Graphics for Inclusion in Electronic Documents

Ian Hutchinson

May 4, 2003

Abstract

How does one produce portable graphics files that can be imported into other documents, especially TeX documents? Ways that images become unportable are discussed as well as good-practice guidelines. The advice is aimed mostly at linux or similar operating systems which have a wealth of open-source command-line tools.

Contents

1 The Problem **2**

2 Postscript **2**
2.1 Encapsulated Postscript 3
2.2 Bounding Boxes 3
2.3 Binary Garbage 5
2.4 Line Ends 5
2.5 Orientation 6

3 Portable Document Format (PDF) **6**
3.1 Converting from PDF to PS 6
3.2 Converting from PS to PDF 7
3.3 Fonts 7

4 Using pstoedit and xfig **8**

5 Bitmap Formats **9**
5.1 PNM to PS 9
5.2 JPEG 9
5.3 GIF 10
5.4 PNG 10
5.5 Combined Vector and Bitmap Graphics 10

6 Scanned Images of Line Drawings **11**

7 LaTeX Desperation Tweaks **11**

8 Summaries **12**
8.1 Problem diagnosis 12
8.2 Tools 12
8.3 Instructions to graphics authors 13

9 Conclusion **14**

1 The Problem

In producing modern documents, images are often essential. The ability of computer viewing to provide graphic information, and the bandwidth of most modern networks makes graphic presentation truly feasible. However, graphics have many different incompatible formats. Many formats are inappropriate for high quality graphics. Many are not portable between operating systems or even different computers. Therefore many embarrassments occur from presentations that have not paid sufficient attention to portability, and endless hours are spent trying to correct weaknesses and portability problems in graphics.

I was motivated to write this document by experience producing a high-quality publication of some 200 pages with input from a dozen people, all using different computers and software tools. LaTeX was the ultimate layout engine. It is an extremely powerful way of producing a properly structured document, and can handle graphics in the form of postscript (PS) or PDF (when producing PS or PDF output respectively). In principle the requirement that was placed on authors, to provide PS files with proper bounding boxes, ought to have been sufficient to enable the process to go smoothly. That was far from the case. So I wrote down some of the lessons learned from this and other related experience.

This document is not intended to explain how to do graphics importation in TeX or LaTeX. It is assumed the reader knows how to do that, or doesn't need to know. Many web resources exist that address these importation techniques as well as files in the local tex installation, available on linux as `/usr/share/texmf/doc/latex/graphics/epslatex.ps`. The emphasis here is on getting figures into a form that will import properly. Even seriously broken figures can sometimes be fixed with internal tweaks of the TeX importation, but that is very troublesome, and with care can be avoided.

2 Postscript

Postscript is, of course capable of representing the entire spectrum of graphic objects: vector graphics (lines, regions, fills etc), bitmaps, and fonts. But its bitmap capabilities are cumbersome in some cases. Vector graphics are by far the best way to represent high quality line drawings, the staple of much technical publishing. For professional purposes the loss of resolution and visual quality caused by converting to a bitmap should never be permitted till the final display.

2.1 Encapsulated Postscript

Much confusion surrounds the difference between encapsulated postscript (EPS) and just plain vanilla postscript (PS). The EPS format specification is at `http://partners.adobe.com/asn/developer/pdfs/tn/5002.EPSF_Spec.pdf`. A useful summary may be found at `http://netghost.narod.ru/gff/graphics/summary/epsf.htm`. There is no difference between "EPSF" and "EPS".

The requirements that make a PS file also an EPS are

1. EPS is a single page;
2. a few obscure PS commands are forbidden in EPS;
3. EPS must have certain comments, including a bounding box comment.

Any self-respecting PS output ought also to have the key comments, in which case it will generally be an EPS file if it is one page; but some PS files aren't self-respecting.

Some kinds of EPS files are not valid PS files. So we have the unpleasant situation that though they intersect, neither PS nor EPS is a subset of the other. Postscript is ASCII (plain text). Some EPS files have *binary* data added to them representing a preview bitmap. This possibility is unfortunately part of the EPS specification. Binary preview data is a cause of many incompatibility problems and prevents a file containing it from being portable to different platforms in many instances. There is therefore a more portable preview format, EPSI, standing for encapsulated postscript interchange, which is ASCII, and leaves a file in the state that it is (also) a valid PS file. For present purposes, EPSF binary previews are poison; EPSI previews are more benign but still unwanted for TeX.

