EPS: a Programming Language

MJ Rutter
mjr19@cam

Lent 2005

The Boring Stuff

PostScript is a registered trademark of Adobe Systems Inc.

The best place to read more is ‘PostScript Language Reference, 3rd Ed’, pub. Addison-Wesley. This 900+ page monster can be downloaded from Adobe’s WWW site as a 7.5MB PDF. It is two clicks from
`http://www.adobe.com/supportservice/devrelations/`
and that URL is in the printed edition, so should be stable!

Why EPS?

Encapsulated PostScript is a subset of PostScript which is designed to be readily included in other PostScript documents.

PostScript is the language used by all modest printers (all printers which have been in TCM in the past decade), and there are free PostScript interpreters for most flavours of UNIX and Windows (i.e. ghostscript).

Most sensible document preparation systems (e.g. LaTeX) love PostScript for figures, for it is trivial to include into the final printed output. Most sensible graphics packages love PostScript for it is easy to machine-generate.

Why learn EPS?

Knowing EPS well enough to be able to make trivial changes to the output of programs is almost essential.

Knowing it well enough to be able to write figures directly in it is a luxury for PhD students: there may be better things to do with your three years. However, if you intend to have a long career in academia, it can be useful. One can produce small, portable, precise diagrams in a manner which even the best graphics packages struggle to match.

The History of PostScript

Adobe defined the original (level 1) PostScript in 1985.

A major enhancement occured with level 2 in 1990: colour was formally supported, along with vastly improved support for bitmapped graphics.

The current version is level 3, and was defined in 1999. It adds little to level 2.

Adobe permits anyone and anything to read and write PostScript. However, if one wishes to claim that a printer uses PostScript, then one must have the printer pass Adobe's test suite, and pay royalties to Adobe. Many vendors sell 'PostScript compatible' printers, which avoid this charge.

The above is not a precise statement of the legal position.

A Programming Language. . .

Whereas PDF and PCL are little more than structured data files which describe graphics, PostScript is a genuine programming language, with loops, conditional expressions, procedures, and most other things.

It suffers from using a global namespace by default, being interpreted, having no double precision floating point type, and being strongly stack based.

‘Flattened’ PostScript is PostScript without the structure: it is simply a linear list of things to draw.

. . . with Comments

A comment begins with a ‘%’ character and continues to the end of the line.

The first line of a PostScript file must start with ‘%!’.

The first line of an EPS file must start:
`%!PS-Adobe-2.0 EPSF-2.0` (the version numbers may vary)

Comments starting with ‘%%’ at the beginning of a line describe the structure of the file, and are considered later.

A Calculator

```
tcma:~$ gs -dNODISPLAY
AFPL Ghostscript 8.14 (2004-02-20)
Copyright (C) 2004 artofcode LLC, Benicia, CA.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
GS>3 4 add =
7
GS>quit
tcma:~$
```

In Detail

PostScript is a stack-based language. The above code places 2 on the stack, then 3, then `add` adds the top two elements on the stack, replacing them with their sum. Finally `=` prints and discards the top element.

PostScript has the usual maths functions, but it does not use the conventional arithmetic symbols as binary operators. Rather, the function name is spelt out.

It also deals with bases from 2 to 36, using the syntax ‘base#number’ for bases other than 10:

```
GS>16#1a =
26
GS>50 7 (xxx) cvrs =
101
```

Don’t worry about the details of the `cvrs` operator yet.

PostScript Maths Functions

```
add, sub, mul, div
neg, abs
sqrt
log, ln, exp
sin, cos, atan
floor, ceiling, round, truncate
idiv, mod
```

Notes: angles in degrees, not radians
`atan` takes two arguments
`idiv` and `mod` require integer arguments

The Stack

The usual stack manipulation operators exist. The obvious ones are:

`dup`	duplicate top element
`exch`	exchange top two elements
`pop`	discard top element
n `copy`	duplicate top *n* elements

Hence `6 dup mul` is 36,
and `2 3 add 1 exch div` is 0.2.

Operator	Stack
2	2
3	2 3
add	5
1	5 1
exch	1 5
div	0.2

Data Types

PostScript has two numeric datatypes: integer (32 bit) and real (single precision only). Conversion between the two is usually automatic:

```
3 2 div =
```

will yield 1.5.

It has a boolean type. (`true` and `false`)

It supports strings, and uses `(` and `)` to delimit them.

It supports names.

It supports procedures.

Other things too, including dictionaries and (1-D) arrays.

Object Classes and Function Definitions

Objects fall into two classes: ‘literal’ and ‘executable’.

Literal objects are simply placed onto the stack, whereas excutable objects are immediately interpreted. Whitespace separates objects – space and newline are equivalent.

Placing code in parentheses delays execution and creates a single object.

```
/double {dup add} def
6 double
=
```

prints 12. The forward slash causes `double` to be placed on the stack as a name, the parentheses cause the enclosed objects to be placed on the stack as a group, without being interpreted, and the the `def` associates the two objects on the stack: the name with the *executable array*, or procedure, defined.

An Interpreted Language

```
/double {dup add} def
/add {mul} def
6 double
=
```

prints 36. One can redefine the standard operators, and the definitions in effect at the point at which a procedure is called, not the point at which it was defined, are used.

This behaviour can be over-ridden with the `bind` operator.

```
/double {dup add} bind def
```

forces the definitions used at that point to be frozen into the procedure, and gives marginally faster execution.

Loops

Simple for loops exist, where the loop counter is placed on the stack before each new iteration.
The form is
start increment end proc `for`
e.g.
```
1 1 5 {=} for
```
prints the numbers one to five, and `0 1 1 5 {add} for =`
sums them (N.B. extra 0 to start the sum).

A Programmable Calculator

```
/factorial {
  /n exch def
  1
  1 1 n {mul} for
} def
5 factorial =
```

This is about as close as one gets to a variable, and is clumbersome. What in most languages would be written:

```
n=5
n=n+2
```

becomes

```
/n 5 def
n 2 add /n exch def
```

(or `/n n 2 add def`).

And With Recursion

```
/factorial {
  dup 0 eq
    {pop 1}
    {dup 1 sub factorial mul}
  ifelse
} def
5 factorial =
```

If top of stack is 0, replace it by a one and return.

Else place on stack one less than is there already, call factorial, and multiply its return value by the orginal value on the stack.

More advanced than Fortran 77!

More Structure

The comparison operators are `eq`, `ne`, `lt`, `le`, `ge`, `gt`. The operands must be of the same type, and one can compare numbers or strings.

The logical operators `and`, `or`, `xor` and `not` exist for boolean and integer types.

The control operators are
boolean {*true proc*} `if` and
boolean {*true proc*} {*false proc*} `ifelse`

Graphics

So far we have looked at printing to “stdout” only, and not at producing any graphics.

PostScript can draw, and fill (with repeated patterns), lines, arcs, and text. It can also print bitmaps.

Drawing is done in a ‘user co-ordinate space’ which is mapped to the ‘device co-ordinate space’ by the ‘current transfer matrix’ (CTM). This ‘matrix’ is a 2×2 matrix and a translation vector, and can thus describe any linear transform.

There is a single ‘current font’, which will be used for showing text, a single current linestyle, used for drawing lines, a single ‘current point’, the position at which the next object will be placed, etc.

Simple Text

```
%!
/Times-Roman 12 selectfont
100 100 moveto
(Hello) show
```

Choose a font and a size, move to a given point, and then show a given string. The default units are points, and there are 72 points to an inch. The origin is at the bottom left.

Although this will display with Ghostview, it will not print. Printers require a final `showpage` operator, which commits the image drawn to paper.

There are 25.4 millimetrics to an inch, or approximately $2\sqrt{2}$ points to a millimetric.

Ghostview draws as it goes along, for it can erase again should such an operator surprise it.

The Current Point

In order to draw anything, the current point must be defined, and the `moveto` operator is one way of doing so. The current point will be modified to be the end of what has been drawn, so that

```
%!
100 100 moveto
/Symbol 12 selectfont (p ) show
/Times-Roman 12 selectfont (is less than 4) show
```

is sensible.

Use '`\)`' for a closing bracket in a string, and *octal* to enter a character code directly, e.g.
`(p \273 3.14)`
in the Symbol font gives $\pi \approx 3.14$.

