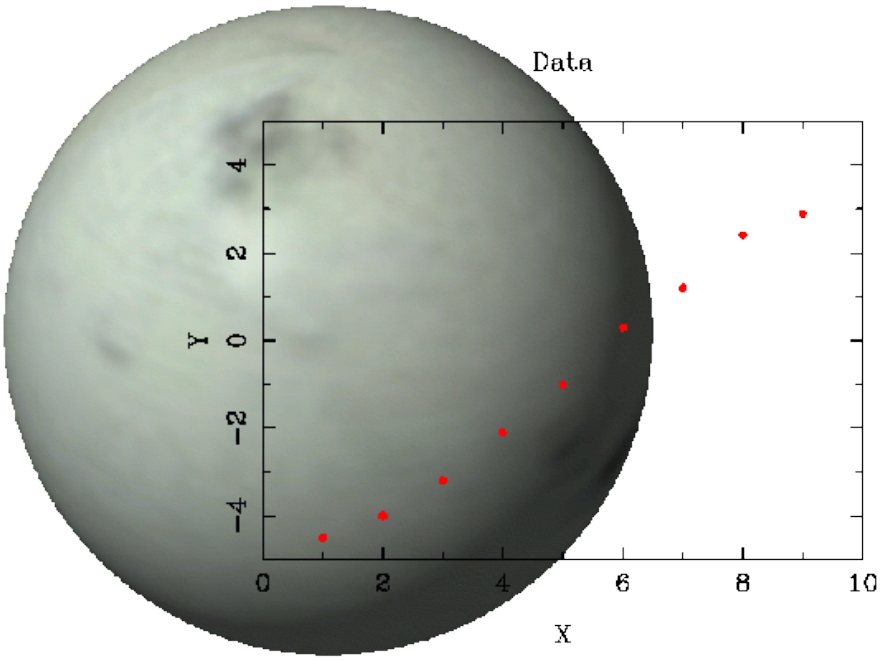
SC/15.2

Starlink Project
STARLINK Cookbook 15.2

A. Allan, D. Terrett

22nd August 2000

The Graphics Cookbook



Abstract

This cookbook is a collection of introductory material covering a wide range of topics dealing with data display, format conversion and presentation. Along with this material are pointers to more advanced documents dealing with the various packages, and hints and tips about how to deal with commonly occurring graphics problems.

Contents

1 Introduction **3**

2 Call for contributions **3**

3 Subroutine Libraries **3**
3.1 The PGPLOT library . . . 3
3.1.1 Encapsulated Postscript and PGPLOT . . . 5
3.1.2 PGPLOT Environment Variables . . . 5
3.1.3 PGPLOT Postscript Environment Variables . . . 5
3.1.4 Special characters inside PGPLOT text strings . . . 7
3.2 The BUTTON library . . . 7
3.3 The `pgperl` package . . . 11
3.3.1 Argument mapping – simple numbers and arrays . . . 12
3.3.2 Argument mapping – images and 2D arrays . . . 13
3.3.3 Argument mapping – function names . . . 14
3.3.4 Argument mapping – general handling of binary data . . . 14
3.4 Python PGPLOT . . . 14
3.5 GLISH PGPLOT . . . 15
3.6 `ptcl` Tk/Tcl and PGPLOT . . . 15
3.7 Starlink/Native PGPLOT . . . 15
3.8 Graphical Kernel System (GKS) . . . 16
3.8.1 Enquiring about the display . . . 16
3.8.2 Compiling and Linking GKS programs . . . 17
3.9 Simple Graphics System (SGS) . . . 17
3.10 PLplot Library . . . 17
3.10.1 PLplot and 3D Surface Plots . . . 18
3.11 The `libjpeg` Library . . . 19
3.12 The `giflib` Library . . . 20
3.13 The `libungif` Library . . . 20
3.14 The `angif` Library . . . 21
3.15 The PNG Format . . . 21
3.16 The MNG Format . . . 21
3.17 The Python Imaging Library . . . 22
3.18 The `gd` Library . . . 22
3.18.1 `gd` from other languages . . . 23

4 Plotting Packages **23**
4.1 QDP . . . 23
4.1.1 QDP Basic Stuff . . . 23
4.1.2 Plot devices and PostScript output . . . 25
4.1.3 Error Bars . . . 26
4.1.4 That "Date and Time" Thing . . . 27
4.1.5 Fitting using QDP . . . 27
4.1.6 QDP Files . . . 30
4.1.7 COD and QDP models . . . 30
4.2 PONGO . . . 31
4.3 SM . . . 31

4.4 GNUplot . . . 31
4.4.1 Co-ordinate systems . . . 32
4.4.2 Plotting 3D data . . . 33

5 Image Display **35**
5.1 KAPPA . . . 35
5.2 SAOimage . . . 35
5.2.1 Printing in SAOimage . . . 35
5.3 GAIA . . . 35

6 Visualisation **37**

7 Other Applications **37**
7.1 ImageMagick . . . 37
7.1.1 `display` . . . 37
7.1.2 `import` . . . 38
7.1.3 `animate` . . . 39
7.1.4 `montage` . . . 39
7.1.5 `convert` . . . 39
7.1.6 `mogrify` . . . 39
7.1.7 `identify` . . . 40
7.1.8 `combine` . . . 40
7.1.9 `XTP` . . . 40
7.1.10 XMagick . . . 40
7.1.11 PythonMagick . . . 42
7.2 XV . . . 43
7.2.1 Screen Capture . . . 43
7.2.2 Problems with small images . . . 43
7.2.3 Getting XV, patches and enhancements . . . 43
7.2.4 Compiling XV on RedHat 6.0 . . . 43
7.2.5 XV is **not** under the GPL . . . 45
7.3 XPaint . . . 45
7.4 Xfig . . . 46
7.4.1 `pstoedit` . . . 46
7.5 Sketch . . . 47
7.6 GIMP . . . 47
7.6.1 Plug-ins, `Script-Fu`, GIMP-Perl and `Gimp::Fu` . . . 50
7.6.2 GIMP-Python . . . 51
7.6.3 GIMP Plug-In Registry . . . 53
7.6.4 The GIMP and layers . . . 53
7.6.5 Mapping images to solid objects . . . 60
7.7 Electric Eyes . . . 62
7.8 WhirlGIF . . . 62

8 CAD Applications **65**
8.0.1 QCad . . . 65
8.0.2 XCircuit . . . 67

9 Format Conversion **67**

9.1 CONVERT . . . 67
9.1.1 NDF2GIF . . . 68
9.2 PBMplus . . . 69
9.2.1 PBM, PGM, PPM or PNM images? . . . 69
9.3 Image Resizing . . . 69

10 Postscript and PDF **69**
10.1 Ghostscript . . . 70
10.2 GV and Ghostview . . . 70
10.3 Acrobat . . . 70
10.4 `psmerge` . . . 72
10.5 `epsutil` . . . 72
10.6 `prescript` and `pstotext` . . . 72
10.7 Postscript to PDF . . . 72
10.8 PS Utils . . . 72
10.9 Generating Postscript Output . . . 76

11 X Window Displays **76**
11.1 Pseudocolor . . . 76
11.2 Grey Scale . . . 76
11.3 Static Grey . . . 76
11.4 Directcolor . . . 76
11.5 Truecolor . . . 77
11.6 What does this mean for you? . . . 77
11.7 Pseudo Colour applications on True Colour desktops . . . 78

12 Virtual Computing **79**
12.1 Virtual Network Computing (VNC) . . . 79
12.1.1 Pseudo colour displays . . . 79
12.1.2 Computing by remote control . . . 81
12.2 VMWare . . . 81
12.3 plex86 (previously FreeMWare) . . . 84
12.4 The VMWare and plex86 Patent Position . . . 84

13 Hardware **85**
13.1 Scanners . . . 85
13.2 Digital Cameras . . . 85

14 The Web **87**
14.1 Transparent GIFs . . . 87
14.2 Animated GIFs . . . 88
14.3 Beveled Images . . . 88
14.4 “Web Safe” Colour Maps . . . 89
14.5 Browser support of PNG images . . . 89

15 The GIF Legal Position **89**

16 From the Quick Archives **90**
16.1 FITS to MPEG . . . 90

17 Package Availability **92**

List of Figures

1 Output from the simple PGPLOT Fortran program. . . . 4
2 Example of an application using the BUTTON PGPLOT extension library. . . . 8
3 Output from the simple `pgperl` example script. . . . 12
4 The default appearance when plotted using QDP. . . . 24
5 Plotting in QDP using a Roman font. . . . 26
6 Our second test data set plotted using QDP with the default options. . . . 27
7 Our second test data set plotted using QDP with best fit model. . . . 29
8 A plot of a function using GNUplot . . . 32
9 A plot in polar co-ordinates using GNUplot. . . . 33
10 A 3D plot using GNUplot. . . . 34
11 A 3D plot of a digitised blue whale using GNUplot. . . . 34
12 The GAIA interface. . . . 36
13 The ImageMagick `display` GUI. . . . 38
14 The `xv` interface. . . . 44
15 The `xv` interface being used to grab an image from the display. . . . 45
16 The `xpaint` interface. . . . 46
17 Creating a figure using `xfig`. . . . 47
18 The GIMP interface. . . . 48
19 Tux the penguin. . . . 53
20 The GIMP Layers & Channels dialog. . . . 54
21 Making a “floating selection” in GIMP. . . . 54
22 A new layer has been created for the image. . . . 55
23 The pop-up dialog. . . . 55
24 Our renamed text layer is selected in the dialog. . . . 55
25 The GIMP Levels dialog. . . . 56
26 The Levels dialog again, note the position of the bottom slider. . . . 56
27 Tux with his darkened text layer. . . . 57
28 The GIMP Text dialog. . . . 57
29 Tux with additional layers. . . . 58
30 The Layers & Channels dialog again with the first text layer selected. . . . 58
31 The final product, Tux with text. . . . 59
32 The GIMP File dialog. . . . 59
33 Original star spot image generated using PGPLOT . . . 60
34 Star spot image mapped on sphere. . . . 61
35 Star spot image projected on plane. . . . 61
36 Star spot image with an inverse bump map and gaussian blur. . . . 62
37 Bump mapped star spot image projected on a plane . . . 63
38 The GIMP Map Object Dialog “Options” . . . 63
39 The GIMP Map Object Dialog “Light” . . . 64
40 The GIMP Map Object Dialog “Material” . . . 64
41 The GIMP Map Object Dialog “Orientation” . . . 65
42 The Electric Eyes interface. . . . 66
43 The QCad application running under KDE. . . . 67
44 The Xcircuit interface. . . . 68
45 The GV interface. . . . 71
46 VNC displaying a pseudo colour desktop on a true colour display. . . . 80

47 VNC displaying a Windows 98 desktop on a X Windows display. 82
48 VMWare Schematic 82
49 VMWare booting the virtual machine using Linux as the “host” OS. 83
50 VMWare running Windows 98 as a “guest” OS, using Linux as the “host” OS. . . 84
51 plex86 running DOS 6.22 as a “guest” operating system. 85
52 The `gphoto` interface. 86

Revision history

(1) 2nd February 2000; Version 1. Release version (AA)

(2) 22nd August 2000; Version 2. Checked URLs + minor updates (AA)

1 Introduction

Data display and visualisation has become more complex over the last few years. Considering that it was not trivial to begin with it is unsurprising that this is an area that throws a few problems in the path of getting your paper into press.

This cookbook will attempt to address some of the more common problem areas, however it is unlikely that it will deal with your specific graphics problem (unless you are really lucky) since the sheer scope of the topic precludes such an approach. Instead of discussing solutions to a few individual problems, we'll focus on the tools that will allow you to solve a range of problems. Hopefully it will help you to help yourself. This document is therefore more of a collection of basic material with pointers to move advanced and in-depth documents, than a cookbook which provides set recipes to approach your graphics problems. Indeed, perhaps it should be referred to as an "ingredients" book rather than a cookbook.

The cookbook is probably best browsed in the HTML version since much of its content is pointers to packages, anonymous FTP sites and further information about the various packages, libraries and applications discussed. All of these are more easily accessed from an online copy than a bound one. I have, however, made an effort to include URLs in the text whereever possible in the LaTeXversion.

Details of where the packages discussed in this cookbook can be obtained, if not explicitly mentioned in the relevant section, can be found in Section 17.

2 Call for contributions

As with the *Theory and Modelling Resources Cookbook* the *Graphics Cookbook* is a wide ranging and (virtually) open ended project. Hopefully I've covered enough ground so that you at least know which tool you should be using to solve you problem, even if you are not sure (quite yet) how to use it. I would welcome comments, contributions and corrections to this document, since I have been aware while compiling it of bias towards my own favourite applications, such as PGPLOT. Comments should be sent either to me at `aa@astro.keele.ac.uk` or to the Starlink software librarian `starlink@jiscmail.ac.uk`.

3 Subroutine Libraries

3.1 The PGPLOT library

The PGPLOT library is `Fortran` or `C` high-level callable graphics library. It has become the de-facto standard for display of astronomical data. Along with the standard primitives to draw lines, write text and annotate plots, there are also high level routines which use these primitives to build up more complicated graphs such as histograms and contour maps. PGPLOT also has the capability for interactive graphics where the user can interact with the plots using the cursor.

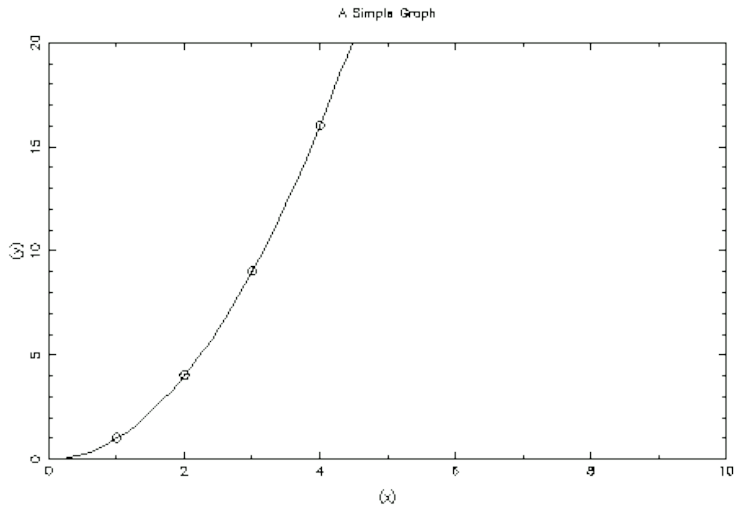


Figure 1: Output from the simple PGPLOT Fortran program.

While it would be quite possible for me to fill the entire cookbook with PGPLOT information an excellent tutorial style manual already exists for the library written by Tim Pearson of CalTech the package's author. If you work at a Starlink node this manual should be available to you, if not it's available on the web at `http://astro.caltech.edu/~tjp/pgplot/contents.html`.

A simple example program is shown below, taken from Chapter 2 of the PGPLOT manual. It shows how a simple plot can be generated in just a few lines of code, the output from the program is shown in Figure 1.

```
      PROGRAM GRAPH

      INTEGER I, IER, PGBEG
      REAL XR(100), YR(100)
      REAL XS(5), YS(5)

      DATA XS/1.,2.,3.,4.,5./
      DATA YS/1.,4.,9.,16.,25./

      IER = PGBEG(0,'?',1,1)                      ! Start PGPLOT
                                                  ! Specify the device

      IF (IER.NE.1) STOP                          ! No response? Stop execution
      CALL PGENV(0.,10.,0.,20.,0,1)               ! Initialise the axes
      CALL PGLAB('(x)', '(y)', 'A Simple Graph')  ! Label the axes
      CALL PGPT(5,XS,YS,9)                        ! Plot 5 points using symbol 9

      DO 10 I=1,60
          XR(I) = 0.1*I                           ! Calculate X^2 function
          YR(I) = XR(I)**2
   10 CONTINUE
      CALL PGLINE(60,XR,YR)                       ! Draw a line
      CALL PGEND
      END
```

3.1.1 Encapsulated Postscript and PGPLOT

A commonly asked question is how to get PGPLOT to produce valid encapsulated postscript (EPSF) files. Most PGPLOT postscript files are valid EPSF files, except for multi-page plots. Some problems do exists however, valid EPSF files should have a `%%BoundingBox` comment line in the header of the file, PGPLOT places this line in the trailer of the file where some programs will fail to find it. This comment can be moved into the file header using any text editor. If you do not wish to do this you can also modify the way PGPLOT deals with bounding boxes using the `PGPLOT_PS_BBOX` environment variable.

Additionally PGPLOT Postscript files do not contain a screen preview section. A device-independent screen preview can be added to PGPLOT files with the program `ps2epsi` by George Cameron, available with the GhostScript PostScript interpreter.

3.1.2 PGPLOT Environment Variables

Some of the more useful environment variables which control the behaviour of PGPLOT are listed below, the list is not exhaustive, for a full list you should see the relevant sections of the PGPLOT manual.

`PGPLOT_DIR` Directory name. Unless told otherwise by environment variables `PGPLOT_FONT` and `PGPLOT_RGB`, PGPLOT looks for the files it needs at run-time in this directory. The binary font file is `grfont.dat` and the color-name database is `rgb.txt`. If this variable is undefined, or if the specified file does not exist in this directory, PGPLOT looks in the current default directory, *e.g.* `setenv PGPLOT_DIR /usr/local/lib/pgplot/`. For Starlink users this environment variable will be set at login when you source the Starlink `login` file, or by typing the `starlink` command at the UNIX prompt.

`PGPLOT_DEV` Device specification. If this variable is defined, it is used as the default device specification: if the device specification given to `PGBEG` (or supplied by the user in response to the PGPLOT prompt) is a blank string, this device specification is used, *e.g.* `setenv PGPLOT_DEV /xwin`. The Starlink distributed version of Native PGPLOT has an additional device, `/gwm`, that allows plotting in GWM Windows.

`PGPLOT_TYPE` Device type. If this variable is defined, it is used as the default device type: if the device specification supplied to PGBEG consists of a file name without a trailing slash (/) and device type, this device type is assumed, *e.g.* `setenv PGPLOT_TYPE ps`

`PGPLOT_BUFFER` If this variable is defined, with any non-null value, PGPLOT buffers output. The effect is the same as if `PGBBUF` is called immediately after opening the graphics device, and PGEBUF immediately before closing it. It will have no effect on programs that already include these calls. On some devices, buffering output can lead to large improvements in speed, but enabling buffering may upset synchronization between graphical output and other program activity, *e.g.* `setenv PGPLOT_BUFFER yes`

3.1.3 PGPLOT Postscript Environment Variables

Problems dealing with PGPLOT's Postscript output are amongst the most common complaints about the package. PGPLOT has several environment variables which control the output from its Postscript device driver

`PGPLOT_PS_WIDTH (default 7800)`

`PGPLOT_PS_HEIGHT (default 10500)`

`PGPLOT_PS_HOFFSET (default 350)`

`PGPLOT_PS_VOFFSET (default 250)` These variables tell PGPLOT how big an image to produce. The driver uses resolution elements of 0.001 inches, (*i.e.* milli-inches)giving an "apparent" resolution of 1000 pixels per inch. The true resolution is device-dependent. The image dimensions are therefore 7.8 inches horizontally by 10.5 inches vertically (portrait mode) by default. These defaults while defined with American 8.5 x 11-inch paper in mind, rather than the European A4 size sheets, are appropriate in most circumstances. The maximum dimensions of a PGPLOT image are `WIDTH` by `HEIGHT`, with the lower left corner offset by `HOFFSET` horizontally and `VOFFSET` vertically from the lower left corner of the paper. The "top" of the paper is the edge that comes out of the printer first.

`PGPLOT_IDENT` If this variable is defined (with any value), the user name, date and time are written in the bottom right corner of each page.

`PGPLOT_PS_BBOX` Normally, PGPLOT computes the bounding box for the entire plot (the smallest rectangle that includes all the graphics) as it creates the PostScript file, and writes this information in a %%BoundingBox comment in the file trailer. Some programs that read encapsulated PostScript files expect to find the %%BoundingBox comment in the file header, not the trailer, and may not display the plot correctly. To fix this problem, you may need to move the comment from the trailer to the header with a text editor or special program. Alternatively, you can define `PGPLOT_PS_BBOX = MAX`. This tells PGPLOT to put a %%BoundingBox comment in the header of the PostScript file; the bounding box is one which encompasses the whole plotable area, not a minimal one, because PGPLOT does not know the correct bounding box until it has finished writing the file.

`PGPLOT_PS_DRAW_BBOX` If this variable is set, the bounding box (the smallest rectangle that includes all the graphics) is drawn on each page.

`PGPLOT_PS_VERBOSE_TEXT` If this variable is set, the text of each plotted character string is included in the PostScript file as a comment before the sequence of vectors that represents the string. This makes the file slightly larger, but it can be useful if you want to hand edit the PostScript file.

`PGPLOT_PS_EOF` Normally the output file does not contain special end-of-file characters. But if environment variable `PGPLOT_PS_EOF` is defined (with any value) PGPLOT writes a Control-D job-separator character at the beginning and at the end of the file. This is appropriate for Apple LaserWriters using the serial interface, but it may not be appropriate for other PostScript devices.

`PGPLOT_PS_MARKERS` Specify `NO` to suppress use of a PostScript font for the graph markers; markers are then emulated by line-drawing. If this option is not requested, PGPLOT graph markers are scaled geometrically with the character-height attribute and the line-width attribute is ignored. This is different from most of the other drivers, where the line-width used for markers is set by the line-width attribute rather than the character-height attribute. Requesting this option makes the PostScript driver behave like the other drivers, but it also makes the PostScript files larger.

3.1.4 Special characters inside PGPLOT text strings

Another group of commonly asked questions about PGPLOT concern how to put Greek characters, or super- and sub-scripts into text strings (such as axis annotations) in PGPLOT. The `PGPTXT` subroutine, along with all the routines that call it such as the higher level `PGTEXT` or `PGLAB`) uses escape sequences embedded in the text string to print these characters. Escape sequences are characters which are not plotted, but instead instruct the program to change font, draw superscripts, subscripts or non-ASCII characters (*e.g.* Greek letters). All escape sequences start with a backslash character "\". The following escape sequences are defined:

`\u` Start a superscript, or end a subscript

`\d` Start a subscript, or end a superscript

`\b` Do not advance text pointer after plotting the previous character

`\fn` Switch to Normal font (1)

`\fr` Switch to Roman font (2)

`\fi` Switch to Italic font (3)

`\fs` Switch to Script font (4)

`\\` Print a backslash character "\"

`\x` Multiplication sign (×)

`\.` Centered dot (·)

`\A` Ångström symbol (Å)

`\g`*x* Greek letter corresponding to roman letter *x*

`\m`*n*

`\m`*nn* Standard graph marker number *n* or *nn* (1-31)

`\(`*nnnn*) Character number *nnnn* (1 to 4 decimal digits) from the Hershey character set; the closing parenthesis may be omitted if the next character is neither a digit nor ")". This makes a number of special characters (*e.g.* mathematical, musical, astronomical, and cartographical symbols) available. See Appendix B in the PGPLOT manual for a list of available characters.

3.2 The BUTTON library

While plots can be made interactive using standard PGPLOT functions, *e.g.* `PGCURS`, the BUTTON library extends this functionality to allow you to easily create interactive FORTRAN applications using graphic buttons, see Figure 2.

The program which produced the plot shown in Figure 2 is shown below, and is compiled using the command `f77 -o sample sample.f libbutton.a -lpgplot -lX11` (assuming that `libbutton.a` shared library has been built and is in the current directory).

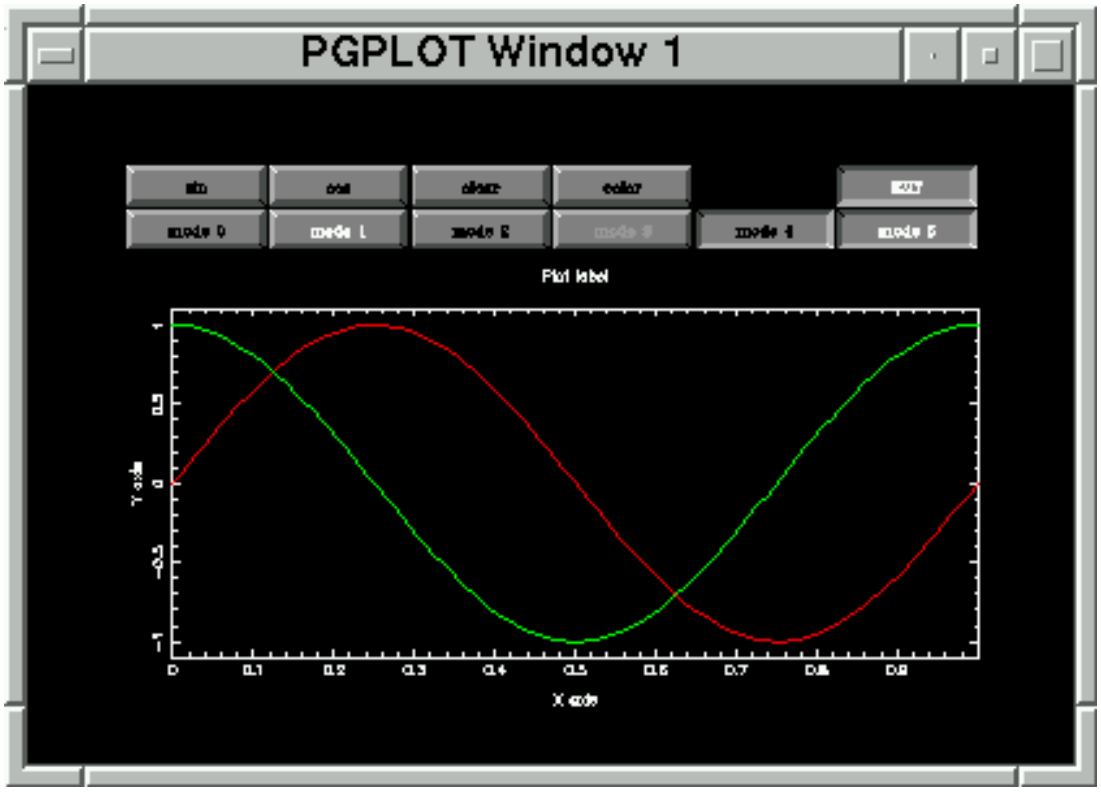


Figure 2: Example of an application using the BUTTON PGPLOT extension library.

```
!-----------------------------------------------------------------------------
! Version 23-July-1998                                          File: sample.f
!-----------------------------------------------------------------------------
! Copyright N. Cardiel and J. Gorgas, Departamento de Astrofisica
! Universidad Complutense de Madrid, 28040-Madrid, Spain
! E-mail: ncl@astrax.fis.ucm.es or fjg@astrax.fis.ucm.es
!-----------------------------------------------------------------------------
! This program is free software; you can redistribute it and/or modify it
! under the terms of the GNU General Public License as published by the Free
! Software Foundation; either version 2 of the License, or (at your option) any
! later version. See the file gnu-public-license.txt for details.
!-----------------------------------------------------------------------------

      PROGRAM SAMPLE
      IMPLICIT NONE

      INTEGER I,NB,NCOLOR
      INTEGER NTERM,IDN(8),ITERM
      REAL XC,YC
      REAL XX(100),YY(100)
      REAL XV3,XV4,YV3,YV4
      LOGICAL LCOLOR(8)
      CHARACTER*1 CH

! Open graphic output
```

```
      CALL RPGBEGIN(NTERM,IDN,LCOLOR)

! Plot and label buttons

5     CALL BUTTON(1,'sin',0)
      CALL BUTTON(2,'cos',0)
      CALL BUTTON(3,'clear',0)
      CALL BUTTON(4,'color',0)
      CALL BUTTON(6,'EXIT',0)
      CALL BUTTON(7,'mode 0',0)
      CALL BUTTON(8,'mode 1',0)
      CALL BUTTON(8,'mode 1',1)
      CALL BUTTON(9,'mode 2',0)
      CALL BUTTON(10,'mode 3',0)
      CALL BUTTON(10,'mode 3',3)
      CALL BUTTON(11,'mode 4',4)
      CALL BUTTON(12,'mode 5',5)

! Plot box

      DO ITERM=NTERM,1,-1
        CALL PGSLCT(IDN(ITERM))
        IF(ITERM.EQ.1)THEN
          CALL RPGENV(0.,1.,-1.1,1.1,0,0)
        ELSE
          CALL PGENV(0.,1.,-1.1,1.1,0,0)
        END IF
        CALL PGLABEL('X axis','Y axis','Plot label')
      END DO
      NCOLOR=1

10    CONTINUE                        ! main loop: button handle

      CALL RPGBAND(0,0,0.,0.,XC,YC,CH)
      CALL IFBUTTON(XC,YC,NB)

      IF(NB.EQ.0)THEN

        WRITE(*,100)'Cursor at:'
        WRITE(*,*)XC,YC

      ELSEIF(NB.EQ.6)THEN

        CALL BUTTON(6,'EXIT',5)
        WRITE(*,100)'Press  to EXIT'
        READ(*,*)
        GOTO 20

      ELSEIF(NB.EQ.1)THEN             ! plot sin function

        CALL BUTTON(1,'sin',5)
        DO I=1,100
          XX(I)=REAL(I-1)/99.*2.*3.141593
          YY(I)=SIN(XX(I))
```

```
      XX(I)=XX(I)/(2.*3.141593)
    END DO
    DO ITERM=NTERM,1,-1
      CALL PGSLCT(IDN(ITERM))
      IF((NCOLOR.NE.1).AND.(LCOLOR(ITERM))) CALL PGSCI(NCOLOR)
      CALL PGLINE(100,XX,YY)
      IF((NCOLOR.NE.1).AND.(LCOLOR(ITERM))) CALL PGSCI(1)
    END DO
    CALL BUTTON(1,'sin',0)


  ELSEIF(NB.EQ.2)THEN              ! plot cos function

    CALL BUTTON(2,'cos',5)
    DO I=1,100
      XX(I)=REAL(I-1)/99.*2.*3.141593
      YY(I)=COS(XX(I))
      XX(I)=XX(I)/(2.*3.141593)
    END DO
    DO ITERM=NTERM,1,-1
      CALL PGSLCT(IDN(ITERM))
      IF((NCOLOR.NE.1).AND.(LCOLOR(ITERM))) CALL PGSCI(NCOLOR)
      CALL PGLINE(100,XX,YY)
      IF((NCOLOR.NE.1).AND.(LCOLOR(ITERM))) CALL PGSCI(1)
    END DO
    CALL BUTTON(2,'cos',0)

  ELSEIF(NB.EQ.3)THEN              ! clear plot

    CALL BUTTON(3,'clear',5)
    DO ITERM=NTERM,1,-1
      CALL PGSLCT(IDN(ITERM))
      CALL BUTTQBR(XV3,XV4,YV3,YV4)
      CALL RPGERASW(0.,1.,0.,YV3)
    END DO
    GOTO 5

  ELSEIF(NB.EQ.4)THEN              ! change color

    CALL BUTTON(4,'color',5)
    WRITE(*,100)'Current PGPLOT color is number: '
    WRITE(*,*)NCOLOR
    WRITE(*,100)'Enter new PGPLOT color number: '
    READ(*,*) NCOLOR
    CALL BUTTON(4,'color',0)
  END IF

  GOTO 10


20  CONTINUE                       ! end of main loop: button handle

  CALL PGEND
  STOP
```

```
100     FORMAT(A,$)
        END
```

The BUTTON library is available from http://www.ucm.es/info/Astrof/button/button.html along with installation instructions and a description of the library functions.

