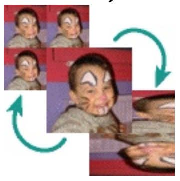
BouMaton v1.5, 2006-02-10



Frédéric Boulanger

1 What is it?

BouMaton is an application which computes two transforms on pictures:

- the **photomaton** transform dispatches the pixels of an image toward its four corners, so that the result looks like four identical sub-images, just like in the automatic photo machines you use to get a picture for your ID card;
- the **boulanger** transform stretches the image horizontally and then folds it so that it has the same size as the original. This is similar to what a baker does when he kneads bread, hence the name (boulanger is the french word for baker).

Both transforms retain all the information in the picture, so it is always possible to retrieve the original image by applying the reverse transform. Moreover, these transforms are periodic: After a given number of iterations, the original image comes back.

BouMaton is able to apply these transforms and their reverse transforms any number of times to a picture. It also computes the periods of the transforms, and keeps a history of the transforms so far applied. It can also save the resulting picture in JPEG format if the necessary Java classes are available on your computer.

2 Quick start

BouMaton is a Java application. On many platforms, double-clicking the BouMaton.jar file will launch it. In a command-line shell, use `java -jar BouMaton.jar` to launch BouMaton. On Mac OS X, double-click the BouMaton application (but you can also use the BouMaton.jar archive or the command line).

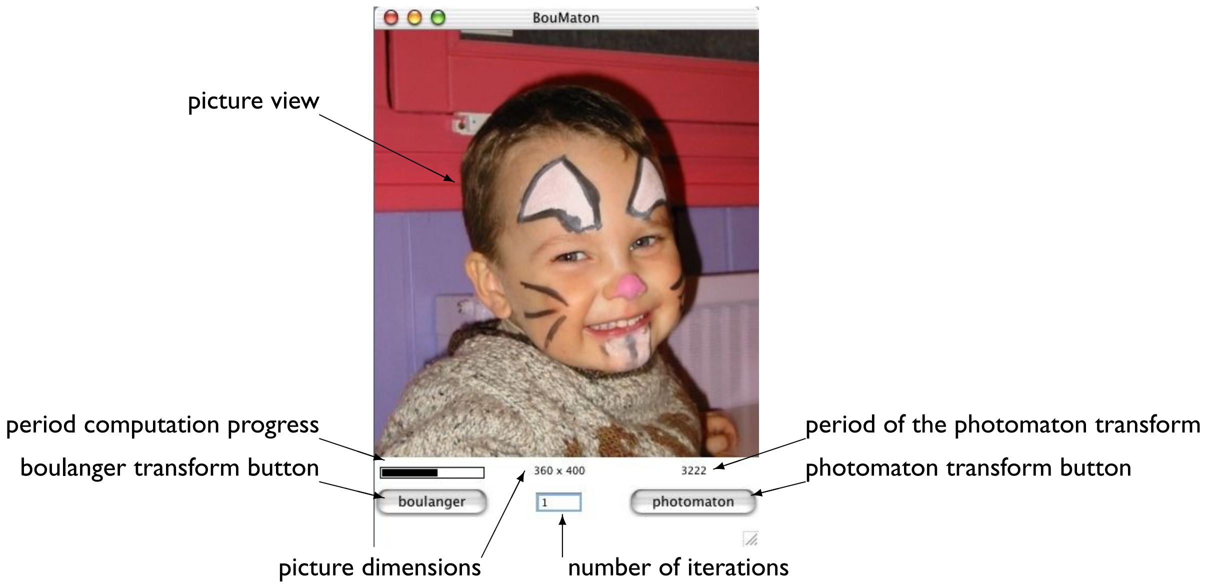
Once BouMaton is running, select the **Open...** item in the **File** menu to open a picture. The formats allowed for the pictures are those supported by the javax.imageio package: at least jpeg, gif and png.

Click the **boulanger** or **photomaton** button, or select the corresponding item in the **Transforms** menu to apply a transform to the picture. You can set the number of iterations (negative for reverse transforms) in the text field between the two buttons.

If you started the computation of a large number of transforms on a big picture and find that it takes too much time to complete, you can interrupt the computation with the **Interrupt** item of the **Transforms** menu.