From now on, the initial `%!` will be omitted from the examples.

Where Are We?

```
100 100 moveto
/Symbol 12 selectfont (p ) show
/Times-Roman 12 selectfont (is less than 4) show
currentpoint pop 100 sub 2 add
99 99 moveto
0 rlineto 0 12 rlineto 99 111 lineto
closepath stroke
```

The `currentpoint` operator returns the x and y co-ordiantes of the current point. We discard the y, and subtract 100 from the x. This gives the width of the text shown. This value, plus two, is left on the stack. We move back to the beginning, and create a *path* of the length calculated in x, zero in y, and then complete the rectangle, mixing absolute and relative (to currentpoint) coordinates, and using `closepath` to join the end of the current path back to its beginning. The `stroke` operator causes the path to be drawn, rather than doing something else with it. It also causes currentpoint to become unset.

π is less than 4

Centred Text

```
/show-ctr {
  dup stringwidth pop
  -2 div 0
  rmoveto show
} def
/Times-Roman 12 selectfont
200 200 moveto (These lines) show-ctr
200 170 moveto
(are both precisely centred) show-ctr
```

The `stringwidth` operator consumes a string and returns the x and y displacement of the current point if that string were shown with the current font. Dividing the x displacement by -2 gives the offset of the start of the text from its centre.

These lines

are both precisely centred

Linear Transforms

```
/Times-Roman 12 selectfont
100 100 moveto
(Normal) show
100 120 moveto
2 1 scale
(Stretched) show
50 140 moveto
40 rotate
(And rotated) show
```

Note the effects are cumulative, and one can quickly lose track of reality.
Note that `50 140 moveto` would have been `100 140 moveto` in the coordinate system before the scale operator, and that a rotation in a non-isotropic coordinate system might not be quite what was expected.

And rotated

Stretched

Normal

Reality

The operator `gsave` saves the current graphics state, and `grestore` restores it. These commands can be nested to a depth of 30. Things saved include:

current point (if any)
any partially constructed path
current linestyle, font, fill style, etc.

Nothing is placed on the stack by `gsave`, nor removed from it by `grestore`

As above, but

```
gsave
2 1 scale
(Stretched) show
grestore
100 140 moveto
```

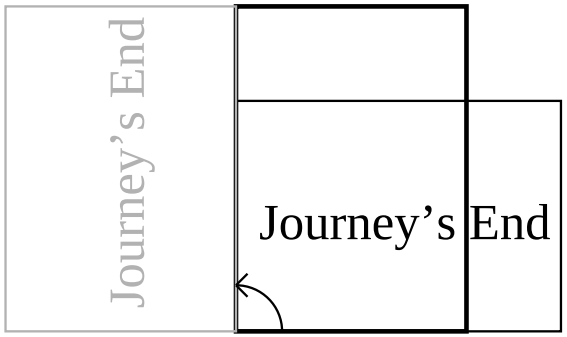
And rotated

Stretched

Normal

Landscape to Portrait

Converting landscape documents to portrait often confuses. The transform is not simply a 90 degree rotation, but a translation as well. The rotation takes the document wholly off the page.



The required transform is `90 rotate 0 -595 translate` for A4.

More CTM

The general form of the current transfer matrix is a 6-component vector
[a_{11} a_{12} a_{21} a_{22} x y]

Another similar matrix can be combined with it using the `concat` command.

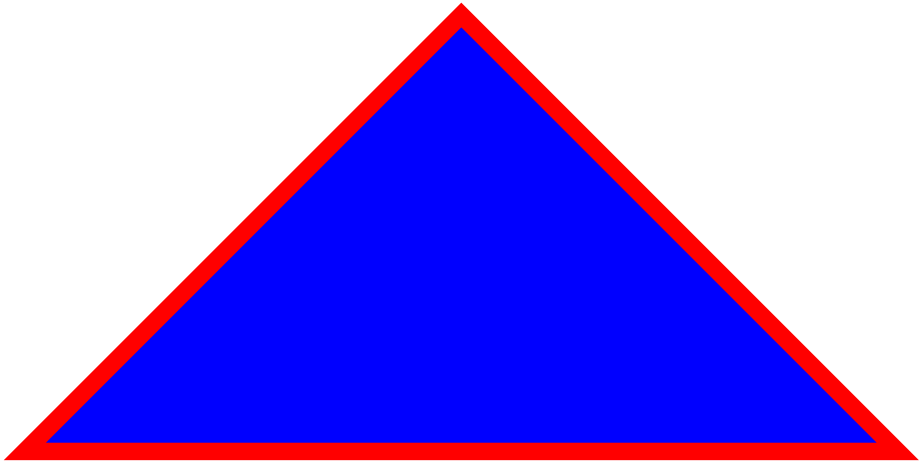
The `scale` and `rotate` commands are simply a shorthand for particular transforms of this matrix. So
`x y scale` is equivalent to
`[x 0 0 y 0 0] concat`, and
`x rotate` is equivalent to
`[x cos x sin x sin neg x cos 0 0] concat`, and for translations
`x y translate` is equivalent to
`[1 0 0 1 x y] concat`.

Note that between the `[` and `]` one must merely place six numbers on the stack. Any sequence which achieves this can be used.

Filled Objects

```
0 0 1 setrgbcolor
100 100 moveto
200 100 lineto
150 150 lineto
closepath
gsave fill grestore
1 0 0 setrgbcolor
2 setlinewidth
stroke
```

A blue triangle with a red border. Note that `fill` destroys the current path, so the `gsave` / `grestore` are needed to recover it for the `stroke` operation. The `closepath`, which draws a line from the current point to the start of the path, is very useful for filling objects: it ensures that the end of the path joins back to the beginning.

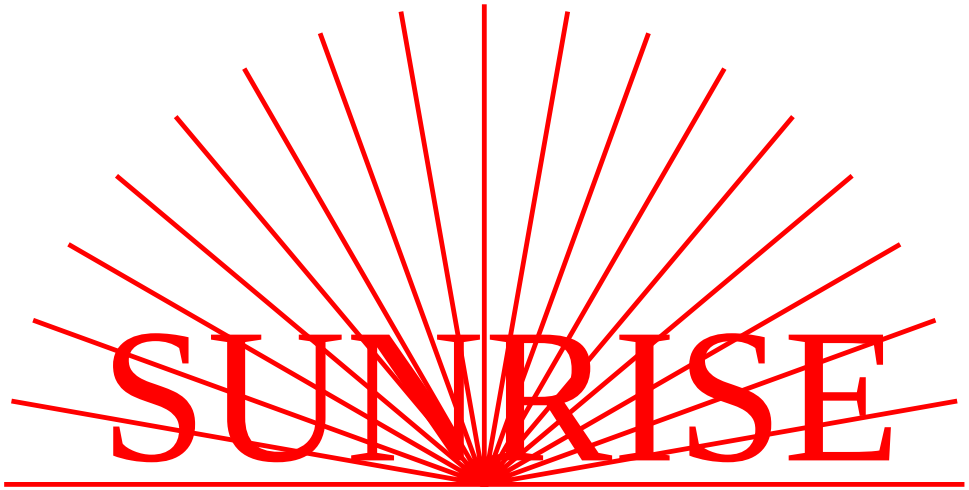


Text and Graphics

```
100 0 translate
1 0 0 setrgbcolor
gsave
19 {0 0 moveto 100 0 lineto 10 rotate} repeat
stroke
grestore
/Times-Roman 40 selectfont
-80 5 moveto (SUNRISE) show
```

count {*proc*} `repeat` – another loop construct.

PostScript has little concept of transparancy: things just overwrite each other in the order of execution.


SUNRISE

Text as a Mask

```
100 0 translate
1 0 0 setrgbcolor
gsave
/Times-Roman 40 selectfont
-80 5 moveto (SUNRISE) true charpath clip newpath
37 {0 0 moveto 100 0 lineto 5 rotate} repeat
stroke
grestore
```

PostScript does support complex masks, or clipping paths. Once a clipping path is defined, *nothing*, except a `grestore`, can cause a mark to be placed outside the clipping path. Here, the text is converted to such a path using `charpath`.