3.3 The `pgperl` package

The `pgperl` package is a dynamically loadable Perl module which interfaces to `Fortran` PGPLOT library. Perl provides a superset of the features of the useful UNIX utilities `awk` and `sed` and is the "Swiss-Army Chainsaw" of UNIX programming. Users of PONGO and SM packages will be familiar with this style of programming.

The simple example shown below, taken from the `pgperl` distribution, shows how the `Fortran` routines are interfaced into a simple Perl script, the output from this script is shown in Figure 3.

```
#!/usr/local/bin/perl

use PGPLOT;                                  # Load PGPLOT PERL module

print "\nTesting simple point plot...\n\n";
print "PGPLOT module version $PGPLOT::VERSION\n\n";

pgqinf("VERSION",$val,$len);                 # Query PGPLOT version number
print "PGPLOT $val library\n\n";

$dev = "?" unless defined $dev;              # "?" will prompt for device

pgbegin(0,$dev,1,1);                         # Open plot device

pgscf(2);                                    # Set character font
pgslw(4);                                    # Set line width
pgsch(1.6);                                  # Set character height

pgenv(0,10,-5,5,0,0);                        # Define data limits and plot axes

pglabel("X","Y","Data");                     # Print axis labels
pgsci(5);                                    # Change plot colour

@x=(); @y=(); $i=0;


while(<DATA>){
                                             # Read data in 2 columns
                                             # from file handle and
                                             # put in two perl arrays
   ($x[$i], $y[$i]) = split(' ');
   $i++;
}


pgpoint($i,\@x,\@y,17);                     # Plot points
```

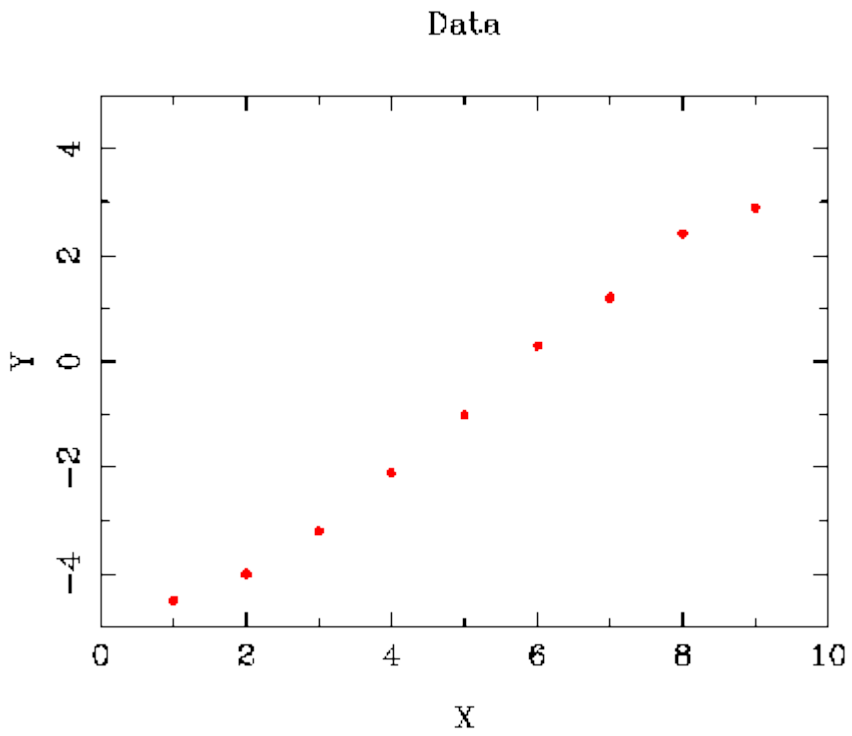


Figure 3: Output from the simple `pgperl` example script.

```
                                        # note how perl arrays are passed
pgend;                                  # Close plot

__DATA__
1 -4.5
2 -4
3  -3.2
4 -2.1
5 -1
6 0.3
7 1.2
8 2.4
9 2.9
```

For every PGPLOT `Fortran` function the `pgperl` module provides an equivalent Perl function with the same arguments. Thus the user of the module should refer to the PGPLOT manual to learn all about how to use `pgperl` and for the complete list of available functions. More information on `pgperl` can be found at `http://www.aao.gov.au/local/www/kgb/pgperl/`.

3.3.1 Argument mapping – simple numbers and arrays

Passing simple numbers and arrays to the PGPLOT subroutines via the `pgperl` module any `Fortran REAL`, `INTEGER` or `CHARACTER` scalar variable maps to a Perl scalar, since Perl doesn't care about the differences between strings, integers or reals. Therefore to draw a line to point `(42,$x)`:

```
pgdraw(42,$x);
```

To plot ten points of data held in in Perl arrays `@x` and `@y` with plot symbol 17 the Perl arrays are passed (by reference) to the PGPLOT module as follows:

```
pgpoint(10, \@x, \@y, 17);
```

Label the axes:

```
pglabel("X axis", "Data units", $label);
```

Draw a single point, note that when $N = 1$, `pgpoint()` can take a scalar argument rather than an array:

```
pgpoint(1, $x, $y, 17);
```

3.3.2 Argument mapping – images and 2D arrays

Many of the PGPLOT calls (*e.g.* `pggray`) take 2D arrays as arguments. Several methods to access these subroutines are provided by `pgperl`:

(1) Pass a reference to a 2D array:

```
# Create 2D array
$x=[];

for($i=0; $i<128; $i++) {
  for($j=0; $j<128; $j++) {
    $$x[$i][$j] = sqrt($i*$j);
  }
}
pggray( $x, 128, 128, ...);
```

(2) Pass a reference to a 1D array :

```
@x=();
for($i=0; $i<128; $i++) {
  for($j=0; $j<128; $j++) {
     $x[$i][$j] = sqrt($i*$j);
  }
}
pggray( \@x, 128, 128, ...);
```

Here `@x` is a 1D array of 1D arrays. Alternatively `@x` could be a flat 1D array with 128x128 elements, 2D routines such as `pggray()` are programmed to do the right thing as long as the number of elements match.

(3) If your image data is packed in raw binary form into a character string you can simply pass the raw string:

```
read(IMG, $img, 32768);
pggray($img, $xsize, $ysize, ...);
```

Here the `read()` function reads the binary data from a file and the `pggray()` function displays it as a grey-scale image. This saves unpacking the image data in to a potentially very large 2D perl array. However the types must match. The string must be packed as a "`f*`" for example to use `pggray`. This is intended as a short-cut for sophisticated users. Even more sophisticated users will want to download the PDL module which provides a wealth of functions for manipulating binary data.

3.3.3 Argument mapping – function names

Some PGPLOT functions (*e.g.* `pgfunx`) take functions as callback arguments. In Perl simply pass a subroutine reference or a name, for example:

```
# Anonymous code reference:
pgfunx(sub{ sqrt($_[0]) },  500, 0, 10, 0);
# Pass by ref:
sub foo {
  my $x=shift;
  return sin(4*$x);
}
pgfuny(\&foo, 360, 0, 2*$pi, 0);
# Pass by name:
pgfuny("foo", 360, 0, 2*$pi, 0);
```

3.3.4 Argument mapping – general handling of binary data

In addition to the implicit rules mentioned above PGPLOT now provides a scheme for explictly handling binary data in all routines.

If your scalar variable (*e.g.* `\$x`) holds binary data (*i.e.* 'packed') then simply pass PGPLOT a reference to it (*e.g.* `\$x`). Thus one can say:

```
read(MYDATA, $wavelens, $n*4);
read(MYDATA, $spectrum, $n*4);
pgline($n, \$wavelens, \$spectrum);
```

This is very efficient as we can be sure the data never gets copied and will always be interpreted as binary.

3.4 Python PGPLOT

Python is one of the new breed of object-oriented programming languages. It is commonly used both for scripting and as a stand alone rapid development language. One of the properties of the language is that it provides facilities for the easy integration of external services. It should therefore come as no surprise that there are currently several different Python interfaces to the PGPLOT subroutine libraries. Due to the nature of the language, being a rapid development tool, it should also come as no surprise that the documentation is a bit on the patchy side.

The most well documented seems to be an interface to PLplot. While based on PGPLOT, and having a similar API, PLplot is *not* derived from the PGPLOT source and care must be

taken when using it if you are used to PGPLOT. More information on PLplot can be found in Section 3.10.

Both of the other interfaces require the installation of the NumPy libraries. NumPy is a collection of C extension modules to the Python programming language which add multi-dimensional array objects. These new objects give Python the number crunching power of numeric languages like Matlab and IDL while maintaining all of the advantages of the general-purpose programming language Python. If you are running Linux it is possible that NumPy may already be installed, else or otherwise, it can be found at `http://andrich.net/python/`.

One interface also requires you to install SWIG. The Simplified Wrapper and Interface Generator (SWIG) is is a software development tool that connects programs written in C, C++, and Objective-C with a variety of high-level programming languages. SWIG is primarily used with common scripting languages such as Perl, Python, and Tcl/Tk but has been extended to include languages such as Java.

More details of the interfaces available, including some basic usage and installation instructions, can be found on the UBC Python Page at `http://www.geog.ubc.ca/~phil/ubc_python.html`.

3.5 GLISH PGPLOT

For users of aips++ a PGPLOT binding for GLISH has been developed. While the PGPLOT library itself has a large number of device drivers, only the Tk and PostScript drivers are available from GLISH. More information on the PGPLOT bindings in GLISH can be found at `http://aips2.nrao.edu/released/docs/reference/Glish/node97.html`.

3.6 `ptcl` Tk/Tcl and PGPLOT

`ptcl` registers PGPLOT functions as `tcl` commands. It allows you to create plots from the command line or from scripts. If the `tk` extensions are installed it is simple to create graphical user interfaces (GUIs) allowing you to directly interact with the plots. More information on `ptcl` can be found at `http://www.InfoMagic.com/~nme2/ptcl/ptcl.html`.

3.7 Starlink/Native PGPLOT

Unbeknown to some, PGPLOT commonly comes in two flavours on Starlink supported machines. The original or "Native" version which uses the low level graphics package GRPCKG, which was also written at Caltech, and a version developed by Starlink, in collaboration with Dr Pearson, which uses RAL GKS. The two versions have identical subroutine interfaces and in most cases, applications can be moved from one version to the other simply by re-linking.

Starlink currently supports both the native version and the Starlink versions. Most packages available in the Starlink Software Collection (USSC) currently are linked against the Starlink version. Work is ongoing to port these applications to the Native version. More information can be found in SUN/15 which describes the use of PGPLOT on Starlink systems. Use of the Starlink/GKS version is deprecated. The Starlink distributed version of Native PGPLOT has an additional device, `/gwm`, that allows plotting in GWM Windows.

To compile a program and link it to the native version of PGPLOT you should use the following command line:

```
% f77 prog.f -L/star/lib `pgplot_link`
```

While to use the Starlink version you should use:

```
% f77 prog.f -L/star/lib `pgp_link`
```

Or to link the code in an ADAM application:

```
% alink prog.f -L/star/lib `pgp_link_adam`
```

More detailed discussion of the differences between the two versions can be found in SUN/15.

3.8 Graphical Kernel System (GKS)

The Graphical Kernel System (GKS) is a device independent low level graphics system designed to be the kernel of a wide variety of higher-level graphics systems. It is very comprehensive but does not itself set out to provide the most convenient or user-friendly interface for all applications. For high level graphics you are recommended to use the PGPLOT library, while for low-level graphics you may prefer to use SGS rather than play with GKS directly.

More information on the GKS system can be found in the Starlink GKS document SUN/83. While detailed API information can be found in the RAL GKS User Guide and the RAL GKS Reference Manual (obtainable from your Starlink site manager or Starlink user support at RAL).

3.8.1 Enquiring about the display

With the current movement away from pseudo- to true-colour XWindow displays, a common problem when writing software is trying to find what sort of X display the person running your package has available. A good indicator as to the type of X display available is whether the colour table is writable. If it is, then your user is probably sitting in front of a pseudo-colour display. If it's not, then it is more likely that the display is True Colour (see Section 11).

GKS provides a function to enquire whether the colour table (amongst other attributes) is writable, formally called "Inquire Dynamic Modification of Workstation Attributes", *i.e.*

```
      CALL GQDWKA( WTYPE, IERR, PLBUN, PMBUN, TXBUN, FABUN,
     :             PAREP, COLREP, WKTR)
```

Where the variables are defined as:

`WTYPE = _INTEGER` (Read)
: workstation type

`IERR = _INTEGER` (Returned)
: error indicator

`PLBUN = _INTEGER` (Returned)
: polyline bundle representation changeable

PMBUN = _INTEGER (Returned)
polymarker bundle representation changeable

TXBUN = _INTEGER (Returned)
text bundle representation changeable

FABUN = _INTEGER (Returned)
fill area bundle representation changeable

PAREP = _INTEGER (Returned)
pattern representation changeable

COLREP = _INTEGER (Returned)
colour representation changeable

WKTR = _INTEGER (Returned)
workstation transformation changeable

If the colour table is writable, COLREP will be returned as 1. More information on other useful GKS functions can be found in the RAL GKS Reference Manual, which can be found online at http://www.itd.clrc.ac.uk/Publications/RAL-GKS/gks_cat.html.

3.8.2 Compiling and Linking GKS programs

Before compiling a program that uses the GKS include file GKS_PAR you must first execute the command:

```
$ gks_dev
```

Programs are linked with GKS by:

```
$ ld objmodule -L/star/lib ‘gks_link‘
```

3.9 Simple Graphics System (SGS)

The Simple Graphics System (SGS) is a low-level graphics subroutine library sitting above the GKS package allowing easier access to GKS features. Full details of the SGS library package can be found in SUN85.

3.10 PLplot Library

PLplot is a library of C functions that are useful for making scientific plots from a program written in C, C++, or Fortran. The PLplot library can be used to create standard x-y plots, semilog plots, log-log plots, contour plots, 3D plots, mesh plots, bar charts and pie charts. Multiple graphs (of the same or different sizes) may be placed on a single page with multiple lines in each graph. Different line styles, widths and colors are supported. A virtually infinite number of distinct area fill patterns may be used. There are almost 1000 characters in the extended character set. This includes four different fonts, the Greek alphabet and a host of mathematical, musical, and other symbols. The fonts can be scaled to any desired size. A variety of output devices are supported. More information on PLplot can be found at http://emma.la.asu.edu/plplot/.

3.10.1 PLplot and 3D Surface Plots

One important feature available in PLplot, which is not (trivially) available in PGPLOT is the ability to represent a single-valued function of two variables as a surface.

As usual, we would like to refer to a three dimensional point (X, Y, Z) in terms of some meaningful user-specified coordinate system. These are called three-dimensional world coordinates. We need to specify the ranges of these coordinates, so that the entire surface is contained within the cuboid defined by $xmin < x < xmax$, $ymin < y < ymax$ and $zmin < z < zmax$. Typically, we shall want to view the surface from a variety of angles, and to facilitate this, a two-stage mapping of the enclosing cuboid is performed. Firstly, it is mapped into another cuboid called the normalized box whose size must also be specified by the user, and secondly this normalized box is viewed from a particular azimuth and elevation so that it can be projected onto the two-dimensional window.

This two-stage transformation process allows considerable flexibility in specifying how the surface is depicted. The lengths of the sides of the normalized box are independent of the world coordinate ranges of each of the variables, making it possible to use "reasonable" viewing angles even if the ranges of the world coordinates on the axes are very different. The size of the normalized box is determined essentially by the size of the two-dimensional window into which it is to be mapped. The normalized box is centered about the origin in the x and y directions, but rests on the plane $z = 0$. It is viewed by an observer located at altitude, alt, and azimuth, az, where both angles are measured in degrees. The altitude should be restricted to the range zero to ninety degrees for proper operation, and represents the viewing angle above the xy plane. The azimuth is defined so that when $az = 0$, the observer sees the xz plane face on, and as the angle is increased, the observer moves clockwise around the box as viewed from above the xy plane. The azimuth can take on any value.

The routine `PLWIND` or `PLENV` (equivalent to the PGPLOT `PGENV` routine) is used in the usual way to establish the size of the two-dimensional window.

```
XMIN2D = -2.5;
XMAX2D =  2.5;
YMIN2D = -2.5;
YMAX2D =  4.0;
PLENV(XMIN2D, XMAX2D, YMIN2D, YMAX2D, 0, -2);
```

The routine `PLW3D` must then be called to establish the range of the three dimensional world coordinates, the size of the normalized box and the viewing angles. After calling `PLW3D`, the actual surface is drawn by a call to `PLOT3D`.

```
BASEX = 2.0;
BASEY = 4.0;
HEIGHT = 3.0;
XMIN = -10.0;
XMAX = 10.0;
YMIN = -3.0;
YMAX = 7.0;
ZMIN = 0.0;
ZMAX = 8.0;
ALT = 45.0;
AZ = 30.0;
```

```
SIDE = 1;
PLW3D(BASEX, BASEY, HEIGHT, XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX, ALT, AZ);
PLOT3D(X, Y, Z, NX, NY, OPT, SIDE);
```

The values of the function are stored in a two-dimensional array $z[\,][\,]$ where the array element $z[i][j]$ contains the value of the function at the point x_i, y_j. (The two-dimensional array z is a vectored array instead of a fixed size array. z points to an array of pointers which each point to a row of the matrix.) Note that the values of the independent variables x_i and y_j do not need to be equally spaced, but they must lie on a rectangular grid. Thus two further arrays $x[nx]$ and $y[ny]$ are required as arguments to plot3d to specify the values of the independent variables. The values in the arrays x and y must be strictly increasing with the index. The argument opt specifies how the surface is outlined. If $opt = 1$, a line is drawn representing z as a function of x for each value of y, if $opt = 2$, a line is drawn representing z as a function of y for each value of x, and if $opt = 3$, a net of lines is drawn. The first two options may be preferable if one of the independent variables is to be regarded as a parameter, whilst the third is better for getting an overall picture of the surface. If side is equal to one then sides are drawn on the figure so that the graph doesn't appear to float.

The routine `PLMESH` is similar to `PLOT3D`, except that it is used for drawing mesh plots. Mesh plots allow you to see both the top and bottom sides of a surface mesh, while 3D plots allow you to see the top side only (like looking at a solid object). The side option is not available with `PLMESH`.

Labelling a three-dimensional or mesh plot is somewhat more complicated than a two dimensional plot due to the need for skewing the characters in the label so that they are parallel to the coordinate axes. The routine `PLBOX3` thus combines the functions of box drawing and labelling.

3.11 The `libjpeg` Library

Let us get it clear right from the start, JPEG is not an image format, instead the ANSI JPEG specification lays down a definition for a family of compression algorithms. In fact the JPEG specification is commonly used in two file formats JFIF and TIFF. The JFIF format is a simple format used for applications that just need to store image data, this is the format that is commonly referred to as being "JPEG" and files in this format usually have `.jpg` or `.jpeg` endings. The second, more complex, TIFF file format is used by applications that need to store extra data about images (*e.g.* colour correction curves). However TIFF, while more flexible, are far less portable than JFIF since different applications implement different subsets of TIFF specification. It should be noted that the official standard for JPEG image compression is not available on-line, you have to order a paper copy from ANSI (or ISO).

The forthcoming JPEG Part 3 standard defines a file format called SPIFF. This format should be backwards compatible with the JFIF image standard, although it has some technical advantages. However its major advantage is that it is an official ANSI standard, where JFIF and TIFF are not. At this time it is unclear as to whether SPIFF will replace JFIF, or whether JFIF will continue to be widely used with the new SPIFF standard being ignored by the rest of the world.

The JPEG standard is optimised for "real-world" images, cartoons and other non-realistic images are not handled well, since JPEG is a lossy algorithm. This means that the output image is not identical to the image, you trade off output image quality against a smaller file size, by adjusting a compression parameter (how lossy the image will be) on image generation.

The Independent JPEG Group (IJG) has a freely redistributable implementation of the JPEG (JFIF) image compression/decompression algorithms. The distributed programs provide conversion between JPEG format and image files formats such as PPM/PGM, GIF, BMP, and Targa. The core programs used to do this is `cjpeg` to compress an image file into JPEG format and `djpeg` to decompress a JPEG file back into a conventional format. The core compression and decompression library, `libjpeg.so`, which is written in C can easily be reused in your own programs programs.

The code is available for both commercial and non-commercial use, and the latest version of the code can be obtained via anonymous FTP from `ftp://ftp.uu.net/graphics/jpeg/`. Detailed documentation on how to code using the library API is provided along with the distribution (see the `libjpeg.doc` file).

If you are using a Linux system it is likely that the library, and associated applications, are already installed. RedHat 6.0 ships with `libjpeg.so` shared object library as part of the standard distribution. The `libjpeg` library is also distributed as part of the Starlink Base Set, if you are using a Starlink supported machine you can link your program to the Starlink distributed version.

3.12 The `giflib` Library

GIF files use Lempel-Ziv-Welsh (LZW) compression algorithm to encode the image data to save space, however there is a great deal of controversy over the GIF legal position (see Section 15) due to the Unisys patent issue.

Eric S. Raymond, the maintainer of `giflib` has this to say about Unisys's licensing: "Due to Unisys's increasingly aggressive interpretation of its patent claims on the LZW compression format, I can no longer recommend the use of the `giflib` library or utilities. `giflib` may be withdrawn in the near future".

If you need to deal with GIF images it is recommended that you use the the `libungif` library.

3.13 The `libungif` Library

The way round the entire legal mess surrounding the GIF image standard is simply not to use the LZW compression algorithm. The `libungif` library follows this approach and is designed to handle uncompressed GIFs. These are image files that, while not using LZW compression, are still recognizable as GIF files by decoders which expect normal (compressed) GIFs. The obvious problem is that the uncompressed GIF images will be larger than those encoded using the LZW algorithm. This library speaks both GIF87a and GIF89.

The latest version of the `libungif` library can be obtained from the anonymous FTP archive at `ftp://prtr-13.ucsc.edu/pub/libungif/`. Extensive documentation on how to code using the library API is included with the distribution. However if you are using a Linux system it is likely that the library, and associated applications, are already installed. RedHat 6.0 for instance ships with both `libgif.so` and `libungif.so` shared object libraries as part of the standard distribution.

3.14 The `angif` Library

ANGIF is a C library to generate GIF format output. It can generate animated or, a bit of a standard breaker here, true colour (24bpp) GIFs. Due to the legal problems surrounding the format ANGIF is LZW free. Command line test programs are included with the distribution.

It should be noted that ANGIF is in pre-beta release, the only documentation available is the source code comments. Although there doesn't appear to be a home page for the library yet, it can be downloaded via HTTP from `http://phil.ipal.org/freeware/angif/`.

3.15 The PNG Format

The Portable Network Graphics (PNG) format was designed to replace the older and simpler GIF format and, to some extent, the much more complex TIFF format.

PNG format several major advantages over GIF. Firstly it uses alpha channels, allowing you to have variable transparency images. Unlike GIF, which implements a simple binary transparency (either a pixel is transparent or opaque) PNG specifies 254 levels of partial transparency. Instead of storing three bytes for each pixel for red, green and blue (RGB), four are now stored, these being red, green, blue and alpha (RGBA). PNG supports both true colour, greyscale and palette-based (pseudo) colour images, unlike GIF which supports only pseudo colour images. All three types of PNG image support alpha channels, although the size of true colour PNG images effectively rules them out for use on the web. Additionally, the format makes use of gamma correction, allowing cross-platform control of image brightness. Finally, the PNG format specifies two-dimensional interlacing (progressive display) rather than the one-dimensional scheme used by GIF images.

PNG also compresses better than GIF in almost every case, but the difference is generally only around 5 to 25 per cent. Additionally, and quite importantly, PNG is free of any legal entanglement.

For those of you wanting to implement programs to handle PNG images, the official PNG library `libpng` is available via anonymous FTP from `ftp://swrinde.nde.swri.edu/pub/png/src/`. This library requires `zlib`, a general purpose lossless compression library. A copy of the library can be found at the same FTP site, but the latest version and more information about the library can be found at `ftp://ftp.freesoftware.com/pub/infozip/zlib/index.html`

More information on the PNG format, programming resources and supporting applications can be found online at `http://www.libpng.org/pub/png/`.

3.16 The MNG Format

The Multiple-image Network Graphics (MNG) format has been implemented by the same people that brought you PNG, and it therefore shares the same modular philosophy. The idea behind the format is to provide a home for all of the multi-image capabilities that have no place in PNG. While it has fairly extensive animation and image-manipulation capabilities, there is no serious expectation that it will ever integrate audio or video. In other words this format it intended to replace multi-image GIF animations.

Though the MNG specification itself has not yet been promoted to release status, as of 11 May 1999 it was officially frozen by a vote of the MNG developers. Although relatively mature, MNG

is still a draft proposal. There is therefore no general use MNG reference library (along the same lines as `libjpeg` for example). However, there are already several applications with partial MNG support, the main UNIX application being ImageMagick (see Section 7.1).

3.17 The Python Imaging Library

The Python Imaging Library (PIL) adds an image object to your Python interpreter. You can load image objects from a variety of file formats, including BMP, EPS, GIF, JPEG, PNG, PPM, TIFF and XBM, and apply a rich set of image operations to them. See the feature sheet at `http://www.python.org/sigs/image-sig/Imaging.html` for more details.

3.18 The `gd` Library

`gd` is a C graphics library that allows you to quickly draw images with lines, arcs, text, multiple colors, cut and paste from other images, and flood fills, and write out the result as a PNG file. More information can be found at `http://www.boutell.com/gd/gd.html`.

A simple example of the `gd` library in use, taken from the documentation, is shown below:

```
/* Bring in gd library functions */
#include "gd.h"

/* Bring in standard I/O so we can output the PNG to a file */
#include <stdio.h>

int main() {
        /* Declare the image */
        gdImagePtr im;
        /* Declare an output file */
        FILE *out;
        /* Declare color indexes */
        int black;
        int white;

        /* Allocate the image: 64 pixels across by 64 pixels tall */
        im = gdImageCreate(64, 64);

        /* Allocate the color black (red, green and blue all minimum).
                Since this is the first color in a new image, it will
                be the background color. */
        black = gdImageColorAllocate(im, 0, 0, 0);

        /* Allocate the color white (red, green and blue all maximum). */
        white = gdImageColorAllocate(im, 255, 255, 255);

        /* Draw a line from the upper left to the lower right,
                using white color index. */
        gdImageLine(im, 0, 0, 63, 63, white);

        /* Open a file for writing. "wb" means "write binary", important
                under MSDOS, harmless under Unix. */
        out = fopen("test.png", "wb");
```

```
            /* Output the image to the disk file. */
            gdImagePng(im, out);

            /* Close the file. */
            fclose(out);

            /* Destroy the image in memory. */
            gdImageDestroy(im);
    }
```

When run, this program creates an image, allocates two colours (the first colour allocated becomes the background colour) and draws a diagonal line (note that 0, 0 is the upper left corner) before writing the image to a PNG file.

3.18.1 `gd` from other languages

The `gd` library can also be accessed from languages other than C. There is an API for both the Perl, see `http://stein.cshl.org/WWW/software/GD/GD.html`, and Tcl scripting languages, see `http://www.tcltk.com/ftp/ellson/`.

4 Plotting Packages

4.1 QDP

QDP is an interactive graphics plotting package that began life in the late seventies on a PDP 11/70 at the Goddard Space Flight Center, it doesn't seem that likely that any of code has survived from that first version.

QDP is now an interactive front end to Tim Pearson's PGPLOT package, and is distributed by HEASARC as a stand alone segment of their XANADU X-ray analysis package. Most Starlink sites should have XANADU, and hence QDP, already installed. Ask your site manager, or any random X-ray astronomer if you have one to hand, for site specific instructions on how to setup the package for use. The QDP manual is available on the web at `http://heasarc.gsfc.nasa.gov/docs/software/ftools/others/qdp/node3.html`.

