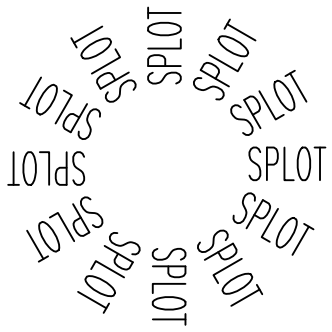


Splot Manual

T.W. Steiner

1994

T.W. Steiner, 312 - 1230 Haro St.
Vancouver, BC, V6E - 4j9, Canada
e-mail steiner@sfu.ca

Contents

Chapter 1

Introduction

Scientific plotting programs fall into two main categories. The first is the non interactive type represented by most mainframe plotting programs which require that the plot be specified as a file of plotting instructions. This file is edited using a text editor and then the plot is generated by executing this file. The second major type of plotting programs are those popularized on PCs which allow a user to specify the drawing using a mouse and are inherently interactive. This type of program is easier to use and it is possible to rapidly make changes. It is however not ideal since the drawing is usually stored in a proprietary, non-text format which is difficult to interpret. This makes the task of reusing parts of a previous drawing in a new drawing or simply plotting new data in an old figure difficult or impossible. Furthermore, these programs invariably do not have the flexibility to handle the needs of preparing scientific plots. Splot has been developed with the goal of bridging these two disparate approaches in an attempt to gain the benefits of both.

Splot is geared towards generating two dimensional, publication quality scientific plots as painlessly as possible without sacrificing flexibility and power. Splot contains much of the functionality of Postscript™ and can thus be used as a general purpose drawing tool. Furthermore, it has functions specifically aimed at producing plots of x,y data pairs. Splot generates a drawing by interpreting a C program which specifies the drawing. Unlike traditional mainframe plotting programs this happens quickly since the interpreter and editor are integrated into splot. Specifying the plot in a powerful programming language has several advantages. Since C is currently the most popular programming language in existence many people will already be familiar with its use. This makes C a much better choice than developing a specialized language to be interpreted. Having the drawing specified by a programming language also gives algorithmic capabilities thus allowing repetitive components of a drawing to be generated using a standard loop construct. Data to be plotted can also be manipulated before drawing using the built in standard C math library. C also allows parts of a drawing that are to be used more than once to be specified as a subroutine. This sub component can then be drawn anywhere in the final drawing by first specifying a translation, scale and rotation and then calling the subroutine. Lastly, the description of the drawing is in plain ASCII text allowing it to be easily understood and transferred electronically. Splot contains a built in drawing and plotting function library whose members are called by the C interpreter to generate the drawing.