Text as an Outline

```
/Times-Roman 40 selectfont
1 0 0 setrgbcolor
0 0 moveto (SUNRISE) show
0 1 0 setrgbcolor
0 0 moveto (SUNRISE) false charpath stroke
```

SUNRISE

Fonts

Fonts come in several flavours. The original, Type 1, are scalable outlines formed of lines and curves. They are thus similar to TrueType and Metafont, but no direct, lossless translation exists between these three different scalable formats.

There is also a Type 3, which are generally used to encode bitmap fonts. Bitmap fonts should not be used, but are often used by LaTeX systems. The result works well at precisely one resolution of output device.

PostScript Level 3 introduced Type 42 fonts, which are TrueType by another name.

One can embed any of these font types into a PostScript document.

Strictly Type 3 fonts can contain arbitrary PostScript commands for each character, whereas Type 1 fonts use a very limited number of operations. A Type 1 font renderer, such as that built in to most X servers, is thus much simpler than a full PostScript interpreter.

Type 2 and Type 14 also exist, but are of little interest.

Available Fonts

PostScript printers traditionally shipped with 35 fonts predefined. Other fonts needed to be embedded. Then Adobe upset people by saying ‘While there is not a standard set of fonts most PostScript products include software for 13 standard fonts. . . ’

Thus strictly one cannot rely on *any* fonts pre-existing, although most people do.

There is no automatic mechanism for generating bold or italic fonts. Thus Times Roman and Times Bold are two completely distinct fonts.

Fonts on Parade

The thirteen standard fonts, then the rest of the 35 at 75% of the size of the first, follow. Notice that there is little standard about the naming of ‘Italic’ vs ‘Oblique.’

Times-Roman	**Times-Bold**	*Times-Italic*	***Times-BoldItalic***
Helvetica	**Helvetica-Bold**	*Helvetica-Oblique*	***Helvetica-BoldOblique***
Courier	**Courier-Bold**	*Courier-Oblique*	***Courier-BoldOblique***
Σψμβολ (Symbol)			
AvantGarde-Book	**AvantGarde-Demi**	*AvantGarde-BookOblique*	***AvantGarde-DemiOblique***
Bookman-Light	**Bookman-Demi**	*Bookman-LightItalic*	***Bookman-DemiItalic***
Helvetica-Narrow	**Helvetica-Narrow-Bold**	*Helvetica-Narrow-Oblique*	***Helvetica-Narrow-BoldOblique***
NewCenturySchlbk-Roman	**NewCenturySchlbk-Bold**	*NewCenturySchlbk-Italic*	***NewCenturySchlbk-BoldItalic***
Palatino-Roman	**Palatino-Bold**	*Palatino-Italic*	***Palatino-BoldItalic***
ZapfChancery-MediumItalic	✺❁❒❁✿❉■✳✲❁▼▲ (ZapfDingbats)		

More fonts

TrueType and Type 1 fonts are, legally, programs, and subject to the same restrictions on copying. Commericial printers dislike non-standard fonts, for even if you have a licence to use a font on your PC and your locally-attached printer, this does not necessarily give you a licence to embed the font in a document and print it on a large-scale commercial printer.

Embedding Type 1 fonts in PostScript is trivial. One takes a `.pfa` file, perhaps produced from the corresponding `.pfb` file by `pfb2pfa`, and sticks the result in the PostScript before one wishes to use the font. Ideally one might reduce the font to avoid defining characters not used, but this can cause complications.

Curves

There are two types of curves which can be appended to paths.
$x\ y\ r\ \theta_1\ \theta_2$ `arc`
draws a counter-clockwise section of a circle centre (x,y) radius r, starting at an angle of θ_1 (where $\theta = 0$ is the +ve x axis) and ending at θ_2.

$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ curveto
draws a Bézier cubic section from the current point to (x_3, y_3), using (x_1, y_1) and (x_2, y_2) as control points.

Note that PDF and Type 1 fonts permit Bézier curves, but not circular arcs.

There are also `arcn` for drawing clockwise arcs, and `rcurveto` as a form of `curveto` where all co-ordinates are relative to the current point.

Bitmaps

PostScript offers an ‘image’ operator, which paints a bitmap image with unit pixels at the origin. In its usual form, it accepts a *dictionary* as its single argument. A dictionary is an unordered list of key / value pairs, and is delimited by << and >>.

The image data are stored in a string, usually encoded as the raw data will be binary. The simplest encoding is hexadecimal, and such strings are delimited by < and >

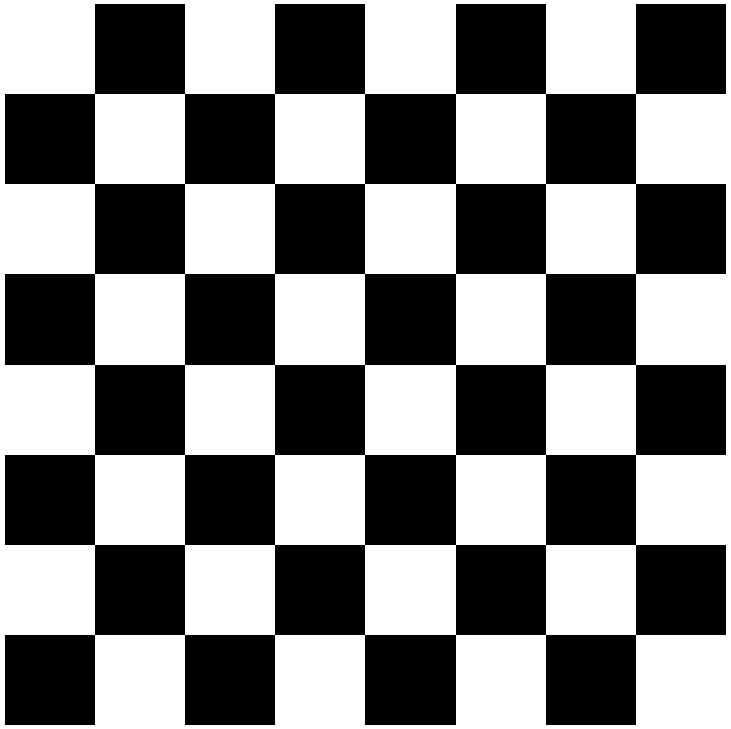
Mandatory entries in the image dictionary include `/ImageType` (must be one), `/Width`, `/Height` and `BitsPerComponent`, the size and colour depth of the image, `/ImageMatrix`, a transform to use in addition to the CTM, and `/DataSource`, which describes how to find the raw image data. The answer `currentfile` means ‘right here’, but this dictionary entry is meaningful when the `image` operator is executed, so ‘here’ means after the corresponding `image`. Note there is no start of string marker when using `currentfile`.

The image scans start at the bottom left (not top left as is the case for GIF/JPEG/PNG), and the `/ImageMatrix` can be used to perform an inversion if required.

Chess?

```
/DeviceGray setcolorspace
gsave
  100 120 translate
  10 10 scale
  << /ImageType 1
   /Width 8 /Height 8
   /ImageMatrix [1 0 0 1 0 0]
   /BitsPerComponent 1
   /Decode [0 1]
   /DataSource currentfile /ASCIIHexDecode filter
  >> image
  55aa55aa55aa55aa>
grestore
/Times-Roman 12 setfont
105 100 moveto
(A chess board) show
```