4.1.1 QDP Basic Stuff

QDP takes standard ASCII text files as input, there need to be at least two columns. The first column is taken to X values for the data points, while the second column is taken to be Y values. If there are more than two columns then QDP (by default) assumes that further columns are additional Y values data points (with the same X values). Comments can be included either as separate lines, or at the end of a line of data using the "`!`" character to signify the start, and the end of a line to signify the end, of a comment. Data values may be separated by spaces, a comma, or tabs. However, each row should contain the same number of columns, although if some data are missing you can enter the word `NO` instead of an actual number. QDP regards the `NO` to mean no data available. An example data file is shown below:

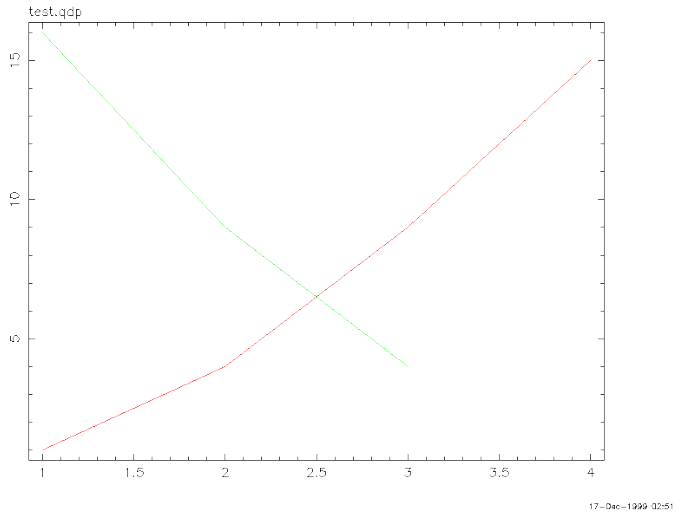


Figure 4: The default appearance of the `test.qdp` data file when plotted using QDP.

```
! A Comment Line
   1   1  16
   2   4   9 ! Another Comment
   3   9   4
   4  15   NO
```

By default QDP will look for files with the `.qdp` extension, if your file has a `.qdp` extension you do not need to pass this to QDP as it will automatically assume its presence. Although the program is perfectly happy to read files with any extension, it will in fact refuse to read files that do **not** have a filename extension. This sounds complicated, it is not really, for instance if we create a file called `test.qdp`, with the above data, you'd plot it by typing:

```
% qdp
QDP file name: test
 To produce plot, please enter
PGPLOT file/type: /xw
PLT>
```

There are two important features in this short exchange that you should take notice of, firstly, since QDP sits on top of native PGPLOT any device that you can access with PGPLOT you can use with QDP (and is specified in the same way you'd specify native PGPLOT devices, see the PGPLOT Manual. Here we specified that we wished to use X Windows to display our plot so that we can interact with it later. Obviously non-interactive devices like PostScript files are exactly that, you can't interact with your plot once it is created. Which leads to the second important point, the `PLT>` prompt, QDP has processed your data file and it is now waiting for commands. The plot from this command is shown in Figure 4.

Since the file contained three columns of numbers, the default mode assumes there are three plot groups. The first plot group determines the X coordinate. The next two columns are plotted as two lines. The name of the QDP file appears in the top left of the plot and your userid, current date, and time appear in the bottom right of the plot. All this can now be changed interactively

from the PLT prompt. A useful command at this point is help which provides access to the online help for all QDP commands.

Plots can be rescaled using the rescale command and the axes labelled using the label command, which also controls other labelling like the filename at the top left of the plot. The default is to draw the plots using a line, this can be turned off using line off which will result in each data point being plotted as a dot. This is not that desirable, and the data point marker can be changed using the mark command. A list of markers, and associated numbers can be found by typing mark ? (this will overwrite the current plot on the graphics display, it can be gotten back simply by typing plot). More information on all these commands, and all the others, can be found in the online help from the application which is excellent.

It should be noted that it is important to type plot after making changes to a plot, otherwise the current display will **not** be changed, and that most QDP commands can be abbreviated (for instance typing ma ? and mark ? will produce the same result).

4.1.2 Plot devices and PostScript output

QDP generates hard copy in the same way it writes to any other device, therefore you question shouldn't be "How do I get hardcopy of my plot?", but instead "How do I change the plot device I'm currently using?". The answer is to issue a dev command to change your current plotting device, for instance:

```
PLT> dev /ps
PLT> plot
PLT> dev /xw
```

would change our current plotting device to a landscape PostScript file and re-plot our current plot. The final dev command to (yet again) change out current plot device is important since this closes the PostScript file, without which the file would be empty. By default QDP writes PostScript output to a file called pgplot.ps.

Despite the fact that PostScript output is "just another device" there is a specific command to deal with this common case of device switching, the hard command. You can find out your default hard copy device by:

```
PLT> hard ?
 Current hardcopy device is: /PS
PLT>
```

which tells us we're currently going to generate a PostScript file as our default hardcopy output. Typing hard on its own will result in the current plot being written to the default pgplot.ps output file. It is possible to override the default hardcopy device, for instance the hard /vps command would generate a vertical (portrait) mode PostScript file no matter what the current default hardcopy output device. It should be noted that the default hardcopy device can be set using the PLT_HARDCOPY environment variable.

By default QDP uses the Simple font since this is fastest, for hard copy output you may wish to change this to something more professional looking like the Roman font. Additionally, since most journals photo-reduce the size of supplied figures before printing it may be advisable to increase the size of the characters on the labels, and also to increase the line width (which by default is one pixel). For instance:

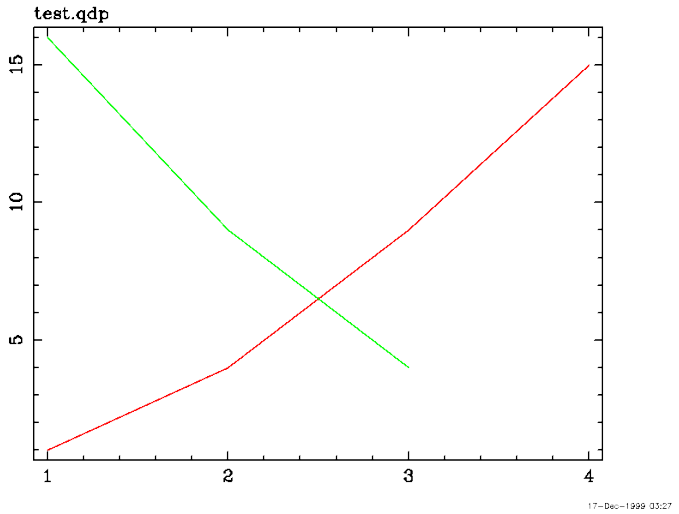


Figure 5: The appearance of the `test.qdp` data file when plotted in QDP using a Roman font with `csize 1.3` and `lwidth 7`.

```
PLT> font Roman
PLT> csize 1.3
PLT> lwidth 7
PLT> hard
```

would set the current font to `Roman`, increase the character size by a factor of 1.3 and set the current line width to 7 pixels before generating a hardcopy of the current plot. Our `test.qdp` data plotted with these options is shown in Figure 5 (compare with the same figure plotted with the default options, shown in Figure 4).

4.1.3 Error Bars

How do we go about telling QDP that the one of the columns in a data file is not data, but instead is a list of errors on our data values? This is done from the QDP file, by putting the `READ Serr` QDP command at top. This tells QDP that the data has *symmetric* errors. For instance:

```
READ Serr 1 2
1.0  0.25   1.24  0.5
1.5  0.25   1.86  0.5
2.0  0.25   3.76  0.5
4.0  1.75  16.43  4.8
7.0  1.25  49.06  0.5
```

would tell QDP that the third and fourth columns of numbers in our data file are the X and Y errors respectively. Confused? Don't worry, this confuses many of people to being with, let us take a step back. Basically when it thinks about columns QDP doesn't count columns that contain errors, so that to QDP there is not really four column in our file. Instead there are only two, a column containing X data and the associated errors, and a column containing the Y data and the associated errors. Perhaps this will make more sense if we separate the columns QDP perceives using commas. Hence:

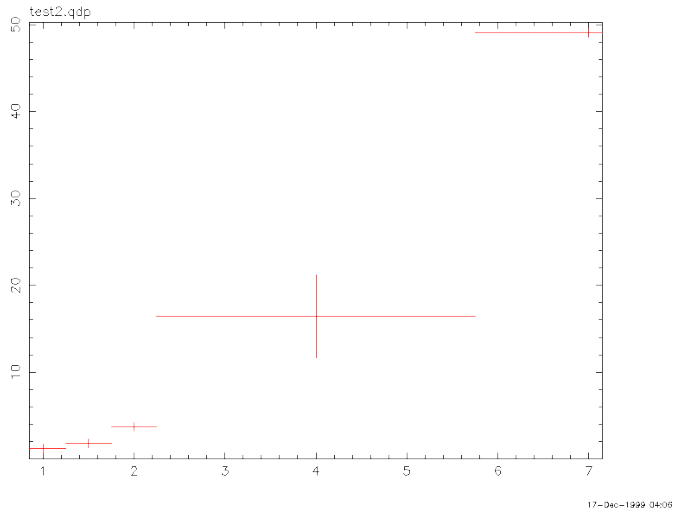


Figure 6: Our second test data set plotted using QDP with the default options.

```
READ Serr 1 2
1.0  0.25 ,  1.24  0.5
1.5  0.25 ,  1.86  0.5
2.0  0.25 ,  3.76  0.5
4.0  1.75 , 16.43  4.8
7.0  1.25 , 49.06  0.5
```

Does it make more sense now? The `READ Serr` command basically tells QDP that the the two lists of numbers should each have two real columns (one for data, one for errors). You can see what QDP makes of this file in Figure 6.

You can also tell QDP to use two-sided errors using the `READ Terr` command. It takes three real columns to specify a two-sided error. The first column is the central value, the second (which must be positive) specifies the upper error the third column (which must be negative or zero) specifies the lower error. For instance the file:

```
READ Serr 1
READ Terr 2
 1. .1    2. +.1 -.2
```

would plot a point at $(1 \pm 0.1, 2^{+0.1}_{-0.2})$

4.1.4 That "Date and Time" Thing

A commonly asked question by QDP novices is how to turn off the display of the date and time in the bottom right hand corner. This is done simply through use of the `time` command: `time off` will suppress printing the current date and time, while `time on` will restore this default behaviour.

4.1.5 Fitting using QDP

QDP has some basic fitting capabilities, for instance to fit our second lot of test data with a model consisting of a constant, linear and quadratic components (in other words fit it to the

equation $Y = X^2 + X + C$) we must first define a model, *e.g.*

```
PLT> model cons linr quad
  1 CO: VAL( -1.000    ), SIG(  0.000    ), PLO(  0.000    ), PHI(  0.000    )?

  2 LI: VAL(  1.000    ), SIG(  0.000    ), PLO(  0.000    ), PHI(  0.000    )?

  3 QU: VAL(  1.000    ), SIG(  0.000    ), PLO(  0.000    ), PHI(  0.000    )?

PLT>
```

Here we have simply accepted the default values (`VAL`) by hitting return, I could have instead entered likely estimates for our model parameters. This procedure becomes necessary for more complicated data and models. Having established which model we want to fit to the data, now type `fit`.

```
PLT> fit
 Fitting group   2,  from  0.850     to   7.15
 Fitting      5 points in a band of      5.
 -1.  1.  1.
 ( -3)   W-VAR=0.5758
 ( -4)   W-VAR=0.4192
 ( -5)   W-VAR=0.4189
  0.764678299 -0.742543042  1.09179008
PLT>
```

You'll see that we've got an acceptable fit, if you look at the display you'll see that the a line has been drawn showing the fit. The fit doesn't look very nice on screen however since QDP has illustrated its fit with a line having only the same number of points as our initial data values. If you want a smoother curve for a publication you should now type:

```
PLT> fit plot 200
PLT> plot
```

This will replot the fit with 200 points interpolated along the best fit model, which looks much nicer, as can be seen in Figure 7. Note that the fit parameters and variance, along with the number of data points fitted, is shown down the right hand side of the plot.

You can get a list of the different possible models available by typing `model`, *e.g.*

```
PLT> model
 Possible components are:
  CONS  LINR  QUAD  CUBI  X4    X5    POWR  SIN   GAUS
  EXP   AEXP  BURS  SBUR  PEAR  WIND  KING  LN    LORE
  DEMO  SPLN  AKIM
PLT>
```

While to get help on specific model components you must turn to the online help system, *e.g.*

```
PLT> help model cons
```

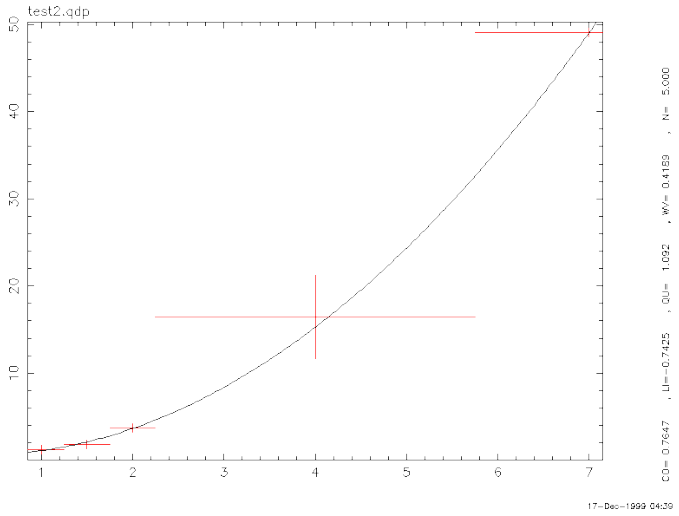


Figure 7: Our second test data set plotted using QDP with best fit model.

```
      HELP
           MOdel
                CONS

Select a model with a constant component:

FNY=FNY+CO.

help>
```

You'll note that the `CONS` model has, obviously, only one component while:

```
PLT> help model gaus
GAUS

     HELP
          MOdel
               GAUS

Select a model with a Gaussian component:

FNY=FNY+GN*EXP(-Z*Z/2.),

where Z=(X-GC)/GW and with integral SQRT(2*PI)*GN*GW.

help>
```

has three, `GN`, `GC` and `GW`.

It is possible to save the current model to disk using the `wmodel` command, *e.g.*

```
PLT> wmodel file
```

will create a `file.mod` file. To read this model back into QDP use the command `model @file`. If you do not enter a file name with the `wmodel` command, then the model is written to the screen.

The `uncertain` command can now used to estimate uncertainties in the parameter values, *e.g.*

```
PLT> uncertain 1
 Delta parm   Delta Chi^2
  -2.19          2.70
   2.19          2.70
 Parameter   1, Delta CHI^2=  2.700      -1.424       2.953
PLT>
```

would give an error estimate for the first parameter of our fit (`C0`). You'll see that this shows us that we really didn't need to add a constant value to our model, since our value 0.765 ± 2.188 is consistent with zero. More information on fitting models can be found in the QDP manual and online help.

4.1.6 QDP Files

You've already seen that you can instruct QDP to treat the columns in your `.qdp` file differently using the `READ Serr` command. QDP also lets you put commands that you would normally type at the `PLT` prompt into you `.qdp` file, for instance:

```
READ Serr 1 2
label X Time
label Y Distance
1.0  0.25  1.24  0.5
1.5  0.25  1.86  0.5
2.0  0.25  3.76  0.5
4.0  1.75 16.43  4.8
7.0  1.25 49.06  0.5
```

Would plot our second example data file with the X axis labelled "Time" and the Y axis labelled "Distance". You can in fact have QDP files that are entirely commands, these have the default extension `.pco` instead of the usual `.qdp`. For instance if we have a file `test.pco` which contains:

```
label X %1%
label Y %2%
label T %3%
```

and at the `PLT` prompt we typed:

```
PLT> @test Time Distance "A Test Plot"
```

we would execute the commands contained in the `test.pco` file, putting the label "Time" on the X axis, "Distance" on the Y axis and the string "A Test Plot" along the top of the graph.

4.1.7 COD and QDP models

You can use something called COD (COmponent Definition) to generate new functions that can be used as components in QDP models. COD can also be used interactively as a reverse polish calculator. More information on COD is available in the QDP manual and from COD's own online help, *e.g.*

```
% cod
 Type HELP for help.

COD> help

 cod

The COD (COmponent Definition) program is intended to fill two roles:

First, it can be used interactively as a reverse-Polish calculator
using all the functions described in the 'dictionary' section.

Second, it can be used to test COD programs as described in the 'files'
section.

In all cases, COD will accept several commands on a single line.


dictionary   files        forth        future       GEt          List
Newpar       Quit         RUn          Step

cod topic?
```

4.2 PONGO

PONGO is another interactive plotting package that, like QDP, is based on PGPLOT. The PONGO package can used from both the Starlink/ICL and IRAF/CL command languages and integrates with other Starlink software packages.

Using graphics between different applications packages is often difficult because once one package has finished plotting and completed execution all information concerning the contents of the plot is lost. PONGO makes use of the Starlink Applications Graphics Interface (AGI) libraries, see SUN/48, which allows different graphics applications to be used to display images, draw contours and annotate, for example, on the same plot without each application losing access to the plot dimensions. This means, for instance, that you can display an image using the KAPPA `display` application and then annotate it using PONGO.

Full details of the PONGO package can be found in SUN/137 which should be available from your Starlink site manager.

4.3 SM

Another interactive plotting package that may be found at your site is SM. Full details about SM can be found in the *SM User Guide* (MUD/159) and the *SM Tutorial* (MUD/160) which can be obtained from your Starlink site manager.

4.4 GNUplot

GNUplot is a command-driven interactive function plotting program that, for once, doesn't sit on top of PGPLOT. It can be used to plot both functions (*e.g.* sin) and data points in both 2 and 3D plots (*i.e.* contour plot, mesh, *etc.*). Despite its name GNUplot is not written/maintained by the FSF, nor is it released under the GPL.

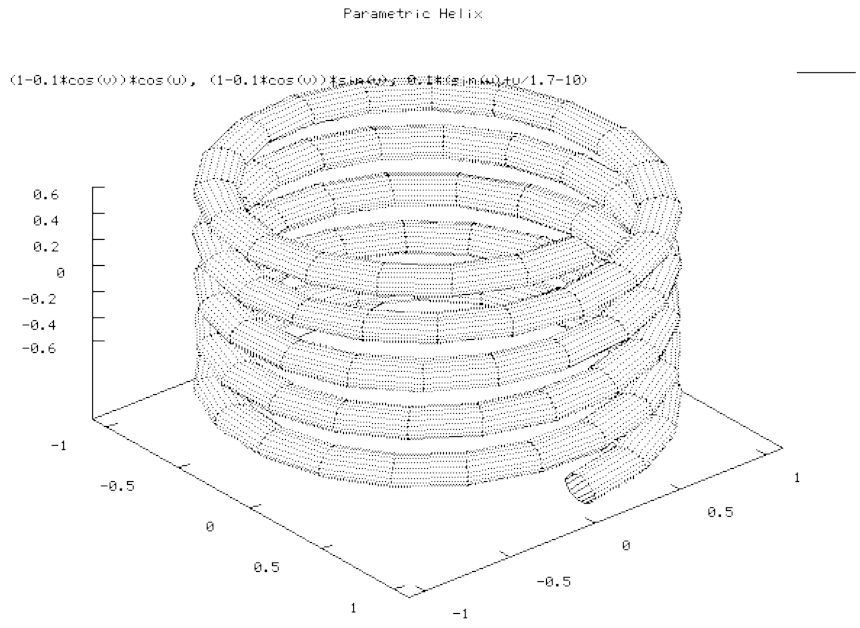


Figure 8: A plot of the function $(1 - 0.1\cos(v))\cos(u), (1 - 0.1\cos(v))\sin(u), 0.1(\sin(v) + u/1.7 - 10)$, a parametric helix, using GNUplot

An important thing to note about GNUplot is that its commands are case sensitive (like UNIX) and should be entered in lower case. Commands may extend over several input lines (for clarity) by ending each line but the last with a backslash (\). The backslash must be the **last** character on each line. Strings are indicated with quotes, although they may be either single or double quotation marks.

Since GNUplot accepts the name of a command file as a command line argument or as a redirection from standard input, *e.g.*

```
% gnuplot file.dat
```

or

```
% gnuplot < file.dat
```

the application is therefore very suitable for batch processing of data files. Extensive documentation on GNUplot is available on the web at `http://www.cs.dartmouth.edu/gnuplot_info.html`, and while the application can be used as at the simplest level as a plotting tool for 2D data, GNUplot has many powerful features including plotting of 3D parametric functions (see Figure 8) and data, and the ability to integrate functions and show the result graphically.

4.4.1 Co-ordinate systems

One of GNUplot's strengths its ability to plot functions, and data, in different co-ordinate systems. For instance Figure 9 shows a plot of the function $\cos(2x)$ in polar co-ordinates.

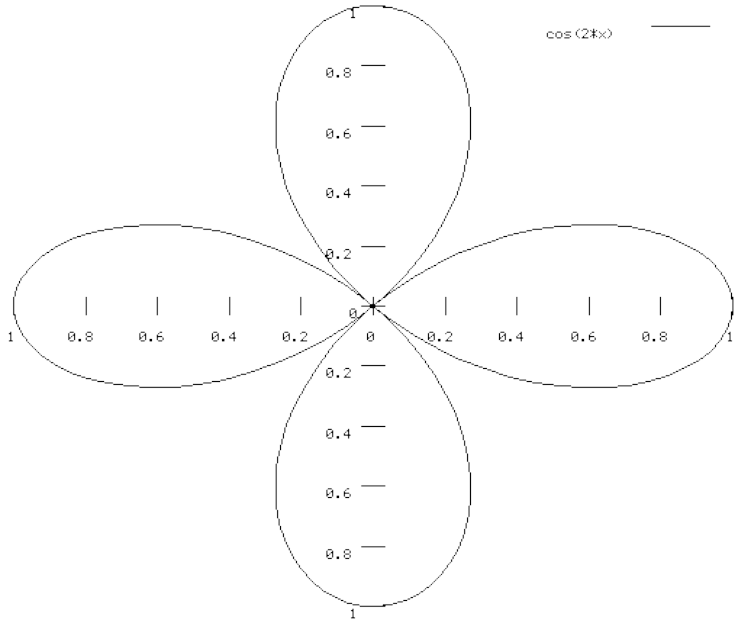


Figure 9: A plot of the function $\cos(2x)$ in polar co-ordinates using GNUplot.

4.4.2 Plotting 3D data

GNUplot has quite powerful 3D plotting abilities, using the `splot` command, reading either from an ASCII or binary file. ASCII data files should have the data stored in a format that looks something like:

```
<x0> <y0> <z0,0>
<x0> <y1> <z0,1>
<x0> <y2> <z0,2>

<x1> <y0> <z1,0>
<x1> <y1> <z1,1>
<x1> <y2> <z1,2>
  .    .     .
  .    .     .
  .    .     .
```

while binary files should have single precision floats stored as follows:

```
<ncols> <x0> <x1> <x2> ...
<y0> <z0,0> <z0,1> <z0,2> ...
<y1> <z1,0> <z1,1> <z1,2> ...
```

GNUplot will automatically determine if the data is in binary or ASCII format when it is read in using the `load` command.

An example of a 3D data file plotted using the `splot` command is shown in Figure 10. While another, very impressive, example from the GNUplot manual is shown in Figure 11.

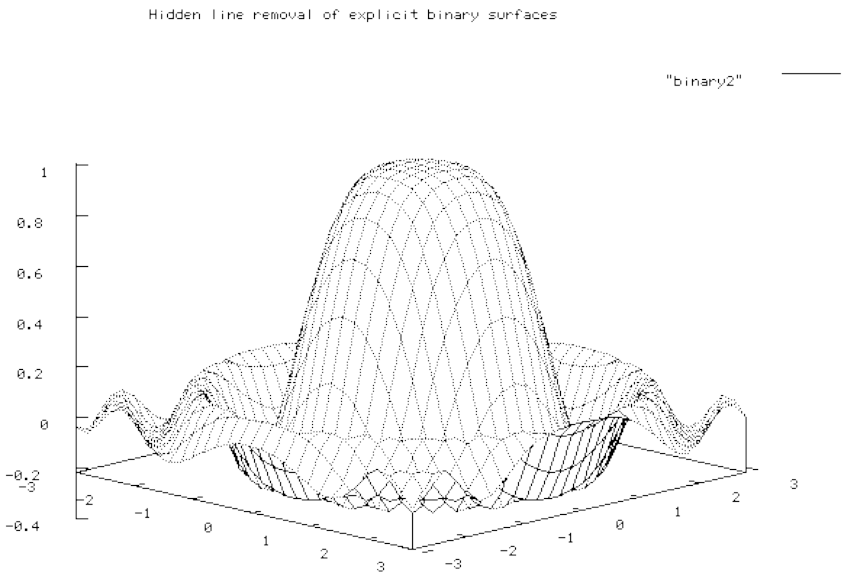


Figure 10: A 3D plot using GNUplot.

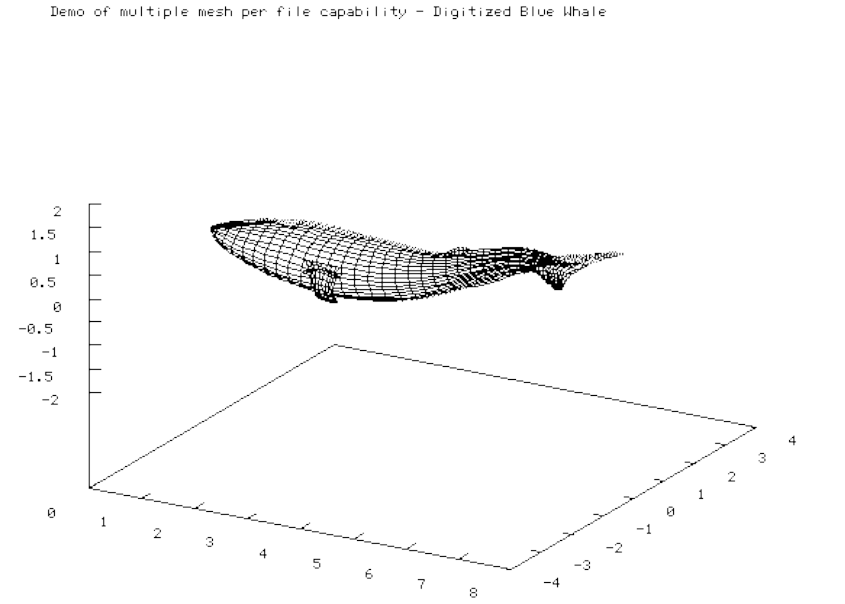


Figure 11: A 3D plot of a digitised blue whale using GNUplot.

5 Image Display

5.1 KAPPA

The KAPPA `display` program displays a 1- or 2-dimensional NDF in (by default) a GWM X Windows window. This GWM window can then be accessed by other Starlink applications (including other KAPPA applications) and the image manipulated. For instance, before carrying out photometry using the PHOTOM package you would display your CCD image using the KAPPA `display` command.

More information about KAPPA, and the `display` application, can be found in SUN/95.

5.2 SAOimage

SAOimage is a utility for displaying astronomical images in the X Window environment. It was written at the Smithsonian Astrophysical Observatory by Mike Van Hilst in 1990 and is now maintained by Doug Mink. Image files (including NDF) can be read directly, or image data may be passed through a named pipe from IRAF display tasks.

SAOimage provides a large selection of options for zooming, panning, scaling, coloring, pixel readback, display blinking, and region specification.

The SAOimage desktop includes, a main image display window, a button menu panel, a display magnifier, a pan and zoom reference image, and a color bar. A color table graph window can be brought up by clicking on the color bar at the bottom of the SAOimage desktop.

A quick introduction to SAOimage can be found in SUN/166. Further information can then be found in the *SAOimage User Manual* (MUD/140) which is distributed in LaTeX form with the SAOimage source code in the 'doc' subdirectory. Further information is also available on the web on the SAOimage Home Page at `http://tdc-www.harvard.edu/software/saoimage.html`.

5.2.1 Printing in SAOimage

One of the most commonly asked questions when dealing with image display in SAOimage is how to change the default printer. Under `csh` or `tcsh` (the default shell at Starlink sites) the following lines should be put in your `.cshrc` file:

```
setenv PRINTER printer
setenv PSPRINTER printer
setenv R_DISPOSE 'lpr -Pprinter -r %s'
```

where `printer` is the name of the printer you want to make your default printer.

5.3 GAIA

GAIA is an image display tool written in C++ by Peter Draper based on the SkyCat image display and catalogue browsing tool developed as part of the VLT project at ESO. GAIA is extendable and can integrate other applications. Currently extensions are provided that allow the user to do aperture and optimal photometry, automatic source detection, contouring, arbitrary region analysis, celestial coordinate readout, calibration and modification, grid overlays, blink comparison, image defect patching and the ability to connect to resources available in on-line catalogues and archives. An example of GAIA in action is shown in Figure 12

A full discussion of GAIA's capabilities can be found in SUN/214.