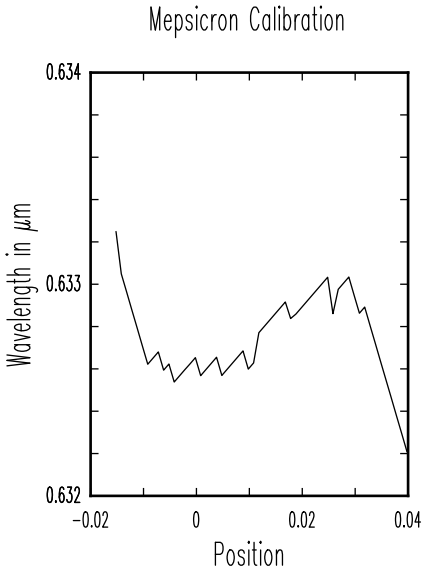
Chapter 2

Using Splot

2.1 Splot Basics - Getting Started

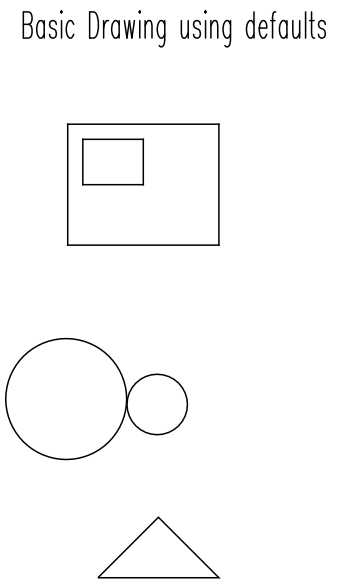
When splot is started the user is presented with two windows. The left window is the drawing area where the image will be drawn. On the right is a text window were the C program specifying the drawing can be displayed. The text is edited using the built in 'e' editor and the drawing is generated by 'executing' the file, i.e. by hitting CTRL G or selecting the menu item 'exec' at the top. Splot can be used for drawing, graphing data or both, and since it is basically C programming, has essentially all the C programming capabilities. To introduce you to the program and demonstrate the basic concept, three simple examples are given below. To begin, an example of a simple splot program to read in and plot a data file using splot default parameters is given. The resultant drawing is shown next to it.

```
/* demo of simplest possible data */
/* plotting using all the defaults */

#include <splot.h>
double *data;
main()
   {
   readdata("demo\data1.dat",data);
   plotdata(data);
   label(LOWER,"Position");
   label(LEFT,"Wavelength in !m!m");
   text(6.20,22.98,"Mepsicron Calibration");
   }
```



Here is a simple example of a drawing using some of the built-in functions and the corresponding drawing.

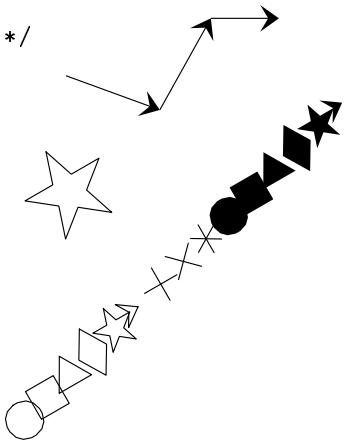
```
/* file basic.spt */
/* Demo of a simple drawing using */
/* some built-in functions */

#include <splot.h>
main()
   {
   /* draw two boxes */
   box(6.0,20.0,11.0,16.0);
   box(6.5,19.5,8.5,18.0);
   stroke();
   /* draw two circles */
   /* input coordinates with mouse */
   arc(5.95,10.91,2.0,0,360);
   stroke();
   arc(8.96,10.73,1.0,0,360);
   stroke();
   /* draw a triangle */
   moveto(7.0,5.0);
   lineto(9.0,7.0);
   lineto(11.0,5.0);
   lineto(7.0,5.0);
   stroke();
   /* add some text */
   text(4.45,23.22,"Basic Drawing using defaults");
   }
```



Lastly, here is an example using some C programming capabilities such as loops.

```
/* This demo file shows the available symbols */
/* which can be used for data points. Symbols names */
/* equating to the numerical constants are defined */
/* in splot.h */

#include "splot.h"
main()
   {
   int i;
   /* make the symbols large. Since they are just a special font */
   /* make the font width large. */
   set(FONTWIDTH,4);
   /* change the symbol orientation just for fun */
   set(FONTDIR,30);
   moveto(1,1);
   /* show all 17 symbols in a diagonal line */
   for (i = 0;i < 17;i++)
      {
      /* use a relative move here */
```

```
    rmoveto(1,1);
    symbol(i);
    }
/* try other form of symbol */
set(SYMMULT,2);
symbol(4,12,OSTAR);
set(SYMMULT,0.5);
/* add some extra arrows to illustrate the */
/* arrowto() function */
moveto(3.82,17.19);
arrowto(7.96,15.68);
arrowto(10.21,19.72);
rarrowto(3,0);
stroke();
}
```



There are two ways a drawing can be made.

1 . by typing in a C program text file using the built-in editor.

2 . by using the mouse and pull-down menus to insert commands into the text file (see section 2.1.3).

Depending on which is more convenient for the task at hand, one can either draw interactively with the mouse or enter text. The drawing is only generated upon execution of the text file.