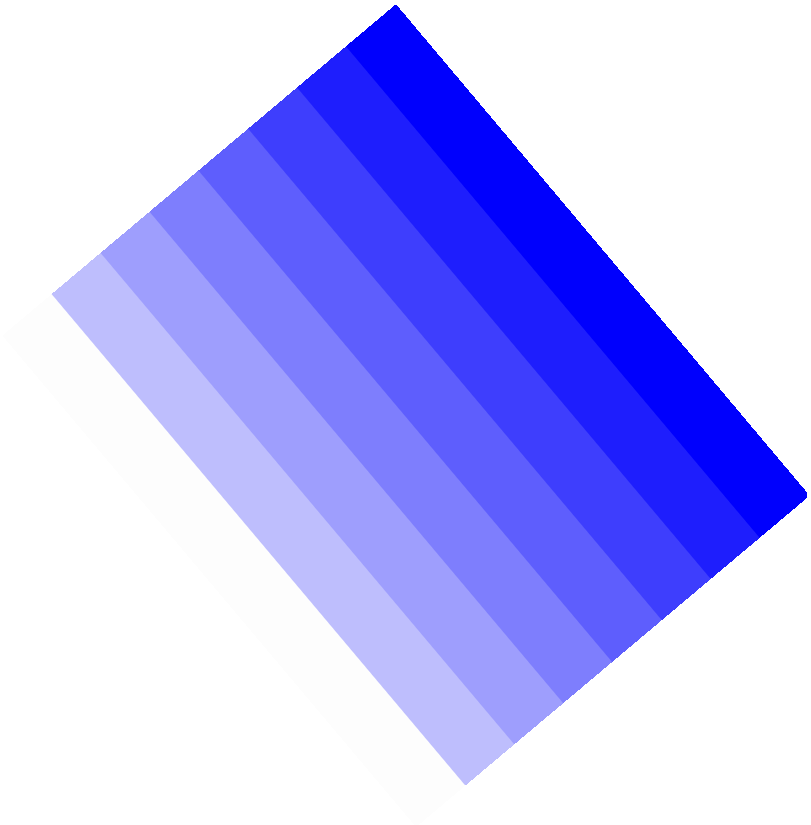
`aa` has the bit pattern `10101010`
`55` has the bit pattern `01010101`



A chess board

Colour Images

```
/DeviceRGB setcolorspace
100 100 translate
40 rotate
10 100 scale
<< /ImageType 1
   /Width 8 /Height 1
   /ImageMatrix [1 0 0 1 0 0]
   /BitsPerComponent 8
   /Decode [0 1 0 1 0 1]
   /DataSource currentfile /ASCIIHexDecode filter
>> image
ffffff c0c0ff a0a0ff 8080ff
6060ff 4040ff 2020ff 0000ff>
```



Compression

Encoding 24 bit colour bitmaps in hex is *BAD*.

Each pixel will take over six characters, so the result is huge.

PostScript, from Level 2 onwards, offers two improvements. An ASCII85 encoded string (delimited by <~ and ~>) uses 85 ASCII symbols to encode four bytes into five characters (cf. hex using 16 symbols to encode one byte into two characters). There are also decompression algorithms available: CCITTDecode (cf. Fax), LZWDecode (cf. GIF), DCTDecode (cf. JPEG) and, in Level 3 only, FlateDecode (cf. PNG).

Therefore a GIF, JPEG or PNG file should grow by no more than about 20% when converted to EPS. If this is not the case, use a better EPS generator.

`xv` is very poor, as it likes producing Level 1 PostScript.

`convert` is non-ideal.

`bmp2eps` is perfect.

More Cmprssn

Many programmers do not believe in vowels, or in words of more than about six characters (the FORTRAN77 syndrome). PostScript can help here:

```
/bd  {bind def} bind def
/m   {moveto} bd
/l   {lineto} bd
/rl  {rlineto} bd
/col {setrgbcolor} bd
/f   {gsave fill grestore} bd
```

Such tricks are common, and can lead to smaller PostScript files.

Gradient Fills

There are several ways of creating a gradient fill background:

1. Draw filled rectangle covering whole area, then a rectangle marginally smaller covering almost whole area and of next shade, etc.
2. Draw abutting filled rectangles.
3. Draw slightly overlapping filled rectangles.
4. Scale a 1D bitmap appropriately.

Of these, method one takes for ever: for a 256 step gradient, the total area filled is 128× the area seen. Method 2 can leave tiny gaps due to rounding errors, although a small overlap cures this. Method 4 is quick and simple.

Remember that if the document is converted to PDF, any loops used will be flattened (eliminated).

‘Fountain fills’ (circular contours) are hard to do efficiently.

The Stack in Detail

Objects are either *simple* or *composite*. Simple objects can be placed directly on the stack, whereas composite objects are stored elsewhere in memory, and a pointer to them is placed on the stack. Items on the stack are of fixed length (probably eight bytes), and the data type is stored along with the value or pointer.

Simple objects include booleans, integers, reals and operators. Composite objects include strings, dictionaries and procedures.

Traditional programming languages (C, Fortran, etc) do not store any information about data type with the data. PostScript does. Therefore a procedure can trivially query the type of its arguments, and behave differently according to the result. What C++ programmers would call ‘function overloading’ is thus simple.

It also means that run-time type-checking occurs.

Amusing Consequences

```
(Hello, World) show
```

Allocate memory for the string ‘Hello, World’.
Place on stack pointer to the string.
Consume the pointer with the `show` operator.

At this point, the string still exists in memory, but nothing points to it anymore – it is *garbage*.

PostScript does automatic *garbage collection*. It will notice, and automatically reclaim the memory. Neither C nor Fortran does this.

In `(Hello, World) dup show`, the `dup` duplicates the pointer, not the actual composite object, and no garbage collection occurs.

Operators returning strings rarely allocate the strings themselves, but need to be passed a sufficiently-long string which they overwrite with the return value. Hence

```
50 7 (xxx) cvrs
```

DSC

Certain comments are called *Document Structure Comments*. These describe the structure of a PostScript, or EPS, file, and are mostly optional. However, if present, they must be correct!

Printers ignore them all. Ghostview reads some of them. This confuses people.

They are intended to be informational, and to assist programs which wish to rearrange documents. The final imaging process should not use them. Programs such as `psnup` and `pstops` rely very heavily on them, and `gv` needs them in order to move backwards or to jump forwarsd in a document: otherwise it can merely display pages sequentially.

PostScript documents beginning
`%!PS-Adobe-`n.m
are thus claiming to be DSC compliant. Those that are not should start simply ‘%!’ or ‘%!PS’.

Structured PostScript Programming

A well-structured PostScript program starts with global setup commands and definitions, which cause nothing to be drawn. Then it has a number of wholly independent pages.

If such a structure is made clear by DSC comments, then removing, extracting, and reordering pages is trivial. Ghostview requires this to be able to jump directly to a given page. Without this, it can only step forwards through the document one page at a time.

Without the imposition of DSC, one can define a function on page 4, and use it on pages 4 and 10. Worse, one could define it, differently, on pages 4 and 10, and use it on pages 4, 6, 10 and 20.

DSC can be used in other tricks. A document should declare which fonts it needs, which it has embedded in itself, and any embedded fonts should be surrounded by comments. Fonts which are needed but not embedded can thus be supplied, and those which are embedded by not needed (because the interpreter already supplies them) can be removed. In my experience, this way madness lies.

Useful Comments

`%%Pages:` 34
total number of pages

`%%BoundingBox:` llx lly urx ury
rectangle which is larger than area used by file, required by EPS.

`%%Page:` iii 4
the start of page label iii, the forth page in the document

`%%Orientation:` landscape
`gv` uses this one

`%%Creator:` dvips
`%%CreationDate:` Ides Martis
`%%Copyright:` SPQR
free format comments, which can reveal all sorts of information.

And note again that no printer uses the `%%BoundingBox:` or `%%Orientation:` comments for anything.

Not EPS

A PostScript program can change the state of the imaging device:

```
<< /Duplex false
   /MediaType (Transparency)
   /PageSize [595 842] >> setpagedevice
```

You never want an included figure doing this sort of thing. Nor may an EPS file contain multiple pages.

Note that specifying the MediaType may change both the tray used *and* the printing process used (fuser temperature, etc.).

Note too that, in PS files, neither any `%%BoundingBox:` comment nor the area drawn influence paper selection. The `setpagedevice` command is one of the few ways of selecting a non-default paper size.

Genuine EPS

A real EPS file must not call `setpagedevice`, `initgraphics`, `initclip` or several other operators.

It must make sure its use of the stack and of `gsave` is balanced.

It must not redefine any previously defined names, nor leave any new ones around. (It may use a private namespace.)

If given a redefined operator, it must not attempt to access the original, nor may it attempt to break any clipping path imposed.

It must include a correct `%%BoundingBox:` comment.

It should not contain a `showpage` operator.

It must start
`%!PS-Adobe-2.0 EPSF-2.0` (the version numbers may vary)
and not simply `%!` or `%!PS-Adobe-2.0`.

