FastPlot

COLLABORATORS			
	TITLE : FastPlot		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 30, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

FastPlot

1.1 FastPlot Guide

1.2 Disclaimer

2) FastPlot is distributed in its original form. No changes are made and this documentation file and the supplied examples are distributed with it.

3) FastPlot is not distributed as part of a commercial product, unless you have the written permission of the author. Also, FastPlot is not distributed on a disk or other data carrier which contains commercial data as well.

4) The author can not be held responsible for damage or data loss, which are caused by failures in the program, incorrect use of the program, errors in the manual or any other reason. All responsibility remains with the user of the software.

5) FastPlot is Freeware. This means you need not to pay for using it. However, if you appreciate this program and like to see new versions or other improvements, it is not forbidden to send cash of any kind to the author. If you have any bug reports or suggestions for improvement then they are even more welcome and you can send me a letter to my address.

1.3 Introduction

For some time now I have been searching for a general-purpose plotting library that is both easy to use and offers fairly sophisticated functionality which can produce professional-looking graphical plots. Commercial packages are generally too specific (i.e., business graphics only) to be used in a wide range of applications and also are fairly pricy. AREXX plotting libraries would be another alternative but they are interpreted and thus are considerably slower than a custom software package. In addition, the plotting capability should be easy to integrate into other programs with very simple calls and a minimum of set up. The PlotLibrary package meets these goals with the following features:

- produces bar charts, linear, curved, scatter, logarithmic and semi-log plots
- fast due to the use of a compiled language (Modula-2)
- any number of plots on a custom screen with up to 32 colours
- automatic x- and y-axis labelling or user-definable labels
- automatic grids
- automatic local minima/maxima labelling
- information box
- user-definable text/line colours

Some sample plots produced by the PlotLibrary routines are shown in Figures 1 to 5. The program in Listing 1 produced these different plots. Listing 2 is the complete source for the PlotLibrary module. Listing 3 shows the source for a support module called StringUtils.

The remainder of this article demonstrates how to create your own plots by walking through the program in Listing 1 and describing the PlotLibrary routines, as needed, to give a basic understanding of how to use them.

1.4 Starting FastPlot Demo

From the Workbench, just double-click on the TestPlots icon. A set of hard-wired sample plots will be displayed with about a five delay between plots. Once you've whetted your appetite, read on to see how easily you can create your own plots.

1.5 Creating Your Own Plots

The first step is to call the InitPlot plot routine with a set of arguments which define the plot characteristics as follows:

```
PROCEDURE InitPlot(
  VAR Plot                : PlotType;
      MainTitle               : ARRAY OF CHAR;
      PlotKind                : PlotKindType;
  xMin, xMax, yMin, yMax : REAL;
  width, height           : INTEGER;
  xDiv, yDiv              : CARDINAL;
  xSubDiv, ySubDiv        : CARDINAL;
  xDec, yDec              : CARDINAL;
  NumberOfColours         : CARDINAL)
            : BOOLEAN;
```

where the Plot is a variable which is used as a handle to identify a specific plot; MainTitle is the title which appears at the top of the plot; PlotKind is one of

1) normal (Figure 1) plot continuous curves,
2) line (Figure 2) plot points connected with line segments
3) bar (Figure 3) bars of various heights,
4) log x (not shown) plot the log of the x-axis;
5) log y (not shown) plot the log of the y-axis;
6) log-log (Figure 4) plot the log of the x- and y-axis;

xMin, xMax, yMin, and yMax define the real-world plot window; width and height define the screen dimensions of the plot in pixels; xDiv and yDiv give the number of divisions in the x and y directions where 0 values disable the grid; xSubDiv and ySubDiv give the number of subdivisions in the x and y directions; xDec and yDec give the number of decimal places used for x and y labels; and NumberOfColours gives the maximum colours desired on a plotsubject to some limitations discussed later.