The text editor that is built into Splot is a fully featured editor with some very powerful capabilities. Ordinary operation should, however, be very intuitive. For a description of the text editor component of Splot refer to Chapter 8. Only a brief introduction to give you the basics to get going is presented here. The text cursor represented by an underscore in the text window can be moved using the arrow key pad or the mouse. Clicking the left mouse button positions the text cursor at the mouse cursor location. Scrolling is accomplished with the page up/down keys or by clicking on the scroll fields along the right side of the text window. A file is written to disk with the name specified on the highlighted status line at the bottom of the text window with a CTRL W (write) command (or use the save menu item). To remove a file use the CTRL Q (quit) command (or use the quit menu item). Splot itself terminates when the last file is removed. To add a new file to the ring of files to be edited or executed type '`<esc> e file.nam`' where file.nam is the name of the desired file or select the open menu item.

2.1.1 Making a Drawing

In order to make a drawing it is useful to understand the underlying philosophy which is based on the postscript imaging model. A very brief introduction to the postscript imaging model will be presented here. To make a postscript like drawing it is first necessary to generate a current path. A path is built using functions such as `moveto(x1,y1);` and `lineto(x2,y2 );`. These functions by themselves do not cause anything to appear on the page. Once a current path is specified then it can be stroked or filled. A path is stroked to the page using the current line width, style, and colour with the `stroke();` function. This function also resets the current path to null. The current values of width etc. are set using the `set(attribute,value);` function were it is advisable to use the predefined values in splot.h for the attributes and values to improve readability of the code (The easiest thing is to select them from the pull down menu). Alternatively a closed current path can be filled with the current colour using the

`fill();` command. This also resets the current path to null. At this point a virtual page in memory has been marked but the actual physical page has not yet been affected. This allows subsequent drawing commands to draw over what is already there. The "paint" is in effect opaque. This means that it is possible to draw over something with the colour set to white and in effect white it out. This can be put to good use for fixing blemishes as shown in some of the example files. The virtual page is transferred to the physical page using the `showpage();` or function or at the end of file execution by default. However, only things within the current clip window are transferred to the physical page. The crucial points to remember are that in order to see anything on the physical page (either on the screen or a printer page) a path must first be stroked or filled and then finally transferred using `showpage();`. This imaging model is very powerful and allows essentially anything to be drawn given the a suite of path building functions. More details on these and other postscript issues can be found in a standard postscript reference.

2.1.2 Making Plots

Splot also contains some custom functions specialized for generating scientific plots. These, such as `plotdata( );`, depart from the postscript model in the sense that they are implicitly stroked. Implicitly stroked functions do not require a `stroke();` in order to mark the page. It is possible to override implicit stroking by starting with a `newpath( );` command.

There is a function called `readdata();` which reads in tabular data from a data file into an array. The data file must be in ASCII format with one line per data point with the x, y and optional error values each in separate columns. This data can then be drawn inside an axes box using `plotdata();` if the default tick positions, size and labelling are acceptable. Alternatively, custom axes boxes, tick marks and labels can be generated using the `axes_box();`, `tickmarks();`, and `ticklabel( );` commands followed by a `drawdata();` or `plotdata();` to draw the data points found in the data array. The difference between `drawdata();` and `plotdata();` is that `plotdata();` will pick default values for the axes box, tickmarks or ticklabels if not previously explicitly specified. The plotted data uses the current line colour, style and width to connect the data points if an ordinary plot was selected. Alternatively, the current symbol can be drawn at each data point. There is a function which automatically draws error bars as well as one which fits and draws the best straight line to a set of data points. The data points actually plotted are easily restricted to those within a range of x or y values. The data can also easily be shifted and scaled before plotting.

2.1.3 Using The Mouse

Clicking the left mouse button on a window changes the current focus if the window does not already have the input focus. On the drawing window the left mouse button is also used to draw the rubber zoom box by holding down the left mouse button. In the editor window double clicking brings up help for the item under the cursor. Help exists for function names, constants and global data values. Double clicking the left mouse button with the mouse on the drawing page inserts the current x,y coordinates into the text file at the current text cursor location. If the mouse was positioned at the beginning of a coordinate pair that coordinate pair is replaced with the new one. If the mouse cursor is in the text region then hitting the left mouse button will move the current text cursor to the mouse cursor location. The left mouse button is also used to select items in drop down menus.