Including EPS

Most EPS files are naughty, so programs including them are recommended to:

- redefine `showpage`
- set a clipping path to the bounding box
- offer a private namespace
- expect and remove rubbish from the stack

Not all programs do all of these things, and none should be necessary.

```
/showpage {} def
```
is a useful trick for including dodgy EPS files.

Conversions

Converting EPS to PS is trivial. Converting PS to EPS is very hard. So every sane person uses EPS in preference.

If one believes that EPS files may contain a `showpage`, then a single-page document can be simultaneously EPS and PS. This is what some programs do (e.g. gnuplot).

If an EPS file does not contain a `showpage`, then it will not print. Even if it does, its natural scaling and positioning may be inappropriate for A4 paper…

Pretending that one can convert PS to EPS by changing the initial line and adding a BoundingBox comment, or, worse, renaming the file, is madness.

Trimming EPS files by editing the BoundingBox comment is valid only if one is removing whitespace.

The Wacky World of Windows

Computers running Windows or MacOS used to be thought of as incapable of interpreting PostScript. The existance of Ghostscript for these platforms did not convince unbelievers, so, for a long time programs like MS Word could include EPS figures, but not display them, and print them only to PostScript printers.

To make life slightly more pleasant, there are ways of attaching bitmap previews to EPS files. Programs can then readily read the preview and display it, and send it to non-PostScript printers, but use the real thing for PostScript printers.

Previews in Triplicate

There are three different ways of doing this:

- Preview as uncompressed, hex-encoded bitmap in PostScript comments. Huge, but the file remains conformant EPS.
- Preview in resource fork, EPS in data fork. What? Well, meaningful on MacOS's filing system only.
- Short binary header, binary preview, then EPS. The DOS/Windows 'solution', but no longer a valid EPS file.

In the UNIX world, where `xfig`, `xdvi`, etc. can all read EPS without previews, one does not want to touch the things.

Specific Programs

Now a brief discussion of some programs often-used with EPS.

dvips

LaTeX's `dvips` does not constrain EPS files which it included by `\includegraphics` to their BoundingBox. This is because `\includegraphics` is clearly specified as *not* cropping files by default.

If one wants cropping from LaTeX, then `\includegraphics*` is the appropriate command.

Most installations of `dvips` default to sending their output directly to the printer (via `lpr`) and using embedded bitmapped fonts. Using '`-Ppdf`' sends the output to a file with the extension `.ps` and includes Type 1 fonts. This is usually much more useful.

The naming of '`-Ppdf`' comes from the days before `dvipdfm` when `dvips` was the first stage in converting from DVI to PDF.

Wordprocessors

These things rarely produce EPS, even if printing to a file via a PostScript driver trivially produces PostScript.

Modern wordprocessors are very fussy about letter spacing (kerning). Whereas older wordprocessors adjust the inter-word spacing in order to achieve a straight right margin, modern ones usually adjust the inter-letter spacing too. This leads to every character being individually positioned with its own `moveto` and `show` commands. The resulting file is huge and ugly.

```
/x {moveto show} bind def
...
100 100 (H) x
110 100 (e) x
108 100 (l) x
114 100 (l) x
120 100 (o) x
```

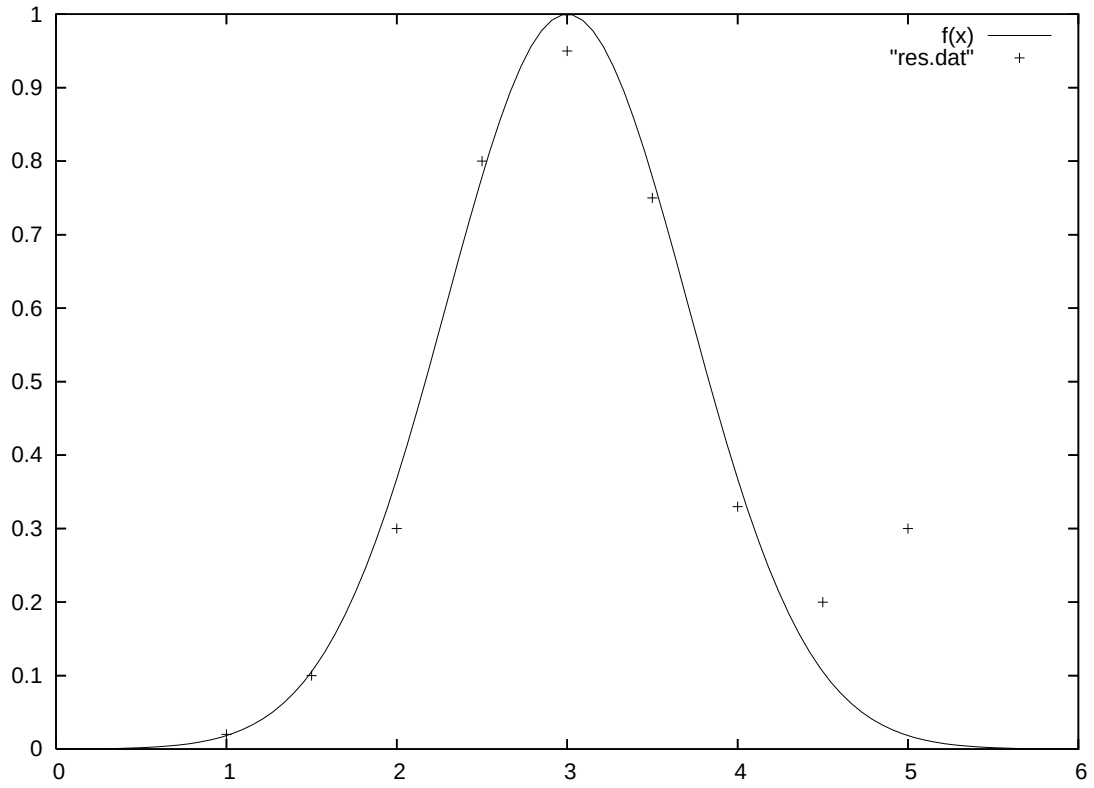
Gnuplot

This program positively encourages one to edit its EPS output:

> A PostScript file is editable, so once gnuplot has created one, you are free to modify it to your heart's desire.

Its defaults are thin, weedy and unexciting, although much can be done with options from within the program.

```
f(x)=exp(-(x-3)**2)
set term post eps
set output "plt1.eps"
plot [0:6] f(x), "res.dat"
```


f(x)
"res.dat"
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
1
2
3
4
5
6

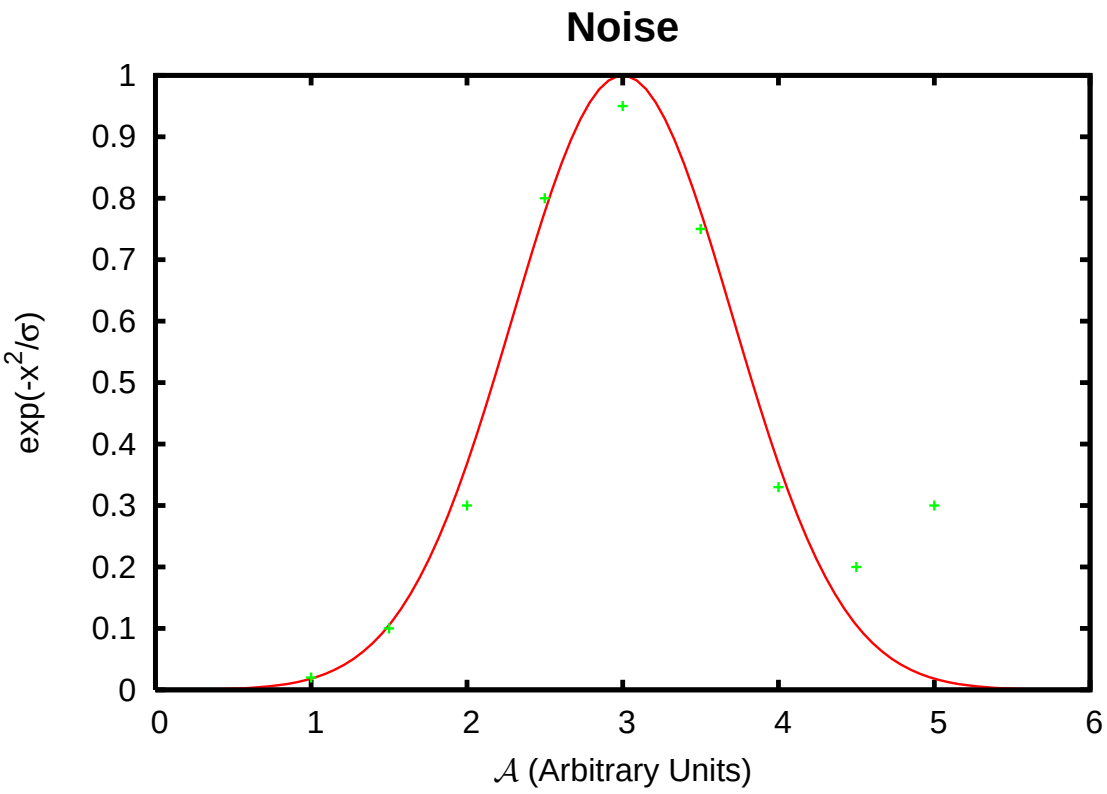
Gnuplot (2)