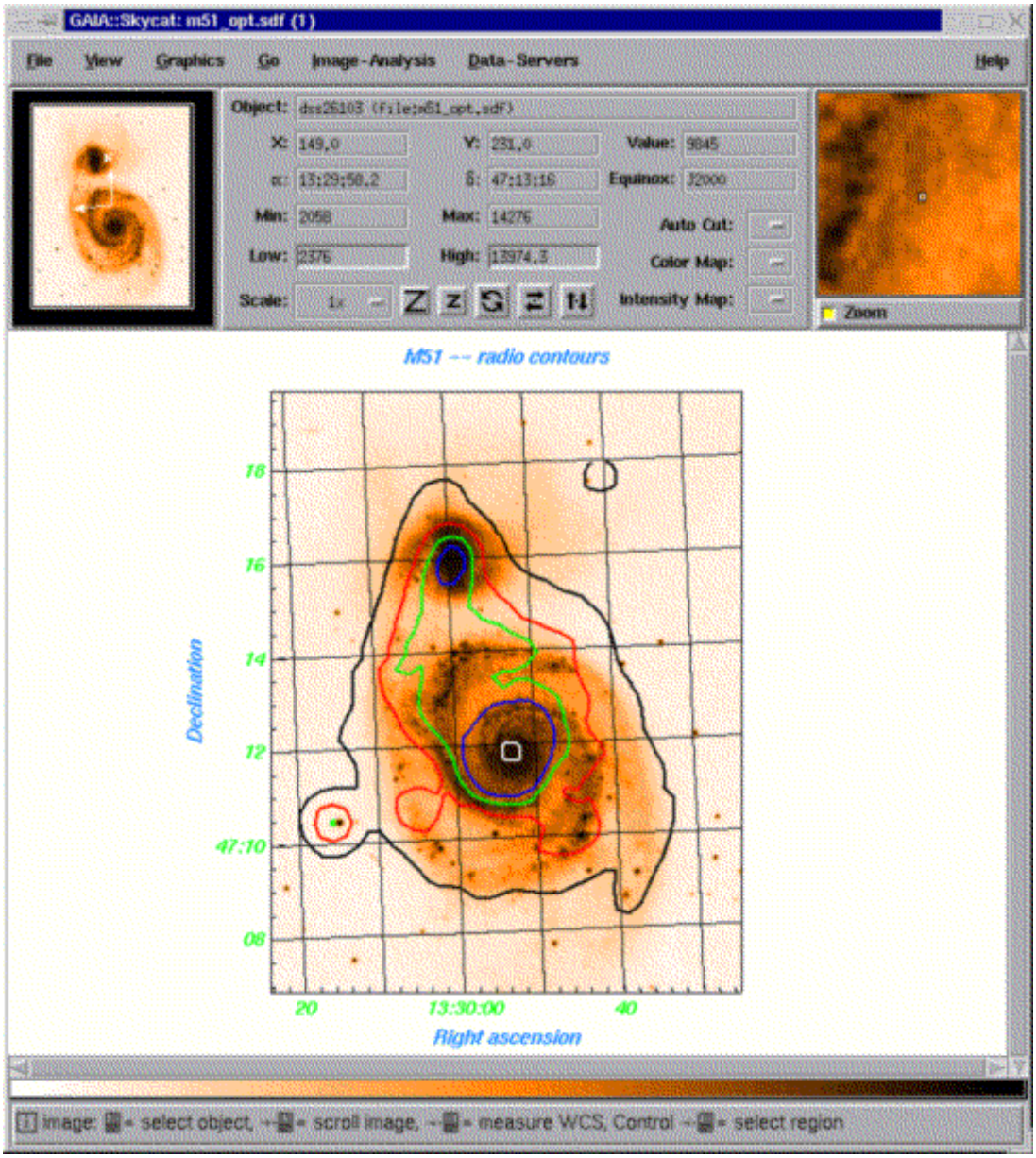


Figure 12: The GAIA interface.

6 Visualisation

For astronomers with complex datasets the standard tools, such as PGPLOT, may not be sufficient for your needs. Instead for display of three-dimensional scalar data you should look towards IDL or DX.

IBM DX (Data Explorer) is the data visualisation package recommended by Starlink, particularly for the visualisation of three-dimensional scalar and vector data. The package is covered in detail in *The DX Cookbook* (SC/2). However, additional information about DX and documentation dealing with SX, the Starlink enhancements to DX, can be found in SUN/203.

The Interactive Data Language (IDL) is designed to be used for data analysis, visualization, and cross-platform application development. IDL may be available at your site, see your Starlink system administrator for details.

An Introduction to Visualisation Software for Astronomy, SG/8, provides a general overview of visualisation software for astronomers and includes details of a number of packages. The author of SG/8, Clive Davenhall, also maintains a web page covering visualisation software at `http://www.roe.ac.uk/~acd/vissys/index.html`.

7 Other Applications

There is so much graphics and image manipulation software available, usually under the GNU Public License (GPL), that I couldn't possibly hope to cover it all. Although I have tried to cover the major packages, new (sometimes better) applications appear every day.

7.1 ImageMagick

ImageMagick, is a collection of GUI and command line tools along with callable libraries allows you to read, write, and manipulate images in any of the more popular image formats including GIF, JPEG, TIFF, PNG and PDF.

There are two methods for accessing the capabilities of ImageMagick. Firstly you can incorporate its functionality directly into your own code by linking to the ImageMagick libraries, you can do this from several languages, including Perl, C, C++, Java and Python. More information can be found in the ImageMagick Users Guide at `http://www.wizards.dupont.com/cristy/ImageMagick.pdf`.

You can also access ImageMagick functions directly from the command line using the `display`, `import`, `animate`, `montage`, `convert`, `mogrify`, `identify`, and `combine` tools. These tools allow powerful manipulations to be carried out on images in a manner understandable to anyone familiar with the UNIX way of doing things, for instance:

```
% convert file.jpg HISTOGRAM:- | display -
```

pipes the output of `convert` into the `display` application to get a intensity histogram for the image `file.jpg`. Here the use of "`-`" for the file specifier (for both `convert` and `display`) directs the applications to use standard input and output (*i.e.* streams).

7.1.1 `display`

Not to be confused with KAPPA `display`, the ImageMagick `display` program is an image processing tool. The program is invoked from the command line as below, Figure 13 shows the interface to the application.

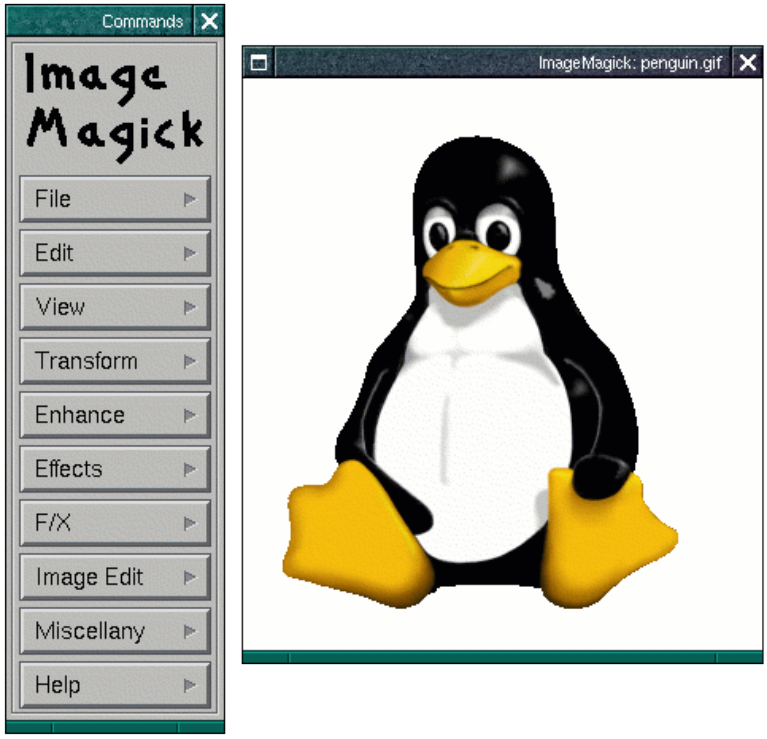


Figure 13: The ImageMagick `display` GUI.

```
% display penguin.tif
```

The `display` application is a powerful image manipulation and processing tool, full details of its functions can be found at `http://www.wizards.dupont.com/cristy/www/display.html`.

One important point for people used to `xv` is that the command menu is brought up by *left* clicking on the image rather than *right* clicking (as with `xv`).

7.1.2 `import`

`import` reads an image from any visible window on your X Windows desktop and outputs it as an image file. You can capture a single window, the entire screen, or any rectangular portion of the screen. Full details of its functions can be found at `http://www.wizards.dupont.com/cristy/www/import.html`.

For instance, to capture the entire X Windows desktop screen in the JPEG image format in a file titled `root.jpg`, use:

```
% import -window root root.jpg
```

While to select a a specific window, using the mouse, and save it in Encapsulated Postscript format use:

```
% import figure.eps
```

A common problem with `import` running on pseudo colour displays is that the image it captures sometimes has the wrong colour map. To correct this use the `-descend option`, *e.g.*

```
% import -descend image.miff
```

By default, `import` quickly grabs the image from the X server. However, it may not always have the correct colors in some areas. This can happen when a sub-window has a different colour map than its parent. With `-descend`, `import` descends the window hierarchy. Descending involves grabbing the image and colour map of each window or sub-window associated with the window you select and combining them on a blank canvas. This can be *significantly* slower than just grabbing the top-level window but ensures that the final composite image will have the correct colour map.

7.1.3 `animate`

`animate` displays a sequence of images. To help prevent color flashing on pseudo colour displays, `animate` creates a single colourmap from the image sequence. This can be rather time consuming. You can speed this operation up by reducing the colours in the image before you "animate" them. Use `mogrify` to colour reduce the images to a single colourmap. Full details can be found at `http://www.wizards.dupont.com/cristy/www/animate.html`.

7.1.4 `montage`

`montage` creates a composite by combining several separate images. The images are tiled on the composite image with the name of the image optionally appearing just below the individual tile.

7.1.5 `convert`

Not to be confused with the Starlink CONVERT package, the ImageMagick `convert` program converts an input file from one image format to an output file with a differing image format. Various types of image processing can be performed on the converted image during the conversion process.

For example, to convert a TIFF image to an A4 Postscript page with the image in the lower left corner we would use:

```
% convert -page A4+0+0 image.tiff document.ps
```

Or to annotate an image with the word "Stuff" written in blue at position (100,100) in the Helvetica 12x24 pixel font, we would use:

```
%convert -font helvetica -pen blue -draw "text 100,100 Stuff" in.jpg out.miff
```

In this case we read in a JPEG file and write out a MIFF (ImageMagick internal format) file. Full details of the different file formats handled by `convert` can be found on the ImageMagick web site at `http://www.wizards.dupont.com/cristy/www/convert.html`.

7.1.6 `mogrify`

`mogrify` performs transformations such as scaling, rotation and colour reduction on a image or series of images.

For instance to scale an image to 640×40 pixels we would use:

```
% mogrify -geometry 640x480! image.miff
```

7.1.7 `identify`

`identify` describes the format and characteristics of one or more image files. It will also report if an image is incomplete or corrupt. The information displayed includes the scene number, the file name, the width and height of the image, whether the image is colourmapped or not, the number of colors in the image, the number of bytes in the image, the format of the image (JPEG, PNM, *etc.*), and finally the number of seconds it took to read and process the image. An example line output from `identify` follows:

```
images/image.miff 640x480 PseudoClass 256c 308135b MIFF 1s
```

If `-verbose` is set, expect additional output including any image comment, *e.g.*

```
Image: images/image.miff
  class: PseudoClass
  colors: 256
  signature: eb5dca81dd93ae7e6ffae99a5275a53e
  matte: False
  geometry: 640x480
  depth: 8
  bytes: 308135
  format: MIFF
  comments:
  Imported from MTV raster image:  image.mtv
```

7.1.8 `combine`

The `combine` program is used to, well combine, two or more images into a single new images. For instance, to compute the difference between two images in a series you could:

```
% combine -compose difference series.1 series.2 difference.miff
```

Or to combine a red, green and blue colour plane into a single composite image:

```
% combine -compose ReplaceGreen red.png green.png red-green.png
% combine -compose ReplaceBlue red-green.png blue.png composite.png
```

7.1.9 `XTP`

Not strictly an image-processing application, `xtp` is a non-interactive replacement for `ftp`.

7.1.10 XMagick

XMagick is a C library, which allows integration of the ImageMagick library calls with any X application by providing functions which convert between the native X image format (XImage) and the native ImageMagick format (Image).

The example code below, which is included in the library, reads an image `image.xpm`, converts the Image to an XImage and back, and stores the result as `image2.xpm`.

```
#include <magick/magick.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "xmagick.h"

static void add_pixel(unsigned long pixel, unsigned long *pixels,
                int *npixels, int max_pixels)
{
        int i;

        if (*npixels >= max_pixels) return;
        for (i = 0; i < *npixels; i++) {
                if (pixel == pixels[i]) return;
        }
        pixels[i++] = pixel;
        *npixels = i;
}

static int npixels_in_ximage(XImage *img, int max_pixels)
{
        int x, y;
        int npixels = 0;
        unsigned long *pixels = malloc(max_pixels*sizeof(*pixels));
        for (y = 0; y < img->height; y++) {
                for (x = 0; x < img->width; x++) {
                        unsigned long pixel = XGetPixel(img, x, y);
                        add_pixel(pixel, pixels, &npixels, max_pixels);
                }
        }
        return npixels;
}

int main(int argc, char **argv)
{
        Image *image, *image2;
        ImageInfo image_info;
        XImage *ximage;
        Display *display;

        display = XOpenDisplay(NULL);
        if (display == NULL) {
                fprintf(stderr, "Can't open X display\n");
                exit(1);
        }

        GetImageInfo(&image_info);
        strcpy(image_info.filename, "image.xpm");
        image = ReadImage(&image_info);
        if (image == NULL) return 1;

        printf("%ld colors in image\n", GetNumberColors(image, NULL));
        ximage = XMagickImageToXImage(display, image);
        printf("%d pixels in ximage\n", npixels_in_ximage(ximage, 256));
        image2 = XMagickXImageToImage(display, ximage);
```

```
        printf("%ld colors in image2\n", GetNumberColors(image2, NULL));
        strcpy(image2->filename, "image2.xpm");

        /* Save to disk */
        WriteImage(&image_info, image2);

        /* Free resources */
        DestroyImage(image);
        DestroyImage(image2);
        XDestroyImage(ximage);
        XCloseDisplay(display);

        return 0;
}
```

The prototypes for the XMagick conversion calls are as follows:

`XImage *XMagickImageToXImage(Display *display, Image *image)`
: Creates XImage from Image. Returns `NULL` if not successful.

`Image *XMagickXImageToImage(Display *display, XImage *ximage)`
: Creates Image from XImage. Returns `NULL` if not successful.

The Xmagick library is available via anonymous FTP from `ftp://siag.nu/pub/xmagick/`.

7.1.11 PythonMagick

PythonMagick, is a interface to ImageMagick. It is similar in function to the Python Imaging Library and calls to PythonMagick can be combined with calls to the Imaging Library. A simply example script from the PythonMagick distribution is shown below:

PythonMagick, is a interface to ImageMagick. It is similar in function to the Python Imaging Library and calls to PythonMagick can be combined with calls to the Imaging Library. A simple example script from the PythonMagick distribution is shown below:

```
#!/usr/local/bin/python
import Magick

# Use the Python Imaging Library to create a Tk display
dpy = Magick.TkDisplay(startmain=0)

# Read the image
img = Magick.read('test.gif')

# Display the image
dpy(img)
dpy(img.Swirl(90))

dpy.startmain=1
dpy.show()
```

7.2 XV

The XV program seems to have been around forever, it was first released sometime during 1990, with the current release version being 3.10a. For a long time it was not only the best image manipulation package available for UNIX, it was the only image manipulation package available for UNIX.

Despite many people using it to convert images from one format to another, XV doesn't have any support for bulk conversions, if you want to convert a lot of images from one format to another you should probably look at PBMplus. XV is definitely not a paint program, if you are interested in creating graphics you should look towards the GIMP, `xpaint` or `xfig`. While XV does have some features of a paint program they aren't that high powered, for instance I very heavily recommend **against** using the text annotation command in XV.

XV is primarily a graphics display and manipulation package, you can see an example of the interface in use in Figure 14.

While the interface is usually fairly self explanatory, the package has a very good online manual available at `http://www.trilon.com/xv/manual/xv-3.10a/cover.html`.

7.2.1 Screen Capture

Most of the screen captures you'll see in this cookbook were done using XV. Simply run `xv`, right click on the XV window to bring up the control panel, and hit the GRAB button bring up the Grab Dialog (see Figure 15). Check the "Hide XV Windows" tick-box, so that the XV windows will disappear while you are doing the screen grab, and hit the GRAB button. A single click on your desktop will now carry out a screen grab.

7.2.2 Problems with small images

XV has a problem displaying images which are less than 100 pixels wide, it will actually stretch the images to this minimum size if you try and get it to display something smaller. This problem is due to your window manager rather than XV itself. XV tries to automatically resize the image window to the size of your image, however most modern window managers enforce a minimum window size (so there is room for the minimise/maximise buttons and other such things in the title bar). There are two workarounds. Firstly, you can start XV with the `-nodecor` command line argument, this should allow XV to resize the window to the correct dimensions. However, depending on your window manager this may cause you problems trying to move the XV window around the screen. Alternatively, you can accept that the image is going to be displayed incorrectly, so long as you check the NORMAL SIZE box in the XV save dialog the image will be written to disk correctly.

7.2.3 Getting XV, patches and enhancements

The current version of XV is 3.10a, there probably won't be a version 3.2 (or the long awaited version 4.0) until the author is successful in sorting out a licence to use the LZW compression algorithm with UniSys.

Both source code and binary distributions are available for download, along with a selection of patches and enhancements, from `http://www.trilon.com/xv/downloads.html`. The patches provided at the site are **not** pre-applied to the standard distributions. Enhancements include patches to read/write PNG and PDF files, so it is worth applying them if you are going to the trouble to compile the source.

7.2.4 Compiling XV on RedHat 6.0

There are two, minor, source-code changes that need to be made to get XV to compile under RH6.0. Firstly in `xv.h` lines 119-121 should be commented out, otherwise the compilation will fail with an "`already defined in stdio.h`" error.

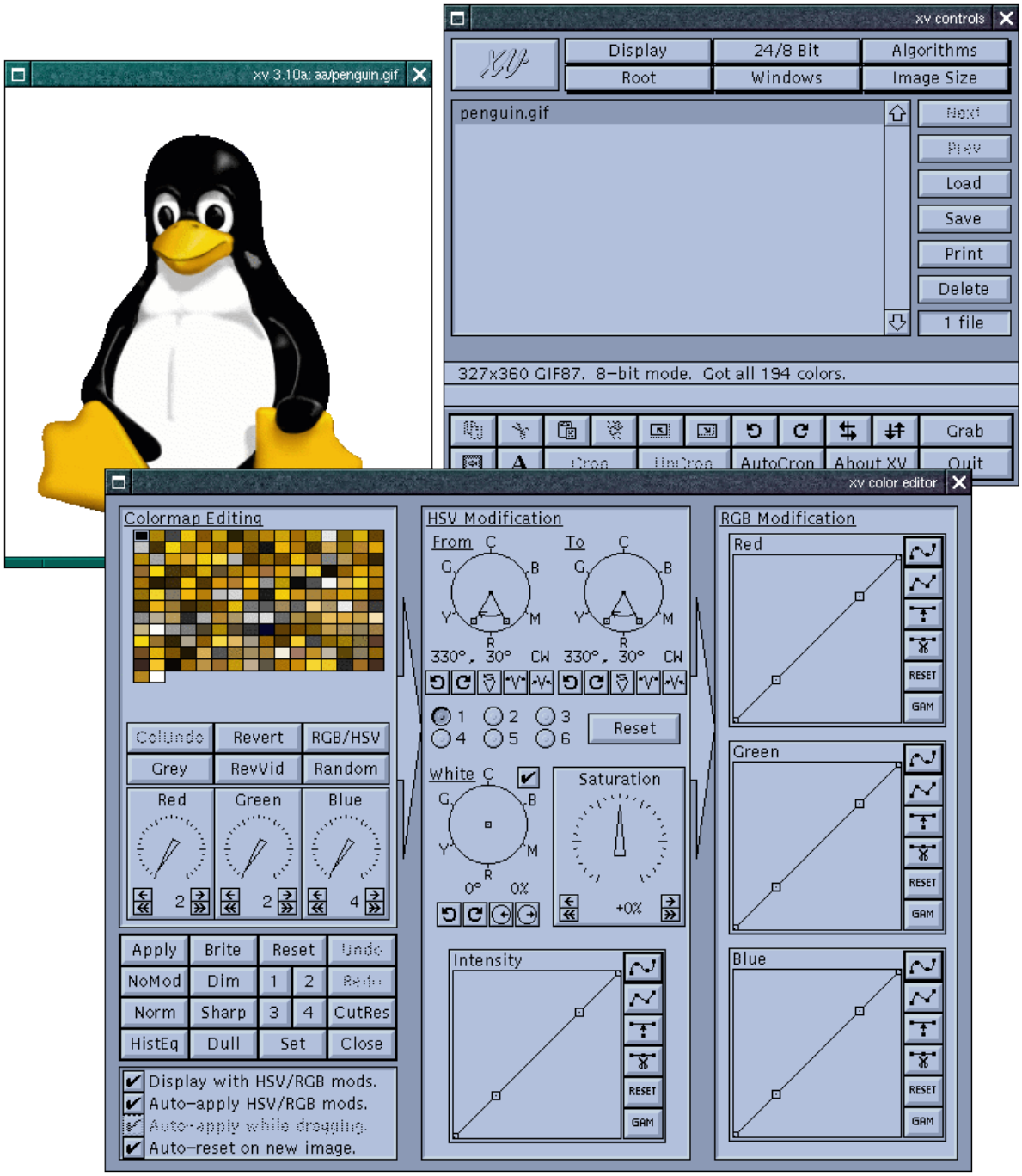


Figure 14: The xv interface along with the Color Editor Dialog (accessed from the Windows pull down menu).

Figure 15: The xv interface being used to grab an image from the display.

Additionally, in the machine-specific options of the package `Makefile` the Linux entry need to specify the full path to the X11 libraries. So line 105 needs to read:

```
MCHN = -DLINUX -L/usr/X11R6/lib
```

7.2.5 XV is not under the GPL

It may surprise some people but XV has not been released under the GPL or another similar community licence. For personal use XV is shareware, if you find it useful you should register the program with the author ($25). Commercial, government, and institutional users *must* register their copies of XV. Users at Starlink sites are covered by the project wide XV licence purchased by Starlink. More information about XV licensing issues can be found `http://www.trilon.com/xv/pricing.html`.

7.3 XPaint

XPaint is a color image editing tool which features most standard paint program options. It allows for the editing of multiple images simultaneously and supports most of the common formats, including PPM, XBM, TIFF, JPEG, *etc.*

XPaint is divided into a toolbox area, for selecting the current paint operation, and paint windows for modifying/creating images. Each paint window has access to its own color palette and set of patterns, although the paint operation in use is globally selected for all windows. An example of the interface in use is shown in Figure 16.

More information on `xpaint` can be found at `http://home.worldonline.dk/$\sim$torsten/xpaint/`.

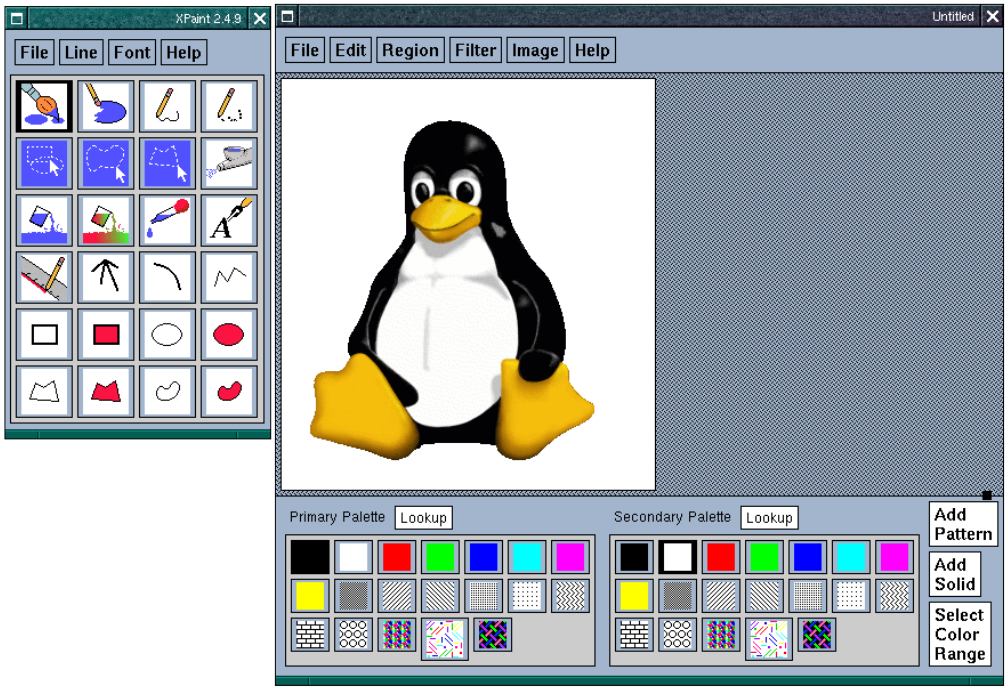


Figure 16: The `xpaint` interface.

7.4 Xfig

Unlike `xpaint`, `xfig` is a drawing rather than a painting package. The difference is subtle. In a paint package if you have for instance drawn a line, the application no longer remembers that it is a line, it is simply a bunch of individual (unrelated) pixels that are now a different colour than they were previously. In a drawing package the application considers the line as an "object" that can be moved, modified or deleted without disturbing anything else. `xfig` allows you to construct figures using objects such as circles, boxes, lines and spline curves. It is also possible to import images in formats such as GIF, JPEG or PostScript, imported images are treated as "objects". A simple example of what can be easily done with the package is shown in Figure 17.

`xfig` saves figures in its native format, Fig format, but they may be converted into various formats such as PostScript, GIF or JPEG. There are other applications that are capable of producing output in Fig format that can subsequently be read into `xfig`. For example, `xfig` doesn't have a facility to create graphs, but tools such as GNUPLOT can create graphs and output them in Fig format.

More information on `xfig` can be found at `http://epb1.lbl.gov/xfig/`.

7.4.1 `pstoedit`

The `pstoedit` application can convert PostScript and PDF graphic files into a variety of vector formats, including the `xfig` Fig format and the Sketch internal format. `pstoedit` can be found at `http://www.geocities.com/SiliconValley/Network/1958/pstoedit/`.

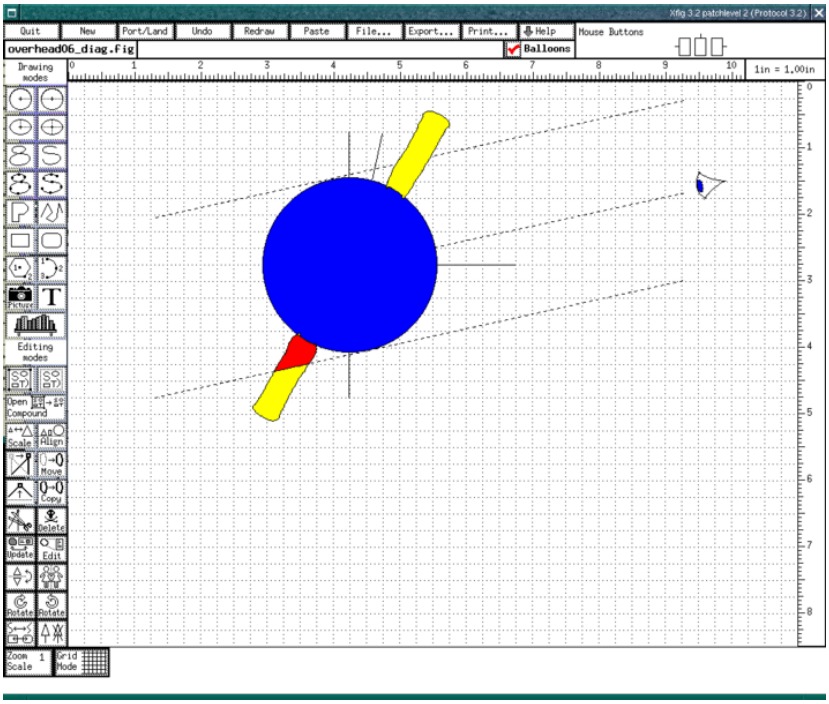


Figure 17: Creating a figure showing line of sight effects on a magnetically accreting white dwarf using the `xfig` package.

7.5 Sketch

Sketch is another interactive drawing program which somewhat resembles Corel Draw or Adobe Illustrator. Unusually for such a beast Sketch has been implemented entirely in Python, and to use it you must have Python installed along with the Python Imaging Library. In addition the application also requires Tk/Tcl to be present. More information can be found on the Sketch home page at `http://sketch.sourceforge.net/`.

Sketch also has a scripting function (using Python) to automate tasks and add new functionality. Since Sketch was been developed in Python, scripts have access to all areas of the application (including internal data structures) which allows user written scripts to be very powerful tools.

7.6 GIMP

The GNU Image Manipulation Program (GIMP) is package suited to tasks as photo retouching, image composition and image authoring. It can be used as a simple paint program but is perhaps better suited to tasks such as photo retouching. However, the GIMP is extensible. It has been designed to be augmented with plugins, and includes a scripting interface to simplify repetitive image processing, making it suitable for batch processing tasks and image format conversion. Figure 18 shows the interface during a typical GIMP session.

It requires the presence of the GTK+ library. If you are working on a Linux machine it is likely that both GTK+ and the GIMP will already be installed. The latest stable version of the GIMP available before going to press is 1.1.13 (released 29th Nov 1999).

