Threshing EPS files

Bogusław Jackowski, Piotr Pianowski, and Piotr Strzelczyk
BOP s.c.
ul. Piastowska 70, Gdańsk, Poland
B.Jackowski@gust.org.pl\\ P.Pianowski@gust.org.pl\\ P.Strzelczyk@gust.org.pl

Abstract

In this article we describe the CEP package for compressing EPS files. It belongs to the public domain and was released at the GUST meeting in Bachotek, 1997.

The amount of disk space occupied by bitmap graphics is a well-recognized problem. For example, a 300 dpi picture (A4) contains ca 8700000 pixels; assuming that each CMYK pixel occupies four bytes, one obtains ca 35MB of disk space needed to store the picture.

Now, imagine a TEX-er, who is not allowed to use binary graphic data (because of the otherwise magnificent DVIPS); thus our poor TEX-er usually converts the binary data to hexadecimal EPS files, thus doubling the required space, and next, after compiling a document with TEX+DVIPS, the whole graphic data is put into the resulting PostScript file, so the required space is doubled again — altogether 140MB per one A4 page. The nightmare begins...

This problem is not a new one; it was recognised by Adobe a relatively long time ago. In the PostScript Level 2 specification, they included objects called filters which enable data compression. In particular, instead of hexadecimal data, one can use ASCII85 encoding (there are explanations of abbreviations at the end of the article), run length compression, LZW compression, DCT (used in JPEG files), and many others. Why not make use of these tools? The question is not as silly as it may look at the first glance, as there exist relatively few applications capable of generating well-compressed PostScript graphics.

We decided to patch somehow this gap. We developed a little package enabling the compression of "normal" (non-compressed) graphic data. The nature of the problem is more complex, however, than one might expect. In particular, a universal, always efficient compression technique does not exist. Choice of an optimal algorithm depends upon the kind of data, form of the file and the expected application. Hence, the package has several "buttons" which enable controlling various aspects of compression.

Actually, the name CEP is derived from "compressed EPS". Coincidentally, the name in Polish means "the flail". We hope that others find threshing EPS files useful, in order to get rid of chaff, i.e., redundant data.

About the program

Our package consists of two pairs of AWK programs (`cep.awk`–`uncep.awk` and `cop.awk`–`uncop.awk`), four MS-DOS batch files and text information. `cep.awk` and `cop.awk` generate (on-the-fly) PostScript programs which, processed by Ghostscript, yield the appropriate data compression. UNCEP and UNCOP accomplish (using a similar technique) the reverse process, i.e., uncompression.

CEP is devised for the compression of the usual bitmapped EPS files, containing a single, hexadecimally-coded image; COP can be used to compress any PostScript data.

The question arises: Why use two packing techniques? The answer is simple: the efficiency of compression is higher if a compression program knows in advance which kinds of data are to be expected. In general, bitmaps are more regular (redundant) than arbitrary PostScript data, hence even simple algorithms turn out to be more efficient.

Tests show that in the best case (screen dumps) squeezing up to 10% of the original size is nothing unusual. Sometimes, however, no compression method gives a satisfactory result. In such a case, one can always use encoding data using the ASCII85 filter, obtaining a reduction of a hexadecimal bitmap size by approximately 35%.

Below we give a brief description of CEP and COP. So far, only the MS-DOS version of the PostScript-compressors is available, but it should be easy to adapt our package to any platform, where GAWK and Ghostscript are available. In this version the GNU implementation of AWK (`GAWK-EMX.EXE`) and Aladdin Ghostscript interpreter (`GS386.EXE`) are used.

We tested the package using several Ghostscript and GAWK implementations; now we use Ghostscript 5.10 and GAWK 3.0.3.

CEP

The CEP subpackage consists of the MS-DOS batch files `cep.bat` and `uncep.bat` and the AWK programs `cep.awk` and `uncep.awk`. First, AWK inspects the source EPS file doing its best to recognize a position of a hexadecimal bitmap; next it creates an appropriate PostScript program; and then the control is passed on to Ghostscript which just performs the submitted program: encodes the bitmap and copies verbatim the remaining lines. The original preamble is slightly modified; nevertheless, all DSC comments are left intact.

If the bitmap cannot be found or the AWK suspects that troubles may arise, the CEP engine gives up.

The resulting file should be verified prior to removing the original one, as the CEP heuristic tricks may fail to fix the bitmap properly; moreover, due to Ghostscript bugs, premature removal of the source may also be painful.

CEP never generates binary output — only hexadecimal or ASCII85 encoding are supported. This is due to the fact that CEP-compressed EPS files are primarily meant to be used by TEX+DVIPS. Nevertheless, the resulting files can be used in other typesetting systems as so-called placeable EPS files. The applicability to non-TEX applications, however, is somewhat limited, as binary TIFF previews (required by WYSIWYG applications) may be misinterpreted by (G)AWK.