Better use of Gnuplot's commands produces a significant improvement.

```
f(x)=exp(-(x-3)**2)
set title "{/Helvetica-Bold=26 Noise}"
set xlabel "{/CMSY10 A} (Arbitrary Units)"
set ylabel "exp(-x^2/{/Symbol s})"
unset key
set term post enh eps color lw 3 fontfile "cmsy10.pfa" 20
plot [0:6] f(x), "res.dat"
```

Note the use of superscript (`^`), selection of 'standard' PostScript fonts (`{/Symbol s}`), including with a size for the title, and also the inclusion of a `.pfa` font file (LaTeX's `\mathcal` font) and the selection of it. The `.pfa` file was generated from the `.pfb` file in `texmf/fonts/type1/bluesky/cm/` by `pfb2pfa`. The linewidth is also set to 3, not one, and the default font size to 20 (not 14).

The font-changing commands and superscript require the 'enh' option to the PostScript terminal type.

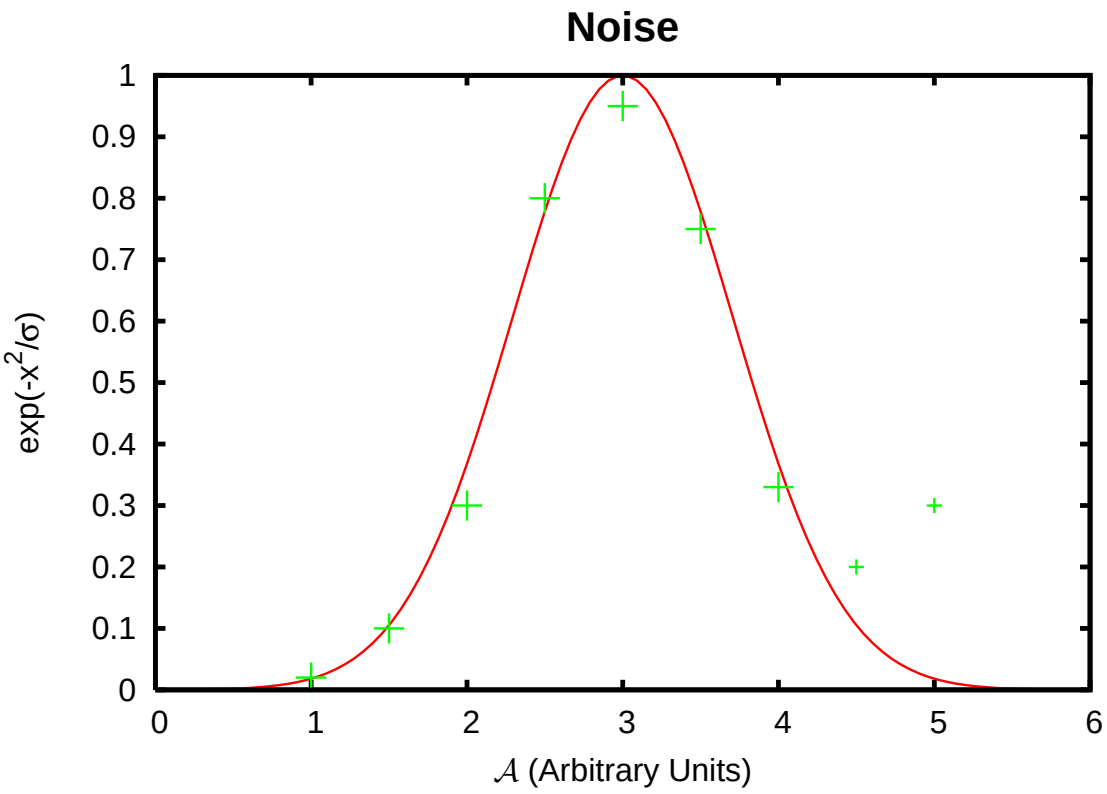

Noise
exp(-x^2/σ)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
6
A (Arbitrary Units)

Gnuplot (3)

Still not ideal: one would like to make the green crosses larger, and to divert attention from inconvenient outlying points. So one can edit the EPS and:

treble the values of `hpt_` and `vpt_`
halve the values of `hpt`, `hpt2`, `vpt`, `vpt2` just before drawing the final two points.

Many more tricks are described in the file `docs/psdoc/ps_file.doc` distributed with Gnuplot.


Noise
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
exp(-x2/σ)
0
1
2
3
4
5
6
A (Arbitrary Units)

XMGrace

Less need to edit the PostScript, as more options can be set from the GUI. (Plot | Set Appearance)

Has habit of starting by drawing a white rectangle in the wrong place.

In xmgrace one can simply select ”Plot”, then ”Plot appearance”, and uncheck the ”Page Background” ”Fill” box. (The dots which then appear on screen are not printed.) If one cannot rerun xmgrace, the solution involves editing the EPS file. Load it into your favourite editor, and seach for ”%%EndSetup”. Then delete (or comment out by prefixing with ”%”) the lines reading:

```
n
0.0000 0.0000 m
0.0000 1.0000 l
1.2937 1.0000 l
1.2937 0.0000 l
c
[/DeviceRGB] SCS
Color0 SC
fill
```

Mathematica

Need to embed Mathematica’s maths fonts if anything else is to read the file.

Complex surface plots can produce huge files, full of thousands of neatly rendered triangles. In an ideal world one removes the text, converts the figure to a bitmap, then adds the text back in using scalable fonts.

xfig

The lazy way of combining EPS files, or adding labels etc. to them. Tends to prefer input files to be EPS, so convert JPEGs etc. to EPS before loading them. The result looks dreadful on screen, but the final EPS is great.

Cannot readily add non-standard fonts this way.

(When `xfig` reads an EPS file, it converts it to a 256 colour bitmap for screen preview purposes. If the figure gets resized, it rescales the bitmap rather than reconverts. Thus the on-screen representation is dreadful, but the final EPS output is fine as it inserts the original EPS file.)

Converters: User Beware!

Suse/Alpha convert EPS:	2148K	
xv	2144K	
Suse convert EPS2:	1067K	binary
gimp2	334K	
bmp2eps -Z	265K	
Alpha convert EPS2:	140K	level 3!
bmp2eps	137K	
bmp2eps -8	108K	binary
Original PNG	78K	

What you get is what you pay for, and you paid nowt for the above.

This whole document is under 500KB, and one image on the penultimate page accounts for over a third of that.

Gallery

Finally, the obligatory gallery, showing off what can be done in very few lines.

