

Inserting figures into TeX documents

Szabó Péter

Budapest University of Technology and Economics

<http://www.inf.bme.hu/~pts/> (without the backslash)

Abstract

* My article deals with the practical considerations of inserting sampled figures into TeX documents. Sub-topics:

- Choosing the appropriate compression for sampled images. Software patents. PostScript and PDF compatibility.
- Conversion of PNG, JPEG, TIFF and GIF files to PDF (for pdfTeX) and EPS (for dvips). `sam2p`, a conversion utility I've developed recently.
- Creating pixel-based figures with transparent pixels. Transparency support in `sam2p`.

The `LATEX` `graphics` package and `epsf.tex` for plain TeX provide an easy and fairly standard way for including EPS figures into TeX documents with PostScript target, and pdfTeX together with the `graphics` package supports PDF figures in `LATEX` documents with PDF targets. This works fine for vector images, because serious editors have EPS export capability, and one can convert an EPS file to PDF with several utilities (such as `epstopdf` bundled with `teTeX`, `ps2pdf` bundled with `Ghostscript`, `PSTi11` and `Acrobat Distiller`).

Inclusion of raster (bitmap) images seems to be even simpler and less troublesome at the first glance. Under UNIX, the `convert` utility of `ImageMagick` can be used to create a PBM or PGM file, which can be converted to EPS by the `pnmtops` utility of `NetPBM`. Filters mentioned in the previous paragraph can be used to create a PDF file from the EPS. There are several other approaches, which will be described later in this article.

But high-quality raster image files tend to be huge, and thus documents containing raster images can be only transferred on the network only slowly, and printing is even more slower. It is common that a PostScript version of a small article with a few dozen raster illustrations becomes larger than 100 MB, and printing each page on a personal printer takes half an hour. Software techniques such as compression and colorspace transformation can be used to make these images much smaller, and PostScript and PDF support much of these techniques, but unfortunately currently no free utilities

exist that give the user full control over the image representation parameters.

This article analyses the problem and possible solutions in detail, and presents the new utility `sam2p` written by the author of the article, that implements these solutions.

Concepts Although PostScript and PDF are most often used for faithfully representing two-dimensional vector graphics, they have sophisticated features for displaying sampled image data as well. Unfortunately the PostScript and PDF output facilities of existing image converters and editors don't make use of these advanced features: they often produce large, slow and incompatible PostScript or PDF code. `sam2p` is a new sampled image conversion utility that gives the user full control to adjust compliance, compression and encoding of the PostScript and PDF output files.

The concept of sampled images is rather simple: the image is a rectangular array of *pixels*, in which the color of each pixel is independent of the others. A color consists of one or more *components* (*channels*, *planes*): for example, the colors of the RGB color space are composed of a red, a green and a blue component. The name *sampled image* comes from the fact that only a finite number of pixel samples of the continuous picture are stored. (The sampled image is also quantized, e.g. the components are integers in the range 0...255, where 0 means darkest, 255 means lightest.) Sampled images are also called *raster images* or *bitmap images*. Some images have an *alpha channel*, which describes the opacity of the pixels: a pixel having $\alpha = 0$ is fully transparent, and a pixel with $\alpha = 1$ is opaque. Non-rectangular shapes can be represented with a

* The author thanks GUST for funding his participation in EuroBachTeX 2002, and Ferenc Wetzl for testing `sam2p`.

rectangular image having transparent pixels outside the shape, and opacity can be used this way to draw arbitrary shapes atop of each other.

Recent versions of the PostScript and PDF formats fully support the notions defined above. Many features are not available in older PostScript printers, and there are even some features (such as opacity) that are documented in the file format, but not implemented in any common free renderers. Thus, compatibility must be always considered when creating a PostScript or PDF file containing a sampled image. These two file formats are based on a common graphics model, and thus they are almost equivalent. The actual image data (being either compressed or not) is stored the same way in PostScript and PDF files, only the syntax of the surrounding meta-information (such as the image width, height and color space specification) is different.