2.2 Bounding Boxes

The main thing that a publishing program needs to know about a graphic it is importing is its size and shape. Normally the program has the ability to scale the graphic to fit a specified area provided it knows the original size. Postscript files conventionally provide this information through comments that conform to the "Encapsulated" postscript conventions. The most important such comment for present purposes is the BoundingBox comment. It has the form:

```
%%BoundingBox: 0 0 520 685
```

which conventionally defines the smallest rectangle that encloses all the drawn parts of the graphic with little or no margin.

Perhaps the most frequent problem for importing graphics is that postscript files are often produced with an incorrect bounding box. This is the fault of the software that produced it, or occasionally of the user, in not realizing that a large margin should be *avoided* when preparing figures for importation into another document. Frequently, PS files produced for printing use the size of the paper as the bounding box rather than the size of the drawn area.[1]

[1] According to Scott Pakin, you can sometimes tell this by the values of the box: 0 0 612 792 (U.S. letter) and 0 0 596 842 (A4).

Assuming that one does not have access to the software and source files that originally produced the figure there are several ways to fix bounding boxes.

Manual Editing Postscript files are plain text ASCII. Any decent editor can be used to adjust the bounding box by hand. Its four parameters are the (x, y) coordinates of the bottom left and top right corners. To tell where you want to put these points, view the file in Ghostview (`gv`) move the pointer to the place you want the corner to be, and read out the position from gv's display. Edit the file to change the values, then view it again with `gv` or some similar postscript viewer. Ghostview adjusts its window to the bounding box, so you can quickly tell whether it is correct. Trial and error will give a reasonable bounding box in one or two tries. If your editing does not seem to change the way the figure is rendered by `gv`, make sure you are editing the right bounding box comment. Preferably edit out all but the first one.

Automatic BoundingBox Adjustment Ghostscript (`gs`) has a driver bbox whose purpose is to discover the bounding box by drawing the entire graphics file internally and keeping track of what is drawn. It outputs the information. Not all gs executables have bbox compiled in. Execute the command `gs -h` to list the built in drivers. One can call `gs` with the bbox driver and edit the PS file to replace the old BoundingBox comment with the values output. Alternatively a simple script can do the work for you automatically, here is an example:

```
#!/bin/bash
# bbget
# Get a bounding box and put it on the second line of the file.
if [ $# -lt 2 ] ; then
    echo Usage: $0 filein fileout [extrafiles]
    echo Put a bounding box on the second line of the file fileout.
else
#Remove previous bounding boxes.
 echo Removing the following BoundingBox lines:
 sed -n -e "/BoundingBox/!w temp.ps" -e "/BoundingBox/ p" $1
 if [ $# -gt 2 ] ; then cat $3 $4 $5 >> temp.ps; fi
 if [ `fgrep -c showpage temp.ps` != "0" ] ; then
    echo "File has a showpage. No additions."
    showpage=""
 else
    echo "File has no showpage. I'll add one temporarily."
    showpage=' -c "showpage"'
 fi
#Get the BBox info to bb.out
 gs -sDEVICE=bbox -sNOPAUSE -q temp.ps $showpage -c quit 2> bb.out
#Read it in to the second line of the output ps file.
 sed -e"1 r bb.out" temp.ps > $2
```

```
 echo "Bounding Box Information:"
 cat bb.out
#Clean up.
 rm -f bb.out
 rm -f temp.ps
fi
```

This script uses `sed` the stream editor, as well as `gs` and a few other standard linux tools. A point to note is that some PS files do not end with the command `showpage`, which means that they won't print on some printers, but more importantly here that they won't output the bounding box for some `gs` versions. So this script ensures that showpage is called at the end.

The problem with the automatic approach is that `gs` can't always find the bounding box correctly. For bitmaps it often fails. Also, more often than one might suppose the postscript file has some sort of hidden drawing points outside the real visible area. Many Mac programs seem to put spurious invisible points in strange off-canvas places. These cause difficulties with this approach. Also if the postscript clips off vectors outside a certain area by specifying a clip-path then ghostview does not seem to observe the clipping when tracing the bounding box.