The right mouse button is used for highlighting elements of a drawing. If the mouse cursor is on top of a displayed element on the drawing page and the right mouse button is hit then the corresponding element is highlighted and the text cursor jumps to the corresponding line in the text that generated that element. Conversely, if the mouse cursor is on the text window hitting the right mouse button will highlight any graphics elements generated by the line under the mouse cursor. This feature is very useful for finding the code associated with a particular element of a drawing or the reverse. For the pull down selection lists, the right mouse button opens a help window for those items for which help is available.

When the mouse cursor is on the drawing page the current coordinates are printed at the top. The units and origin used are those in effect at the end of the last program execution. The default units are

cm with the origin in the bottom left corner of the page. Pressing the middle mouse button (or both the left and right mouse buttons together on two button mice) with the mouse pointer on the drawing page causes the current coordinate reference point to be reset so that the status line subsequently reports the distance from this point if the relative option has been selected from the misc menu.

2.1.4 Actions Associated with the Mouse Buttons

There are several mouse menu buttons along the top of the plot window. The action of these buttons is as follows. Starting at the left top there are four buttons labelled 'draw', 'sopt', 'slib' and 'keyw'. These activate drop down menus which allow a function to be selected and inserted into the text. The 'draw' functions are a suite of drawing routines which cause lines, arcs, text etc. to be drawn to the page. The 'sopt' menu is a list of textual names for numerical constants which are used in conjunction with the `set();` command to change attributes of the current graphics state such as the current colour, line width, line style, axes type, etc. Choosing set from the list of drawing functions automatically opens a menu with only an appropriate subset of these and is the recommended approach. The 'slib' button allows selection of a standard C library function. Most of these are math functions such as `sin()` and will not be required for making ordinary plots. They are however useful for manipulating data before plotting or generating data mathematically all from within Splot. The last button of this group 'keyw' opens a list of C keywords for selection. The only one of these that is of interest for ordinary plots is 'main'. Every file that generates a plot or drawing must start with

```
#include <splot.h>
main
   {
   /* list of drawing commands go here between the braces on as many */
   /* lines as necessary */
   }
```

This is all automatically inserted in the text by selecting 'main'. The remaining keywords in this list are only of interest when constructing loops or branches. Note that the actual text inserted for the draw, slib and keyw menu choices can be customized by editing the corresponding macros in the splot.cfg startup file.8.5

The next set of menu buttons along the top are concerned with displaying the drawing. The default size of the drawing is one which fits on a standard 8 x 11 inch piece of paper if printed. However, as the screen resolution is not as good as that of printers it is possible to zoom in on part of a drawing in order to get a better view.

2.1.5 Hard Copy

In order to send a graph to the printer select the print option in the file menu of the editor window. Output can be directed to a file by opening the printer settings notebook and selecting 'print to file' Hard copy is device independent and relies on the OS/2 printer drivers. Splot can also generate encapsulated postscript (.eps) files of a figure for inclusion in a document. It is better to let Splot generate an .eps file rather than to rely on a postscript printer driver with the output directed to a file as the Splot generated file is much smaller. Splot also adds the required %%BoundingBox statement to the generated ps file as required for encapsulated postscript files.

2.2 Miscellaneous notes

A drawing can be made in either of two ways. First the drawing can be specified by simply typing in a C program using the built in editor. Of course with this editor it is easy to import components of previous

drawings. A drawing can however also be made using the mouse. First the desired drawing function is selected using the drop down menus on the top row. There are menu buttons for the C keywords, drawing functions, miscellaneous functions, and the defined constants. Selecting an item causes it to be inserted in the text at the current text cursor position. Many functions require coordinate points as parameters. These can be entered by moving the mouse to the desired location on the drawing page and double clicking the left mouse button. if the text cursor is currently located at the start of a coordinate pair, that coordinate pair is erased and replaced with a new coordinate when the left mouse button is doubled clicked. In this way it is possible to draw interactively with the mouse or by entering text which ever is more convenient for the task at hand. Parameter errors are not detected until the file is executed using CTRL G or the 'exec' menu button.