The most important difference between PostScript and PDF formats is that PostScript is a full-featured programming language, and it gives more power to the programmer to represent the image. For example, one can write a PostScript program that would compute and draw a Mandelbrot or Julia set as a sampled image, but there is no similar capability in PDF (i.e. the pixels have to be pre-computed before the generation of the PDF file). The programming constructs of PostScript can be used to compensate the weaknesses of the interpreter: for example, older interpreters don't support the Flate (ZIP) compression, but this can be overcome by including a pure PostScript implementation of the `/FlateDecode` filter into the EPS file, just before the image data, and executing the supplied code when the filter is missing from the interpreter itself. `sam2p` does this and several other tricks to increase compatibility of PostScript files. Unfortunately this doesn't work for PDF; there is no way to extend the list of available filters.

Raster images are most commonly used as illustrations in more complex documents, thus the holding PostScript and PDF files should be generated embeddable. The design of the PDF file format makes embedding any PDF file (including the most complex ones) very easy, even without the special attention of the generator. PostScript figures, on the other hand, share a common namespace (and very limited memory) with the document they are embedded into, and extreme caution should be taken to avoid conflicts with the document itself and with other figures. Encapsulated PostScript (EPS) is the most common file format of embeddable PostScript code, and most utilities (including

`sam2p`) output sampled PostScript images as EPS. There is a thumbnail/preview feature in the EPS format, but `sam2p` doesn't emit one, because most utilities (including the utilities related to `TEX`) ignore it, and they would call `Ghostscript` to render an independent preview anyway.

Smaller output Sampled images, especially when rendered in high resolution (≥ 300 DPI), tend to be large and occupy much space on disk. A PDF file with many high resolution figures in it gets downloaded slowly from the web, and a PDF file with low resolution images looks ugly when printed. A long PostScript file containing large images prints very slowly, and might even fill up the disk space of the computer running the printer spooler. (Just imagine multiple PostScript files of 200 MB being printed on an old, dedicated print server with small hard disks. This situation often happened to the author.) Large PostScript files print especially slowly, because the parallel-port interface on which the printer is connected to the computer is too slow, or the processor embedded to the printer running the PostScript interpreter is slow. It is really annoying to wait more than a minute for a single A4 page, and even more annoying to wait for someone else's job running at this "speed".

When running `TEX` on an old machine, the user must wait several minutes for `TEX` finishing skipping long EPS files after finding geometry (bounding box) information. The situation is even worse today, when one expects WYSIWYG, or – at least – instant preview. Several minutes is definitely not instant.

Thus, the generated PostScript and PDF files should be as small as possible. This can be accomplished by:

- apply filtering to reduce noise, increase contrast etc.: This should be done manually in graphical image editors. Sometimes even a size reduction of a factor of 4 can be achieved by removing noise and unnecessary information. As part of the filtering, the user should adjust the color palette: black should be exactly black (RGB triplet `#000000`), and white should be completely white (RGB triplet `#ffffff`), because otherwise `Ghostscript` would print glaring gray regions.
- using less bits for representing a single color component: A black-white image needs only 1 bit per pixel, and it would be big waste to include it as an 8-bit RGB image with 24 bits per pixel. Even color images can be encoded with a few bits when using only a palette of very few number of different colors. PostScript and PDF support may

sample formats ranging between 1...33 bits per pixel.

- using compression adequate for the image types: There is no universal method that finds the compression with best ratio, but in general, continuous-tone color or gray images (such as photos or scanned paintings) should be compressed with the JPEG (DCT) compression filter, black-white images should be encoded with the the Group3 Fax (CCITTFax) filter, and ZIP compression (Flate) is quite good for other sample formats. When ZIP compression is not available on the printer, LZW (patented till 19 June 2003) or RLE (poor ratio) can be used.
- image data should be preconditioned for compression: There are several predictors (such as horizontal differencing) that convert similar patterns found in the image to repeating data bytes, so the compression ratio of some filters will be improved.

Sometimes the user has a priori knowledge about the structure of the image, and knows the best combination of compression filters, parameters and predictors. Although many converters support some kind of Postscript or PDF compression, unfortunately free utilities that would respect the exact wish of the user haven't existed before **sam2p**.