UNCEP requires that a CEP-compressed file was not changed. In particular, it relies on the information in a quasi-DSC comment `%UNCEPInfo:`. This information can be destroyed by a seemingly innocent modification (e.g., by adding or removing a comment line). Note that the technique employed by CEP destroys, by its nature, the information about the line-breaking structure of the hexadecimal bitmap. Therefore, UNCEP cannot retrieve the original file. Line-breaking structure does not make any problem for a PostScript interpreter. There exist programs, however, that read their own bitmapped EPS files, which for unknown reasons make use of such (sub)lexical information; Aldus PhotoStyler is a notable example.

The command line invoking CEP is pretty simple:

`cep.bat` ⟨*in_file*⟩ ⟨*out_file*⟩ ⟨*options*⟩

One should remember that the names of input and output files must differ. The program recognizes the following options:

`8` — use ASCII85 coding (default)
`h` or `H` — use HEX (hexadecimal) coding
`r` or `R` — use RLE (RunLength) compression (default)
`l` or `L` — use LZW compression
`f` or `F` — use Flate compression (PDF and Level 3)
`n` or `N` — don't compress

Invoking UNCEP is even simpler:

`uncep.bat` ⟨*in_file*⟩ ⟨*out_file*⟩

As mentioned, decompression and decoding methods are taken from an input file.

COP

The subpackage consists of the MS-DOS batch files `cop.bat` and `uncop.bat`, and the AWK programs `cop.awk` and `uncop.awk`. COP reads and encodes appropriately the supplied data. No analysis of the PostScript data is performed, as the entire file is encoded without changing even a bit. The only aspect that is taken into account is the DSC comment `%%BoundingBox:`; if it is found, COP inserts this comment in the preamble, otherwise the resulting file does not contain the bounding box information.

COP-generated files are readable by any PostScript Level 2 interpreter.

UNCOP scans the header and deduces from it the method of decompression, hence no options are needed. UNCOP, unlike UNCEP, retrieves precisely the original file. It is still recommended, however, that a user verifies whether the resulting file is properly interpreted by Ghostscript. Due to Ghostscript bugs, premature removal of the source file after compression or decompression may turn out to be painful.

Since COP can be used to compress any data for arbitrary applications, binary encoding is allowed also. The resulting files can be used with typesetting systems that accept so-called placeable EPSs. Unfortunately, binary TIFF previewers make files after compression illegible for PostScript.

The usage of COP is similar to that of CEP:

`cop.bat` ⟨*in_file*⟩ ⟨*out_file*⟩ ⟨*options*⟩

The program recognizes the following options:

`8` — use ASCII85 coding (default)
`b` or `B` — use binary coding
`h` or `H` — use HEX (hexadecimal) coding
`r` or `R` — use RLE (RunLength) compression (default)

`l` or `L` — use LZW compression
`f` or `F` — use Flate compression (PDF and Level 3)
`n` or `N` — don't compress

Observe that binary encoding is, in fact, no encoding at all.

The reverse process, i.e., UNCOP decompression, is also straightforward:

`uncop.bat` ⟨*in_file*⟩ ⟨*out_file*⟩

As with UNCEP, decompression and decoding methods are taken from an input file.

A heap of remarks concerning our package

The applied solution addresses several problems:

1. It is not at all obvious how to determine syntactically where a hexadecimal bitmap begins in an EPS file; semantic analysis (by redefining PostScript primitives `image`, `imagemask` and `colorimage`) is possible, but it also has its limitations; anyway, we decided to recognize a bitmap syntactically, which implied a problem of recognizing such artifacts as `add` or `def` which look like fragments of a bitmap but, in fact, are not.
2. Also, it is not obvious which compression method should be applied for a given data type; usually, ASCII85 encoding is advisable; for pure bitmaps (CEP) RLE compression is satisfactory, although LZW and Flate filters usually produce much better results (the latter seems to be the best); nevertheless, both LZW and Flate encodings have limited usability:
   (a) LZW encoding is not implemented in Ghostscript ver. > 4 due to USA patent law; as a by-pass, Aladdin implemented an LZW-compatible filter which produces non-compressed data (in fact, enlarged by some 10%) readable for any `LZWDecode` filter. You can use an old Ghostscript version, or compile a Ghostscript version containing the real LZW filter at your own risk, but...
   (b) Flate encoding (the same that is used in GZIP) is available on photo-typesetters having implemented PostScript Level 3; it is also used in PDF files. It is safe to assume that Ghostscript ver. > 4 has this filter built-in. With other PostScript devices, in particular commercial ones, the test described in point 6 may prove useful.