Scatter plots (Figure 5) are special in that they are produced by plotting character symbols at each graph point instead of joining the adjacent graph points with line segments. These plots are described in more detail later. Figure 6 illustrates the difference between the real-world coordinate system and the pixel coordinate system inherent in the Amiga's graphic utilities. In effect, the real-world coordinate system defines the domain in which the plotted functions live. For example, the plotted sinusoidal wave in Figure 1 has an x-range (xMin to xMax) of 0 to 7 and a y-range (yMin to yMax) of -1 to 1. This real-world space is mapped onto the Amiga screen within the pixel space defined by the width and height parameters passed to the InitPlot function by the xyToCoords mapping function in Listing 2 .

If InitPlot returns a TRUE value, the plot has been successfully initialized and the plot can be drawn using the PlotFx procedure. This routine takes two arguments: the first is the plot handle which was defined by the InitPlot function; the second is a function which takes a real number (x) and returns some value F(x). Eight different functions are defined in Listing 1 : DampSine, Cubic, RandomSine, AmpModulated, Response, StockPrices, ProfitValues, and ProfitValues2. As you can see from these examples, the processing performed by F(x) varies somewhat depending on the kind of plot to be drawn. Normal, scatter, log, and semi-log plots can accept any continuous function; the bar chart and linear plot require a discrete function which returns one value for each bar or point on the linear plot.

Congratulations, you have successfully produced your first plot! And it took only two calls to routines in the PlotLibrary! Of course, this first plot can be enhanced in a number of ways which include labeling the plotted minima and maxima, displaying an information box, changing text, line, and screen colours, adding your own x-axis labels, and plotting multiple functions on a single axis. I'll describe each of these special effects next.

1.6 Special Effects

I remember being in school and having to determine the local minima and maxima of functions by taking the first derivative of the function. Well, the LabelMinMax routine does exactly that and then labels the resultant points right on the plot. LabelMinMax takes the same arguments which you also passed to the PlotFx procedure. You usually call LabelMinMax right after calling PlotFx but the calling order could be reversed if you want the plotted curve to overlay the minimum/maximum point labels. LabelMinMax works with any of the plot types defined in the PlotLibrary. For the scholarly reader, the local minima and maxima are determined by looking for sign reversals of the derivative of the plotted functions. In school you solved this same problem by determining where the derivative (or slope) was zero; however, with computers, tests against zero don't always work so the sign reversal approach gives more consistent results.

The InformationBox routine provides a powerful mechanism for displaying details about the plotted function. The preceding plots demonstrate the use of the information box. To use the information box, define an array of strings (LineType is exported from PlotLibrary for this purpose) which contains enough elements to hold all the lines of display text. For example, to display five lines you would declare:

```
Message : ARRAY [1..5] OF LineType;
```

and initialize this array with the information to be displayed in the information box. This information box can be positioned in the upper or lower plot area and can be centered, left-justified, or right-justified in the plot region. Both the information box's background and outline colours are specified in the call to InformationBox; text colour is set by the SetTextColour routine described below. The ScientificPlot routine in Listing 1 whose plotted output is shown in Figure 1 demonstrates how to use the InformationBox routine.

Four routines handle the colour manipulation of the plotted output.

SetColourMap changes which of the 4096 available colours get mapped to a subset of colours available for the plot screen. An eight-colour screen would allow you to select eight of the 4096 colours to be displayed at one time. Each call to SetColourMap sets one screen colour, so eight calls are necessary to fully define all the colours for the example screenalthough unused colours don't need to be mapped. Defining names for the colour indices helps you to remember which colour is mapped where. The declarations in Listing 1 define a set of constants like Red = 5 so that a call to SetColourMap(Plot, Red, 11, 0, 0) can be used with a colour name instead of a number. The last three arguments define the amount (0 to 15) of red, green, and blue, respectively, for screen colour six (the first colour is zero). These eight colours are used to define the text colour (SetTextColour), the plotted curve colour (SetPlotColour), and the grid colour (SetGridColour). In each case, these procedures take, as an argument, the colour number (from 0 to 7 for an eight-colour screen). A maximum of 32 colours can be used with a plot screen width of 320 pixels; 16 colours for a 640 pixel plot screen width.