A beginner who doesn't want to know about all the details, can use the output profile created by an expert. An output profile consists of several independent rules. For example, if the profile contains a rule with DCT compression, and another rule with Fax compression, **sam2p** will choose automatically the first rule for continuous-tone RGB images, and the second rule for black-white images. When two rules are in conflict (i.e. they are both applicable), the one that has been declared first is applied.

More compatible output The PostScript format has evolved much over time. Newer versions provide color images and more compression filters:

- *PSL1*. This is the very first PostScript specification, called LanguageLevel1, as defined by Adobe in 1985. It doesn't have any predefined filters, and supports only grayscale images directly. Very limited support for RGB images with few different colors exists, using the `setrgbcolor` and `imagemask` operators.
- *PSLC*. This is Level1 PostScript with the CMYK extension. The CMYK color space is not important for **sam2p**, but the extension also adds the `colorimage` operator with the ability of drawing arbitrary RGB (and CMYK) images. This is the standard that ancient PostScript printers follow.

- *PSL2*. PostScript LanguageLevel 2 supports several color spaces (e.g. "indexed"=paletted), and all compression filters except ZIP. Ghostscript, PS till, Acrobat Distiller, and most PostScript printers today understand this LanguageLevel.
- *PSL3*. The most important feature added by Level3 is Flate compression. Ghostscript supports this and many other features of this level. Unfortunately Level3 is too new, it has not been implemented by major printer vendors yet.

The history of PDF files is shorter: *PDF1.0* roughly corresponds to PSL2, and *PDF1.2* added ZIP compression. It is important to note that most PostScript printers cannot print PDF files, because random access (i.e arbitrary file seeking, positioning) would be required, but printers are connected with parallel, USB or TCP interface, which provides only serial data stream access. There are, however, advanced printers that read the whole PDF file into their memory before processing it. As a conclusion, neither PDF or PostScript can fully replace the other one, so **sam2p** supports both.

In **sam2p**, the user can specify the exact version of the PostScript or PDF format the output should comply to, and the program gives an error message on illegal combinations (such as ZIP compression in PSL2). In some cases, tricky PostScript code is used to emulate features of higher levels. For example, RLE and ZIP (de)compression is provided even in PSL1 (!), by inlining the pure PostScript implementation of the decompressor into the EPS file. More examples: indexed images are emulated in PSLC, and a grayscale-mapped version of the `colorimage` operator is provided for PSL1. The implementations of these PostScript tricks are currently incompatible with each other, but the architecture of **sam2p** makes adding more combinations possible. This way, compatibility is not an alternative of file size, but it is an alternative of decompression and printing speed.

sam2p **sam2p** is a UNIX command line utility written in ANSI C++ that converts many raster image formats into Adobe PostScript or PDF files. It gives full control to the user to specify standards-compliance, compression, and bit depths. The images are not vectorized. It is common that **sam2p** can compress an image down to an 50kB Level1 PostScript file without quality loss, while other popular converters produce multi-megabyte output. **sam2p** is unique, because it provides some Level3 compression filters even on Level1 devices.

The program source code for UNIX systems is available from <http://www.inf.bme.hu/~pts/sam2p-latest.tar.gz>. The license is GNU GPL. Documentation, compilation instructions and examples are included. The author can be reached via e-mail at pts@fazekas.hu with the Subject starting with [sam2p]_.

The most important limitations are:

- only DeviceRGB color space is supported, with the Indexed, Gray and RGB image types
- Color depth is 8 bits maximum.
- Indexed images can have up to 256 colors
- alpha channel and transparency supported only for Indexed images: only one color may be transparent

Temporary limitations at 4 April 2002:

- GUI was not working
- the software is in beta state (the author welcomes bug reports)
- PDF output was not implemented
- transparent PS and PDF output is very limited
- some PSL1 tricks were not included

Although `sam2p` is a command line utility (i.e. without a graphical user interface), it doesn't accept any command line options: it must be controlled from "job" files. It expects a single command line argument: the name of the job file. `sam2p` runs that single job, prints debug, info, notice, warning and error messages (etc.), and creates a single output file: a PS or a PDF. For multiple jobs and multiple output files, one has to run the program multiple times.