   As a rule of thumb we would suggest not to use any compression but ASCII85 for detailed colour photo images. It is just a weakness of all non-lossy techniques — algorithms employed by ARJ, ZIP, LHARC, and others would yield poor results also. A reasonable alternative for data of this kind would be DCT (JPEG) compression.
3. As was mentioned above, ASCII85 encoding can usually be recommended; it added, however, some troubles. First, due to Ghostscript bugs, we decided to add the (dummy) `NullEncode` filter which seems to cure the problem. But there is one more problem: ASCII85-encoded bitmaps may contain lines looking like DSC comments, i.e, they may begin with double percent signs, `%%`, or with a percent-exclamation sign pair, `%!` — why didn't Adobe exclude the percent from ASCII85? Some programs may try to interpret pseudo-DSC lines. For example, DVIPS just removes such lines, unless the option `-K0` is not used; on the other hand, leaving DSC comments intact may stupefy document managers.
4. It would be convenient to have some more filters implemented, in particular DCT and CCITT-Fax; both of them, however, make use of some additional input data which makes using them more complex; moreover, it is not clear whether one can find the optimal compression parameters for DCT without a WYSIWYG program; we consider a possibility of one-to-one conversion between JPEG files and EPS files making use of DCT filters; also, a similar conversion between GIF files and EPS files making use of LZW filters can perhaps be implemented.
5. The package conserves the working disk space — no large temporary files are created; roughly, the needed disk space is equal to the size of the source plus the size of the target.
6. The following file may be helpful for verifying whether a given PostScript device is able to interpret compressed EPS files:

```
%!PS-Adobe-2.0 EPSF-1.2
%%Pages: 1
%%BoundingBox: 0 0 540 150
%%EndComments
/Helvetica 8 selectfont
90 rotate
1 2 moveto
(*)
{0 -10 rmoveto gsave show grestore}
255 string
/Filter
resourceforall
showpage
```

```
%%EOF
```

Running this program yields the list of filters for a given device. The error reported during the processing of this file proves that the device is not Level 2 compatible. In such a case, using the CEP package should be abandoned.

7. Bugs and traps:
   (a) Apparently, by preceding the `closefile` command by `flushfile`, one neutralizes an error in GS 3.x (tail of output swallowed).
   (b) Adding (a dummy) `NullEncode` filter neutralizes (probably) another Ghostscript bug: an `ASCII85Encode` filter with a target procedure may produce superfluous EOD marks, i.e., "`~>`" (if things go really badly you can obtain thousands of them). Using the target procedure instead of a file object excludes GS ver. $< 3.x$, because early Ghostscripts didn't support all features of PostScript Level 2. Nevertheless, Ghostscript ver. ≥ 2.6 can be used for compression with hexadecimal encoding (it has a "legal" LZW compression).
   (c) The target procedure mentioned in 7b is, in turn, due to special treatment of the ASCII85-encoded lines that look like DSC comments; this special treatment is breaking lines after the first percent character. It is dedicated to the DVIPS driver which has a dangerous option *remove comments* (`-K1`).
   (d) An artificial form of quitting, i.e., `{2 2 .quit}` instead of `{2 .quit}`, is due to an infinite loop of Ghostscript 3.5x caused by the latter form. The Ghostscript internal operation `.quit` was chosen to provide error handling at the level of the operating system.
   (e) Still, there exist bugs in older Ghostscript that we were not able to neutralize; e.g., some EPS files are properly compressed by GS 2.6, but Ghostscript 2.6 breaks while displaying them; GS 3.51 behaves similarly with other bitmaps. So far, Ghostscript > 4 seems to be the most resistant to the "filter trial", but it also reveals some deficiencies. Let's hope that GS 5.5, which is expected to appear soon and is claimed to have most of PostScript Level 3 features implemented, will be still better.
   (f) Summing up, we would strongly recommend using Ghostscript 4.x or 5.x (possibly with `LZWEncode` compiled in) and GAWK 3.x: GS 4.x is nearly complete implementation of the Level 2 PostScript; GAWK 3.x provides regular expressions for record separators, which makes it possible to force handling end-of-lines in exactly the same manner as PostScript does and, moreover, is more reliable than earlier versions.

CEP for everybody

The packages described herein, we have developed for our own purposes. We make them available to the public in the hope that others will find them useful too. We intend to support the packages, but we cannot guarantee that we will be able to follow the frequency of Ghostscript upgrades.

Note that all copyrights, copylefts, copyups, copydowns, or whatever you wish to call them, concerning all the files in the CEP/COP packages are essentially of the public domain character.