The x- and y-axis titles are positioned using the CenterLabelX and CenterLabelY procedures. Both these routines automatically center the text horizontally for CenterlabelX and vertically for CenterLabelY. You only need to specify the vertical position (in pixels) for the x-axis label and the horizontal position (also in pixels) for the y-axis label. The horizontal
pixel numbers increase from left to right and the vertical pixel numbers increase from top to bottom. Some experimentation may be required to perfectly position the titles, although the positions shown in the BarChart routine in Listing 1 (395 for x label and 15 for y label) give a good starting point. Note that these labelling routines are not restricted to axis titles but could also be used for centering other information anywhere on the plot.

LabelX and LabelY are two more general labelling procedures which give you control over both the x and y position of plot text. The only difference between the two is that LabelX produces a horizontal line of text while LabelY produces a vertical text display. The Amiga operating system (V1.3) does not allow text to be rotated so the vertical text is composed of unrotated characters, each offset vertically from the others. Rotating characters for the y-axis is a non-trivial task I'll leave for another time.

The last text labelling procedure, SetLabelRoutine, lets you override the numerical labels which are automatically generated by the PlotLibrary. The example in Figure 3 uses this routine to produce month labels for a bar chart. This example passes the BarLabels procedure (Listing 1) to the SetLabelRoutine just before calling the PlotFx procedure. The passed procedure must have an interface identical to the one shown for the BarLabels procedure. The first argument identifies the plot division to be labelled and the second argument returns the corresponding label string. Currently, only the x-axis labels can be overridden, although it should be fairly simple for you to add y-axis custom labels by duplicating the source code shown for the x-axis custom labels.

Another procedure, SetScatterPlot, overrides the line-drawing modes of the standard plots and allows a character to be plotted instead for each point of the original graph. The default character is an 'O' but you can change to any other character using the SetScatterChar routine. The plotted scatter character colours are changed with the SetPlotColour procedure. (Figure 5) ↩
shows
the output of the ScatterPlot procedure from Listing 1 .

The BarChart procedure in Listing 1 illustrates a method of plotting two data

sets on the same coordinate axes. The first bar chart is created normally as was outlined above with a slight twist: the SetPlotOffset routines is used to offset the bars by four pixels to the left of where they would appear by default. The second bar chart uses a second data set which is offset four pixels to the right of the default position using the same SetPlotOffset procedure just before calling the PlotFx routine. This bar chart also demonstrates the power of passing a function (ProfitValues or ProfitValues2) which is called during the plotting operation since colours are set dynamically as the graph is plotted. In this case the plot line colour is changed with SetPlotColour to produce black and grey bars for a positive profit and red and pink bars for a negative profit. Figure 3 shows the resultant bar chart.

The SetPlotOffset procedure is used to give pixel offsets to reposition the bar charts described above. A negative pixel offset passed to the xoff argument shifts the plotted function to the left; a positive offset shifts the plot to the right. A negative pixel offset passed to the yoff argument shifts the plotted function up; a positive yoff offset shifts the plot down.

Any number of plots of any kind can be overlaid on the same plot as long as their real-world coordinate systems are equivalent (i.e., both bar charts had the same January to December x-axis and profit in thousands of dollars for the y-axis). Overlaid plots are usually not offset from each other except when you want to produce three-dimensional effects or bar charts. By adding offsets in the ShadowPlot routine in Listing 1 , and changing the line colour, a plot with a shadow for emphasis is produced Figure 7 .

You now know how to overlay plots which share the same world coordinates, but what about producing plots which use different world coordinate systems? Fortunately, there is a simple solution if the SetPlotLimits procedure is used. This routine modifies the mapping of real-world coordinate points to the pixel coordinates required by the Amiga's display. By specifying different plot limits from those originally passed to the InitPlot routine, we can dynamically change our window size to accommodate larger or smaller plotted functions in the same plot window.

A related procedure called SetPlotScale in Listing 2 allows us to magnify the centre of the plot by an arbitrary amount.