The syntax of the job file resembles PostScript and PDF. For example:

```
<<%sam2p job file
/InputFile (input.ppm)
/OutputFile (output.eps)
/Profile [
  << /FileFormat/PSL2 /SampleFormat/Mask
    /TransferEncoding/Binary
    /Compression/Fax
  >> ] >>
```

The *Profile* is a list of one or more *Output rules*. Each output rule is a dictionary (written between << and >>), and specifies all the desired properties the OutputFile should have. In the example, there is a single output rule that specifies that `output.eps` should be an EPS file conforming to PostScript LanguageLevel 2, containing the image as a Mask (i.e. only a single, non-transparent color and a transparent color), the file is Binary (can contain arbitrary characters and long lines), and data is compressed with

the CCITT Fax encoding filter. Additional keys and values of the output rule dictionary will be described later.

The Profile contains a single output rule most of the time, but it is possible to specify more than one. `sam2p` tries to follow the output rules in the order they are specified, and if it encounters an inconsistency (not an I/O error!), then it proceeds to the next output rule. For example, one can specify a Profile that works for all grayscale images, and provides the compression adequate for the bit depth:

```
<<%sam2p job file
/InputFile (in.pgm) /OutputFile (out.eps)
/Profile [
  << /FileFormat/PSL1 /SampleFormat/Opaque
    /TransferEncoding/ASCII >>
  << /FileFormat/PSL2 /SampleFormat/Gray1
    /TransferEncoding/A85
    /Compression/Fax >>
  << /FileFormat/PSL3 /SampleFormat/Gray2
    /TransferEncoding/A85
    /Compression/ZIP >>
  << /FileFormat/PSL3 /SampleFormat/Gray4
    /TransferEncoding/A85
    /Compression/ZIP /Predictor 2 >>
  << /FileFormat/PSL3 /SampleFormat/Gray8
    /TransferEncoding/A85
    /Compression/ZIP /Predictor 15 >>
] >>
```

Note that there is no ultimate way to specify best compression; the configuration above may or may not suit the compression of special grayscale images. For example, if the 8-bit images the user would like to compress are photos, then JPEG (`/Compression/IJG /Hints<</Quality 60>>`) is probably a better choice.

The job file is case sensitive, and newlines and spaces are treated as a single space outside strings. The job file contains a single job dictionary describing a single conversion. Other files may be included with the run command, for example:

```
<<%sam2p job file
/InputFile (in.pgm) /OutputFile (out.eps)
/Profile [ (my_profile.job) run ] >>
```

See the documentation shipped with the `sam2p` sources for a complete reference of output rule elements. The most important keys are:

- `/FileFormat`: any of `/PSL1`, `/PSLC`, `/PSL2`, `/PSL3`, `/PDFB1.0`, `/PDFB1.2`, `/PDF1.0`, `/PDF1.2`, `/GIF89a`, `/PNM`, `/Empty`. PDFB creates a PDF without an image dictionary, but with a BI inline image. This is recommended when the PDF file will be

processed by pdfTeX, because pdfTeX forces ZIP compression on non-BI images.