Likened to Adobe Photoshop by its supporters, the GIMP is a complex tool in the right hands, it is a professional quality graphics package. Unfortunately this doesn't mean that anything

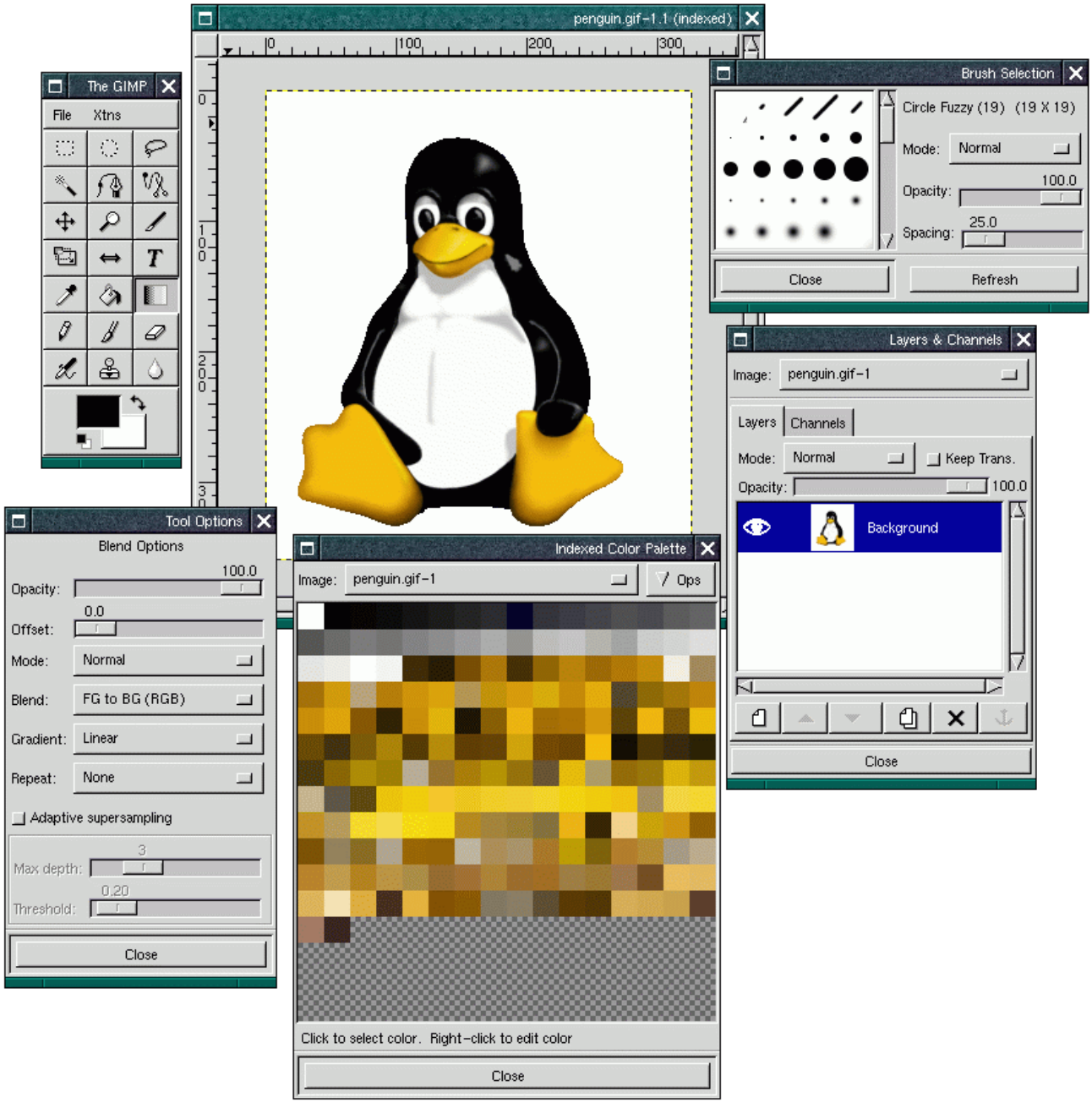


Figure 18: The GIMP interface.

coming out of the GIMP will be of professional quality, it means that if a professional graphics artist goes to use it they'll find all the tools they're used to or need to get the job done. I don't claim to be a professional graphics or layout artist, and am therefore in no position to try and teach anyone how to use the GIMP. However I still make use of the program quite extensively when dealing with graphics, because the truth is you don't have to be a professional to make use of the GIMP at a basic level. In fact, due to the plugin nature of `Script-Fu` some highly advance graphics manipulation techniques are available very easily to the novice user. A good start is the GIMP User Manual (GUM) which you can find at `http://manual.gimp.org/`, however I'd also like point you towards the following books:

- *GIMP: The Official Handbook*
  By Karin Kylander, Olof S. Kylander
  US List Price: $49.99
  UK List Price: £46.99
  Paperback – 895 pages (3 November, 1999)
  Coriolis Group Books; ISBN: 1576105202
- *The Artists' Guide to the GIMP*
  By Hammel, Michael J.
  US List Price: $39.95
  UK Equivalent: £24.54
  Paperback – 332 pages (1 December, 1998)
  Specialized Systems Consultants; ISBN: 1578310113

GIMP: The Official Handbook appears to be a bound copy of the GIMP User Manual (GUM), while *The Artists Guide to the GIMP* is a tutorial style book that seems to be well thought of by the Open Source community.

A good series of introductory articles can be found on *The Graphics Muse* web site which is maintained by the author of *The Artists' Guide to the GIMP*, see `http://graphics-muse.com/`.

- **Mastering GIMP - Part I (5th Nov 1999)**
  `http://graphics-muse.com/graphics-cgi/gmGimp.pl/gimp/articles/1999/15/article.html`
- **Mastering GIMP - Part II (12th Nov 1999)**
  `http://www.graphics-muse.com/graphics-cgi/gmGimp.pl/gimp/articles/1999/16/article.html`
- **Mastering GIMP - Part III (4th Dec 1999)**
  `http://www.graphics-muse.com/graphics-cgi/gmGimp.pl/gimp/articles/1999/17/article.html`
- **Mastering GIMP - Part IV (13th Dec 1999)**
  `http://www.graphics-muse.com/graphics-cgi/gmGimp.pl/gimp/articles/1999/18/article.html`

Another good article, also on *The Graphics Muse* site is "*Better aliens through science*". This article shows some of the power of the GIMP when used to do photo manipulation work.

- **Better aliens through science (May 1999)**
  `http://graphics-muse.com/gimp/tutorials/1999/9/tutorial.html`

7.6.1 Plug-ins, `Script-Fu`, GIMP-Perl and `Gimp::Fu`

There seems to be some confusion between GIMP Plug-ins and `Script-Fu`. Plug-ins are external modules that do the graphics transformations, while `Script-Fu` is the name of the Scheme based scripting interface in the GIMP. Scripts written in `SIOD`, the Scheme subset that GIMP uses, are known simply as `Script-Fu` scripts.

GIMP-Perl is the name of the Perl module that interfaces with the GIMP, it is the primary interface between Perl scripts and the GIMP. However, it does not include easy access to the user interface (the GUI), nor does it abstract the Procedural Database. These two aspects are more properly handled by, yet another, Perl module `Gimp::Fu`. Perl scripts that use GIMP-Perl and `Gimp::Fu` are sometimes referred to as `Perl-Fu` scripts. All this can get somewhat confusing at times. The GIMP-Perl extensions have been part of the core GIMP distribution since version 1.1, however if you are using GIMP on even a recent distribution of Linux (*e.g.* RedHat 6.x) then it is likely that you are using and older version of the GIMP (1.0.x or below). The simplest course if you want to make use of GIMP-Perl interface is to ask your system manager to upgrade your version of the application, alternative you can download the plug-in from the GIMP Plug-In Registry. The official GIMP-Perl web site can be found at `http://www.goof.com/pcg/marc/gimp.html`.

A good introduction to the GIMP-Perl extensions can be found (again) at the *The Graphics Muse* site:

- `Gimp-Perl` **Introduction (July 1999)**
  `http://graphics-muse.com/gimp/articles/1999/5/article.html`
- `Gimp-Perl` **Part II (August 1999)**
  `http://graphics-muse.com/gimp/articles/1999/6/article.html`

An example script, taken from the GIMP-Perl documentation, to add a "scratch effect" to an image is shown below:

```
#!/usr/bin/perl

use Gimp;
use Gimp::Fu;
use Gimp::Util;

sub new_scratchlayer {
    my($image,$length,$gamma,$angle)=@_;
    my $type=$image->layertype(0);
    my($layer)=$image->layer_new ($image->width, $image->height, $image->layertype(0),
                                  "displace layer ($angle)", 100, NORMAL_MODE);
    $layer->add_layer(-1);
    $layer->fill (WHITE_IMAGE_FILL);
    $layer->noisify (0, 1, 1, 1, 0);
    $layer->mblur (0, $length, $angle);
    #$layer->levels (VALUE_LUT, 120, 255, $gamma, 0, 255);
    $layer->levels (VALUE_LUT, 120, 255, 0.3, 0, 255);

    $layer;
}
```

```
register "scratches",
         "Create a scratch effect",
         "Add scratches to an existing image. Works best on a metallic-like background.",
         "Marc Lehmann",
         "Marc Lehmann <pcg\@goof.com>",
         "19990223",
         N_"<Image>/Filters/Distorts/Scratches",
         "*",
         [
          [PF_SLIDER,  "angle_x"   , "The horizontal angle"  ,  30, [  0, 360]],
          [PF_SLIDER,  "angle_y"   , "The vertical angle"    ,  70, [  0, 360]],
          [PF_SLIDER,  "gamma"     , "Scratch map gamma"     , 0.3, [0.1,  10, 0.05]],
          [PF_SPINNER, "smoothness", "The scratch smoothness",  15, [  0, 400]],
          [PF_SPINNER, "length"    , "The scratch length"    ,  10, [  0, 400]],
          #[PF_BOOL, , "bump_map"  , "Use bump map instead of displace", 0],
         ],
         [],
         ['gimp-1.1'],
         sub {
   my($image,$drawable,$anglex,$angley,$gamma,$length,$width)=@_;

   $image->undo_push_group_start;

   my $layer1 = new_scratchlayer ($image, $length, $gamma, $anglex);
   my $layer2 = new_scratchlayer ($image, $length, $gamma, $angley);

   $drawable->displace ($width, $width, 1, 1, $layer1, $layer2, WRAP);

   $layer1->remove_layer;
   $layer2->remove_layer;

   $image->undo_push_group_end;

   $image;
};

exit main;
```

7.6.2 GIMP-Python

GIMP-Python is a package that allows people to write plug-ins for the GIMP in Python rather than `Script-Fu` (Scheme), Perl or C. GIMP-Python binds to the GTK+ Python extention library PyGTK, found at `http://www.daa.com.au/~james/pygimp/`, rather than the more commonly used (under Python) Tkinter extentions for a more consistent *look and feel* interface to the GIMP. The package provides an almost complete wrapper for the `libgimp` plug-in library, and also offers `Script-Fu` capabilities through the `gimpfu` module.

An example of a GIMP-Python plug-in (a translation of the `Script-Fu` clothify plug-in) is shown below, in this example the GUI is generated by the `gimpfu` module.

```
from gimpfu import *

have_gimp11 = gimp.major_version > 1 or \
```

```
                gimp.major_version == 1 and gimp.minor_version >= 1

    def python_clothify(timg, tdrawable, bx=9, by=9,
                        azimuth=135, elevation=45, depth=3):
            bx = 9 ; by = 9 ; azimuth = 135 ; elevation = 45 ; depth = 3
            width = tdrawable.width
            height = tdrawable.height
            img = gimp.image(width, height, RGB)
            layer_one = gimp.layer(img, "X Dots", width, height, RGB_IMAGE,
                                   100, NORMAL_MODE)
            img.disable_undo()
            if have_gimp11:
                    pdb.gimp_edit_fill(layer_one)
            else:
                    pdb.gimp_edit_fill(img, layer_one)
            img.add_layer(layer_one, 0)
            pdb.plug_in_noisify(img, layer_one, 0, 0.7, 0.7, 0.7, 0.7)
            layer_two = layer_one.copy()
            layer_two.mode = MULTIPLY_MODE
            layer_two.name = "Y Dots"
            img.add_layer(layer_two, 0)
            pdb.plug_in_gauss_rle(img, layer_one, bx, 1, 0)
            pdb.plug_in_gauss_rle(img, layer_two, by, 0, 1)
            img.flatten()
            bump_layer = img.active_layer
            pdb.plug_in_c_astretch(img, bump_layer)
            pdb.plug_in_noisify(img, bump_layer, 0, 0.2, 0.2, 0.2, 0.2)
            pdb.plug_in_bump_map(img, tdrawable, bump_layer, azimuth,
                                 elevation, depth, 0, 0, 0, 0, TRUE, FALSE, 0)
            gimp.delete(img)

    register(
            "python_fu_clothify",
            "Make the specified layer look like it is printed on cloth",
            "Make the specified layer look like it is printed on cloth",
            "James Henstridge",
            "James Henstridge",
            "1997-1999",
            "<Image>/Python-Fu/Alchemy/Clothify",
            "RGB*, GRAY*",
            [
                    (PF_INT, "x_blur", "X Blur", 9),
                    (PF_INT, "y_blur", "Y Blur", 9),
                    (PF_INT, "azimuth", "Azimuth", 135),
                    (PF_INT, "elevation", "elevation", 45),
                    (PF_INT, "depth", "Depth", 3)
            ],
            [],
            python_clothify)

    main()
```

The GIMP-Python manual is available as either HTML or DocBook SGML at:

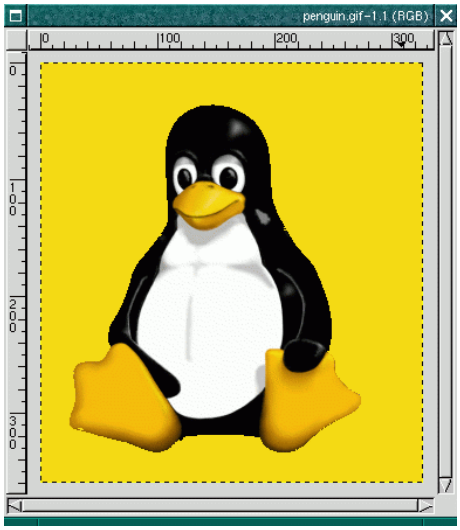


Figure 19: Larry Ewing's image of Tux the penguin.

- GIMP-Python Documentation (HTML)
  `http://www.daa.com.au/~james/pygimp/pygimp.html`
- GIMP-Python Documentation (SGML)
  `http://www.daa.com.au/~james/pygimp/pygimp.sgml`

The package can be downloaded via FTP from `ftp://ftp.daa.com.au/pub/james/pygimp/`. It should be noted that the version available at the GIMP Plug-In Registry is out of date.

7.6.3 GIMP Plug-In Registry

The GIMP Plug-In Registry is a central repository for GIMP extentions, and can be found at `http://registry.gimp.org/`. The available plugins provide everything from support for exotic file format (*e.g.* Adobe Photoshop) to interfacing with SANE. If you have a graphics problem that someone else might have faced, and you are considering using the GIMP, maybe you should look here before trying to write your own code to do the job.

7.6.4 The GIMP and layers

One important feature of the GIMP that most casual users, who aren't used to applications such as Photoshop, may not be aware of is the concept of *layers*. Let us work through an (very) basic example together to show you how they can be useful.

Take an image – I'm going to use Larry Ewing's image of Tux the penguin, the Linux mascot (see Figure 19). As part of the example we are going to want to modify the channel levels of the image, since my example image was a GIF (pseudo colour) image we need to convert it to RGB before we start to play with it. Alot of the GIMP tools work only on RGB (true colour) images rather than indexed (which is how the GIMP refers to pseudo colour).

This is not a problem, if your image is not RGB to begin with *right click* on the image to bring up the actions menu and choose IMAGE→RGB

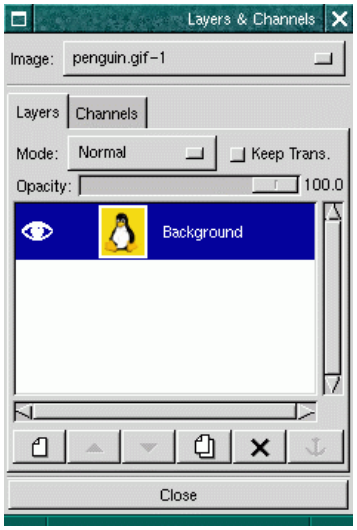


Figure 20: The GIMP Layers & Channels dialog.

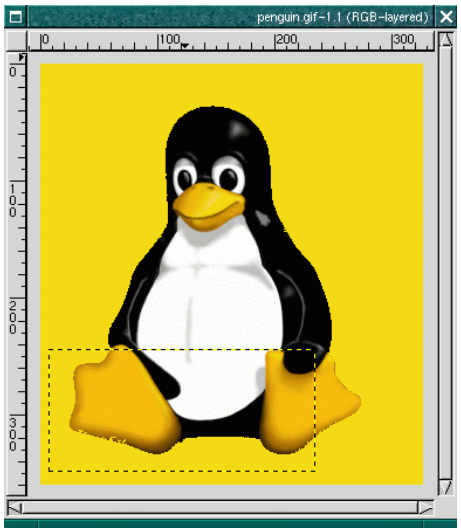


Figure 21: Making a "floating selection" in GIMP.

We now want the Layers & Channels dialog, so bring up the action menu again and choose LAYERS→LAYERS & CHANNELS, this should produce a dialog similar to Figure 20.

As I've done in Figure 21 we want to select a box on the image, we're going to put some text in the box so it needs to be big enough to write in.

We want to select this area to work with, go to the menu and choose SELECT→FLOAT. If you look at the Layers & Channels dialog (Figure 22) you should see that a "Floating Selection" has appeared.

Double click on the Floating Selection layer in the Layers & Channels dialog, a pop-up dialog will appear allowing you to rename it (as in Figure 23). We're going to make this a text layer, so call it something appropriate like "text Box".

Make sure the new layer is selection in the Layers & Channels dialog (see Figure 24).

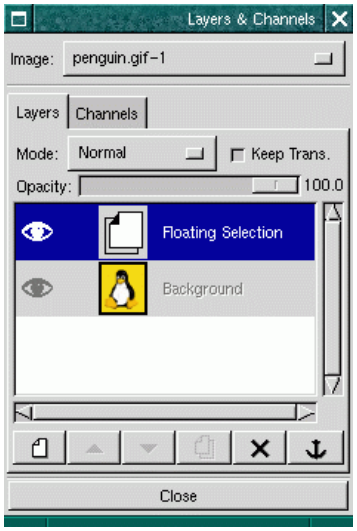


Figure 22: A new layer has been created for the image.

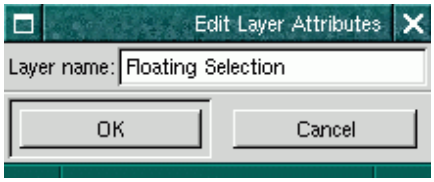


Figure 23: The pop-up dialog.

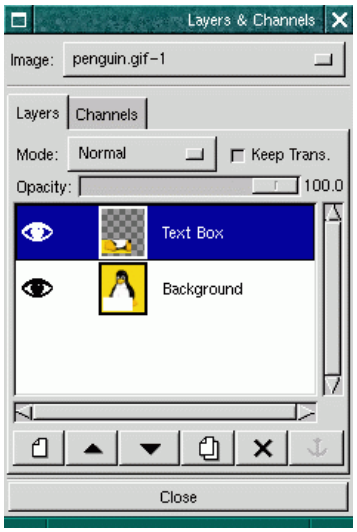


Figure 24: Our renamed text layer is selected in the dialog.

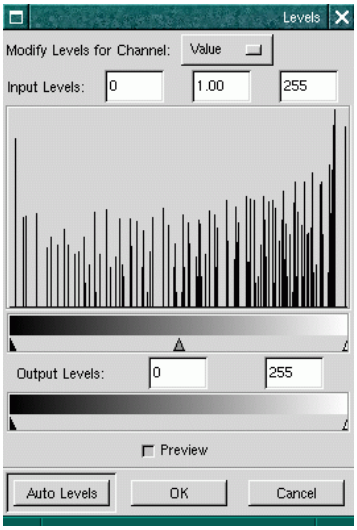


Figure 25: The GIMP Levels dialog.

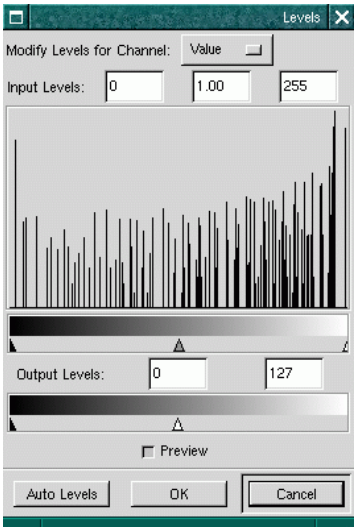


Figure 26: The Levels dialog again, note the position of the bottom slider.

Back to the image and bring up the menu, select IMAGE→COLORS→LEVELS to bring up the Levels dialog (see Figure 25).

We want to change the cut off level in the layer, so move the upper (white) slider in the lower (Output Level) bar from the extreme right to around the centre of the allowed range. So long as the PREVIEW button above this slider is selected you should see out text box getting darker. If you are not sure what I'm talking about here compare Figure 25 (before) and Figure 26 (after).

Hit OK and the you should end up with something like Figure 27.

Now we want some text, bring up the menu and select TOOLS→TEXT. Alternatively, you can also go to the control panel (probably by now exiled to the corner of your screen) and hit the TEXT TOOL button (the **T** button mid-way down the right hand side). Either of these actions will bring up the Text dialog (see Figure 28). Choose your font and whatever font properties you wish, and type some text into the dialog. On hitting OK you'll find that the text appears as

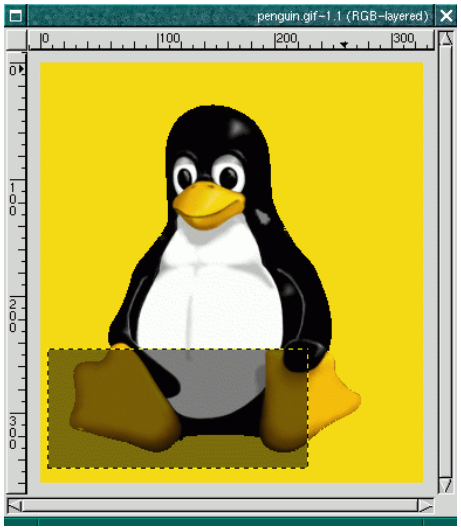


Figure 27: Tux with his darkened text layer.

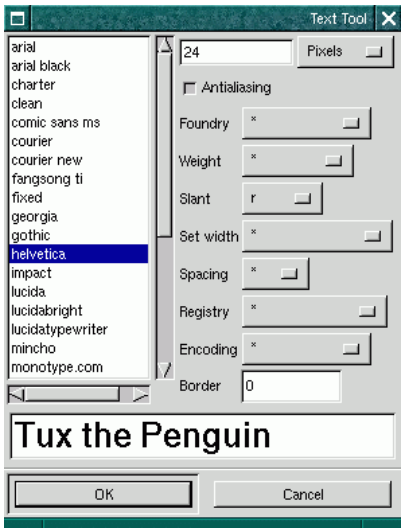


Figure 28: The GIMP Text dialog.

a floating selection which you can move around using the MOVE TOOL (the thing that looks like a addition sign, third down on the left hand side of the control panel, or TOOLS→MOVE from the action menu).

If you look at your Layers & Channels dialog again you'll see that the the text has appeared as another "Floating Selection". We've just created another layer in our image. Rename this layer as before and use the text tool to add any additional text you want to the image, see Figure 29

Figure 30 shows the Layers & Channels dialog after adding two additional layers with the TEXT TOOL, the first text layer is selected (which is why the words "Tux the Penguin" are surrounded with a dotted line in Figure 29.

Click on the "Background" layer in the Layers & Channels dialog, and you should be left with something like the pen-ultimate figure in the series, Figure 31.

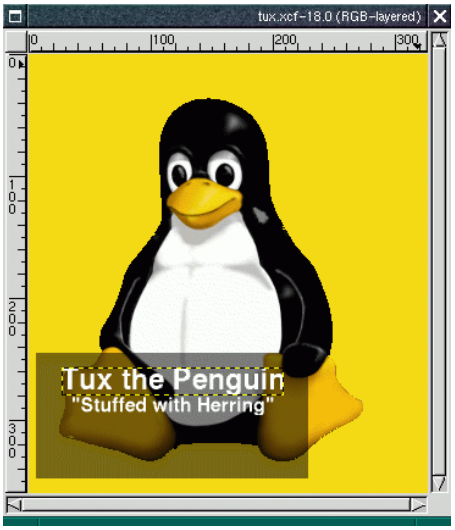


Figure 29: Our Tux image with text added as additional layers ontop of the box we prepared earlier.

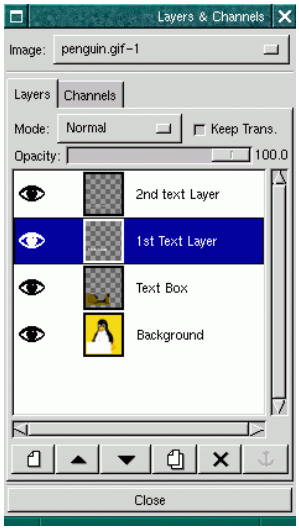


Figure 30: The Layers & Channels dialog again with the first text layer selected.

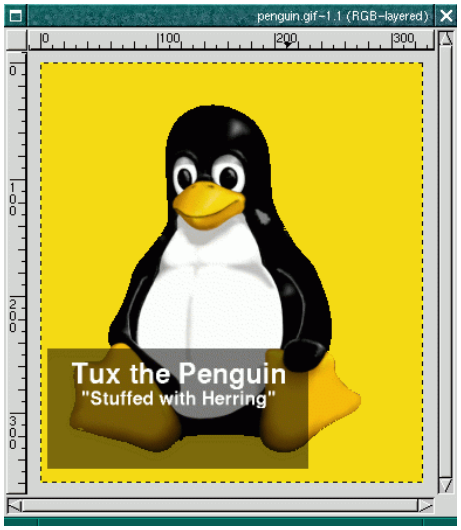


Figure 31: The final product, Tux with text.

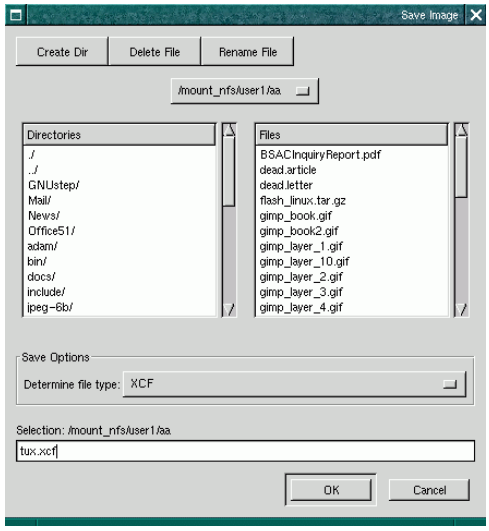


Figure 32:

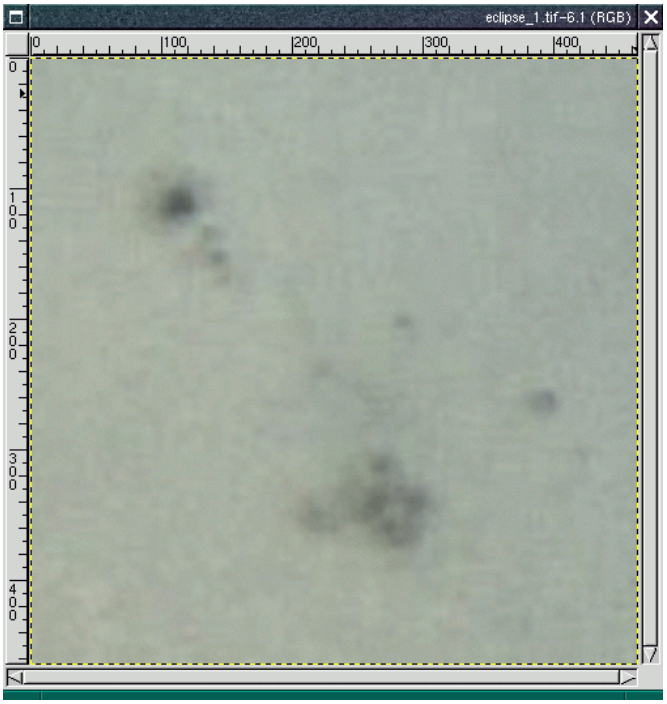


Figure 33: Original star spot image generated using PGPLOT

Now select FILE→SAVE AS from the action menu, see Figure 32, to bring up the File dialog.

Save the image as an XCF file. Then go back to the image and select LAYERS→MERGE VISIBLE LAYERS from the menu, choosing EXPAND AS NECESSARY from the popup menu. If you look at the Layers & Channels dialog you'll see that your carefully crafted layers have disappeared and we now have a flat bitmap. You can now save the image in a usable format, although since we currently have an RGB image if you want to save it as GIF (for instance) you'll have to change the image back to Indexed by selecting IMAGE→INDEXED from the menu.

Okay, you are probably now wondering why we went to all this trouble of creating multiple layers for such little pay back. Well we can now go back to our saved XCF image at any time in the future and change things, GIMP can work with individual layers so, for instance, we can delete or move the text easily by deleting or moving its layer. Layers add flexibility, it would be very difficult to remove the text from our final GIF image, but with the layered XCF image this is a trivial operation.

7.6.5 Mapping images to solid objects

You may have noticed the rather nice star spot image on the front cover of the cookbook. The inital star spot image, see Figure 33, was generated using some modelling code which wrote out a greyscale image using PGPLOT. Mapping the image onto the sphere was done using the GIMP.

Mapping images onto solid objects is just one of the many tools and filters avialable in the GIMP, the map object function can be found by selecting FILTERS→MAP→TO OBJECT. Newer version of the GIMP ($\geq$1.1x) have the ability to map images more complex objects, however the version I was using (v1.0.4, shipped with RedHat 6.0) only allows you to map objects to a sphere (see Figure 34) or a plane (see Figure 35).

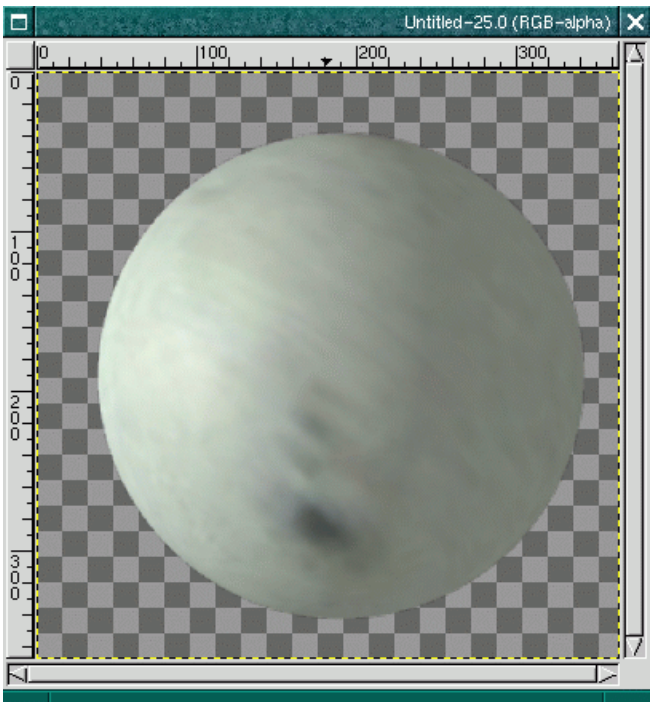


Figure 34: Star spot image mapped on sphere illuminated from the upper left quadrant by a point source.