The ScaledPlot procedure (Listing 1) provides an example which does just this. A cubic curve is shown at different magnifications all on the same set of axes Figure 8 .

The x- and y-axis labels correspond to the first plot and only represent a scaled version of the magnified plots. The plot scale factors passed to the SetPlotScale procedure must be values which are greater than 0.1. Numbers less than one actually reduce the plot size while numbers greater than one enlarge the plot. In this case the cubic function's x- and y-axis have been selectively magnified by two. The enlarged plots have been magnified more than can be shown in the plot window to demonstrate the line clipping algorithm used in the PlotLibrary.

As a final example, four independent plots Figure 9 are shown on the same display screen. A procedure called InitOffsetPlot is used by the QuadPlot procedure in Listing 1 to place multiple scaled plots on the same screen. The initial steps are the same as before. A plot is initialized using InitPlot. This first step creates the canvas for the following four plots so the maximum number of colours and maximum pixel screen size need to be specified in this

call. The second step requires a call to InitOffsetPlot with a reduced pixel area about 1/4 of the total screen area originally specified to give room for four plots. The call is to a new routine:

```
PROCEDURE InitOffsetPlot(
VAR Plot               : PlotType;
    PlotKind               : PlotKindType;
xMin, xMax, yMin, yMax : REAL;
width, height          : INTEGER;
xDiv, yDiv             : CARDINAL;
xSubDiv, ySubDiv       : CARDINAL;
xDec, yDec             : CARDINAL;
xOffset, yOffset       : CARDINAL;
OldPlot          : PlotType)
          : BOOLEAN;
```

The three last arguments in this procedure differentiate this routine from the call to InitPlot where xOffset and yOffset specify the plot offset relative to the pixel coordinate origin at (0,0) in the top left screen corner; and OldPlot is the plot variable initialized when InitPlot was called. The number of x and y divisions should also be cut in half since less text is visible in the reduced display area. The subdivisions have also been eliminated to avoid cluttering the smaller plots.

1.7 Enhancements

A number of enhancements can be made fairly easily to the PlotLibrary to extend its capabilities. These are:

1) Automatically determine the plot minimum/maximum points for the y-axis so the user doesn't have to know these values.

2) Allow text files containing columns to numbers to be used as the function to be plotted.

3) Use rotated and scaled text.

4) Implement a pie chart plot.

These new capabilities require a bit more work, but all these additions would extend the usefulness of this package so that it even rivals the commercial plotting systems. If there is enough interest, I'll tackle these additions in a future article.

1.8 Compiler Dependencies

The PlotLibrary source code was compiled using the M2S compiler and any dependencies on this compiler have been flagged in the source. Specifically, a call to OpenRealTrans is required to grant access to the Amiga's transcendental floating point function libraryother compilers may provide this access automatically. All other Modula-2 specific calls should be available in other compilers. The Amiga system module names and functions may differ slightly in

your compiler but should be similar to those used here. I have provided a StringUtils package to give a string length function and allow conversions from floating point to strings. Both these operations are included because they are required for the PlotLibrary and may not be provided with every Modula-2 compiler. If you have them available, feel free to substitute your own.

If your favourite language is C, you should have no problem translating from Modula-2 to C and several conversion programs can even perform the translation automatically.

1.9 Conclusions

The PlotLibrary provides a set of routines which greatly simplifies the process of visualizing data for students, scholars, business executives, or experimenters. Simple plots can be produced with just two calls but more elaborate capabilities are available to produce an information box, label plot minima and maxima, place text and titles, overlay plots, and display multiple plots on a single screen. The built-in capabilities are powerful enough to unleash the imagination of the PlotLibrary user. Have fun!

1.10 Comments and bug reports

If you have any comments on this program, suggestions for improvements or if you have found a bug, please send me a letter at:

Michael Griebling
c/o Computer Inspirations
150 Clark Blvd., Suite One
Brampton, Ontario
Canada, L6T 4Y8

email: mgriebling@bix.com

P.S. I would be pleased to hear that people are using this program and like it, so if you do, please send me email, a letter, or postcard.