2.3 Binary Garbage

The Postscript drivers on Windows and Macs often put binary codes at the start of postscript files if you just "print-to-file". That is because they are still acting as if this file is going to be sent to the particular printer that happens to be selected, and the binary codes are something that the printer might know, but are not postscript. Any file that does not start `%!PS` is not a proper postscript file. Another cause of binary garbage is the header for binary previews. If the start of your file looks something like ÂDó followed by about a line of garbage and then `%!PS-Adobe-3.0 EPSF-3.0`, then you have a binary header file.

Some hints for avoiding these difficulties are for example here, where Tim Love, the author, says: "Under windows when you're printing to file, look at the PostScript properties (or Advanced options), and choose (depending on the driver you have) either 'Archive Format', 'Encapsulated PS', 'Optimize for Portability' or 'Page Independence'. People seem to have more luck with the free Adobe postscript driver than with the Microsoft one". Never 'include preview' if you are going to import into LaTeX.

If you have postscript files with binary previews, it is possible to clean them up (in some cases) using the command `ps2ps`, but this may turn some fonts into line-drawn shapes (see 3.3). Another way is apparently to read the file into GSview (the Windows PS previewer) and use the 'extract PS file' option.

2.4 Line Ends

Postscript is basically an ASCII format. If you transfer a postscript file from Windows or Mac to Linux or vice-versa using ftp, you ought to use the ASCII rather than BINARY setting. Then the line ends will be correctly converted between the systems. However, if you

transfer as binary, or in many cases if you simply copy the file across filesystems, the line ends will be incorrect for the new operating system. This is particularly a problem if the file has a binary preview, because then it is not an ASCII file (nor PS, though it may be EPS) and so it is not correct to transfer it as such. This is one major reason why binary preview is disastrously unportable.

There are available simple programs on linux to convert from Mac or Windows line ends. For Mac to linux, the simplest conversion program is a one line script:

```
tr "\015" "\012" <mac.file >linux.file
```

However, if you apply such a translation to a file with binary preview it totally breaks it.

Ghostscript itself can mostly cope with line-end problems, but scripts that call it usually cannot. This can cause confusion about diagnosing the problem. Inspecting the PS file directly with e.g. `less file.eps` will usually show the problem immediately.

2.5 Orientation

If a file is saved as if to print in landscape orientation, then it quite likely will import in such a way that it is rotated by 90 degrees. LaTeX has ways of rotating such files internally, but this will sometimes have to be done explicitly, which is annoying. If saving or creating a graphics file for subsequent importation, it is better to use portrait provided the bounding box is generated correctly.

If you acquire a figure with bad orientation, you can use `xfig` to fix the orientation as follows. 1. Open a new xfig file. 2. Place the PS file to be rotated into the xfig file as a graphic (use the camera button). 3. Use the rotate and scale tools to orient the graphic upright. 4. Use the edit tool on the graphic and expand or contract it to the original aspect ratio. 5. Export the xfig file as an eps file. If you started with a decent bounding box the result should be well bounded too. What is more, the postscript will not have been flattened or converted, only encapsulated in a relatively benign wrapper that does the rotation and scaling.

3 Portable Document Format (PDF)

PDF is increasingly popular for producing documents, especially those containing graphics. Such graphics can be cut out of their files using Adobe's Acrobat software tools. Therefore graphics are increasingly coming in this format. Also, this is the format most useful for importing into pdf[La]TeX, the TeX engine that produces PDF directly rather than DVI and then PS.

3.1 Converting from PDF to PS

If you have a graphical file in pdf format with a decent MediaBox (PDFspeak for BoundingBox) around it, you can convert it directly into PS using a tool called `pdf2ps` that executes `gs`; modern versions of `gs` support PDF as well as PS. This will quite often work. But it appears that `gs` implicitly calculates the bounding box that the graphics have, and it won't

necessarily find one that is the same as the MediaBox that the original PDF file has. This seems to be a weakness in the way `gs` works. It would be really helpful if the option existed to observe the MediaBox of a PDF file.