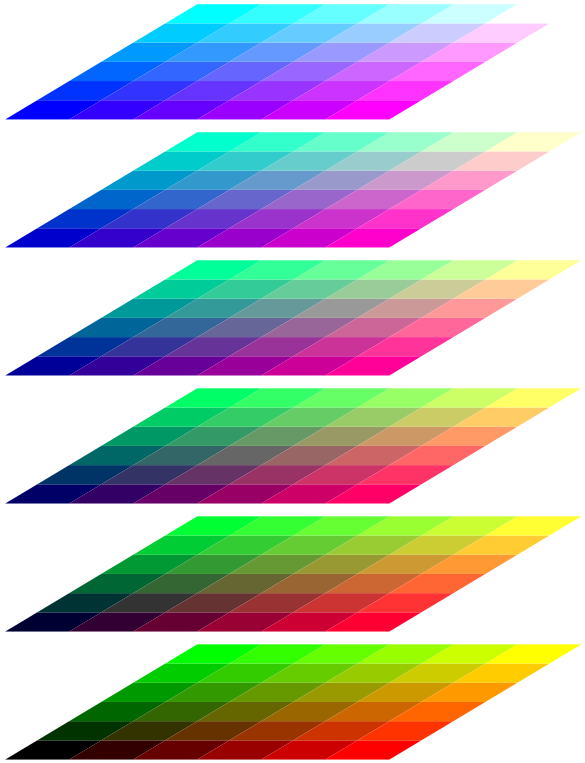
Note that most of these files have been severely trimmed, so do change the global namespace and commit other crimes against the EPS standard. However, writing in 20 lines of 78 characters is hard!

The magic of using a private namespace, or dictionary, to avoid upsetting the global namespace, has not been described in this document. It is achieved by placing '*n* `dict begin`' before the first definition, and '`end`' at the end of the document, where *n* is the size of the required dictionary. For level 1 EPS, *n* must be greater than, or equal to, the number of definitions used. Level 2 dictionaries automatically expand as required, so it can be less.

The Netscape Palette

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 90 135

/square {10 0 rlineto 5 3 rlineto -10 0 rlineto fill} bind def

0 1 5 {
  dup 20 mul /y0 exch def 5 div /b exch def
  0 1 5 {
    dup 3 mul y0 add /y exch def dup 5 mul
    /x exch def 5 div /g exch def
    0 1 5 {
      dup 10 mul x add y moveto 5 div g b setrgbcolor square
    } for
   } for
} for

0 setgray /Times-Roman 8 selectfont
(The Netscape Colour Cube)
dup stringwidth pop 2 div neg 45 add 125 moveto show
```

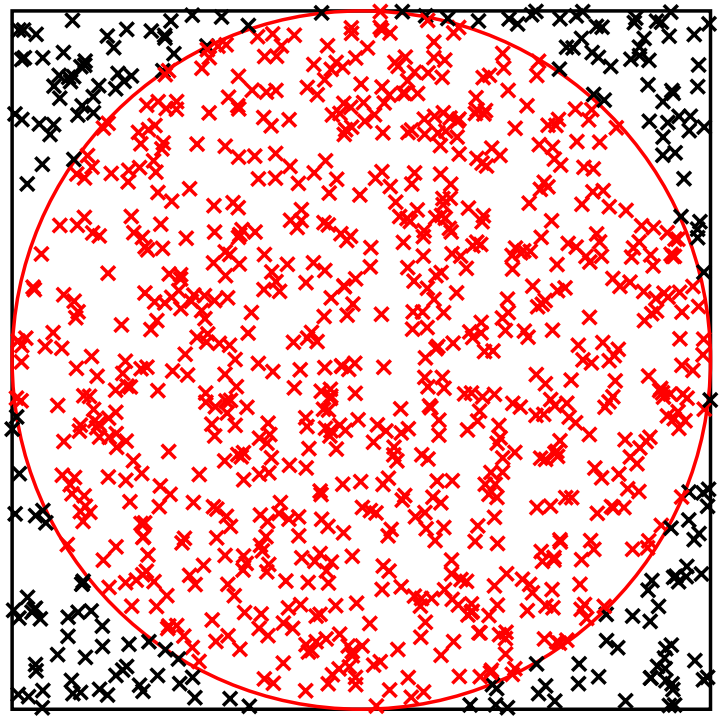
The Netscape Colour Cube



π by Monte Carlo

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 220 260

/trials 1000 def /hits 0 def
/cross { moveto -2 -2 rmoveto 4 4 rlineto -4 0 rmoveto
         4 -4 rlineto stroke } def
rrand realtime revision xor xor 16#7fffffff and srand

110 110 translate -100 -100 200 200 rectstroke
1 0 0 setrgbcolor 0 0 100 0 360 arc stroke
1 1 trials { pop rand 20001 mod .01 mul 100 sub
             rand 20001 mod .01 mul 100 sub
             dup dup mul 2 index dup mul add 10000 ge
             { 0 0 0 setrgbcolor }
             { 1 0 0 setrgbcolor /hits hits 1 add def } ifelse
             cross
           } for

0 setgray /Symbol 25 selectfont -50 120 moveto (p \273 ) show
hits trials div 4 mul 8 string cvs show
```

$$\pi \approx 3.176$$



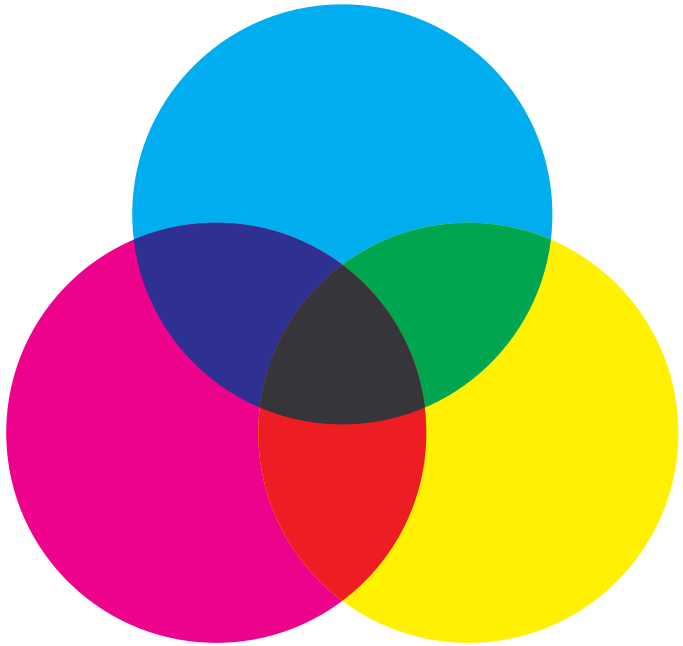
The X11 Logo

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 100 100

gsave 1 dict begin

/half {
  0 0 moveto 38 49 lineto
  0 100 lineto 25 100 lineto
  55 58 lineto 9 0 lineto
  fill
} def

half
100 100 translate -1 -1 scale
half

end grestore
```

This has perfect symmetry and the EPS uses a private namespace and gsave correctly.

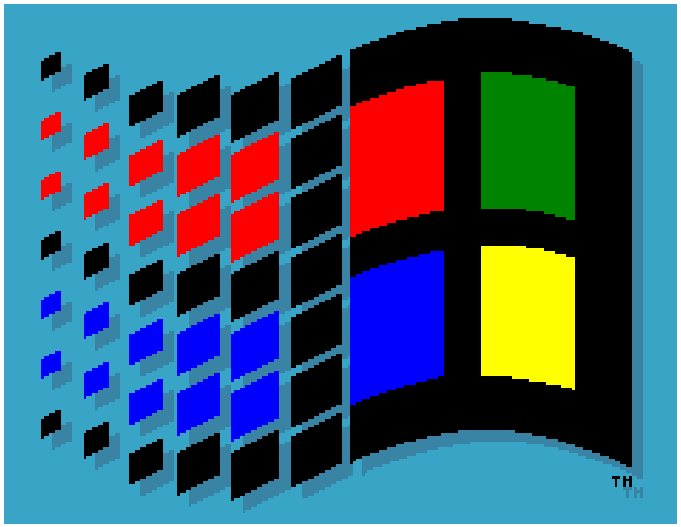
CMY colour mixing

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 20 50 180 202
%%EndComments
4 dict begin gsave
/Ccirc {100 152 50 0 360 arc} def
/Mcirc {70 100 50 0 360 arc} def
/Ycirc {130 100 50 0 360 arc} def
/sc {0 setcmykcolor} def

1 0 0 sc Ccirc fill
0 1 0 sc Mcirc fill
0 0 1 sc Ycirc fill

gsave 1 1 0 sc Ccirc clip newpath Mcirc fill grestore
gsave 1 0 1 sc Ccirc clip newpath Ycirc fill grestore
gsave 0 1 1 sc Mcirc clip newpath Ycirc fill grestore
gsave 1 1 1 sc Ccirc clip newpath Mcirc clip newpath Ycirc fill grestore
grestore end
```



Again a private namespace and gsave are used correctly.