3 A more detailed description

The main window of BouMaton contains the following items:


BouMaton
picture view
period computation progress
boulanger transform button
360 x 400
3222
period of the photomaton transform
photomaton transform button
boulanger
1
photomaton
picture dimensions
number of iterations

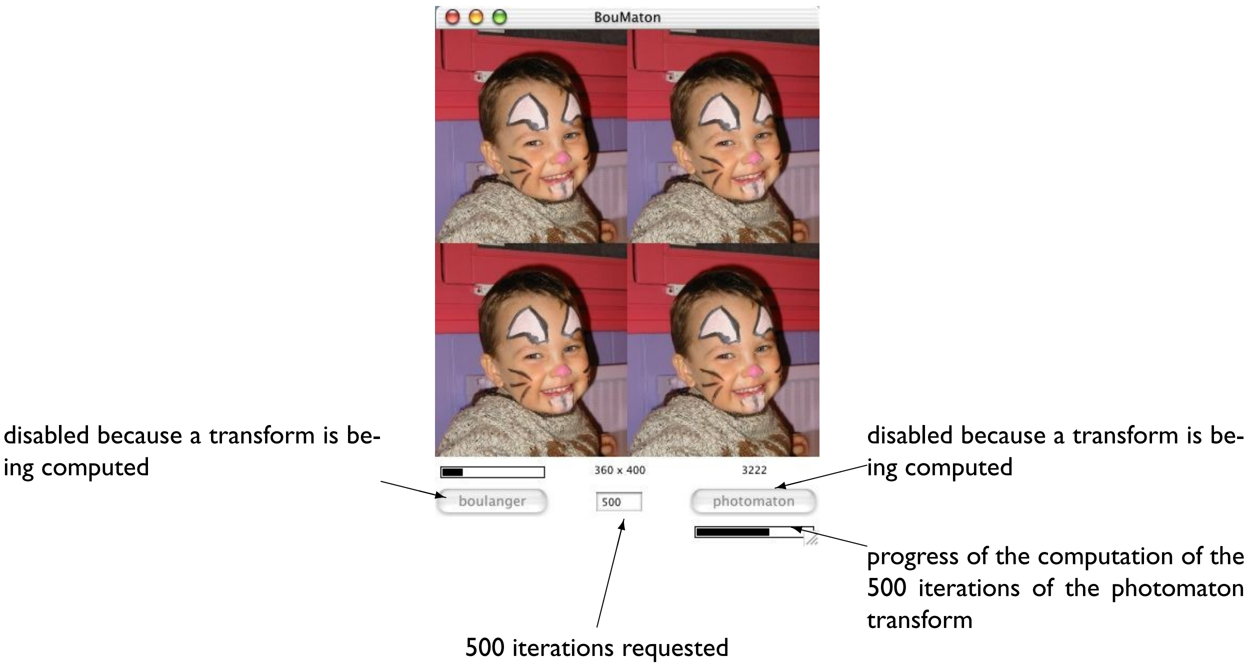
The picture view area displays the result of applying the transforms to the picture. The period computation progress bar indicates that the period of the boulanger transform of this picture is being computed. When the computation is over, the progress bar is replaced by the value of the period.

The **boulanger** and **photomaton** buttons trigger the computation of the corresponding transform, applied as many times as indicated in the number of iterations text field.