Another approach is to use acroread, the Adobe viewer and print to file. Unfortunately this approach will in most cases give a troublesome file. Do not use any of the postscript options such as shrink or expand to fit pages. In the most recent acroreader, these options produce perverse postscript that does internal scaling based on what it thinks the current page size is. This defeats most methods of importing and scaling. If you don't use the fitting options, you get managable postscript but an incorrect bounding box. Moreover, the files (often) are such that the bounding box cannot be automatically fixed; you have to do it by hand.

3.2 Converting from PS to PDF

Since pdflatex can't import PS files, only PDF, the need often arises to produce PDF files from PS. There are (at least) two utilities for this conversion `ps2pdf` and `epstopdf`. They both mostly just call `gs` with the correct options. Unfortunately, the pdfwriter driver that `ps2pdf` uses appears to determine the MediaBox automatically and ignore the specified bounding box. Therefore if you have a problematic file, whose bounding box can't be automatically determined, then even if you have painstakingly edited the BoundingBox by hand, it will likely be to no avail when you produce the PDF file. In this respect `epstopdf` is better than `ps2pdf`, because it pays attention to the bounding box, does some translations to make its values rational, and sets the media box accordingly. It is also sometimes able to strip off binary junk at the beginning of the file. Unfortunately it is not fool proof. In particular it can't cope with files that have broken line ends (see 2.4). For them `ps2pdf` may be better.

Apparently Adobe Distiller can produce PDF files with correct MediaBoxes. It is not freely available on linux.

3.3 Fonts

Fonts in PDF and PS are often problematic. In general it is important to include the entire font into the file, unless it is one of the "standard" Adobe fonts that exist on all printers. If you don't, you may get a PDF file that can't be printed on some printers, or viewed on some operating systems.

If the PDF file is produced from TeX, there are standard ways to produce good looking PDF, by using Type 1 fonts rather than TeX's original bitmapped fonts. These files will work fine.

The problem is fundamentally the use of TrueType fonts. Since these are available mostly only on Windows machines, using them will produce PDF files that are of dubious portability. Acroread will render them, but other conversion tools will often find it difficult. A procedure (provided by Earl Marmar) for making pdflatex compatible pdf files, starting from illustrator is as follows:

`In Illustrator:`

```
1) Check your fonts: pull down the "type" menu, to "find font...";
uncheck the "truetype" box
2) If any of your fonts disappeared from the second list, then they were
truetype. Replace them with type 1 (one of the ones from the list that
remained after unchecking the "truetype" box)
3) An alternative to replacing the truetype fonts, is to convert them to
type 1 during step 7 below, but this seems to create less pleasant
looking pdf
4) From the file menu, use "save as" to save the file in illustrator pdf
format
5) open the resulting pdf file with acrobat

In acrobat:

6) change the bounding box if you want to remove white space around the
figure: use the crop tool for this (double click the crop icon to get a
widget if you prefer)
7) from the file menu, export to eps without preview, using postscript
level 2; font inclusion: all embedded (although this does not seem to
help if you are using truetype fonts).
```

The resulting file should work with epstopdf on linux, and the pdf file you get from that should work with pdflatex on linux.

4 Using pstoedit and xfig

One approach to fixing PS figures that are unsatisfactory is to convert them into an editable format. The program `pstoedit` does a remarkably good job on this for PS vector files that are not too perverse (bitmaps are a different issue). Its forte is converting vector eps into "fig" format, which is the native format of the linux editing tool `xfig`. Once the conversion to fig format is complete, your `xfig` can open the fig file and edit it as needed. Because `xfig` is so good at exporting properly bounded PS (export eps), once you have it in `xfig` as vectors, you can be pretty much guaranteed to get a satisfactory PS file. The `pstoedit` converter also supports other formats, but many of them are through the use of gs. So if `gs` has a problem, the conversion may have the same problem.

Once in `xfig` you can usually fix the problems with the file. The most frequent are that there are spurious drawn lines or points outside the proper drawing. This can happen if a larger drawing is unsatisfactorily cropped. Some programs do this by simply masking out the region. The way to fix it is to delete those extraneous extra points. Sometimes it might seem as if you have deleted them but `xfig` still doesn't output a tight bounding box. This is almost always because there are hidden points still present. The best test for that in `xfig` is to do from the menu: view, zoom to fit canvas. Then you'll usually see where those additional points are and can delete them.