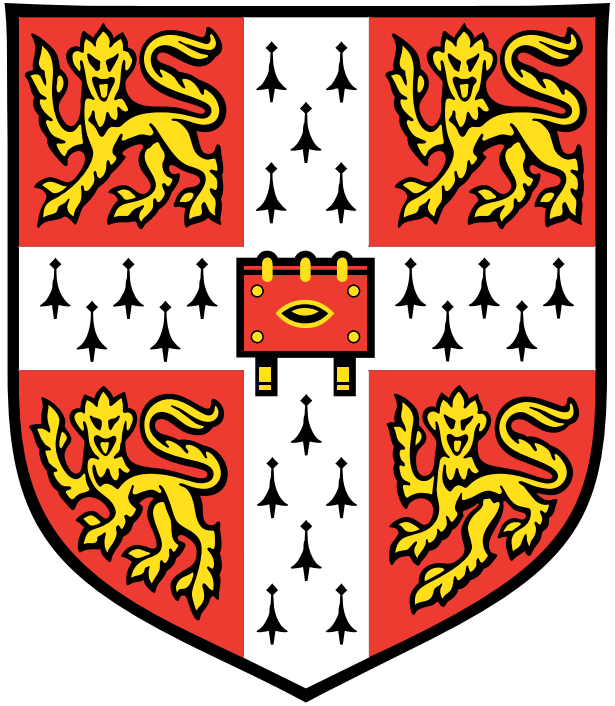
```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 237 183
[/Indexed /DeviceRGB 7 <~z!<3%P!',_/3Mj':^#&bh!<<'!~> ] setcolorspace
<</ImageType 1 /Width 237 /Height 183 /ImageMatrix [1 0 0 -1 0 183]
  /DataSource currentfile  /ASCII85Decode filter /FlateDecode filter
  /BitsPerComponent 4 /Decode [0 15] >> image
GhVQ?bDP%:(^=B5R$heZEWN"<i'Ci'#o:(3JHW*elg<:bI.BS3Fo#XX0O"C=k+=c$&(6AC
4@4FVn*s0a!i%ckgekeJL!^lkl>[)5'.0EXs&<d)@hL'oT.9hn8]j"^7<8baG0(eAj*sGe
'RL-Ykm.2W1>%e?)^O<#3@sc)#0/@eG0OLlr*("j@i)[H1'[K]Mcrhmp,'gfVV#*d&_!Z9
7^guWCAa$"43a4(a&h_DI0sj;L/fJ>MLh=H1qom\.CbLResj#?h$Yd%hp^^%XKc\AIX18g
?=Lm<"jF*Fp@<&:fcFF2-\m#!'iPl%Z\5[Dnm,]^]#KaWm'C)iR/RBRB'@c*:q4\mJX&!W
K'g&0l6DZ@!Htu#4n_ScZOPCM*+Or/Z;MIENC[X<,Q]L7kSV@E;IZn9'3U)9jW]lT00Y64
EY6En9-/YL?8c:tXATX4l*5&_mFCo0c:X./,F6SWV32KFAKMu:7P#K\mFhLt!eiKDmd\_P
SlX*&bQ_+=dFP<gXGLm,q1<116,mrH^5gpE1Zu73d+J=.Ncc44rRn1K%sDnH,I#m!4Kq2m
ZFN?%T2.Y.E94/-2<ng]3l8.!'aTRh')$&e)YUsU<Xe>9nkl;d"hQ(0Wadd+;WZ*@!\<Zq
l^LOaM4_'X/#KDTXL.'/<r5hBij*%9>0PbIl;VQ,rtLfR2=bI)>aV;^^A^qkA.9<?k,4%P
0LlGN<@b=-ZQRJs@?7J+^;M0G7(<>$/rj>MLh&#(\-TN4KX956>:%-hFe,WJiN*A;S$SA(
S5h.&3hFZt&Ph$cpAgI=FcQh&!0oQeNsPI[<B,,^#\PoYboP-d,2C()332HfiUiGGItW/s
8%=%!_Te^[er=V\:UKX:**Id$>VEs2'a3&2>])u-']Hb3!h11g$4<4!#kpd1=HXEXb3GmM
OW#1t$<s]ZOW#+r0KQq,#'$9WMU]]QLL6iH7C\U9s'nPgOW"tn!j"$[Mq"!tf:MbPO\4lk
@qTbb8='..!aB-G?DLK1bYI*[,.5lq=7)=+Xu;'5Y-WV,V\c>'I\pnh'-/7RV6kbY/5eP6
,pQ[/?H;emQE\P$C?LK(G,=5o["Q[T;Ld%;2R>rN'ST74;CN*F=JSh?2R,flA.si"6oLl4
'FG*h<!8M!'WL1FV&;^:0hXYfVFW:MmiEEJUnY)*B>7g6Mc:,9'0'X"'YYUfD#]9+I$'<%
C.IgGXWmGu16->8X9gcLB<QcH8mtBj;#dI5g>[>n"+0P^S@Zl9QM@/6'=6l_?=9c2H[Tt7
/b+JE@)\A,QDO;HenT[A>$P*%=c8];7%J(pp7L?MYXVsCNXKnbhq&s-40??9dX3e>!p3%)
[;%#]C)2I_2XGE'fW-ubKa;l&bJb&E(QiNf#T(>.LiCr!@&l[TX0/dO,+MYaATP:XKX7[i
8&t4!28W30iU3?oVsY/Um[u>5mLeFGTF.Ub&'HF'ZFT#$>D0W##\V]$,FmU'!tjr'IMhj?
'&g"0#^?!6[(b.QE9H=q]^JiFgj.q:H+aJ"GZ8#t_OZThYFGlR:q!2j\2-EYP9X=Hfp3H5
#TB0Fi<U?k']BU*OqHZq=[Fkf>VSl62C?:L(TEN/%_G8kN,Z1^e3J])Ij6g#_Vr7l\e<u1
MH'=,G4_"Soo7HmKHm]I[bG_rZJ!<K,8H)8]d:*kYD/Y$_Ita_8;/8\\lGQ/TcT.qUs@?/
-9=6<LdgUiS1\X8isB50Om8gT\>aUe;K/;BOh-ANg'jLqH4$?bW);GC>s"r7'1R9m<CBO2
I;b$DUc'E/Wc#[*;N]*jgf.]O/P_GN7H@F'1X(Kd/L4SsEfm)22op'2<6]HGk_@_dOd^;k
@8U\RIa(5'Od'HH]5-VSB's@D%(0';-K=aU-rP-9=E_AKbu45/^j1'A/5in-AZG^s)Z%9A
Om9ACj)pc)cZ1)AZ4's3n[_CONLamB4,o7p\&>u4fp9/F3(=Y'W8HXM)h.n.&f'M~>
```



Perhaps a surprising choice, but actually a well-designed logo. Few colours, and good compression – if only everything from Microsoft were this efficient!

The decent information density in the EPS is clear. EPS bitmaps should not contain long sections of obvious structure. This bitmap has a resolution of 237x183 pixels, and the EPS file is 2288 bytes: almost 19 pixels per byte.

A Cautionary Tale



The one on the left is under 7KB. The one on the right is over 165KB.
Guess which one causes more trouble when included in other documents.

Other tricks

Areas can be filled with patterns, and bitmaps used as masks. The shape of the end of lines can be changed (square, rounded, etc), and lines can be dashed rather than continuous. One can use an RGB, HSB, CMYK or paletted colour space (amongst others). And one can enquire what fonts and other resources are available.

Although simple tricks like prepending:

```
%!
2 sqrt dup scale
<< /PageSize [841 1190] >> setpagedevice
/setpagedevice {pop} def
```

might be sufficient to convert an A4 PostScript document to A3, one can easily construct documents for which it won't work.

On the other hand, EPS is sufficiently well-behaved that it can be reliably manipulated.