- `/SampleFormat`: specifies the bit depth and color space in which the samples are stored. Possible values are: `/Opaque`, `/Transparent`, `/Gray1`, `/Indexed1`, `/Mask`, `/Transparent2`, `/Gray2`, `/Indexed2`, `/Transparent4`, `/Rgb1`, `/Gray4`, `/Indexed4`, `/Transparent8`, `/Rgb2`, `/Gray8`, `/Indexed8`, `/Rgb4`, `/Rgb8`, `/Asis`. Use `/Asis` together with IJG compression.
- `/WarningOK`: true or false. With false (non-default), this output rule is disabled for an image if it would cause warnings to be printed.
- `/TransferEncoding`: any of `/Binary`, `/ASCII`, `/Hex=/AHx`, `/A85`. `/ASCII` is useful only when no image data will be stored, for example with `/Opaque` and `/Transparent`.
- `/Compression`: any of `/None`, `/LZW`, `/ZIP=/Flate=/F1`, `/RLE=/RunLength`, `/Fax=/CCITTFax`, `/DCT=/JPEG`, `/IJG`, `/JAI`. DCT is JPEG compression with the PostScript `/DCTEncode` filter (creates large JPEG files), `/IJG` is JPEG compression using `cjpeg` by the Independent JPEG Group (recommended, produces small JPEG files), `/JAI` is mostly verbatim inclusion of a JPEG file read from `/InputFile`. `/ZIP` generally doesn't work in PSL2 or PDF1.0, and other filters generally don't work in PSLC. Because of the PostScript trickery, there is a chance that `/ZIP` and `/RLE` works even in PSL1.
- `/Predictor`: any of 1 (no predictor), 2 (TIFF predictor 2), 10...15 (PNG predictors), 45 (alternative implementation of PNG predictor 15). When used together with `/ZIP` or `/LZW`, it may decrease output file size. See [1] for details.
- `/Hints`: a dictionary of several additional parameters and hints, for fine tuning. See the documentation of `sam2p` for details.

`sam2p` can autodetect and read the following image formats:

- PNM, PBM, PGM, PPM. These are the preferred formats for non-transparent images.
- XPM. This is the preferred format for indexed images with transparency.
- BMP (Windows and OS/2 Bitmap)
- GIF (CompuServe Graphics Interchange Format)
- LBM (IFF ILBM)
- TGA (Targa)
- baseline JPEG JFIF (limited support)

Some other input formats (such as PNG, TIFF, PS, EPS, PDF, XBM, XWD, PCX, Utah RLE) are planned in the future.

Implementation considerations `sam2p` manipulates possibly huge raster images. Today desktop computers tend to have more and more system memory, into which the entire image can be read. But free memory shouldn't be wasted: the programmer must have precise control over the memory allocated for image data. The program should be fast, even when converting large images. Scripting languages and Java provide memory management (i.e. automatic memory deallocation, and safety from illegal memory access), but are slow and they waste resources when dealing with image data. Among the popular programming languages, only C and C++ fulfills the mentioned memory usage and speed requirements.

The code would include stacked encoding filters, for example the `ASCII85Encode` filter writes the actual data to the output file, reads the output of the `LZWEncode` filter, which indeed reads the output of a predictor. The filters and predictors can have multiple implementations (such as a truly internal, one using a shared library such as `zlib`, or one that spawns an external process to do the job). This complexity can be easily handled within the object-oriented paradigm, but would be a mess in pure C. Other reasons for C++ beyond supporting objects are the powerful syntactic sugars and the standardness and standards-compliance of compilers: no more confusion and partial implementations with traditional or ANSI C. So `sam2p` is implemented in Standard C++, but without using `<iostream>` or `STL` (which are not available on many systems). However, advanced C++ features (such as nested templates) are not available on some present C++ compilers, so these should be avoided in `sam2p`.

Another important design choice is between monolithic and modular. The philosophy of the author is that the program should provide most functionality without using external libraries or other programs. Interfaces of external libraries tend to change between versions, so a compiled program using a library very probably won't work when a newer version of the library is installed. And – even worse – the program won't compile 3–5 years later, when the “old” version of library isn't available anymore. (As an example, there are plenty of software on the Internet that work only with a rather old GTK version of the library, which is almost impossible to be installed to a modern system already having a modern GTK. Most users and system administrators don't have time to fiddle with old and unavailable libraries, thus they will refuse to install software using these libraries.)

`sam2p` is modular in the source level, but monolithic after it has been compiled. For example, it is quite easy to include a new type of input image format by writing a file `in_XYZ.cpp`, and registering the loader in `sam2p_main.cpp`; after that, `sam2p` has to be recompiled. The program doesn't use any external libraries, but it is able to call external programs, for example, the `cjpeg` utility is invoked when the user requests IJG compression. In future versions of `sam2p`, the user will have a choice between multiple implementations of the same compression filter. `sam2p` doesn't require any C++-specific libraries, not even STL or `libstdc++`.