The conversion to fig format will sometimes give you horrid courier fonts instead of what

the PS file had. Believe it or not this is a good thing! The conversion was unable to find fonts on the system, so it substituted those courier fonts instead. Therefore you have got rid of incompatible fonts. To get back decent-looking fonts, either do the conversion again with the extra switch setting `-df Helvetica`, or use the update tool to change the font face of the entire figure to one of those available in `xfig`, e.g. Helvetica. Do this by combining everything you want to change into a grouped object and then applying the update tool with just the font face checked.

Problems occasionally arise with vector drawn fonts in pstoedit conversion. The lines that the letters, or sometimes graphical objects, are made of may have their corner styles changed, resulting in less satisfactory aesthetics.

5 Bitmap Formats

Bitmaps are inescapable for photographs and are also used for color contours and a variety of color gradients. There is a long list of formats. Fortunately there exist comprehensive sets of tools for dealing with them, such as the NetPBM utilities[2], as well as freely available image editors such as GIMP. If the graphic consists of nothing but the bitmap, then it is relatively easy to deal with, and the main question is how to avoid getting a PS file that is enormous. In fact it may well be the best policy to ask coauthors if they are supplying photographs to supply the JPEG file itself. Then you can convert it to PS in your own rational way, rather than receiving a butchered file from someone else's attempt to convert.

5.1 PNM to PS

The defacto lowest common denominator of bitmaps on linux is the PNM format. If a a file can be converted into this format, then it is easy to convert this into reasonably compact PS and PDF. There is a utility `pnmtops`, which works but gives very bulky files. The way I usually do it is to use JPEG format inside the PS file, which gives good compression of the bitmap to reduce the size of the PS file. The following commands will do the job:

```
cjpeg -qual 100 filebase.pnm >filebase.jpg
jpeg2ps filebase.jpg > filebase.eps
```

Here `cjpeg` and `jpeg2ps` are linux utilities that are present in many distributions or available off the web. They are converters from PNM to JPEG and JPEG to PS respectively. Choosing quality = 100 ensures minimal image quality loss, but produces larger files than the default (75). It depends on your image whether the extra quality is needed.

5.2 JPEG

It may seem obvious that if you already have a JPEG bitmap, just the second of the commands above is all you need to convert it to PS. Unfortunately, that obvious fact is false. Some JPEGs are coded as "progressive JPEG", which is the type that on a slow web link

[2]Another set of tools is ImageMagick. My experience with NetPBM has been better.

starts out blurred and chunky and gradually gets sharper as more of the data is received. This type cannot be included into eps files. Instead you have to convert it to the standard jpeg first through the step:

```
djpeg -pnm fileold.jpg | pnmcrop >filenew.pnm
```

followed by standard treatment of the resulting pnm file.

5.3 GIF

The GIF standard used to be plagued by a patent that UNISYS was granted on the compression method GIF uses. That patent has lapsed now, but it still leaves a sour taste. Still there are lots of GIFs out there. The utility to convert to PNM is:

```
giftopnm file.pnm | pnmcrop | pnmmargin 10 >file.pnm
```

Notice the optional use of pipes through the utility `pnmcrop` which ensures that uniform color regions are cropped off the outside, and `pnmmargin` which puts back a small margin of 10 pixels here.

5.4 PNG

Portable Network Graphics (PNG) is a standard designed to replace GIF. It is supported by modern browsers. As might be expected, there's a converter. It is called `pngtopnm`, and works just the same as the others.

There are actually a couple of other converters that sound interesting, but I don't have experience with. `imgtops` is available at `http://imgtops.sourceforge.net/`. "It takes bitmap images (in almost any common format) and turns them into Encapsulated PostScript files. It uses PostScript Level 2 and 3 features in order to minimize the size of the output files without reducing image quality." It requires Python to be installed. The other, `bmeps`, is at `http://www.e-technik.fh-schmalkalden.de/personen/dhp/krause` (click to Software (english), bmeps + dvips-add-on). It works on PNG and PNM files (as well as JPEG, but for JPEGs, jpeg2ps is said to be better). These use loss-less compression.