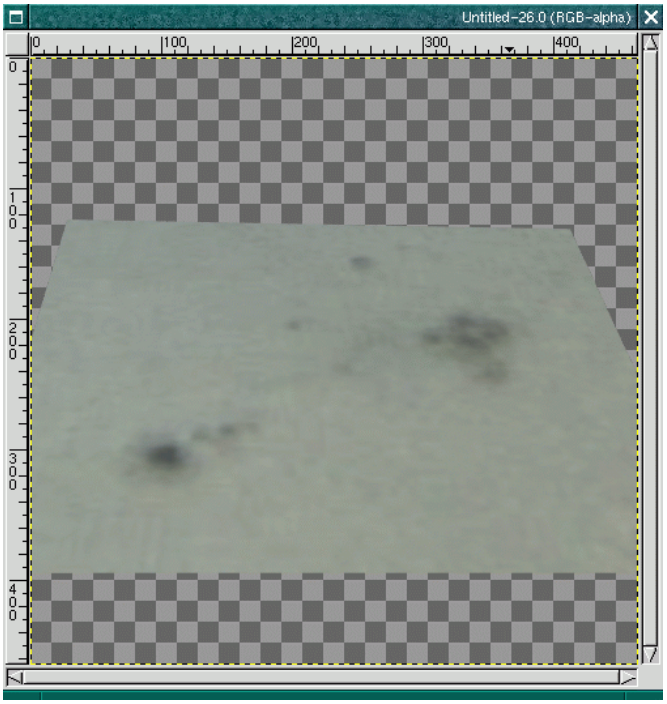


Figure 35: Star spot image projected on plane.

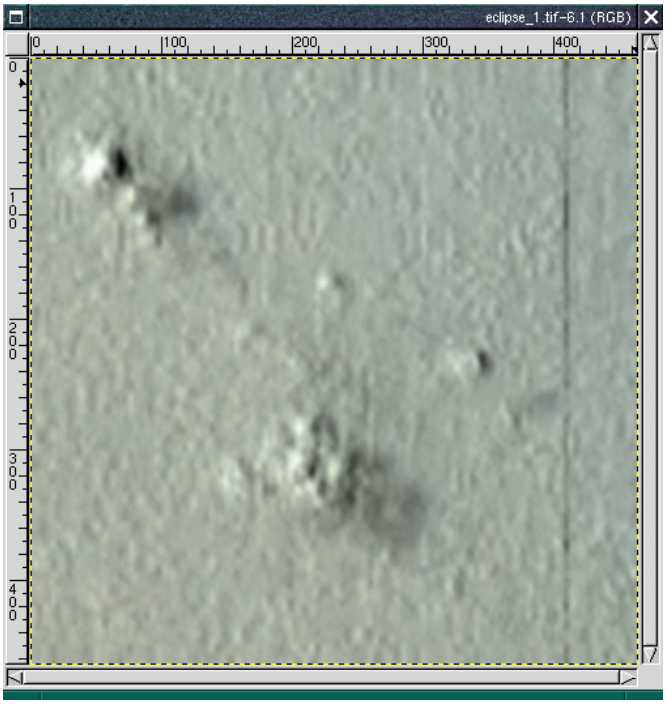


Figure 36: Star spot image with an inverse bump map and gaussian blur.

Another interesting thing that you can do with the mapping functions in the GIMP is bump map our image (see Figure 36) by selecting FILTERS→MAP→BUMP MAP to emphasize the star spots. In this case I've used a bump map followed by a gaussian blur, FILTERS→BLUR→GAUSSIAN BLUR (IIR), to soften the effect. The bump mapped image can be seen in mapped onto a plane in Figure 37. The MAP OBJECT dialog can be seen in Figures 38, 39, 40 and 41.

Another way to present our star spot map is given by the SCRIPT-FU→ANIMATE→SPINNING GLOBE extention. This generates multiple maps onto a sphere and could be used to create an animated GIF of our star.

7.7 Electric Eyes

The development of Electric Eyes is bound up along with GNOME, the GNU Project user environment. It was originally intended to be the default image *viewer* on the GNOME/Enlightenment desktop, although it does now have some limited image editing facilities. An example of the Electric Eyes interface, which is typically GNOMEish, is shown in Figure 42

If you have GNOME, for more information see `http://www.gnome.org/`, installed then Electric Eye will likely also be installed. If you are working on a RedHat Linux box this is almost certainly the case.

7.8 WhirlGIF

WhirlGIF is a command-line utility for generating multi-image GIF animations from a sequence of GIF files. More information can be found at `http://www.msg.net/utility/whirlgif/`. For example:

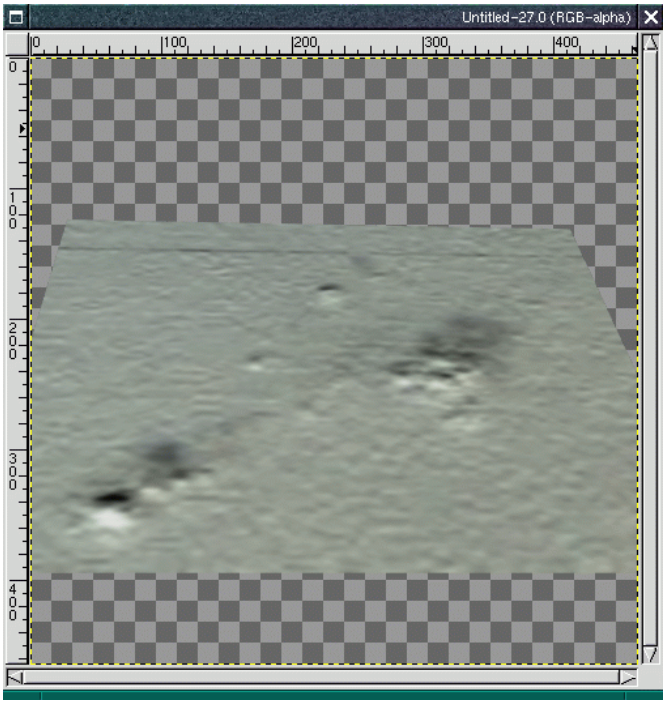


Figure 37: Bump mapped star spot image projected on a plane

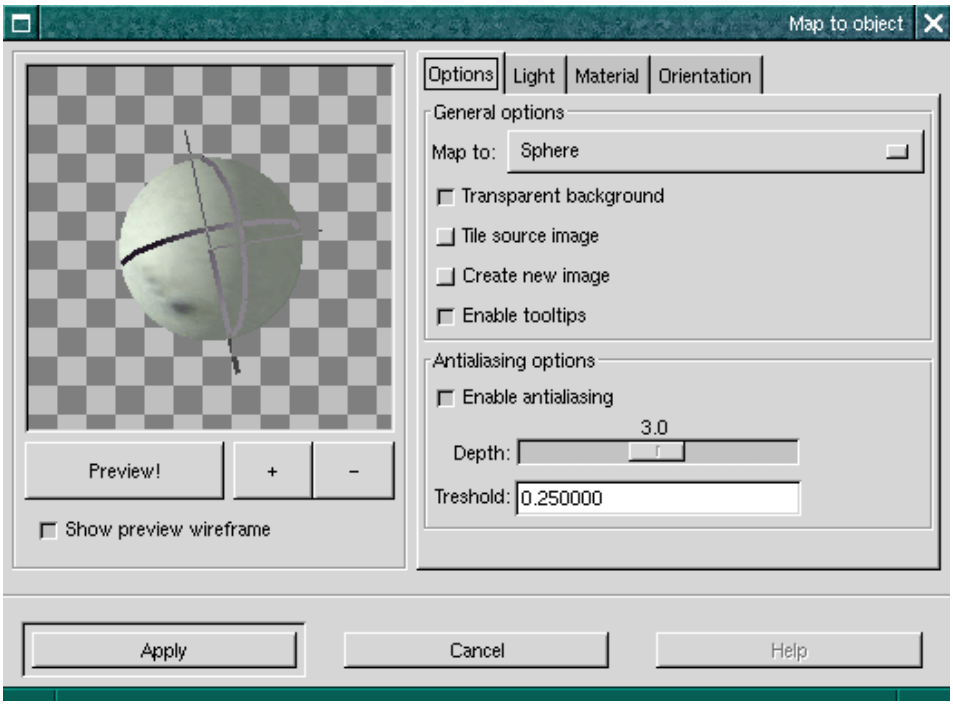


Figure 38: The GIMP Map Object Dialog “Options”

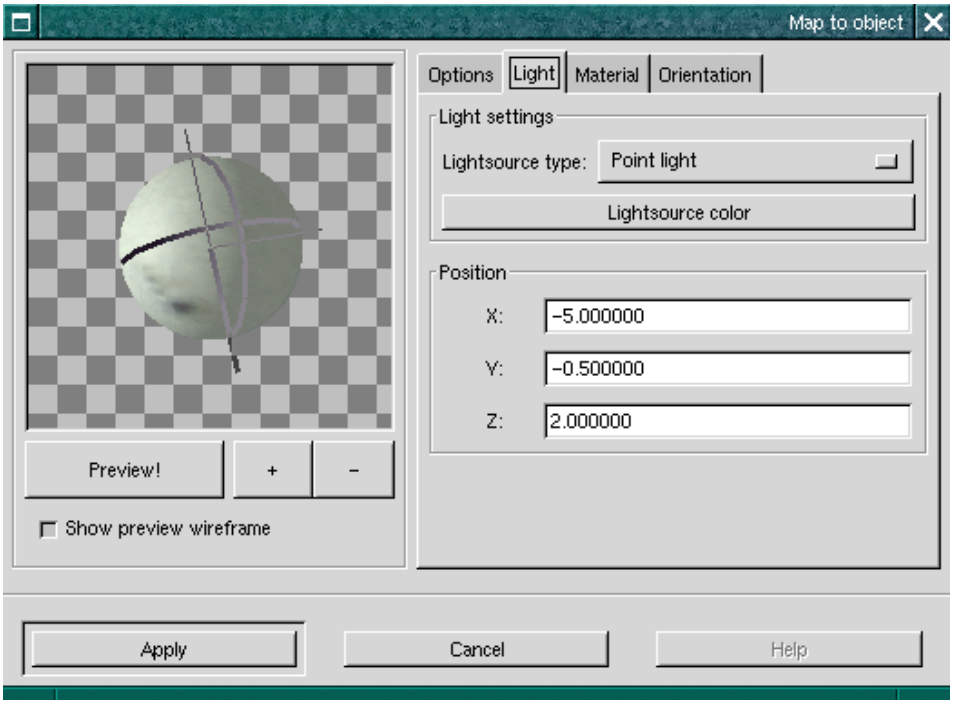


Figure 39: The GIMP Map Object Dialog "Light"

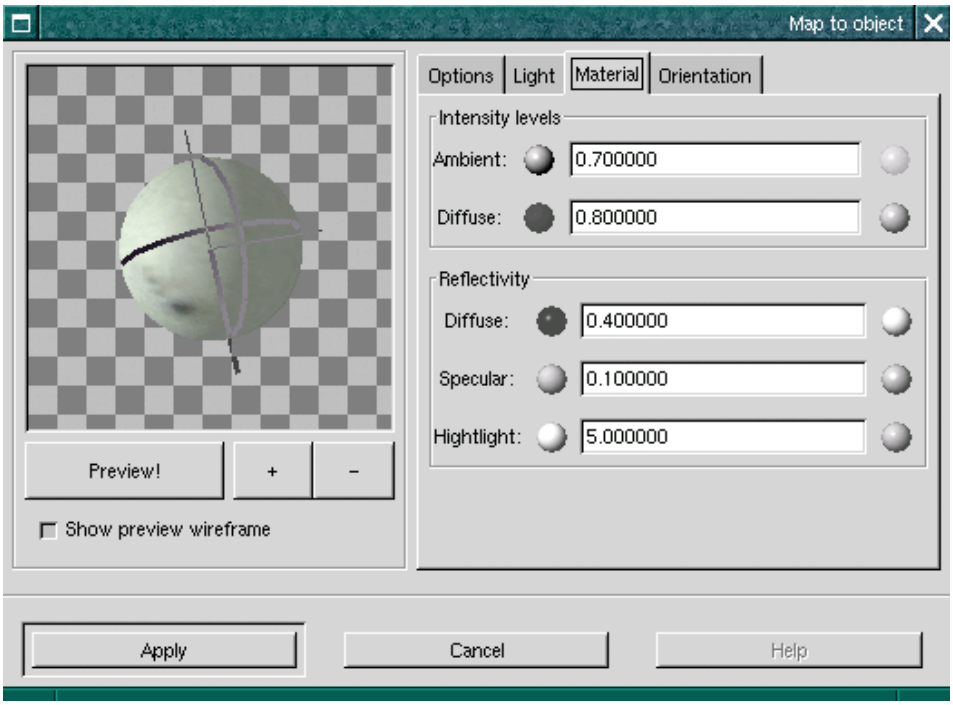


Figure 40: The GIMP Map Object Dialog "Material"

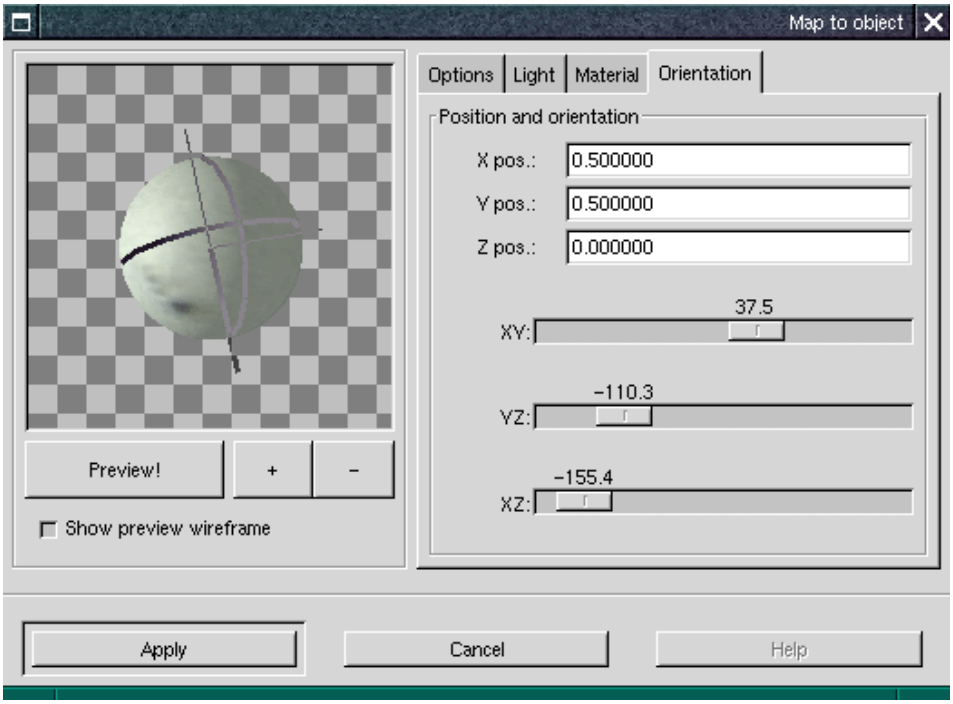


Figure 41: The GIMP Map Object Dialog "Orientation"

```
% whirlgif -loop 5 -trans "#00f10e" file*.gif
```

would create an animated GIF (directed to standard out) from all files in the current directory fitting the pattern `file*.gif`, which would loop five times before stoping and have colour `#00f10e` set to be the transparent index. While:

```
% whirlgif -o out.gif -time 5 a.gif b.gif -time 100 c.gif -time 5 d.gif e.gif
```

would create an animated GIF (called `out.gif`) from the files `a.gif`, `b.gif`, `c.gif`, `d.gif` and `e.gif`. The `d.gif` file would be displayed for 1s, while the remaining frames would be displayed for only 50ms each (delays are in units of 1/100 th of a second).

8 CAD Applications

8.0.1 QCad

QCad is a professional level CAD System. With QCad you can construct and change drawings and save them in DXF format which allows you to import/export your drawing into other CAD systems such as AutoCAD. An example of the QCad interface is shown in Figure 43.

The QCad application requires the presence of the Troll Tech `Qt` Library which underlies the KDE desktop environment. If you are running Linux it is likely that KDE, and hence the `Qt` library, is already installed. If not, more information about the Troll Tech `Qt` library can be found at `http://www.kde.org/` and `http://www.troll.no/`.

Further information on the QCad program can be found at `http://www.qcad.org`.

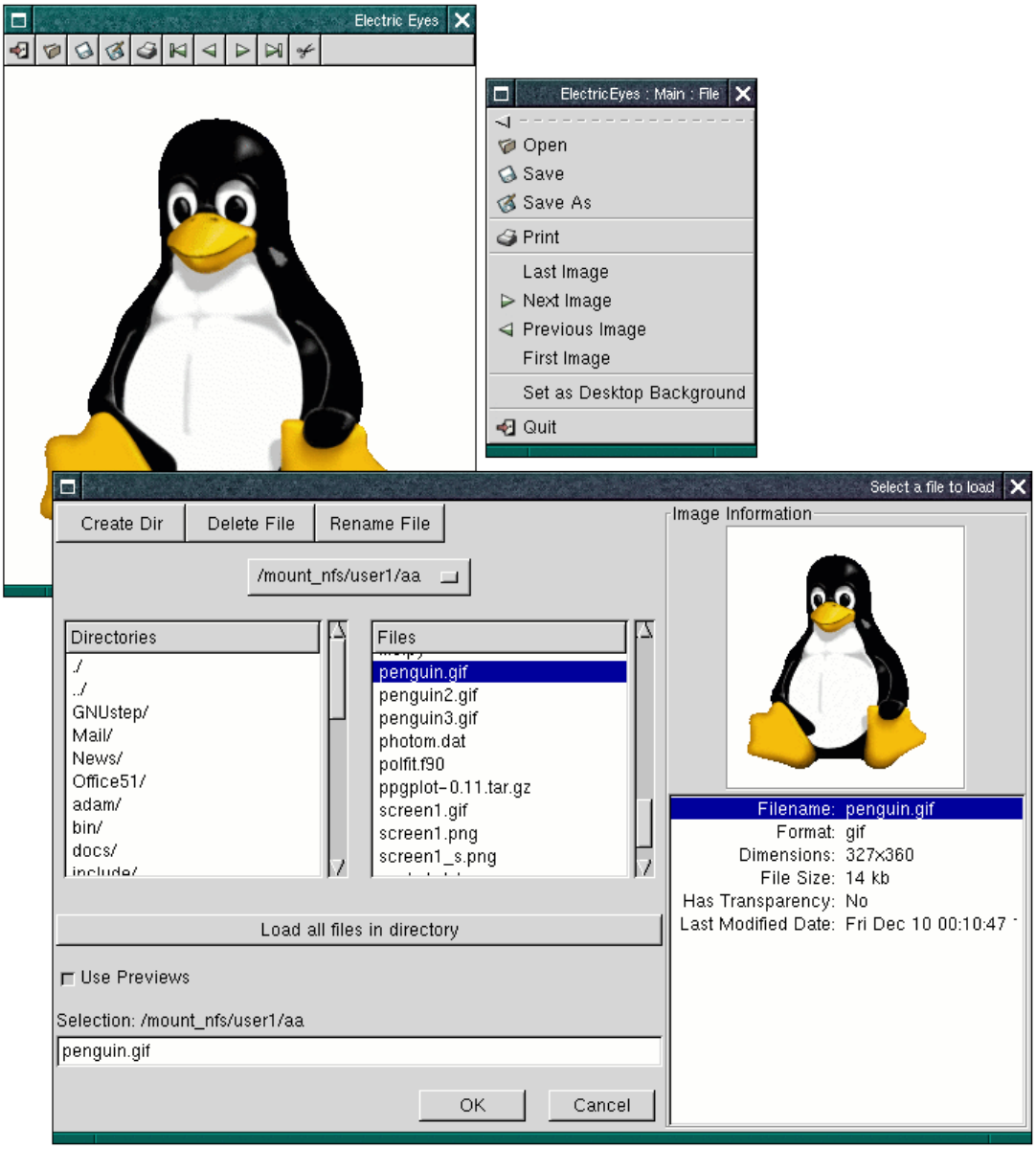


Figure 42: The Electric Eyes interface.

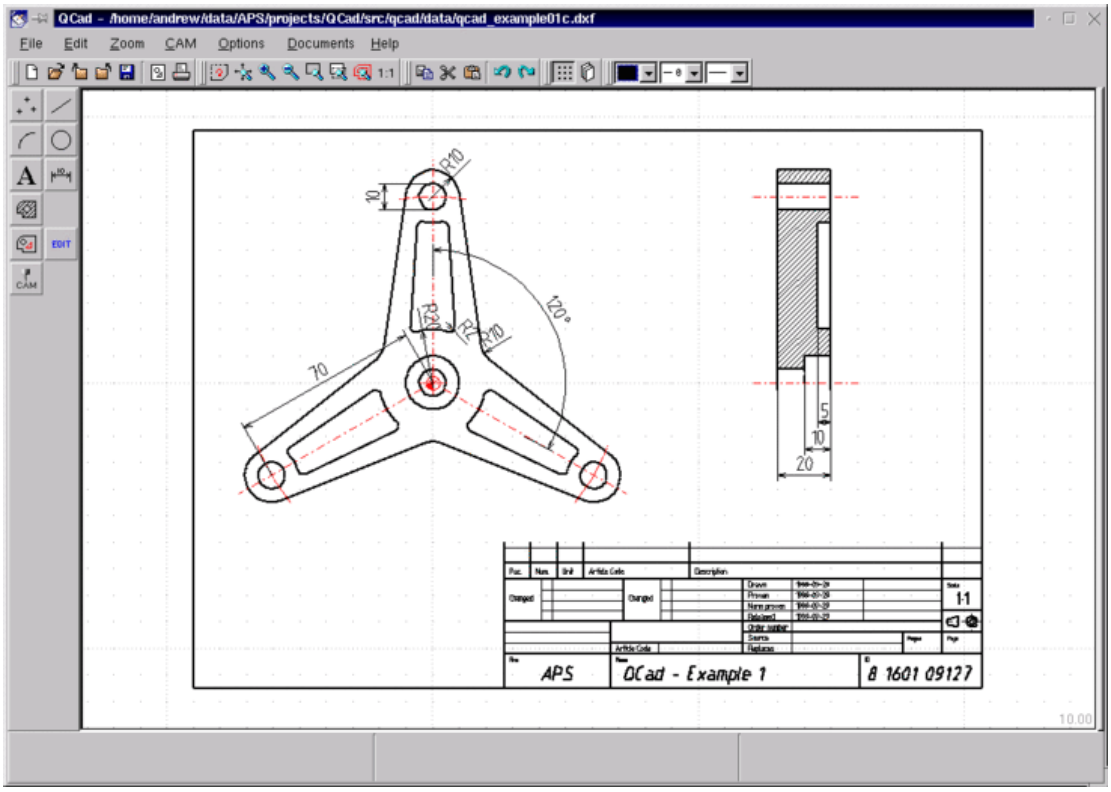


Figure 43: The QCad application running under KDE.

8.0.2 XCircuit

The XCircuit application, while flexible enough to be used as a generic drawing program, is primarly aimed at the production of publishable-quality electrical circuit schematic diagrams and related figures, see Figure 44.

Further information on the XCircuit application can be found on the author's web site at `http://bach.ece.jhu.edu/~tim/programs/xcircuit/`.

9 Format Conversion

9.1 CONVERT

The Starlink CONVERT package is used to convert data files between Starlink's Extensible n-dimensional Data Format (NDF), which is used by most Starlink applications, and a number of other common data formats. Using these utilities, astronomers can process their data selecting the best applications from a variety of Starlink or other packages. The package is discussed in depth in SUN/55.

Starting up the CONVERT package will also set up defaults for the automatic NDF conversion facilities to enable applications which use the NDF library to read and write most of the file formats handled by the CONVERT package.

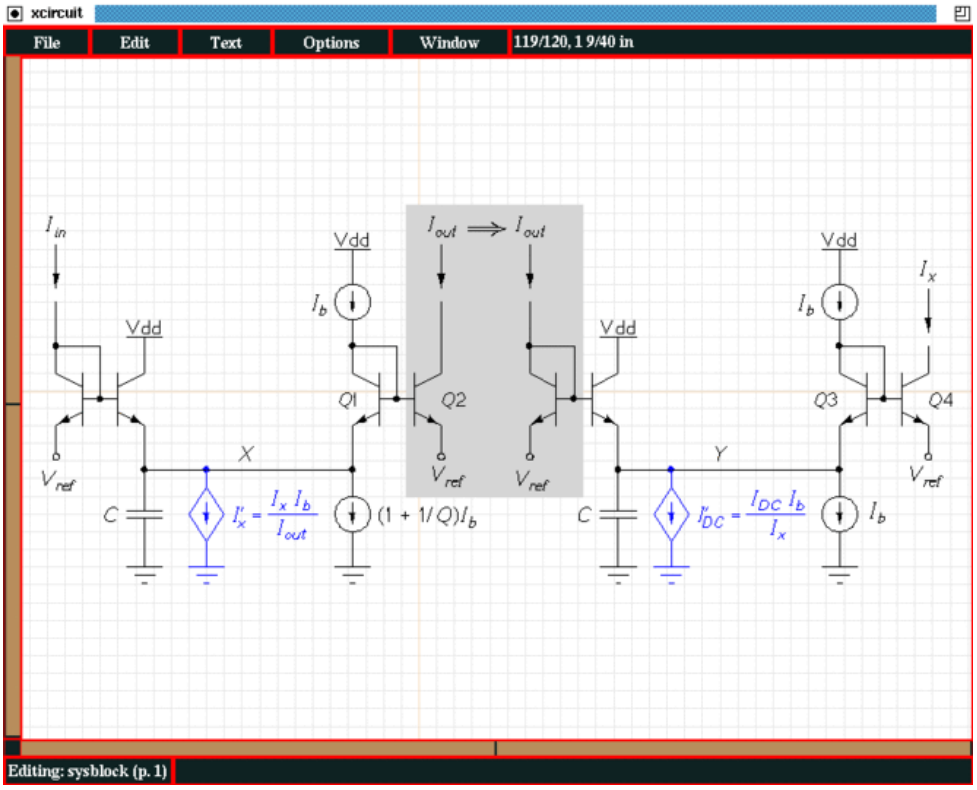


Figure 44: The Xcircuit interface.

An application to convert NDF to a PBMplus format PGM file is one of the conversion utilities available, so further conversion (to formats not handled by CONVERT) can be carried out using the PBMplus package.

9.1.1 NDF2GIF

The `NDF2GIF` application (part of the Starlink CONVERT package, can be used to convert an NDF into the common (and therefore portable) GIF format. Usage:

```
ndf2gif in [ out ] [ scale ] {high=? low=?
                             {percentiles=[?,?], [numbin=?]
                             {sigmas=[?,?]
```

The application converts the imput NDF image into a 256 grey-level GIF image, *e.g.*

```
% ndf2gif image scale=percentiles
```

will convert the file `image.sdf` into a GIF file scaling the image between the values corresponding to two percentiles. In this case, since the thresholds were not provided, the program would prompt for the percentile values between which the image should be scaled.

9.2 PBMplus

The PBMplus package is a command line toolkit for fonverting a large slection of image formats to and from a portable internal format. In addition to the converters the package includes some (simple) tools for manipulation of the images while in the portable format. PBMplus compiles out of the box, however you may want to add support for TIFF, JPEG and PNG format images by installing the relevant libraries, see http://www.acme.com/software/pbmplus/ for details.

A quick example, if you want to convert a Sun raster file to a Postscript file you'd use something that looks like:

```
% rasttopnm < infile | pnmtops > outfile
```

9.2.1 PBM, PGM, PPM or PNM images?

The PBMplus package is split into four distinct suites. PBM programs to handle bitmaps (1 bit per pixel), PGM programs to handle grayscale images, PPM programs to handle full colour images and lastly PNM programs which carry out content-independent manipulations on any of the three internal formats. PBM stands for Portable Bit Map, PGM stands for Portable Gray Map, PPM stands for Portable Pixel Map and PNM stands for Portable Any Map

The suites are upwardly compatible, *e.g.* PGM programs can read both PGM and PBM files, but only write PGM files, and PPM programs can read all three formats, butu only write PPM files. PNM programs will read all three formats and , in general, write out the same type as they read in. Understanding this realtionship is fundamental to understanding the way the package works. for instance if you want to convert an `xwd` to a `gif` file you would read the `xwd` image in using `xwdtopnm` and then convert to `gif` using `ppmtogif`. Since `ppmtogif` is a PPM program it can read whichever of the three formats `xwdtopnm` writes.

9.3 Image Resizing

A common task linked with format conversion is image resizing. Traditionally under UNIX most people tend to use the XV package to carry out this task, unfortunately this package has poor dithering capabilities and resized images tend to be badly pixelated. I'd personally recommend the ImageMagick `display` application which does a superior job of dithering the final resized image to avoid pixelization.

10 Postscript and PDF

Postscript is a page description language. It was introduced by Adobe Systems in the mid-eighties and has become the standard device independent file format for printing graphics files. What this means is that PostScript describes a graphics image in such a way so that it does not make any reference to specific device features (*e.g.* printer resolution) so that the same description (postScript file) could be used on any PostScript compatible printer.

An Encapsulated PostScript File (EPSF or EPS) is a PostScript file structured so that it can be incorporated or included into another PostScript file (so that for example a diagram created with a graphics application can be inserted into a text document created with a word processor).

PDF is another page description language introduced by Adobe to replace PostScript, however it isn't yet in as widespread use as PostScript. For instance its quite hard to find a printer that has a PDF interpreter implemented in hardware, *i.e.* you can not sent a PDF file directly to the printer but must first convert it to PostScript using display software such as Adobe Acrobat.