Modularity is especially important in the following areas: Loaders (that load InputFiles), Appliers (that apply an output rule and create the OutputFiles) and compression filters. Currently it is very easy to add new Loaders and Appliers, because their interface to the main program is small, simple and well-defined. New compression filters will probably never be added, because PostScript and PDF supports only the filters already implemented. However, alternate implementations of existing filters would be important, but the code has to be reorganized a little for that.

`sam2p` is being developed on a Linux system, and currently only flavors of UNIX are supported as run-time platforms. Porting to Win32 or Mac would not be hard, because the user interface is minimal, but the author isn't experienced enough and doesn't have time to do it.

Legal issues The LZW compression is patented: software containing LZW (de)compression code can be distributed only with the prior written permission of Unisys. According to `<mcb@cloanto.com>`, author of `http://lzw.info`, the "exact" answers may come only from lawyers and courts; considering the IBM, the BT and Unisys US patents, then the last of the three would be the Unisys one, expiring on June 19, 2003, 24:00. There cannot be other (new) patents on LZW, as far as he knows.

The GIF file format uses LZW compression, and moreover GIF is a registered trademark of CompuServe.

Thus, to avoid legal problems, LZW compression, GIF import and GIF export are disabled in `sam2p`, until these patents and trademarks expire.

Other utilities `sam2p` was born because the author has tried several existing utilities, and none of them were perfect for his needs. The author has examined the documentation and source code of these utilities, and he has benefited much of them.

- the `convert` utility (from ImageMagick, Debian package `graphics/imagemagick`), with output formats EPS: EPS2: EPSI: EPSF: PS: PS2:. Most of these formats are big and slow, and ImageMagick sometimes produces illegal ADSC comments, so `dvips` cannot embed the image. Apart from ADSC bug, ImageMagick seems to create the most compatible and best embeddable EPS files. The EPS: format should be used for Level1 EPS, and EPS2: for Level2 EPS. The others are obsolete. Partially respects the `-compress` option.
- the `convert` utility with PDF: output format. Only partially respects the `-compress` option, and produces illegal `%PDF-1.1` header with the `/FlateDecode` filter.
- the *Save as PostScript* feature of `xv` (former Debian package `non-free/graphics/xv`). Does RLE compression with PSL1 (!), image can be adjusted to the paper size, but cannot generate EPS.
- the `tiff2ps` utility from `libtiff` (Debian package `graphics/libtiff-tools`). Needs the `-z` and `-e` options; the `-2` (Level2) option is recommended. The scaling (size) of the image is still somewhat obscure, the `scale` operation should be removed from the EPS file by hand. When generating Level2 EPS, `tiff2ps` fully respects the compression of the TIFF file, and produces an equally small EPS. The `tiffcp` utility can be used to adjust the TIFF compression. Unfortunately, old versions of `tiff2ps` don't support Flate compression, and new versions partially support LZW compression due to the Unisys patent. `tiff2ps` seems to be the most advanced EPS-generator, but it still cannot exploit all features of the PDF and PostScript sampled image model.
- the `fax2ps` utility from the same source. Supports only Fax compression.
- the `gif2ps` utility from Debian package `graphics/libungif-bin`. GIF supports only the Indexed color space, and so does `gif2ps`.
- the *Save as PostScript, EPS* feature of The GIMP (Debian package `graphics/gimp`). Creates PSLC EPS with the `colorimage` operator. Compression not supported.
- the *Print to file* feature of The GIMP. Creates Level2 PS, not EPS.
- the `pnmtops` utility from NetPBM (Debian package `graphics/netpbm`). Creates PSLC EPS with `/Hex` and `/RLE`.
- the `g3tops` utility from Mgetty (Debian package `comm/mgetty-docs`). Only Fax compression.
- the `jpeg2ps` utility from Debian package `non-free/graphics/jpeg2ps`. Only `/JAI` and `/A85` is supported.

