MRF image format specification Table Of Contents

MRF format

Updated: 1991

NAME

MRF - monochrome recursive format (compressed bitmaps)

DESCRIPTION

This program is part of Netpbm.

MRF is a compressed format for bilevel (1-bit mono) images. It achieves better compression for some such images than either GIF or PNG. (It's also very easy to implement (about the same difficulty as RLE, I'd say) and an MRF reader needs no tables/buffers, which may make it useful for tiny machines).

In case the above hasn't made it sufficiently clear, I'll make this next point explicitly: MRF cannot represent color at all. Nor can it represent grayscale. It's a specifically mono format. (If you want to compress a color or grayscale image, my advice is to use JPEG2000).

First, here's what goes where in an MRF file. I'll explain how the compression works afterward.

Offset	Description
0	magic number - "MRF1" (in ASCII)
4	width (32-bit, MSB first (i.e. big-endian))
8	height (same)
12	reserved (single byte, must be zero)
13	compressed data

Note that there is no end-of-file marker in the file itself - the compressed data carries on right up to EOF.

The way the picture is compressed is essentially very simple, but as they say, the devil is in the detail. So don't be put off if it sounds confusing.

The image is treated as a number of 64x64 squares, forming a grid large enough to encompass it. (If an image is (say) 129x65, it'll be treated in the same way as a 192x128 one. On decompression, the extra area which was encoded (the contents of this area is undefined) should be ignored.) Each of these squares in turn (in left-to-right, top-to-bottom order) is recursively subdivided until the smallest completely black or white squares are found. Some pseudocode (eek!) for the recursive subdivision routine should make things clearer:

```
if square size > 1x1 and square is all one color, output 1 bit
if whole square is black, output a 0 bit and return
if whole square is white, output a 1 bit and return
output a 0 bit
divide the square into four quarters, calling routine for
each in this order: top-left, top-right, bottom-left, bottom-right
```

(Note that the "output a 0 bit" stage is not reached for squares of size 1x1, which is what stops it recursing infinitely. I mention this as it may not be immediately obvious.)

The whole of the compressed data is made up of the bits output by the above routine. The bits are packed

into bytes MSB first, so for example outputting the bits 1,0,0,0,0,0,0,0 would result in a 80h byte being output. Any ‘unused’ bits in the last byte output are undefined; these are effectively after EOF and their value is unimportant.

If writing that sounds too much like hard work :-), you could always adapt **pbmtomrf** and/or **mrftopbm**. That’s the main reason their source code is public domain, after all.

Above, I said the contents of any extra area encoded (when a bitmap smaller than the grid of squares is compressed) is undefined. This is deliberate so that the MRF compressor can make these unseen areas anything it wants so as to maximize compression, rather than simply leaving it blank. **pbmtomrf** does a little in this respect but could definitely be improved upon.

mrftopbm’s **-1** option causes it to include the edges, if any, in the output PBM. This may help when debugging a compressor’s edge optimization.

Note that the "F" in the name "MRF" comes from "format," which is redundant because it is the name of a format. That sort of makes "MRF format" sound as stupid as "PIN number," but it’s not really that bad.

SEE ALSO

mrftopbm, **pbmtomrf**

AUTHOR

Russell Marks.

Index

NAME
DESCRIPTION
SEE ALSO
AUTHOR