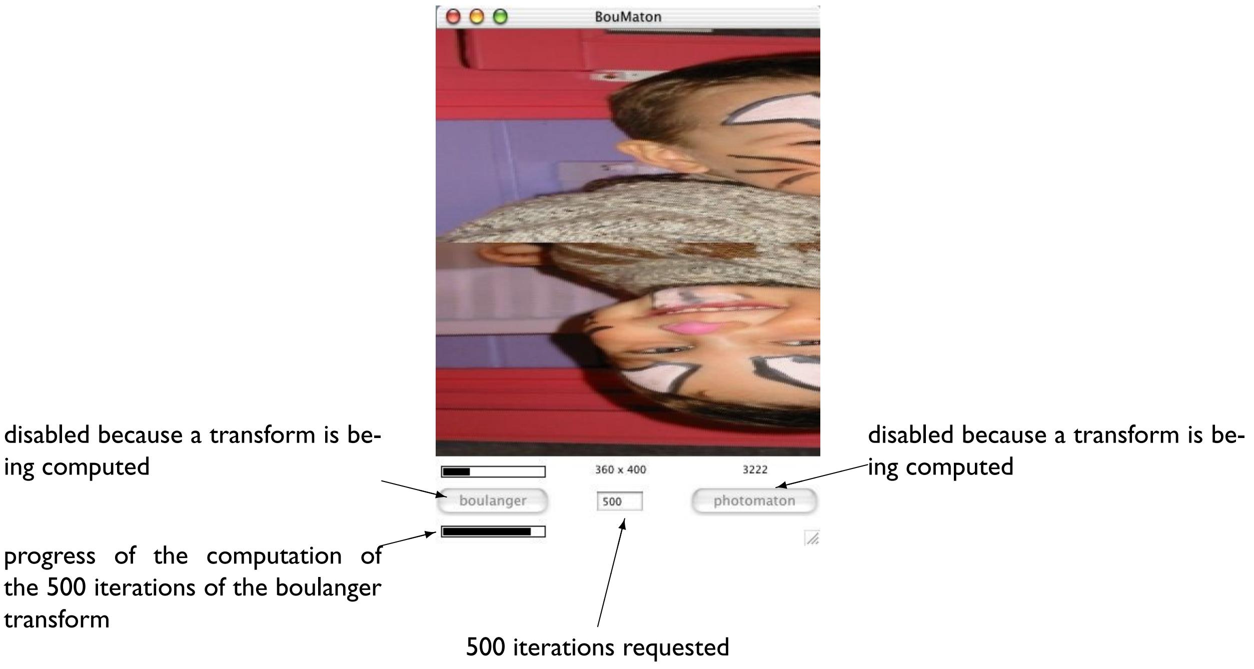
On the right side, you can see the period of the photomaton transform for this picture (which is much faster to compute than the period of the boulanger transform).

In the following example, the view displays the result of the photomaton transform applied once to the original picture, and the program is computing 500 iterations of the photomaton transform.

The picture view area does not display a live update of the picture as it is being transformed, since this would slow down the computation too much. The view is updated only when the computation of the transforms is over. This is why in this example, you see the result of the photomaton transform applied only once to the original picture, even if more than half of 500 requested iterations have already been computed.


BouMaton
disabled because a transform is being computed
360 x 400
3222
disabled because a transform is being computed
boulanger
500
photomaton
progress of the computation of the 500 iterations of the photomaton transform
500 iterations requested

The behavior for the boulanger transform is similar. In the following example, the view displays the result of the boulanger transform applied once to the original picture, and the program is computing 500 iterations of the boulanger transform:


BouMaton
disabled because a transform is being computed
360 x 400
3222
disabled because a transform is being computed
boulanger
500
photomaton
progress of the computation of the 500 iterations of the boulanger transform
500 iterations requested

4 Saving pictures

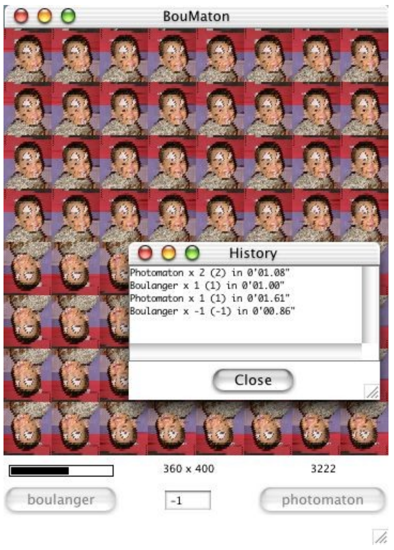
Since version 1.4, BouMaton requires at least Java 1.4 to run, but is able to save pictures in either JPEG or PNG format.

The JPEG format yields smaller files for photographs but ignores details that are considered less important due to the way human eyes work. When you save a file in JPEG format, you can choose how much information will be discarded by setting the quality of the saved picture. The setting ranges from 0 (very aggressive compression, small file, very low quality), to 100 (weakest compression, large file, but high quality). To help you in this choice, BouMaton can compute the estimated size of the file with the current quality setting if you click the **Compute file size** button in this dialog. If the estimated file size is lower than the size of the original JPEG file, your setting is too low. If it is much higher, your setting is too high. However, since the transforms make the picture more and more complex, it is normal that the estimated size grows when you save successively transformed pictures with the same quality setting.