2.3 Carrying On

2.3.1 Text

Text can be added to a figure using the `text();` command. The first two parameters are a coordinate pair giving the position at which the text should start. For subsequent lines of text the coordinates can be omitted in which case the line is positioned below the previous line. The third parameter is the actual text to be printed. There is also an optional fourth parameter which allows the text to be centered on the coordinates or end at the coordinates. Specialized text labels are also generated using the `label();` and `ticklabel( );` commands and the following discussion applies to them as well.

The appearance of the text is affected by several set-able parameters. The size of the letters is determined by `set(FONTWIDTH , size_in_cm);` while the ratio of height to width is set using `set(FONTASPECT , height_div_width);`. Alternatively, the size of the text can be changed from its current value by using `set(FONTMULT , multiplier);`. The spacing between letters can be modified using `set(FONTSPACE , space_in_cm);`. The text is ordinarily printed horizontally but this may be changed by using `set(FONTDIR , angle_in_deg);`. The width of the line used to draw the text is set using `set(LINEWIDTH , width);` and the colour is set using `set(LINECOLOUR , colour_number );`. The text is printed using Hershey stroked fonts. There are two available font families simplex and complex. The complex font is chosen using `set(FONT , COMPLEX);`. Greek and Italic text is selected by using special characters in the text string. Thus Greek may be selected by enclosing a string with '!' as in '!abc! while italics are selected using '#'. Super and sub scripts are handled in the same way by enclosing the super/sub script with '^' or '_' respectively. The position and size of the super/sub script relative to ordinary text may be modified using the `set(SCRIPTSCALE , ratio);` and `set(SCRIPTSHIFT , fraction );` commands respectively. Characters from the symbol font may be included in the text string by enclosing the symbol number with '$'.

The various text possibilities are explored in the two following demo files. The third demo file is a table of all the available text characters included in the font files.

```
/* This file illustrates some of the capabilities of the */
/* text() command to annotate figures */
/* Several fonts are available including full Greek. */
/* Text can be any size, orientation and colour. */

#include <splot.h>
main()
   {
   text(2.63,23,"The default position for the next line");
   /* change to complex font for next line */
   set(FONT,COMPLEX);
   text("right below the previous line");
   set(FONT,SIMPLEX);
   text("There is a complete set of Greek characters");
   text("!abcdefghijklmnopqrstuvxyz!");
   set(FONT,COMPLEX);
   text("!ABCDEFGHIJKLMNOPQRSTUVWXYZ!");
   set(FONT,SIMPLEX);
   set(LINECOLOUR,GREEN);
   /* optional change of script scale and position */
   set(SCRIPTSCALE,1);
   set(SCRIPTSHIFT,1);
   text("Here are ^super^ and _sub_ scripts");
   set(LINECOLOUR,RED);
   text("and #Italics# in different colours");
```

```
set(LINECOLOUR,BLACK);
text("Letters can be large, small, tall, short");
text("fat, thin, vertical and angled.");
set(FONTMULT,2);
text(2.94,9.56,"Large");
set(FONTMULT,0.30);
text(8.90,9.71,"small");
set(FONTWIDTH,1.0);
set(FONTASPECT,3);
text(13.10,9.71,"Tall");
set(FONTASPECT,0.5);
text(17.05,9.71,"Short");
set(FONTASPECT,2);
set(LINEWIDTH,0.01);
text(2.94,7.29,"Thin");
set(LINEWIDTH,0.2);
text(9.02,7.29,"Fat");
set(LINEWIDTH,0.05);
set(FONTDIR,90);
text(3.63,1.26,"Vertical");
set(FONTDIR,130);
text(11.28,1.44,"Angled");
}
```

The default position for the next line
right below the previous line
There is a complete set of Greek characters
αβχδεφγηιϑκλμνοπϑρστυψξψζ
ΑΒΞΔΕΘΓΗΙΘΚΛΜΝΟΠΘΡΣΤΤΨΩΧΨΖ
Here are $^{\text{super}}$ and $_{\text{sub}}$ scripts
and *Italics* in different colours
Letters can be large, small, tall, short
fat, thin, vertical and angled.