10.1 Ghostscript

The Ghostscript software suite is an interpreter for the PostScript language, with the ability to convert PostScript language files to many other formats, display them, and print them on printers that don't have PostScript language capability built in. Additionally Ghostscript also functions as an interpreter for Portable Document Format (PDF) files, with the much the same capabilities. Finally the suite also contains a C subroutine library (the Ghostscript library) that implements the graphics capabilities that appear as primitive operations in the PostScript language.

There are actually two different versions of Ghostscript, these being the Aladdin and GNU distributions. The main difference between them *seems* to be the licencing terms, GNU Ghostscript being distributed under the GPL of course with the Aladdin version being distributed under the Aladdin Free Public Licence. The only difference in the licencing terms appears to be that the Aladdin licence does not allow commercial distribution. If you are using Linux you almost certainly have GNU Ghostscript installed due to the licencing issue.

Further information on Ghostscript can be found at `http://www.cs.wisc.edu/~ghost/`.

10.2 GV and Ghostview

Ghostview is a full fuction X Windows interface for the Ghostscript the PostScript interpreter. Ghostview and Ghostscript function as two cooperating programs. Ghostview creates the viewing window and Ghostscript draws in it. The GUI is fairly self explanatory, however the application ships with an extensive manual page (type `man ghostview` at the UNIX prompt).

GV is a version of Ghostview that was modified for VMS, some enhancements made, and then modified to run again under Unix. It is now replacing Ghostview as the standard desktop tool for viewing PostScript files, and is in fact the default viewier in most Linux dsitributions (*i.e.* if you type `ghostview` on a Linux prompt you'll probably actually start the GV program instead). An example of GV in action can be seen in Figure 45. Further information on GV and Ghostview can be found at `http://www.cs.wisc.edu/~ghost/`.

10.3 Acrobat

Adobe Acrobat Reader allows you to view and print PDF files. While the viewer is free, if you want to create PDF content the tools to do so are not. More information is available at `http://www.adobe.com/products/acrobat/readermain.html`. The Acrobat reader is distributed as part of the Staarlink baseset software, and can be started by typing `acroread`.

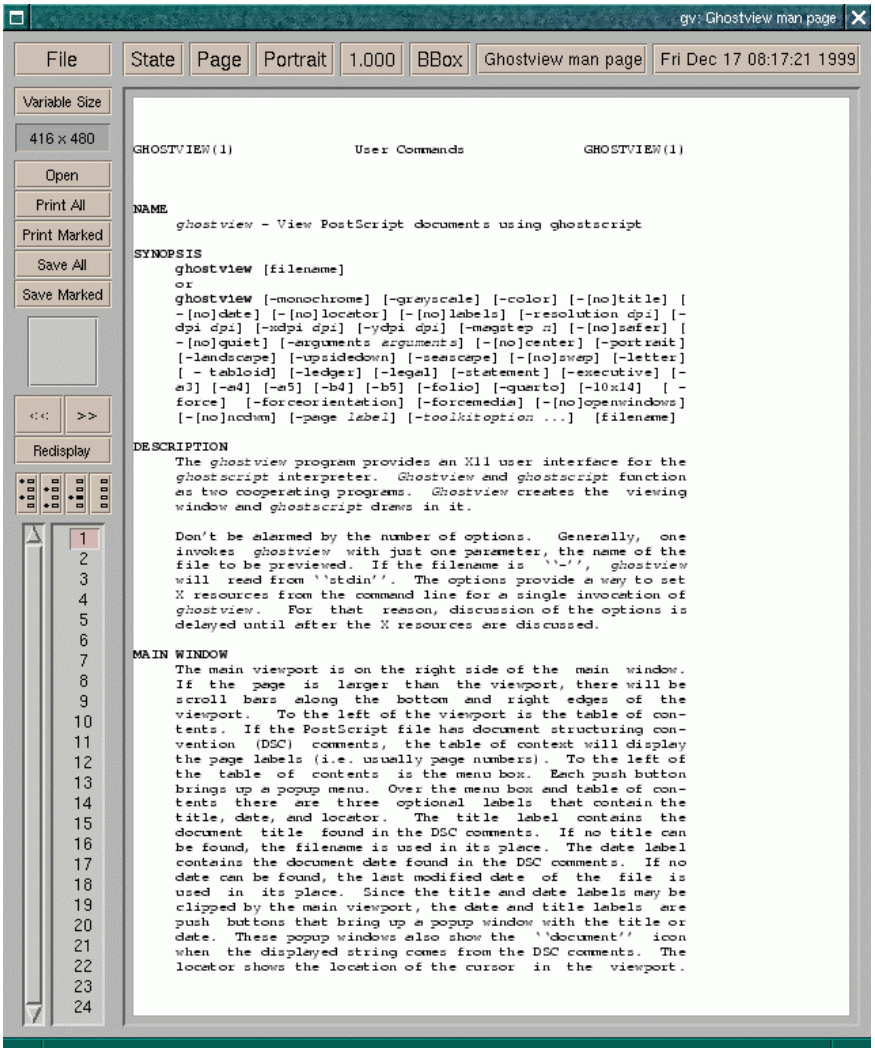


Figure 45: The GV interface.

10.4 psmerge

psmerge is a utility program for merging one or more Encapsulated PostScript Files into a single PostScript file. The input files can be individually rotated, scaled and shifted. The output file can either be Encapsulated PostScript or "normal" PostScript suitable for sending to a printer. The psmerge utility is covered in detail in SUN/164.

10.5 epsutil

epsutil is a utility for manipulating Encapsulated PostScript files. For more information see the manual at http://www.math.utah.edu/~beebe/software/epsutil/epsutil.html.

10.6 prescript and pstotext

prescript extracts text from a PostScript file, storing it either as plain ASCII text, or as HTML according to the mandatory first command-line argument. Usage is:

```
prescript [ html | plain ] [ input.ps ]
```

The output file will be given the same base name as the input file, with its file extension set to one of .html or .txt, according to the first command-line argument.

prescript uses a PostScript interpreter, normally gs, to execute the PostScript program, so that even text that is generated programmatically, rather than being explicitly present in PostScript strings, can be extracted. Particular attention is paid to heuristic recognition of word breaks, to reconstruction of words hyphenated at line breaks, to preservation of paragraph breaks, and to recognition of TEXligatures.

The prescript program can be downloaded from http://www.nzdl.org/html/prescript.html.

A possible substitute for prescript is the pstotext utility. More information can be found at http://www.research.digital.com/SRC/virtualpaper/pstotext.html.

10.7 Postscript to PDF

PDF files can be easily generated using the gs utility using the following command.

```
gs -q -dSAFER -dNOPAUSE -sPAPERSIZE=a4 -sDEVICE=pdfwrite
   -sOutputFile=output.pdf  input.ps
```

10.8 PS Utils

PSUtils, written by Angus Duggan, is a collection of useful utilities for manipulating PostScript documents. Programs included are psnup, for placing out several logical pages on a single sheet of paper, psselect, for selecting pages from a document, pstops, for general imposition, psbook, for signature generation for booklet printing, and psresize, for adjusting page sizes.

- `psbook`

  The `psbook` program rearranges pages from a PostScript document into "signatures" for printing books or booklets, creating a new PostScript file.

  Usage is:

  ```
  psbook [ -q ] [ -ssignature ] [ infile [ outfile ] ]
  ```

  Where `-q` surpresses printing of page numbers below the pages being rearranged (by default page numbers are printed), and `-s`*signature* selects the size of signature which will be used. The signature size is the number of sides which will be folded and bound together; the number given should be a multiple of four. The default is to use one signature for the whole file. Extra blank sides will be added if the file does not contain a multiple of four pages.

- `psnup`

  The `psnup` program puts multiple logical pages onto each physical sheet of paper. The potential use of this utility is varied but one particular use is in conjunction with `psbook`. For example, using `groff` to create a PostScript document and `lpr` as the UNIX print spooler a typical command line might look like this:

  ```
  % groff -Tps -ms file | psbook | psnup -2 | lpr
  ```

  Where file is a four-page document this command will result in a two-page document printing two pages of file per page and rearranges the page order to match the input Pages 4 and 1 on the first output page and Pages 2 then 3 of the input document on the second output page.

  Usage is:

  ```
  psnup [ -wwidth ] [ -hheight ] [ -ppaper ] [ -Wwidth ] [ -Hheight ]
        [ -Ppaper ] [ -l ] [ -r ] [ -f ] [ -c ] [ -mmargin ]
        [ -bborder ] [ -dlwidth ] [ -sscale ][ -+nup ] [ -q ]
        [ infile [ outfile ] ]
  ```

  The `-w` option gives the paper width, and the `-h` option gives the paper height, normally specified in 'cm' or 'in' to convert PostScript's points (1/72 of an inch) to centimeters or inches. The `-p` option can be used as an alternative, to set the paper size to A3, A4, A5, B5, letter, legal, tabloid, statement, executive, folio, quarto or 10x14. The default paper size is A4.

  The `-W`, `-H`, and `-P` options set the input paper size, if it is different from the output size. This makes it easy to impose pages of one size on a different size of paper.

  The `-l` option should be used for pages which are in landscape orientation (rotated 90 degrees anticlockwise). The `-r` option should be used for pages which are in seascape orientation (rotated 90 degrees clockwise), and the `-f` option should be used for pages which have the width and height interchanged, but are not rotated.

  Psnup normally uses "row-major" layout, where adjacent pages are placed in rows across the paper. The `-c` option changes the order to "column-major", where successive pages are placed in columns down the paper.

A margin to leave around the whole page can be specified with the `-m` option. This is useful for sheets of "thumbnail" pages, because the normal page margins are reduced by putting multiple pages on a single sheet.

The `-b` option is used to specify an additional margin around each page on a sheet.

The `-d` option draws a line around the border of each page, of the specified width. If the *lwidth* parameter is omitted, a default linewidth of 1 point is assumed. The linewidth is relative to the original page dimensions, *i.e.* it is scaled down with the rest of the page.

The scale chosen by psnup can be overridden with the `-s` option. This is useful to merge pages which are already reduced.

The *-nup* option selects the number of logical pages to put on each sheet of paper. This can be any whole number; psnup tries to optimise the layout so that the minimum amount of space is wasted. If psnup cannot find a layout within its tolerance limit, it will abort with an error message. The alternative form `-n` *nup* can also be used, for compatibility with other n-up programs. `psnup` normally prints the page numbers of the pages re-arranged; the `-q` option suppresses this feature.

- `psselect`

  The `psselect` program selects pages from a PostScript document, creating a new PostScript file. Usage is:

  `psselect [ -q ] [ -e ] [ -o ] [ -r ] [ -p`*pages* `] [` *pages* `]`
  `[` *infile* `[` *outfile* `] ]`

  Where the `-e` option selects all of the even pages; it may be used in conjunction with the other page selection options to select the even pages from a range of pages, alternatively the `-o` option selects all of the odd pages; it also may be used in conjunction with the other page selection options.

  The `-p`*pages* option specifies the pages which are to be selected. Pages is a comma-separated list of page ranges, each of which may be a page number, or a page range of the form first-last. If first is omitted, the first page is assumed, and if last is omitted, the last page is assumed. The prefix character "_" indicates that the page number is relative to the end of the document, counting backwards. If just this character with no page number is used, a blank page will be inserted in the output.

  The `-r` option causes psselect to output the selected pages in reverse order.

  `psselect` normally prints the page numbers of the pages rearranged; the `-q` option suppresses this. If any of the `-r`, `-e`, or `-o` options are specified, the page range must be given with the `-p` option.

- `pstops`

  The `pstops` program preforms general page rearrangement and selection, creating a new PostScript file. `pstops` can be used to perform a large number of arbitrary re-arrangements of documents, including arranging for printing 2-up, 4-up, booklets, reversing, selecting front or back sides of documents, scaling, *etc.*

  Usage is:

  `pstops [ -q ] [ -b ] [ -w`*width* `] [ -h`*height* `] [ -p`*paper* `] [ -d`*lwidth* `]`
  pagespecs `[` *infile* `[` *outfile* `] ]`

where *pagespecs* follow the syntax:

```
pagespecs = [modulo:]specs
specs = spec[+specs][,specs]
spec = [-]pageno[L][R][U][@scale][(xoff,yoff)]
```

modulo is the number of pages in each block. The value of *modulo* should be greater than 0; the default value is 1. *specs* are the page specifications for the pages in each block. The value of the *pageno* in each *spec* should be between 0 (for the first page in the block) and *modulo*-1 (for the last page in each block) inclusive. The optional dimensions *xoff* and *yoff* shift the page by the specified amount. *xoff* and *yoff* are in PostScript's points, but may be followed by the units 'cm' or 'in' to convert to centimetres or inches, or the flags 'w' or 'h' to specify as a multiple of the width or height. The optional flags L, R, and U rotate the page left, right, or upside-down. The optional scale parameter scales the page by the fraction specified. If the optional minus sign is specified, the page is relative to the end of the document, instead of the start. If page *specs* are separated by '+' the pages will be merged into one page; if they are separated by ',' they will be on separate pages. If there is only one page specification, with *pageno* zero, it may be omitted.

The shift, rotation, and scaling are performed in that order regardless of which order they appear on the command line.

The -w option gives the width which is used by the 'w' dimension specifier, and the -h option gives the height which is used by the 'h' dimension specifier. These dimensions are also used (after scaling) to set the clipping path for each page. The -p option can be used as an alternative, to set the paper size to A3, A4, A5, B5, letter, legal, tabloid, statement, executive, folio, quarto or 10x14. The default paper size is A4.

The -b option prevents any bind operators in the PostScript prolog from binding. This may be needed in cases where complex multi-page re-arrangements are being done.

The -d option draws a line around the border of each page, of the specified width. If the lwidth parameter is omitted, a default linewidth of 1 point is assumed. The linewidth is relative to the original page dimensions, *i.e.* it is scaled up or down with the rest of the page.

pstops normally prints the page numbers of the pages re-arranged; the -q option suppresses this feature.

- psresize

  The psresize program rescales and centres a document on a different size of paper. Usage is:

  psresize [-w*width*] [-h*height*] [-p*paper*] [-W*width*] [-H*height*] [-P*paper*] [-q] [*infile* [*outfile*]]

  The -w option gives the output paper width, and the -h option gives the output paper height, normally specified in 'cm' or 'in' to convert PostScript's points (1/72 of an inch) to centimeters or inches. The -p option can be used as an alternative, to set the output paper size to A3, A4, A5, B5, letter, legal, tabloid, statement, executive, folio, quarto or 10x14. The default output paper size is A4. The -W option gives the input paper width, and the -H option gives the input paper height. The -P option can be used as an alternative, to set the

input paper size. `psresize` normally prints the page numbers of the pages output; the `-q` option suppresses this feature.

10.9 Generating Postscript Output

A common task is to take an image, for instance a GIF or JPEG, and generate a PS or EPS output figure for publication. Depending on which package is used for this task there is a suprising difference between the size of the final postscript image. Of the packages available ImageMagick seems to produce the smallest postscript output files due to its use of vectorised postscript rather than bitmaps which other packages (such as `xv`) use. In extreme cases this can mean the difference between a 2Mb and 50k final postscript file. All the postscript images in this cookbook were generated from the original GIF files using ImageMagick.

11 X Window Displays

11.1 Pseudocolor

The pixel value in the frame buffer selects an entry in a colour table which contains the intensities of the three primary colours (Red, Green, Blue). This is how the traditional image display from the days before X worked and there are typically 8 planes in the frame buffer and therefore 256 entries in the colour table. Each colour table entry typically has 8 bits for each component giving 256 levels of red, 256 of green *etc.* Put another way, at any one time you can have up to 256 different colours out of a palette of 10^{24} ($\sim$16 million) possible colours.

11.2 Grey Scale

This is the same as pseudocolor except that only only one of the primary colours drives the monitor. Since colour monitors are cheap these days (can you get a monochrome monitor for a PC now?) it is of little interest.

11.3 Static Grey

Like Grey Scale but you can't change the colour table—its contents are fixed at some values chosen by either the video adapter manufacturer or the writer of the X server. Even less interesting than grey scale.

11.4 Directcolor

The frame buffer pixel value is divided into three fields; one for each primary colour and each of the fields selects an entry in a separate lookup table each of which holds the intensity of the appropriate primary colour. A typical configuration is a frame buffer with 24 bits per pixel divided into three 8 bit fields each of which addresses a 256 entry colour table with 8 bits per entry. Such a configuration will be described by a manufacturer as supporting 2^{24} (16 million) colours out of a palette of 16 million.

11.5 Truecolor

This is like Direccolor but with the contents of the colour tables fixed.

11.6 What does this mean for you?

Until a few years ago, the best an astronomer could hope for was an 8-bit frame buffer which supported pseudocolor and most applications were written to work with this colour model. Unfortunately, an application that wants to change the colour table has to be aware of whether it is using pseudocolor or directcolor and supporting both in the same application is not trivial—different X lib calls are needed for setting the colour tables in the two models and the way that images are formatted is different. This means that there are still plenty of application in use that display pseudocolor on a directcolor or truecolor display. Most modern hardware supports both pseudocolor and directcolor with a 24 bit frame buffer and you are typically given the choice of either 8 bit pseudocolor or 24-bit directcolor (or 24-bit truecolor—more on this later). But why not 24-bit pseudocolor I hear you ask—the answer lies in the the amount of memory required for the colour tables. For 8-bit pseudo-colour you need (assuming 8 bits per colour) 768 bytes (3×256); for 24-bit directcolor you need the same but for 24-bit pseudocolor you would need a little over 40 Mbytes! (3×2^{24}). So, how is it that direct-colour can give you 16 million colours with less that a Mbyte of colour table while pseudocolor need more than 50 times this to give you the same number of colours? Where's the catch?

Consider what happens when you want to change the colour of something drawn on the screen. When using pseudocolor you change the colour table entry addressed by the pixel value used to draw the object—only things drawn with that pixel value change colour. However, with directcolor this is not true—if, for example, you change the red colour table entry for the selected pixel value not only do the things drawn with that pixel value change but anything drawn with a pixel value with the same value in the red component also changes—and there are 65 thousand pixel values that satisfy this condition. The end result is that if you want to be able to change the colours of things without effecting the colours of anything drawn with a different colour you find that you are back to only 256 colours—exactly what you get with 8-bit pseudocolor (and for 1 third of the video memory).

So, for a typical astronomical image display application which enables you to adjust the colour table interactively you are no better off using a 24-bit display than you are with an 8 bit one. This is not quite true because most applications have some fixed colours and you can potentially use up fewer colour table entries for the same number of fixed colours.

Running X server has a "default visual" (and hence colour model) but may support the creation of other windows with other colour models. However, not all applications are capable of requesting a visual type other than the default. An important exception is applications written in tcl/tk which has a `-visual` qualifier that enables you to select the visual type when you start the application. One undesirable consequence of running applications with different visual types on the same display is "colour flashing" where as the focus moves to one application the display of others is garbled. This effect is only absent on very high-end hardware.

Many X servers allow the default visual type to be specified when they are started (sometimes indirectly by specifying the frame buffer depth—*e.g.* 8 implies pseudocolor while 24 implies truecolor). The choice of truecolor rather then directcolor for the 24-bit mode may, at first, sight seem surprising. The reason is that with 8 bits per colour component in the frame buffer and 8

bits per colour in the colour table you can load the colour tables so that every possible intensity is available simultaneously and applications can draw in every one of the 16 million different colours simply by drawing with the right pixel value. Since the colour tables can't be changed, all applications are guaranteed to have the full palette available—with a directcolor model the available pallet may have been reduced by some other application allocating entries for its own exclusive use. Unless applications are going to change the colours of things after they have been drawn—and most don't (animation of button presses and the like are done by redrawing with a different pixel value)—you are better off with truecolor.

To find out what visual types an X server supports, run `xdpyinfo` (part of the X software) and don't assume that you will necessarily get the same set if you start the server with a different default.

None of the above applies to MS Windows or NT (or Macs)—they handle colour allocation in a quite different way and may well exploit the capabilities of your video adapter rather better—after all it was probably designed to work with Windows from the start.

11.7 Pseudo Colour applications on True Colour desktops

Some applications, including several Starlink supported ones, were designed to be used on Pseudocolor displays. While this is not a problem on the Sun, DEC or NCD hardware, most of which supply a Pseudocolor display by default, the newer Linux machines supplied by Starlink are usually configured to take full advantage of their superior graphics hardware and provide Truecolor displays by default. Trying to use applications which were designed to run uder Pseudocolor displays on a Truecolor desktop may cause the program to display poorly, fail to run, or in extreme cases crash the X server.

There are several approaches to this problem. The least desirable option is to run your Linux X server in Pseudocolor (so called 8 bpp) mode by starting X Windows using the following command:

```
% startx -- -bpp 8
```

this does mean however that you are not utilising your graphics hardware at it best performance. This is somewhat irritating, hence there is a second (better) approach to the problem, you can run two different X servers on the same machine at the same time.

Linux offers the ability to use virtual consoles, these enable you to have several simultaneous sessions on the same machine. You can change between sessions by hitting `Alt-F`$\star$, *e.g.* `Alt-F2` (hold down the `Alt` key and press the `F2` key), or if you are in X Windows `Ctrl-Alt-F`$\star$.

So how does this help us run multiple X servers? Login to your Linux machine as normal and start X Windows using the `startx` command. Once X Windows has booted, hit `Ctrl-Alt-F2` to switch to a second virtual console, and login again. Once you've logged in start a second X server, this one in 8 bpp mode, using the following command:

```
% startx -- :1 -bpp 8
```

You can now switch between the two X servers using `Ctrl-Alt-F7` (the original Truecolor X server on `:0`) and `Ctrl-Alt-F8` (the Pseudocolor X server on `:1`). Applications can even

be started in an `xterm` on one X server and displayed in another by using the `-display localhost:0` and `-display localhost:1` command line options, which most X applications will accept.

You should note that your system administrator may have configured your Linux box to automatically start a Pseudocolor display. If so you can follow the instructions above towards the opposite goal, having a Truecolor display on `:1` by starting the second X server using the following command:

```
% startx -- :1 -bpp 16
```

or if you have more graphics memory (and your X server has been configured to use it):

```
% startx -- :1 -bpp 24
```

There is a down side, since you are now running two X servers you are using twice as much memory, it is inadvisable to try this approach on machines with only a small amount of system memory (RAM). However, yet another additional solution to the problem of Pseudocolor applications and Truecolor desktops is presented in Section 12.1.

12 Virtual Computing

12.1 Virtual Network Computing (VNC)

Virtual Network Computing (VNC) allows the remote display of a computer "desktop" not only on the machine where it is running, but also from anywhere on the internet and on a wide variety of operating systems and hardware architectures.

12.1.1 Pseudo colour displays

As I mentioned in section 11.7 you can use VNC as another way around the problems with Pseudocolor applications on Truecolor displays. Effectively you use VNC to display a self-contained 8-bit Pseudocolor X display on your desktop, programs which require such a display will then run happily in that window, and colour hungry applications which can run in True color can run outside it.

With VNC installed on your system, and the binaries in your `PATH`, to start a Pseudocolor window on any X display you must first start a VNC server using:

```
% vncserver -cc 3 -depth 8

        New 'X' desktop is mypc:1

        Starting applications specified in /home/aa/.vnc/xstartup
        Log file is /home/aa/.vnc/X.log
```

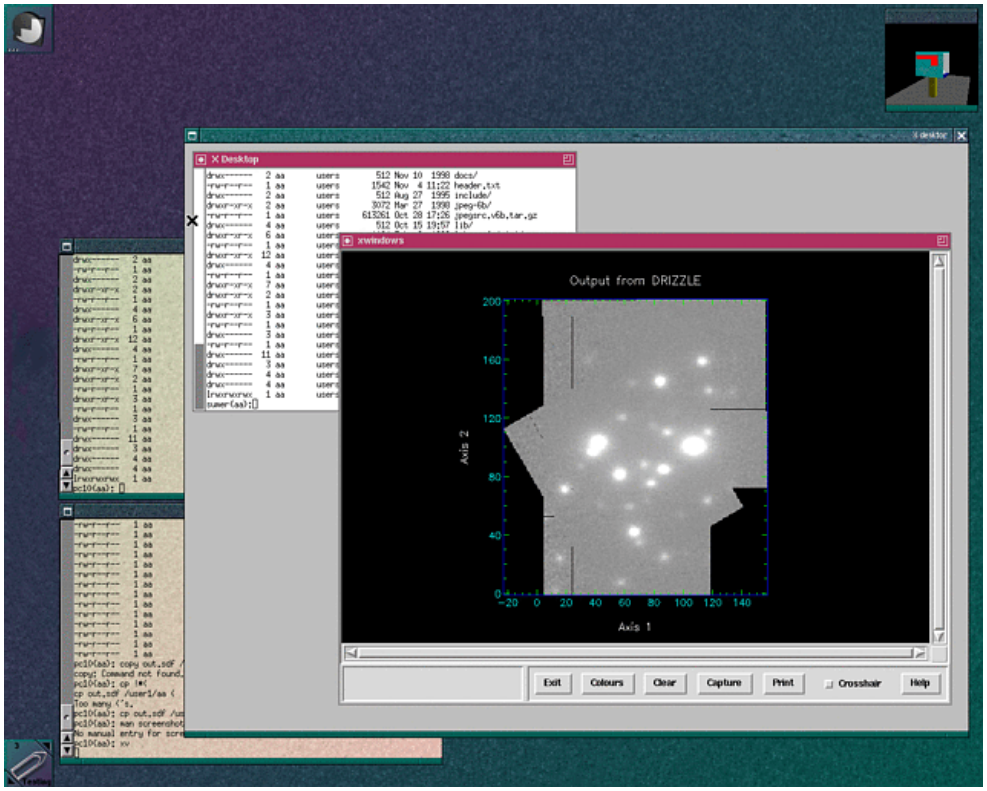


Figure 46: VNC displaying a pseudo colour desktop on a true colour display.

The first time you do this, you will be asked for a password for protecting your VNC sessions. This can be the same or different from the password you use for logging in on that computer. Choose it in the same way as you would a normal Unix password.

You must then start a VNC viewer using the display numebr reported by the preceeding command:

```
% vncviewer :1
        vncviewer: VNC server supports protocol version 3.3 (viewer 3.3)
        Password:
```

At this point type in the password you gave when you first ran `vncserver`. A window will now pop up on your screen, this is a new X Windows desktop. Run Starlink programs from xterms in that window, and they will pop up as sub-windows in that window and run without problem. Outside that window, the X display will carry on working as normal.

When you have finished with the pseudocolour desktop, close down the windows as in a normal X session and terminate the server, using the display number again:

```
% vncserver -kill :1
```

Figure 46 illustrates several aspects of the VNC system. Here we show a Pseudocolor VNC X Windows desktop (1024×768 pixels) running the `twm` window manager on a DEC Alpha using KAPPA to display a CCD image. The VNC desktop is being displayed ontop of a True colour X

Windows desktop (1280×1024 pixels) running the `WindowMaker` window manager on a Linux PC.

You can customise the size of the pop up desktop by using the command line option `-geometry`, *e.g.*

```
% vncserver -cc 3 -depth 8 -geometry 1024x768
        New 'X' desktop is mypc:1

        Starting applications specified in /home/aa/.vnc/xstartup
        Log file is /home/aa/.vnc/X.log
```

While the applications automatically started along with the `vncserver` can be customised from the `~/.vnc/xstartup` script (created the first time `vncserver` is run) in your home directory. By default this script starts one `xterm` and the `twm` window manager. By editing `~/.vnc/xstartup` this behaviour can be modified; by copying or symbolically linking your `.xsession` or `.xinitrc` to it the VNC window can be given the same behaviour as your normal X session.

Additionally, it is not necessary to kill the VNC server at the end of a session, or indeed ever; the only important thing is not to leave an ever increasing number of unused servers running on the system. If it's more convenient, you can simply close the VNC viewer window at the end of a session, and next time you wish to use it start `vncviewer` again, without having to rerun `vncserver`, and your desktop will be exactly as you left it.

12.1.2 Computing by remote control

VNC also comes in other flavours, with versions for many different operating systems. If you think hard enough you can come up with lots of different possible uses, but one that has proved useful for teaching is the remote control of single user machines.

If you run a VNC server on a MS Windows PC and display the screen on your UNIX workstation (or even another Windows machine) you can control the machine remotely using the `vncviewer`, if you move the mouse inside your VNC window the mouse cursor on the actual machine will move with it. If you start a VNC server on every machine in a teaching lab then you can flick between them monitoring student progress, useful for remote diagnostic purposes.

Figure 47 illustrates this setup, showing a MS Windows 98 desktop (1024×768 pixels) being displayed ontop of a Truecolor X Windows desktop (1280×1024 pixels) running the `WindowMaker` window manager on a Linux PC.

12.2 VMWare

What is VMWare? This is actually a fairly complicated thing to explain, although it is obvious whats going on when you see the application running. VMware is software that runs multiple virtual computers on a single PC – at the same time – without partitioning or rebooting. On top of these virtual machines you can then run the OS of your choice. In other words you boot your machine into, say, Linux which VMware refers to as the "host" operating system. You then run the VMWare virtual machine, which can run a variety of other operating systems, *e.g.* MS Windows, as a so called "guest" OS. This proceedure is illustrated in Figure 48, while an example of the application in action is shown in Figures 49 and 50.