The PNG format discards no information. Therefore, pictures saved in this format tend to use much more disk space than pictures saved in the JPEG format. However, PNG uses a lossless compression algorithm that is more efficient than JPEG compression for pictures that contain only lines and large areas of a uniform color. Since the PNG format does not discard information, you should use it to save transformed pictures that you want to restore to their original form later by applying inverse transforms.

5 History of the transforms

BouMaton keeps track of all the transforms you have applied to a picture. You can view this history by selecting the **Show history** item in the **Transforms** menu:



```
Photomaton x 2 (2) in 0'01.08"
Boulanger x 1 (1) in 0'01.00"
Photomaton x 1 (1) in 0'01.61"
Boulanger x -1 (-1) in 0'00.86"
```

Consecutive identical transforms are merged in the history, so that two successive photomaton transforms make only one entry in the history. Each entry shows the nature of the transform, the number of iterations, the effective number of iterations computed, and the time used to compute them. The effective number of iterations may be less than the required number since computing 3225 iterations of a transform which has a period of 3222 for the picture is exactly the same as computing only 3 iterations (but takes more time!). With the same period, if you ask for 3200 iterations, BouMaton will compute 22 iterations of the reverse transform since this is much faster.

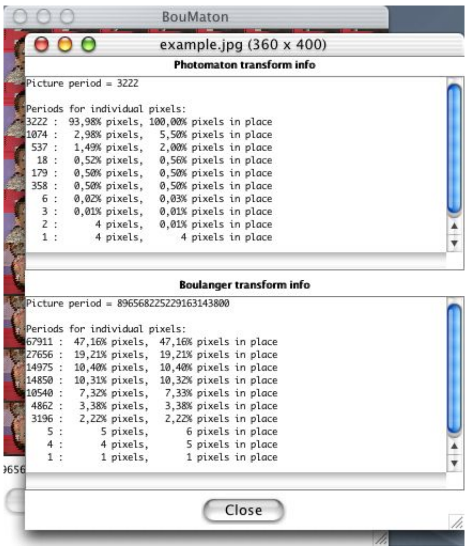
You can always revert to the original picture by choosing the **Revert** item of the **File** menu. This also clears the history.

6 Information about the transforms

While computing the periods of the transforms, BouMaton gathers information about the period of individual pixels. During successive transforms, each pixel follows an orbit (it goes from place to place in the picture) and may come back to its original position well before the whole picture is restored. The period of the transform for the picture is the smallest number of iterations such that each pixel has completed a whole number of turns on its own orbit.

It may happen that after a given number of iterations of a transform, many pixels are back to their original position, so that the original image seems to appear again in the otherwise random-looking set of pixels.

Selecting the **Show info** item of the **Transform** menu will display all the information gathered so far about both transforms. Information about the boulanger transform is quite long to compute and won't be displayed until the period of the transform is computed.



Photomaton transform info

```
Picture period = 3222

Periods for individual pixels:
3222 :  93,98% pixels, 100,00% pixels in place
1074 :   2,98% pixels,   5,50% pixels in place
 537 :   1,49% pixels,   2,00% pixels in place
  18 :   0,52% pixels,   0,56% pixels in place
 179 :   0,50% pixels,   0,50% pixels in place
 358 :   0,50% pixels,   0,50% pixels in place
   6 :   0,02% pixels,   0,03% pixels in place
   3 :   0,01% pixels,   0,01% pixels in place
   2 :       4 pixels,   0,01% pixels in place
   1 :       4 pixels,       4 pixels in place
```

Boulanger transform info

```
Picture period = 896568225229163143800

Periods for individual pixels:
67911 :  47,16% pixels,  47,16% pixels in place
27656 :  19,21% pixels,  19,21% pixels in place
14975 :  10,40% pixels,  10,40% pixels in place
14850 :  10,31% pixels,  10,32% pixels in place
10540 :   7,32% pixels,   7,33% pixels in place
 4862 :   3,38% pixels,   3,38% pixels in place
 3196 :   2,22% pixels,   2,22% pixels in place
    5 :       5 pixels,       6 pixels in place
    4 :       4 pixels,       5 pixels in place
    1 :       1 pixels,       1 pixels in place
```