Large small Tall Short

Thin **Fat**

Vertical Angled

```
/* This file illustrates some of the capabilities of the */
/* text() command to annotate figures */

#include <splot.h>
main()
   {
   text(2,23,"The spacing between letters can be changed");
   set(FONTSPACE,0.5);
   text("to be further apart than usual");
   set(FONTSPACE,-0.2);
   text("or closer together.");
   set(FONTSPACE,0);
   text("The size and position of super/sub scripts can");
   text(2,17.38,"be varied as in");
   set(SCRIPTSHIFT,0.7);
   set(SCRIPTSCALE,0.5);
   text(8.94,17.38,"X_1_\b^2^,");
   set(SCRIPTSHIFT,1.0);
   set(SCRIPTSCALE,0.7);
   text(10.19,17.38,"X_1_\b^2^,");
   set(SCRIPTSHIFT,1.5);
   set(SCRIPTSCALE,1.0);
   text(11.4,17.38,"X_1_\b^2^.");
   text(2.10,15.81,"Note how the superscript was placed above");
   text("the subscript using the backspace char \\b.");
   text("Enter degree sign using '<ALT X> 127'");
   text("Use a \\ before special characters: \_\^\$\#\!\"");
   text("Symbols: $0 1 2 3 4 5 6 7 8 9 : ; < = > $");

   text(2,8,"The default text insertion point is LEFT");
   text(17,6,"but can be set to RIGHT",RIGHT);
   text(10,4,"Or CENTER",CENTER);
   }
```

The spacing between letters can be changed
t o b e f u r t h e r a p a r t t h a n u s u a l
or closer together.
The size and position of super/sub scripts can
be varied as in X_1^2, X_1^2, X_1^2.
Note how the superscript was placed above
the subscript using the backspace char \b.
Enter degree sign ° using '<ALT X> 127'
Use a \ before special characters: _↑$#!"
Symbols: ○ □ △ ◊ ☆ › + x ∗ ● ■ ▲ ♦ ★