5.5 Combined Vector and Bitmap Graphics

The harder situation is when a graphical object has both bitmaps and vectors and fonts in it. The options for dealing with it, if it is somehow broken are either to edit it using the tool that originally produced it or an alternative sufficiently powerful drawing tool, or to convert it completely into a bitmap and edit it from there.

If all you need to do is to rotate it, then follow the steps using `xfig` described under orientation (2.5). When that is done, the resulting PS file will have a simple bounding box line such as

```
%%BoundingBox: 0 0 1173 906
```

near the top (line 6 actually) and a line like:

```
newpath 0 906 moveto 0 0 lineto 1173 0 lineto 1173 906 lineto closepath clip newpath
```

about 50 lines down. You can see that the clip path corresponds to the bounding box. Both can be edited manually to clip and bound the figure appropriately.

6 Scanned Images of Line Drawings

Often one might be interested in including an image scanned from paper. Since a scanned image can usually be saved as a JPEG, this is straightforward for photos and other gradient type images. However, it might be that the image is a line drawing, with mostly white areas. If so, using the JPEG directly would not be the most efficient or satisfactory format for including it.

One might be tempted to scan a monochrome line drawing using black and white settings of the scanner, since, after all, it is a two color only image. This is usually a mistake. By all means use gray-scale if the image has no other colors, but do not use a two-level scale. If you do, the lines will have ugly edges with little teeth on them where the individual pixels are either white or black depending on precise illumination. If you are given such a file, it may be possible to improve its appearance by using a blur tool in an image editing program such as gimp, but it will not be as satisfactory as the following.

Scan the drawing with enough pixels per inch to resolve easily the details of the image. Save it as a some convenient format, PNM or even JPEG of high quality is fine. Look at it and realize that the white background is of course not perfectly white, which is a bad thing, and that the lines are slightly softened at their edges by shades of gray, which is a good thing visually. Now we process the figure to make the white really white by telling the image that anything brighter than some fairly bright level is to be made pure white. That can be done with graphical user interface tools, or more easily for production by the following.

```
djpeg -pnm -gray filebase.jpg | pgmnorm -wvalue 220 | pnmcrop \
 | pnmmargin -white 10 > filebase.pnm
```

Here the assumed initial JPEG is converted to PNM and piped through the utility manipulation step `pgmnorm -wvalue 220` which tells it that anything brighter than 220 (out of 255) is pure white. When this has been done `cjpeg` and `jpeg2ps` will easily convert it into PS. Actually it would be better to use loss-less compression on the final file, such as is used in GIF, because that is more efficient for line drawings. Perhaps `imgtops` or `bmeps` would be good solutions for that.

7 LaTeX Desperation Tweaks

In extremis it is possible to import even files that have bad bounding boxes or improper orientation and fix them by optional commands to the figure inclusion command `\includegraphics`. I consider this a last resort because if you deliberately do clipping or rotation in LaTeX, then you are locking in an incorrect scaling or perhaps orientation of the figure. If the figure is subsequently changed, it might even be resubmitted with correct settings for orientation and bounding box. Then your LaTeX will break it!

What is more, if you want to convert your document to HTML, for example, there is no way for programs that convert your figures to know that they ought to do the rotation, scaling, or cropping that you did with LaTeX.

Many details about including PS figures in LaTeX using the `graphicx` package are given in the document Using imported graphics in LaTeX2e by Keith Reckdahl which most linux installations include as `file:///usr/share/texmf/doc/latex/graphics/epslatex.ps`. It also gives additional information on some of the questions discussed here.