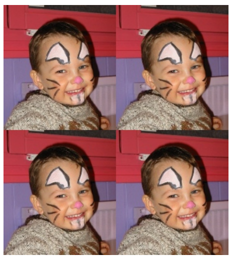
This example shows the information for a 360×400 picture (the periods depend only on the geometry of the picture). We can see that the period of the boulanger transform is quite big, but that after 67911 iterations, 47.16 % of the pixels are back to their original position.

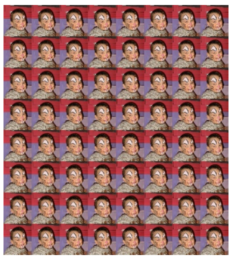
7 Examples

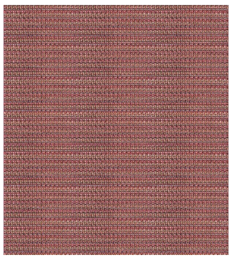
The following series of pictures show the evolution of an image for both transforms, from the original through pictures where all information seems to be lost and others where ghosts of the original appear, and finally back to the original.

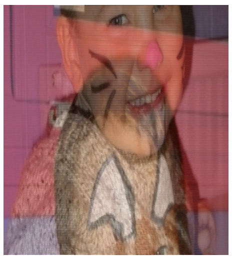
7.1 Iterations of the photomaton transform

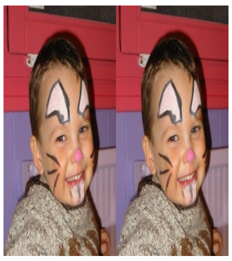
original picture

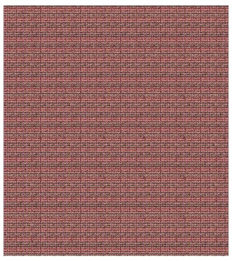

after one iteration

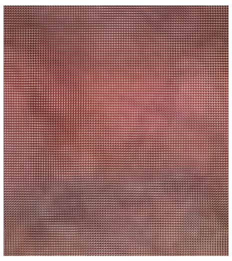

after 3 iterations

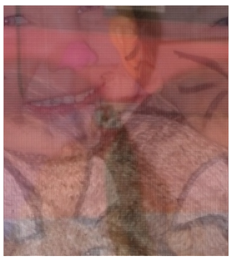

after 8 iterations


something appears after 179 iterations


close to the original after 180 iterations


6 iterations before the original

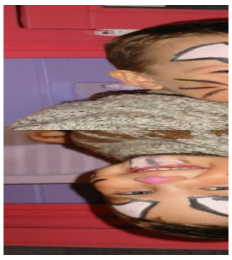

2 iterations before the original

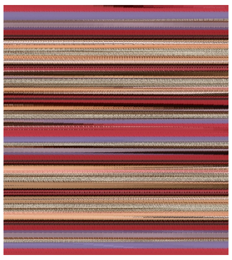

1 iteration before the original

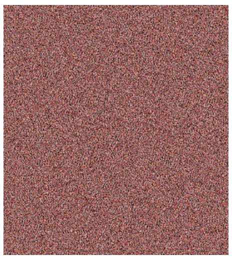
back to the original after 3222 iterations

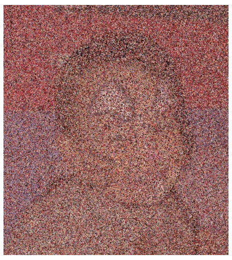
7.2 Iterations of the boulanger transform