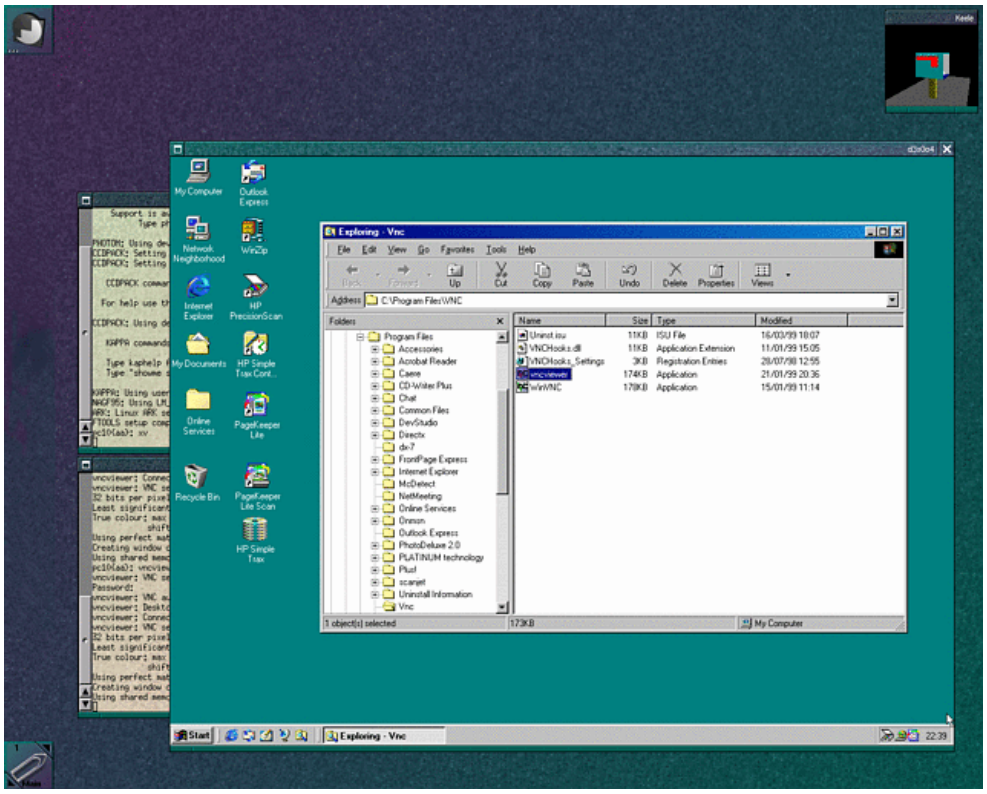


Figure 47: VNC displaying a Windows 98 desktop on a X Windows display.

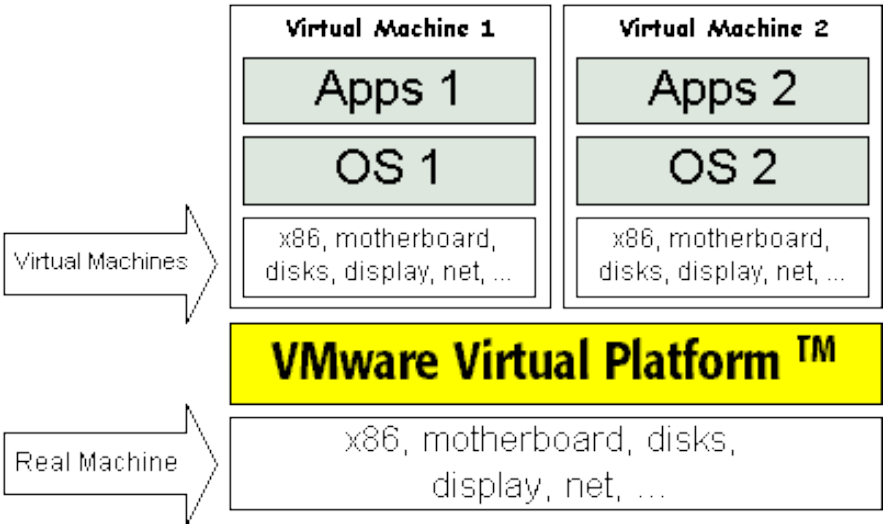


Figure 48: Schematic showing how VMWare is used as a layer between the real and virtual machines.

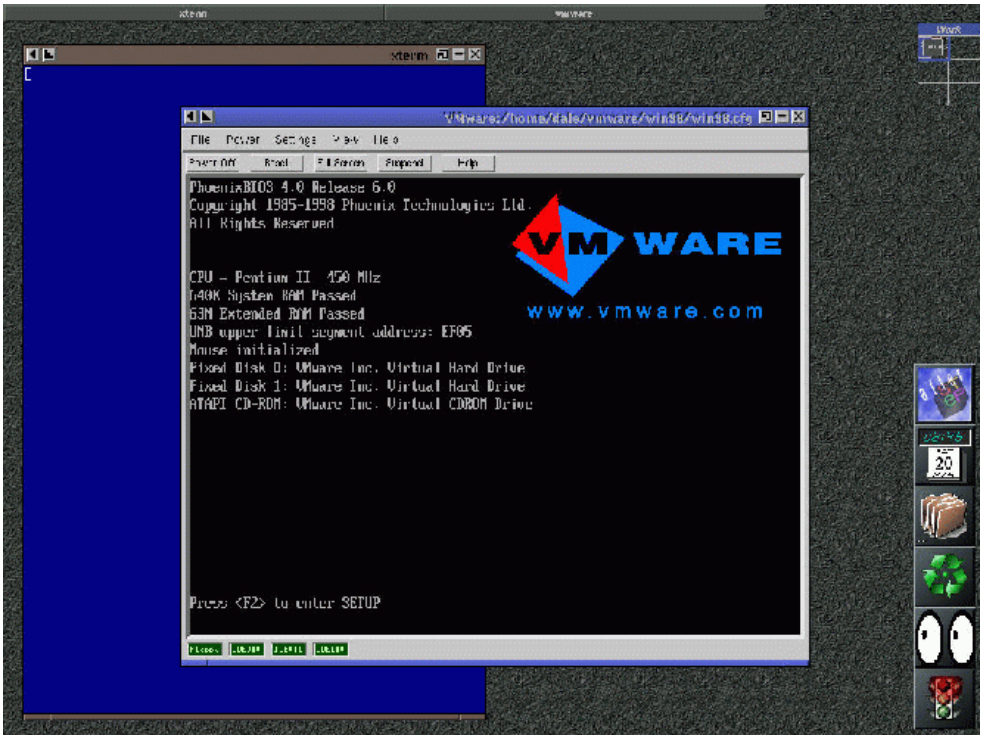


Figure 49: VMWare booting the virtual machine using Linux as the "host" OS.

This is not an emulator, you are not emulating the CPU or the hardware inside the virtual machine, you are just allowing another operating system to use it in parallel with the one you already have running. Each virtual machine can have its own IP number (if your machine is on a network) and you can treat it exactly as it it was another physical computer. Of course you don't really have two computers so your guest operating system ends up sharing resources with your host operating system. Basically, the more RAM you have when using VMWare the better, although I've quite comfortably used it on machines with as little as 64Mb and gotten away with it.

This is not a boot manager; boot managers and VMware are complementary. Boot managers help you select one of several available operating systems and boot it on the computer. Only one operating system is ever running at one time, and moving to another requires rebooting the system. Adding an operating system also typically requires repartitioning a disk. By contrast, VMware allows multiple operating systems to run simultaneously on the same computer. VMware can work with existing disk partitions on an IDE drive or can support new operating systems in VMware logical disks without the need to repartition a disk.

VMWare supports two host operating systems, Linux and Windows NT, and a variety of guest opertaing systems including DOS, Windows, Linux and BSD dervided operating systems. Support for OS/2 and BeOS as guest operating systems is likely to be introduced eventually.

Although it is fairly cheap for academic purchase at US$99.00 per user licence, VMWare is not free. More information about VMWare can be found online at `http://www.vmware.com`.

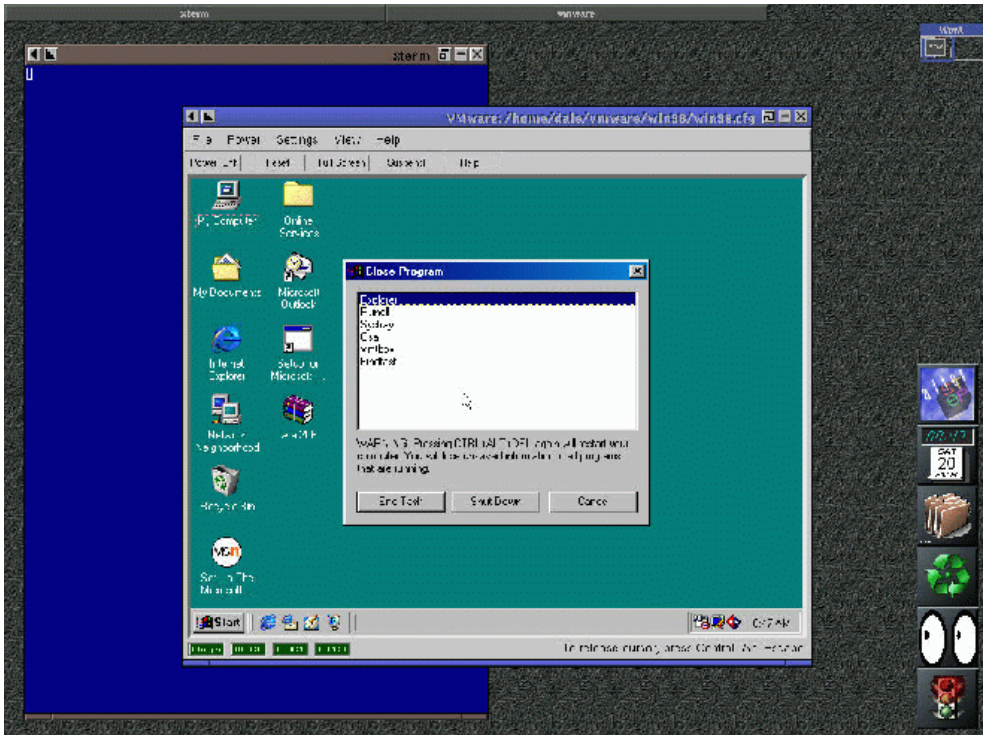


Figure 50: VMWare running Windows 98 as a "guest" OS, using Linux as the "host" OS.

12.3 plex86 (previously FreeMWare)

There is an open source competitor to VMWare, called plex86 (previously called FreeMWare). Since the last edition of this cookbook coding has advanced rapidly and is now actually able to run DOS 6.22 as a guest operating system. However the virtual machine only makes it part way through booting Linux, so for most purposes the application is yet to reach the "usable" stage. More information can be found at `http://www.plex86.org/`.

12.4 The VMWare and plex86 Patent Position

The GIF legal problems are not the only ones in the software world. Very recently a new player in the virtualisation game has appeared called VOS, unknown until recently they have apparently filled broad ranging patent applications across the discipline. If legitimate this casts a shadow across the continued development of both VMWare and plex86. On the other hand the VOS web site looks somewhat suspicious and the current opinion in the community is that it might be a hoax, and since they are currently taking credit card orders, a malicious hoax at that. Even if this is not the case, there is a great deal of prior art in the field and it seems unlikely that these patent applications will be upheld in court. However, for anyone that is a heavy user of one of the main virtualisation products, VMWare or Bochs for instance, this is something worth following fairly closely.

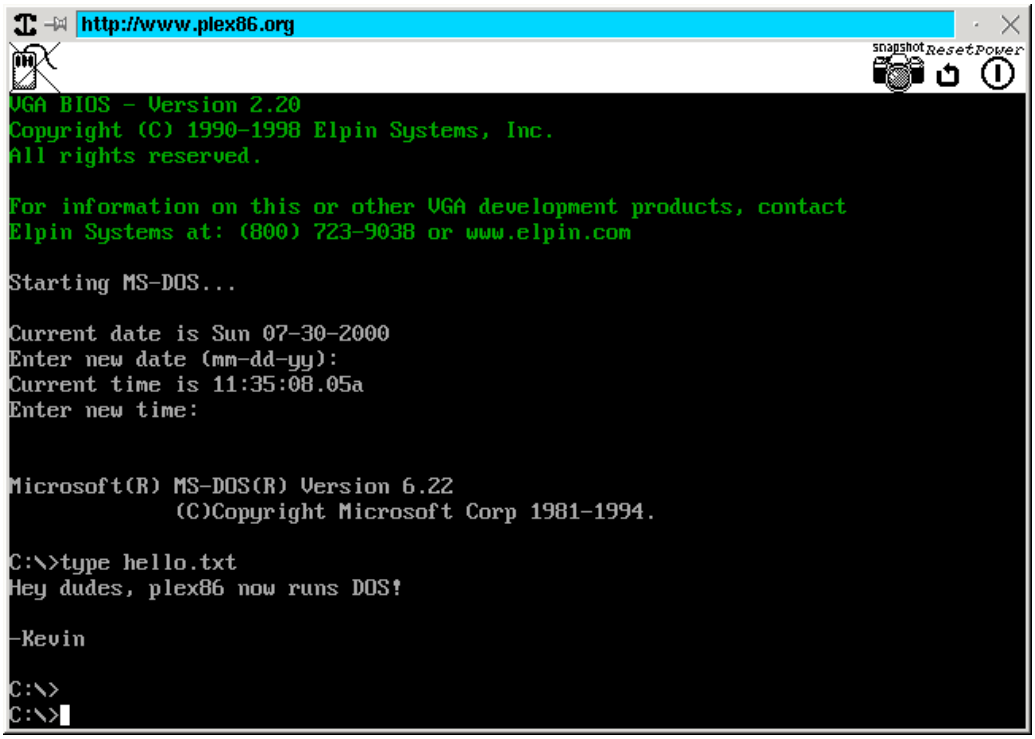


Figure 51: plex86 running DOS 6.22 as a "guest" operating system.

13 Hardware

13.1 Scanners

Only a limited amount of the available scanner hardware is supported under UNIX, if you want to use a scanner with a UNIX workstation (or Linux PC) it is important to investigate the availability of hardware drivers for your chosen platform. SCSI scanners are currently the most heavily supported hardware, support for parallel and USB scanners is much patchier.

The SANE Project attempts to provide a standardised application programming interface (API) for access to any raster image scanner hardware (flatbed scanner, hand-held scanner, video- and still-cameras, frame-grabbers, *etc.*).

The SANE package is divided into two parts: a selection of *backend* hardware drivers which support scanners from different Manufacturers, and a variety of different *frontend* GUI applications that sit ontop of the *backend* drivers.

SANE has been ported to a variety of platforms including Linux, Solaris and Digital UNIX, although the Digital UNIX port is still in need of debugging and is untested. More information on the SANE Project can be found at `http://www.mostang.com/sane/`.

13.2 Digital Cameras

The GNU Digital Camera Project supports over 90 different digital camera models with the `gphoto` program under both Linux and BSD operating systems, see Figure 52. The program is

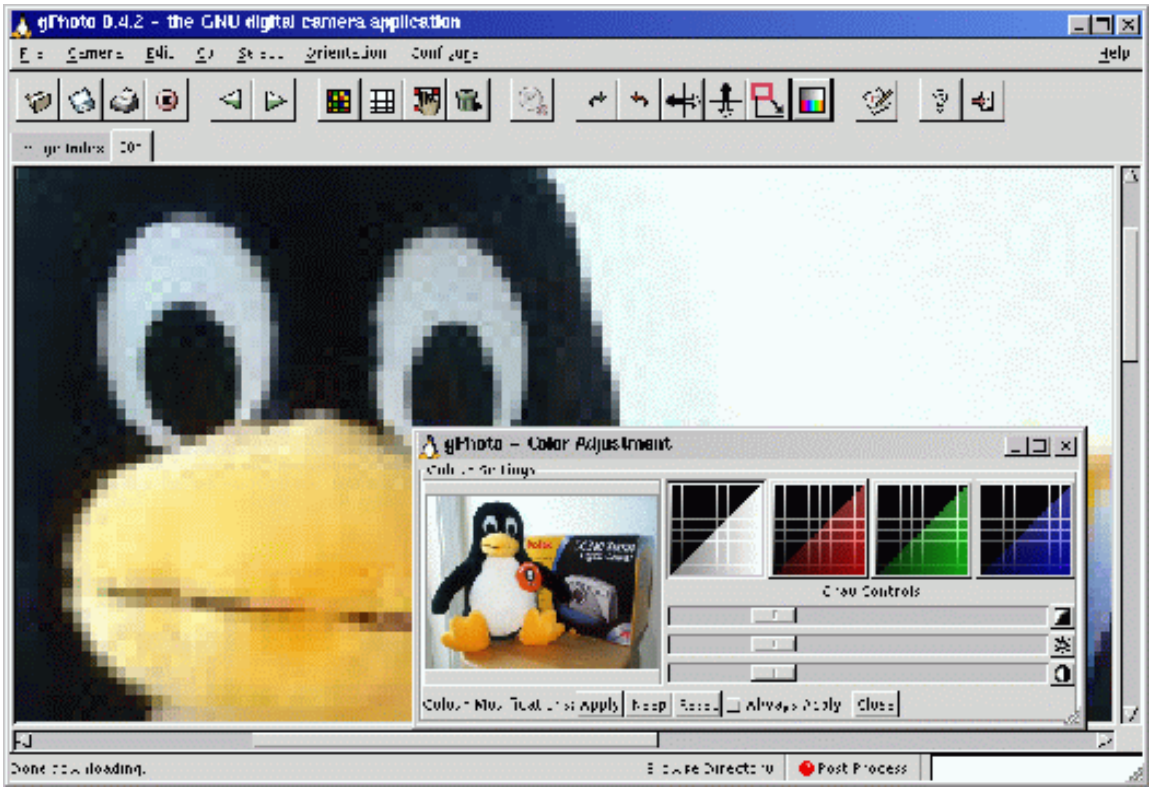


Figure 52: The `gphoto` interface.

known to run under RedHat Linux 6.1, SuSE Linux 6.2, Linux Mandrake 6.1, Debian GNU/Linux 2.1, FreeBSD 3.2, NetBSD 1.4.1 and OpenBSD 2.6. The package requires GNU `sed`, GTK+, `glib` and `imlib`. If you are running Linux these packages will likely already be installed on your system, however if they do not appear to be present then they can be found at the following sites:

- GNU `sed`
  `ftp://gnudist.gnu.org/pub/gnu/sed/`
- GTK+
  `ftp://ftp.gtk.org/pub/gtk/`
- `glib`
  `ftp://ftp.gtk.org/pub/gtk/`
- `imlib`
  `ftp://www.rasterman.com/pub/enlightenment/enlightenment/`

The `gphoto` package can be downloaded from `http://www.gphoto.org/gphoto/download.html`, and is available as source code or binary RPMs. If you want to install the package it is recommended that you download the latest stable version as a binary RPM, and ask your system administraor to install the RPM on your system. Configuration is straight forward, plug your camera into a spare serial port and select the appropriate `COM Port` and `Camera Model` from the configuration menu. To download images from your camera, go to the camera menu and select the `Get Index` option.

14 The Web

14.1 Transparent GIFs

There are many equally valid ways of generating a GIF with a transparent background index. For instance using ImageMagick, first display your GIF image using the `display` program. Choose MATTE from the IMAGE EDIT command menu and identify a pixel with the cursor that has the colour you wish to make transpart. From the new menu select METHOD and choose the most appropriate method:

- **point**
  The point method changes the matte value of the selected pixel
- **replace**
  The replace method changes the matte value of any pixel that matches the color of the pixel you selected
- **floodfill**
  The most useful, floodfill changes the matte value of any pixel that matches the color of the pixel you selected and is a neighbour.

Select your transparent pixel with the pointer and press a button. The image is redisplayed with any transparent pixels recolored to the background color. You can select other pixels or areas to force to transparent. When you are satisfied, press `Return`.

Alternatively you can do this from the command line using the `giftrans` program available via anonymous FTP from `ftp://ftp.rz.uni-karlsruhe.de/pub/net/www/tools/`. The `giftrans` program is distributed as part of the `STAR2HTML` Starlink package and should therefore be available on Starlink supported systems.

To get a list of the current colourmap for the image use:

```
% giftrans -l file.gif
Global Color Table:
    Color 0: Red 0, Green 0, Blue 0, #000000 (black, gray0, grey0)
    Color 1: Red 90, Green 90, Blue 0, #5a5a00
    Color 2: Red 123, Green 123, Blue 0, #7b7b00
    Color 3: Red 156, Green 156, Blue 0, #9c9c00
    Color 4: Red 189, Green 189, Blue 0, #bdbd00
    Color 5: Red 255, Green 255, Blue 0, #ffff00 (yellow, yellow1)
    Color 6: Red 222, Green 231, Blue 222, #dee7de
    Color 7: Red 255, Green 255, Blue 255, #ffffff (white, gray100, grey100)
```

this shows the colour index, the RGB colour value in decimal/hexidecimal and (for some colours) an X Window colour name.

To set colour index zero a the transparent colour you would then use:

```
% giftrans -t 0 file.gif > out.gif
```

You can also specify the color as an RGB triple or an X Windows color name; invoke `giftrans` with the -? option to see a complete usage description.

If you intend to use GIF images on your web pages, you should make yourself aware of the legal position before proceeding.

14.2 Animated GIFs

Again there are may perfectly valid ways of going about this, see for instance the WhirlGIF program, discussed in Section 7.8.

Alternatively we can, make use of the ImageMagick `convert` application with the `-delay` and `-page` options. The `-delay` option is used to specify the delay in 1/100*th* of a second between the display of each frame of the animation. For example:

```
% convert -delay 20 frame*.gif animation.gif
```

You can also declare specific delays for each frame of the image sequence. For example, if the delay was 20, 10, and 5, use:

```
% convert -delay 20 frame1.gif -delay 10 frame2.gif -delay 5 \
  frame3.gif animation.gif
```

Use `-page` to specify the left and top locations of the image frame:

```
% convert frame1.png -page +50"+1"00 frame2.png -page +0"+1"00 \
  frame3.png animation.png
```

If you want the image to loop within Netscape, use `-loop` option, for instance:

```
% convert -loop 50 frame*.png animation.png
```

You can also use the `convert` application in the opposite sense to split a GIF animation into individual image files, *e.g.*

```
% convert animation.gif frame%02d.gif
```

The resulting image files are titled `frame01.gif`, `frame02.gif`, `frame03.gif`, *etc.*

If you intend to use GIF images on your web pages, you should make yourself aware of the legal position before proceeding.

14.3 Beveled Images

Again, most packages including the GIMP and ImageMagick will allow you to add borders to your image to make buttons.

The simplest method is using the GIMP. Make sure you are working on an RGB image (using IMAGE→RGB) and then select SCRIPT-FU→DECOR→ADD BEVEL. You'll be queried as to the width of the bevel in pixels, and it will then be automagically generated for you.

14.4 "Web Safe" Colour Maps

Netscape predefines 216 colours for colour mapped (pseudo colour) workstations. When dithering an image into Indexed mode, the GIMP asks whether you want to use a WWW optimised palette. If you do so the image will be dithered to use the netscape colour map. ImageMagick has similar functionality implemented using the `convert` command, *e.g.*

```
% convert -map netscape: alpha.gif beta.gif
```

14.5 Browser support of PNG images

Due to the legal problems surrounding GIF images the PNG image standard has been put forward as its replacement. Unfortunately support for PNG images is still pretty patchy in the main stream browsers such as `netscape`. Details of the extent of PNG support implemented into the different browsers are listed at `http://www.libpng.org/pub/png/pngapbr.html`.

15 The GIF Legal Position

The Lempel-Ziv-Welsh (LZW) compression algorithm is patented by UniSys. This algorithim is used in the GIF image standard to store the image data inside the GIF image, and because of this software which creates GIFs are subject to licensing fees by UniSys. However Unisys has refused to issue licences to open-source software producers for the use of LZW.

While this contraversy has being going on for several years, Unisys recently (late 1999) raised the stakes by stating that its policy is to require a $5000 fee (so called Intranet or Billboard Web site license) from web sites, even non-commerical web sites, that carry GIF images made by unlicensed software. If you make use of GIF images on your web site that have been generated with an unlicenced piece of software you may be guilty of "contributory infringement". Alarmingly, the LZW compression algorithim is also used in the popular PDF format. Open source software written to handle PDF files is therefore also at risk from this decision.

While it is debatable whether the patent covers LZW decompressors, while the Open Source community take the view that it does not, Unisys argues otherwise. It is possible you may be liable if you distribute code which implements LZW decompression.

For a history of the patent controversy, see `http://lpf.ai.mit.edu/Patents/Gif/Gif.html` and `http://www.cloanto.com/users/mcb/19950127giflzw.html`. To avoid legal problems, it would be a good idea to convert all GIFs on your web sites to PNGs or JPEGs.

When does all this nonsense go away? The basic U.S. patent on the LZW algorithim expires in June of 2003, however patents on variants of the basic algorithim run for another 20 years and further U.S. applications are pending.

16 From the Quick Archives

16.1 FITS to MPEG

A question that has come up more than once as a QUICK request is how to turn a series of FITS files into a movie. Antony Holloway has written a script to automatically convert a series of FITS images into an MPEG movie. The script makes use of the Starlink CONVERT and the Berkely `mpeg_encode` packages.

The script, along with copies of the `mpeg_encode` source and documentation is available via anonymous FTP:

- `fitstompeg` script
  `ftp://ftp.astro.keele.ac.uk/pub/aa/mpeg_encode-script.tar`
- `mpeg_encode` documentation
  `ftp://ftp.astro.keele.ac.uk/pub/aa/mpeg_encode-docs.ps`
- `mpeg_encode` for Linux
  `ftp://ftp.astro.keele.ac.uk/pub/aa/mpeg_encode-linux.tar.gz`
- `mpeg_encode` for Digital UNIX
  `ftp://ftp.astro.keele.ac.uk/pub/aa/mpeg_encode-osf.tar.gz`
- `mpeg_encode` for SunOS/Solaris
  `ftp://ftp.astro.keele.ac.uk/pub/aa/mpeg_encode-sunos.tar.gz`

The `mpeg_encode` program should be installed and accessible somewhere in your `$PATH`, the `fitstompeg` script and `default.param.head` file should be copied into the directory containing the FITS file you want to convert into a movie. By default the script will take all the files in the current directory whose filename ends in "`.FITS`" and convert them into PGM images using the Starlink CONVERT program. At this stage if you have the `pbmplus` package installed (which should be the case for most Starlink sites) you can then modify the output PGM files before they are used to produce the final animation. Two examples of such tinkering are shown in the script (shown below).

```
#!/bin/csh

#+
#  Name:
#     fitstompeg

#  Purpose:
#     Convert a series of FITS images to PGM format and from these
#     generate an mpeg animation file.

#  Description:
#     The script tries to find the files ( *.FITS) which must be in
#     the correct order when listed with ls. The program then generates
```

```
#     the pnm files in the local directory and finally produces the
#     mpeg animation.

#  Authors:
#     AJH: Anthony Holloway (Starlink, Manchester)
#     {add_further_authors_here}

#  History:
#     1-DEC-1998 (AJH):
#       Original version.
#     {add_further_changes_here}
#-

# Source Starlink setup scripts

source /star/etc/login
source /star/etc/cshrc

# Enable Convert commands

convert

# Convert FITS files to NDF

echo "Converting FITS files to NDF"

fits2ndf in="*.FITS" out="*"

# Convert and count each NDF to PGM format, animframeX.pgm

echo "Converting NDF files to PGM format"

set i = 1

foreach file (*.sdf)

    set outname = $file:r

# Original attempt - fails on BITPIX -32 files
#   fits2pnm $file >! $outname".pnm"

    ndf2pgm in=$outname out="animframe"$i".pgm"

# Optional delete of input file
#    rm -f $file


# Optional pgm image manipulation
# NB: Final image must be animframe$i.pgm, hence mv command
# e.g. Scaling by a factor 0.25
#
#    pnmscale 0.25 animframe$i.pgm > animframe-s$i.pgm
#    mv animframe-s$i.pgm animframe$i.pgm
```

```
    # e.g. Change the contrast of the images
    #    pgmnorm animframe$i.pgm > animframe-s$i.pgm
    #    mv animframe-s$i.pgm animframe$i.pgm

        @ i = $i + 1

    end

    @ i = $i - 1

    echo "Total number of frames is $i"
    echo "Editing the default.param file to set this value"

    # Edit the mpeg_encode parameter file to set the number of frames

    cp default.param.head default.param
    echo "animframe*.pgm  [1-$i]" >> default.param
    echo "END_INPUT" >> default.param

    # Run the mpeg encoding command

    mpeg_encode default.param

    # Optional delete of PGM files
    # rm -f animframe*

    # end
```

17 Package Availability

- angif
  http://phil.ipal.org/freeware/angif/
- libjpeg
  ftp://ftp.uu.net/graphics/jpeg/
- libpng
  http://www.libpng.org/pub/png/pngcode.html
- libungif
  http://prtr-13.ucsc.edu/~badger/software/libungif/index.shtml
- **PGPLOT**
  http://astro.caltech.edu/~tjp/pgplot/index.html
- **PGPERL**
  http://www.aao.gov.au/local/www/kgb/pgperl/
- **BUTTON library**
  http://www.ucm.es/info/Astrof/button/button.html

- **PLplot**
  http://emma.la.asu.edu/plplot/
- ptcl
  http://www.InfoMagic.com/~nme2/ptcl/ptcl.html
- **QDP**
  http://heasarc.gsfc.nasa.gov/docs/software/ftools/ftools_menu.html
- **PONGO**
  http://www.starlink.ac.uk/
- **VNC**
  http://www.uk.research.att.com/vnc/
- zlib
  ftp://ftp.freesoftware.com/pub/infozip/zlib/index.html
- gphoto
  http://www.gphoto.org/
- **SANE**
  http://www.mostang.com/sane/
- mpeg_encode
  ftp://mm-ftp.cs.berkeley.edu/pub/mpeg/encode/
- **GTK+**
  http://www.gtk.org
- **PyGTK**
  http://www.daa.com.au/~james/pygtk/
- **GIMP-Python**
  http://www.daa.com.au/~james/pygimp/
- **Python Imaging Library**
  http://www.python.org/sigs/image-sig/Imaging.html
- **PBMplus**
  http://www.acme.com/software/pbmplus/
- gd **Library**
  http://www.boutell.com/gd/gd.html
- **XPaint**
  http://home.worldonline.dk/~torsten/xpaint/
- **PSUtils**
  http://www.dcs.ed.ac.uk/home/ajcd/psutils/index.html

Acknowledgments

In compiling this document I have leant heavily on already available material, usually the packages manual or other documentation, in all cases links to the original sources have been included. Some of the material on VNC was based on a draft SUN written by Mark Taylor. The image of Tux the penguin used throughout this cookbook was created by Larry Ewing (`lewing@isc.tamu.edu`) using The GIMP.