Vocabulary

You may find useful the following short explanation of terms appearing throughout the article.

Ghostscript, GS: A reliable and efficient interpreter of PostScript language by Aladdin Enterprises, available as a free public license product; its current version (in July — 5.10) turns out to be much more reliable than not a few commercial interpreters.

AWK: A popular utility and a programming language for convenient and efficient batch data-reformatting; written in 1977 by Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan.

GAWK: GNU AWK, GNU Free Software Foundation implementation of AWK, written in 1986 by Paul Rubin and Jay Fenlason, with advice from Richard Stallman.

GNU: The initiative of the Free Software Foundation (FSF), a non-profit organization dedicated to the production and distribution of freely distributable software, founded by Richard M. Stallman.

TEX: Public domain typesetting system by Donald E. Knuth of Stanford University.

DVIPS: A popular TEX-to-PostScript driver by Tomas Rokicki of Stanford University.

DSC: Document Structuring Convention — a standard for structuring PostScript documents designed by Adobe.

ASCII85: Algorithm for coding binary data as 7-bit ASCII text consisting of only printable characters; encodes every four bytes as five characters from `%` to `u`; additionally `z` is used to code four zeros (see PostScript Language Reference Manual, second edition, pp. 128–130).

RLE: Run Length Encoding — a standard method of data compression (see PostScript Language Reference Manual, second edition, pp. 133–134).

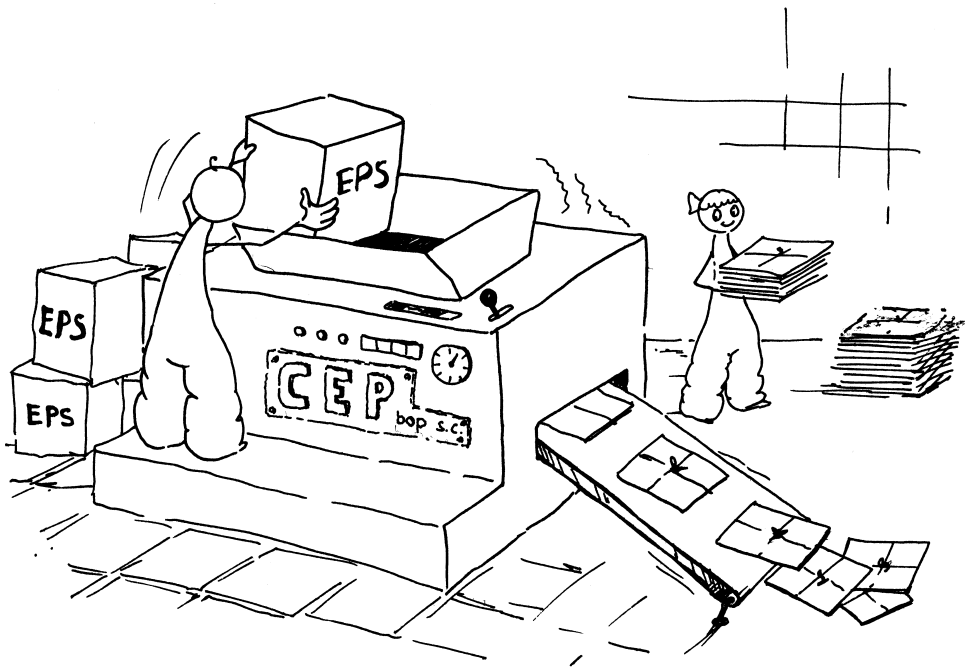
LZW: An algorithm of data compression by J. Ziv, A. Lempel (1978), improved by T. Welch (1984); Unisys, at the time Welch's employer, was granted a US patent in 1985 on Welch's algorithm; a grandfather clause was established by Unisys to make pre-1995 implementations of LZW code free of royalty requirements, thereby eliminating such claims on UNIX compress (information from Nelson H. F. Beebe, e-mail: `beebe@math.utah.edu`).

DCT: Discrete cosine transform compression, an elaborate, very efficient but lossy compression scheme, used in JPEG file format.

JPEG: Joint Photographic Experts Group, an organization responsible for developing an international standard for compression of image data; the PostScript (Level 2) `DCTEncode` filter conforms to the JPEG-proposed standard.

GZIP: Compressing utility by GNU Free Software Foundation, based on a superior and unpatented compression algorithm (modified Lempel and Ziv algorithm), developed in order to get rid of the patented LZW algorithm. It has become a standard compression tool in UNIX systems.

Flate: Filter in PostScript Level 3 based on the compression algorithm used by GZIP; the name is the truncation of the words "inflate" and "deflate".


EPS
EPS
EPS
CEP
bop s.c.