8 Summaries

8.1 Problem diagnosis

Symptoms	Diagnosis
Figure is small object in large whitespace area	BoundingBox incorrect 2.2
Figure is correct but too large for its area, overwriting other stuff	BoundingBox incorrect 2.2
Figure has extraneous additions extending outside its area	Clipping incorrect
LaTeX complaint about Bounding Box	No bounding box in PS 2.2 or Line ends broken 2.4 or Binary corruption in PS 2.3
Figure is sideways or upside down	incorrect orientation 2.5
Page preview ok but won't print on some printers	Non-standard fonts used 3.3
pdfLaTeX can't find figure	figure.pdf not present 3.2 or figure.eps specified in LaTeX
Fonts are ugly and jagged	Bitmapped fonts used 3.3 or Graphic is a bitmap
Background has dirty grey shading	Scanned image not cleaned 6
PS file is gigantic, renders slowly	Graphic is large bitmap 5.1
PDF file has white margin not in PS version	ps2pdf inadequate 3.2

8.2 Tools

Tool	Purpose	Applications and Comments
gs	postscript interpreter	conversion to new format 1
ps2ps	make simpler/cleaner postscript	remove binary garbage 2.3
ps2pdf	make PDF from PS	convert graphics for pdflatex 3.2
epstopdf	make PDF with better MediaBox	needs correct line ends 3.2
pstoedit [-f fig]	make xfig file from PS	edit, vector files only 4
pstopnm	make PNM from PS	image production
jpeg2ps	encapsulate JPEG as EPS	create compact EPS bitmap 5.1
djpeg [-pnm]	translate JPEG [to PNM]	remove progressive JPEG 5.2

cjpeg	make JPEG from PNM	enable additional processing 5.2 prior to jpeg2ps conversion
NetPBM	Suite of bitmap tools	batch conversion, manipulation
pnmcrop	crop PNM	remove margins from bitmap 5.3
pnmmargin	add margin to PNM	add specified margin 5.3
pngtopnm	make PNM from PNG	conversion of www bitmap 5.4
giftopnm	make PNM from GIF	conversion of www bitmap 5.3
pnmtops	make bulky PS from PNM	use jpeg2ps instead
sed	stream editor	file editing in scripts 1
less, cat	examine, type out files	linux standard tools 2.4
xfig	Line drawing program	fixing orientation 2.5 Editing PS file 4
gimp	Bitmap drawing program	5
acroread	PDF viewer	produce PS from PDF 3.1
Illustrator	Adobe drawing program	not on linux 3.3
Acrobat	Adobe PDF manipulation	not on linux 3.3

8.3 Instructions to graphics authors

The following might be useful to provide collaborators or others who are supplying you with graphics, as a way of telling them how to supply graphics in a compatible form.

- Supply Encapsulated Postscript, without preview, in portrait orientation.
- Ensure the margin is minimal, the edges of the media surround the graphic snuggly.
- Don't use print-to-fit settings.
- Don't convert vector drawn figures to bitmap.
- Don't use TrueType fonts; use Type 1 instead. Use standard fonts such as Times, Helvetica, Courier, etc in styles such as roman, bold, italic etc wherever possible. [3]
- Don't scan monochrome line drawings as black/white (two color); use gray-scale.
- Don't supply figures that have been through Powerpoint unless unavoidable.[4] Please use the original, prior to being imported into Powerpoint.
- For vector drawn figures, export as PDF and EPS and supply both.
- Supply (the original) JPEG for photographs if available.
- Transfer EPS files using ASCII (plain text) ftp settings.

[3] In addition to the standard faces mentioned, most postscript devices also have access to AvantGarde-Demi, AvantGarde-Book, Bookman-Demi, Bookman-Light, Helvetica-Narrow, NewCenturySchlbk, and Palatino fonts in one or more style/weight.

[4] A description exists of how to get acceptable EPS figures from powerpoint, but it is truly painful.

- View your graphic using `gv file.eps` on linux. If it works and shows the graphic nearly filling the window, it is probably good. If not, please edit your figure and supply and test it again.

9 Conclusion

The profusion of complicated graphics tools has led to a confusion of formats and annoying and time-wasting incompatibities. The worst culprits are the large software vendors. When they are in monopoly or near-monopoly control of the operating system or graphics format, they have no immediate financial incentive to output compatible, standards-observing files. They sometimes deliberately make their output complicated and incompatible with other tools so as to try to force people to use the monopolist's own products. The only way to oppose this deliberate confusion is to insist on openly defined standards and learn how to produce from your tools files that observe those standards.

If you have corrections to this document, or more or better advice than is contained in it, I would be interested to hear about them at hutch@psfc.mit.edu. However, please give references, describe ways, and/or provide files, that would make it possible for me to verify your claims.