- the `jpg2pdf` shareware utility, available from <http://www.sanface.com/jpg2pdf.html>. Only the RGB color space, with JPEG compression. Simplistic software with stupid legal limitations.
- The author has investigated several PHP modules that generate PDF, but they were not general enough to support all SampleFormats.
- the `imagetops` utility from CUPS (Debian package `net/cupsys`). This is a smart shell script that invokes `djpeg`, `pngtopnm`, `bmptopnm`, `giftopnm`, `tifftopnm`, `ppmtogpm` and `pnmtops`. No way to specify compression.

Better transparency PSL3 and PDF1.3 provide a better way to specify images with transparency pixels. This way uses patterns and masked images. However, most present printers and utilities don't support these features yet, so these are not included into `sam2p`. For complete support, a new type of color space (RGBA: RGB with transparency) should be implemented.

Currently `sam2p` produces the transparent images by calling the PostScript `imagemask` operator for all different colors. This can be slow and make the output file large and less compressible.

Rendering: the reverse direction It is easy to convert a PostScript or a PDF document into a series of bitmap files, using GNU Ghostscript:

```
gs -g100x200 -sDEVICE=ppmraw -q -dNOPAUSE
-dBATCH -q -dSAFER
-sOutputFile=out%d.ppm infile.ps
```

In the command above, `-r` specifies the DPI resolution, `-g` specifies the image size in pixels (thus it should be doubled for double resolution), `ppmraw` is the output file format (should be `pbmraw` for black-white, `pgmraw` for 8-bit grayscale and `ppmraw` for 8-bit RGB). The files `out1.ppm`, `out2.ppm` etc. will be generated. These files can be loaded into The GIMP, or they can be converted to other sampled image formats with ImageMagick's `convert` utility or the `pnmt*` utilities of NetPBM.

Sometimes it is more convenient to specify the sampled image size in PostScript points (72 bp = 1 in):

```
gs -dDEVICEWIDTHPOINTS=100 -dDEVICEHEIGHTPOINTS=200 -r144 -sDEVICE=ppmraw -q
-dNOPAUSE -dBATCH -dSAFER
-sOutputFile=out%d.ppm infile.ps
```

Some malicious PostScript files override the resolution or the page size specified on the command line. This can be avoided by prepending the following a special preamble to the PostScript file. This

preamble can be found in the `sam2p` sources with filename `contrib/aprea.ps`.

Ghostscript can read PDF files besides PS files. However, some PDFs tend to be too complicated for Ghostscript, so they should be converted into a simpler, temporary PostScript first. This can be accomplished by the *File/Export...* menu item of Adobe Acrobat, the *File/Print* menu item of Adobe Acrobat Reader, or the `pdftops` utility found in the XPDF distribution. Unfortunately `pdftops` ignores the bitmap fonts embedded by pdfTeX.

The `psrender` utility written by the author of `sam2p` calls Ghostscript to convert a PostScript file to another PostScript containing all pages as big raster graphics. This is useful when printing onto an old PostScript (Level 2) printer with small memory or buggy implementation. However, the output of `psrender` tends to be big (multi-megabyte for a single page), thus the document gets printed very slowly. But sometimes even an extremely long printing time is better than having no printout at all.

`psrender` is available from the Search facility of <http://www.freshmeat.net/>, the other utilities mentioned are part of major Linux distributions.

Conclusion `sam2p` makes the creation of small, compatible PostScript and PDF figures containing sampled images possible. Documents with these figures are small, and can be printed or transferred quite fast. There is work still to be done, but `sam2p` is already more general and more sophisticated than any other converter the author is aware of.

During further development, the software has to be tested, and bugs have to be fixed. Planned features (such as PDF output, and better transparency handling) have to be implemented.

References

- [1] Ed Taft, Steve Chernicoff and Carline Rose: PostScript Language Reference. Addison-Wesley. 1999.
- [2] Jim Meehan, Ed Taft, Steve Chernicoff and Carline Rose: PDF Reference. Second edition. Addison-Wesley. 2000.