example, by using cells of size 3×3, there are 512 possible cells, and their intensities, between 0 and 9, are arranged in groups according to the 9th line of pascal triangle

$$1\ 9\ 36\ 79\ 102\ 102\ 79\ 36\ 9\ 1$$

   Now, by conflating the three central intensities, we have more than 2^8 possible cells, and we can encode 8 bits of information on their choice, thus reaching an information content of $8/9 = 88.9\%$ on these cells, and about 75% on average (assuming a uniform distribution of the possible cells). By using larger cells it is clear than we can get a theoretical efficiency as high as we want (at the price of final image resolution).

3. So far we have assumed that cells of the same average intensity can be interchanged with negligible information loss. However, this is a gross simplification. For example, in the following table

   A B C D

   the 4×4 cells A and B can be probably identified without loss of visual information, but changing between C and D will have a greater visual impact. This means that cells of the same intensity are not necessarily equivalent, and by carefully building the table of patterns we may obtain much better results for a given carrying capacity.

4. In the light of the previous observation, it might be interesting to build the table of patterns as the optimum of a variational criterion that minimizes visual loss while achieving an optimum carrying capacity. Notice that this algorithm can be very expensive to run, but only upon the creation of the table. Once the table is decided, encoding and decoding are fast, real-time operations. This will surely produce better tables than hand-crafted ones.