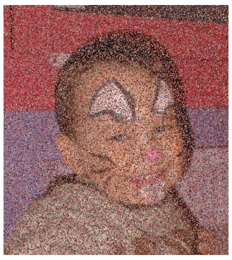
original picture

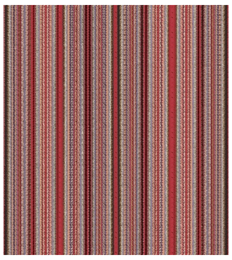

after one iteration

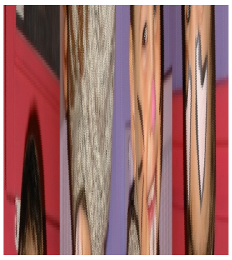

after 4 iterations

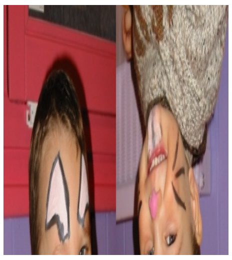

after 20 iterations


19.2% pixels are in place after 27656 iterations


47.2% pixels are in place after 67911 iterations


6 iterations before the original


2 iterations before the original


1 iteration before the original

back to the original after 896568225229163143800 iterations

8 Licence

9 History of BouMaton

Everything began with an article by Jean-Paul Delahaye and Philippe Mathieu in the December 1997 issue (# 242) of “Pour la Science”, the French edition of Scientific American. This article described the photomaton and the boulanger transforms, and I started coding these transforms in C. To avoid writing a real application, I used the plug-in API of GraphicConverter, a very smart application for Mac OS which reads and writes almost any picture file (`http://www.lemkesoft.com/us_gcabout.html`). The result was two plug-ins named PhotoMaton and Boulanger (`http://wwwsi.supelec.fr/fb/Development.html#progs`).

GraphicConverter switched to Carbon when Mac OS X appeared, and instead of converting my plug-ins to Carbon, I decided to write a Java application combining the functionalities of both plug-ins. This allowed me to keep a history of the transforms applied to a picture and to display information concerning the period for the whole image and for individual pixels. Using BigIntegers, I was able to fix a bug in the computation of the period of the boulanger transform which often overflows unsigned long integers.

The result is BouMaton, and I hope that the users of the GraphicConverter plug-ins will appreciate it, and that Windows and Unix users will benefit from the “write once, execute anywhere” Java motto.

Version 1.0 is the first public release of BouMaton.

Version 1.1 fixes a picture update issue.

Version 1.2 improves low-memory conditions handling and adds a dialog to set the JPEG compression quality when saving a picture.

Version 1.3 uses a new algorithm to compute the period of the boulanger transform more quickly.

Version 1.4 uses the javax.imageio package and adds support for saving pictures in PNG.

Version 1.5 fixes a problem with reading PNG images.

10 Technical details

This section explains technical details of BouMaton and may be ignored safely by casual users.

10.1 Translations of BouMaton

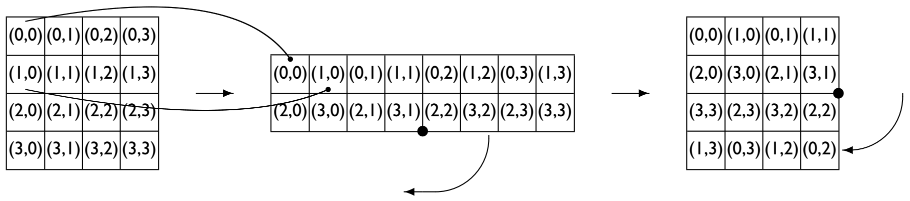
The manual of BouMaton is written in LaTeX. You can start from the English or French versions of the manual and translate it into another language. You should then identify yourself as the translator.

The strings used in BouMaton can also be translated into other languages. The jar file contains a BouMatonStrings.properties file which is for the English (default) version. The BouMatonStrings_fr.properties file is used when the default locale specifies the French language. If you want to translate BouMaton into another language, you have to write a BouMatonStrings_<lang>.properties file, where <lang> is the locale code for your language (for instance de for German, it for Italian and so on). The easiest way to write this file is to start with a copy of one of files I wrote for English and for French, and to translate each phrase which is on the right side of the first "=" sign on a line. The file must be added to the jar file for the new strings to be used when the corresponding locale is active.

If you write a translation for your language, please send me the file so that I can put it in a future release of BouMaton. A translated manual would be appreciated too.

10.2 The boulanger transform

The stretching of the picture in the boulanger transform is done by interleaving the pixels of odd lines with the pixels of even lines. The folding is done by cutting the stretched picture at its original width, and by rotating the right part around its lower left corner, as shown below for a 4×4 picture:



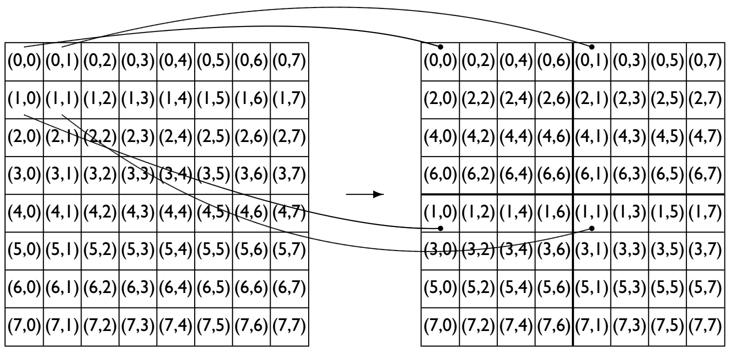
There is no loss of information in this transform, and after a number of iterations, the original image comes back. However, I don't know of a formula to compute this number for all pictures. For a 2^m by 2^m image, the original comes back after $2m + 1$ iterations. For a 2^m by 2^n image, you must wait for $(2m + 1)(2n + 1) + 1$ iterations.

To compute the period of the transform for a given picture, BouMaton computes the period $P(x, y)$ of each pixel of the picture (the number of iterations after which the pixel comes back to its original location). The period P of the picture is the smallest common multiple of all $P(x, y)$. This computation may be quite long, and BouMaton uses a separate thread to make it. Since the period may be very large (896568225229163143800 for a 360×400 picture), BouMaton uses BigIntegers to compute the smallest common multiple of the pixel periods.

Since version 1.3, BouMaton uses a new algorithm to compute this period much more quickly than before. When we are computing the period of a given pixel, we find all the other pixels that belong to its orbit. All those pixels have the same period since they belong to the same orbit, therefore, there is no need to compute their period again. Since there are generally few orbits in a picture, the number of $P(x, y)$ periods to compute is dramatically reduced when using this optimization, and the period of the boulanger transform is now computed as fast as the period of the photomaton transform.

10.3 The photomaton transform

The photomaton transform splits a picture into what looks like four identical copies of the original. However, the four copies are not identical, and the four sub-pictures are obtained by selecting one pixel in each 4×4 square of the original image, as shown in the following illustration for a 8×8 image:



Since no information is lost in this transform, and the number of permutations of the pixels is finite, the original image comes back after a number of iterations. I don't know of a formula to compute the period of the photomaton transform for all pictures. For a 2^m by 2^n picture, the period is the smallest common multiple of m and n, so a 512×512 image will come back after 9 iterations, while a 128×256 one has a period of 56.

However, the period of the photomaton transform is much easier to compute than the period of the boulanger transform because the period of the x coordinate of all pixels with the same x coordinate is the same (all pixels on a same vertical line are back on this line at the same time), and the period of the y coordinate of all pixels with the same y coordinate is the same (all pixels on a horizontal line are back on this line at the same time). The period $P(x, y)$ for a given pixel is the smallest common multiple of the period for x and the period for y. The period for the whole image is the smallest common multiple of all $P(x, y)$.

10.4 Optimizations

When the period P of a transform has been computed, BouMaton uses it to optimize the computation of n iterations of the transform: it computes only $n \bmod P$ iterations. If $n \bmod P$ is closer to P than to 0, it computes only $P - (n \bmod P)$ iterations of the reverse transform. Moreover, for each pixel in the image, BouMaton uses the period for that pixel in the same way to optimize the number of iterations to compute.

10.5 Saving transformed image as JPEG

BouMaton allows to save transformed pictures in either JPEG or PNG format. We have seen that both transforms keep all the information that was in the original picture, but since JPEG compression is lossy, saving a transformed picture in JPEG and loading it later in BouMaton to apply the reverse transforms will not yield an exact copy of the original picture. The global aspect of the picture will be preserved, but colors may be washed out since JPEG compression will have discarded information which seemed not to be critical in the transformed image. There is no such problem with the PNG format since it does not discard any information.

Frédéric Boulanger, 2006-02-10