The default text insertion point is LEFT

but can be set to RIGHT

Or CENTER

```
/* This file displays all the available characters */

#include <splot.h>
#define ROMAN 1
#define GREEK 0
#define ITALIC 0
int i,j;
char c[10];

main()
   {
   set(PAGEROT,ON);
#if ROMAN
   text(13.67,19.14,"Roman",CENTER);
   text(7.5,18,"Simplex",CENTER);
   text(19,18,"Complex",CENTER);
   gsave();
   scale(1.2,1.2);
   translate(1,-11);
   set(FONTWIDTH,0.5);
   set(FONTASPECT,1.5);
   table(0);
   translate(10,0);
   set(FONT,COMPLEX);
   table(0);
   set(FONT,SIMPLEX);
#endif
#if GREEK
   text(13.67,19.14,"Greek",CENTER);
   text(7.5,18,"Simplex",CENTER);
   text(19,18,"Complex",CENTER);
   gsave();
   scale(1.2,1.2);
   translate(1,-11);
   set(FONTWIDTH,0.5);
   set(FONTASPECT,1.5);
   table(1);
   translate(10,0);
   set(FONT,COMPLEX);
   table(1);
   set(FONT,SIMPLEX);
#endif
#if ITALIC
   text(7.5,18,"Italics Simplex/Complex",CENTER);
   text(19,18,"Symbols",CENTER);
   gsave();
   scale(1.2,1.2);
   translate(1,-11);
   set(FONTWIDTH,0.5);
   set(FONTASPECT,1.5);
   table(2);
   translate(10,0);
```

```
   table(3);
#endif
   grestore();
   }

table(int special)
   {
   /* label rows and columns */
   c[1] = '\0';
   for (i = 0; i < 10;i++)
      {
      c[0] = '0' + i;
      tputs(2 + i * 0.7,24.5,c,0);
      }
   c[0] = ' ';
   c[2] = '0';
   c[3] = '\0';
   for (j = 3;j < 10;j++)
      {
      c[1] = '0' + j;
      tputs(0,26 - j,c,0);
      }
   c[0] = '1';
   c[2] = '0';
   c[3] = '\0';
   for (j = 10;j < 13;j++)
      {
      c[1] = '0' + j - 10;
      tputs(0,26 - j,c,0);
      }
   /* prepend each char with '\' so that even special chars get printed */
   c[0] = '\\';
   c[2] = '\0';
   box(1.29,23.77,9.07,13.36);
   stroke();
   /* fill in table */
   for (j = 3;j < 13;j++)
      {
      for (i = 0; i < 10;i++)
         {
         c[1] = j * 10 + i;
         if (c[1] == 'b')
            tputs(2 + i * 0.7,26 - j,"b",special);
         else
            tputs(2 + i * 0.7,26 -j,c,special);
         }
      }
   }

tputs(double x,double y,char *c,int special)
   {
   char tmp[10];
```

```
switch(special)
   {
   case 0:
      text(x,y,c);
      break;
   case 1:
      /* greek */
      strcpy(tmp,"!");
      strcat(tmp,c);
      strcat(tmp,"!");
      text(x,y,tmp);
      break;
   case 2:
      /* italics */
      strcpy(tmp,"#");
      strcat(tmp,c);
      strcat(tmp,"#");
      text(x,y,tmp);
      break;
   case 3:
      /* symbols */
      strcpy(tmp,"$");
      strcat(tmp,c);
      strcat(tmp,"$");
      text(x,y,tmp);
      break;
   }
}
```

Roman

Simplex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	↑	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	°		

Complex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	↑	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	°		

Greek

Simplex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	X	Δ	E
70	Φ	Γ	H	I	Θ	K	Λ	M	N	O
80	Π	Θ	P	Σ	T	ϒ	Ψ	Ω	Ξ	Ψ
90	Z	[	\	]	↑	_	‘	α	β	χ
100	δ	ε	φ	γ	η	ι	ϑ	κ	λ	μ
110	ν	ο	π	ϑ	ρ	σ	τ	υ	ψ	ω
120	ξ	ψ	ζ	{	|	}	~	°		

Complex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	Ξ	Δ	E
70	Θ	Γ	H	I	Θ	K	Λ	M	N	O
80	Π	Θ	P	Σ	T	ϒ	Ψ	Ω	X	Ψ
90	Z	[	\	]	↑	_	‘	α	β	ξ
100	δ	ε	φ	γ	η	ι	ϑ	κ	λ	μ
110	ν	ο	π	ϑ	ρ	σ	τ	υ	ψ	ω
120	ξ	ψ	ζ	{	|	}	~	°		

Greek

Simplex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	X	Δ	E
70	Φ	Γ	H	I	Θ	K	Λ	M	N	O
80	Π	Θ	P	Σ	T	ϒ	Ψ	Ω	Ξ	Ψ
90	Z	[	\	]	↑	_	‘	α	β	χ
100	δ	ε	φ	γ	η	ι	ϑ	κ	λ	μ
110	ν	ο	π	ϑ	ρ	σ	τ	υ	ψ	ω
120	ξ	ψ	ζ	{	|	}	~	°		

Complex

	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	’
40	(	)	*	+	,	−	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	Ξ	Δ	E
70	Θ	Γ	H	I	Θ	K	Λ	M	N	O
80	Π	Θ	P	Σ	T	ϒ	Ψ	Ω	X	Ψ
90	Z	[	\	]	↑	_	‘	α	β	ξ
100	δ	ε	φ	γ	η	ι	ϑ	κ	λ	μ
110	ν	ο	π	ϑ	ρ	σ	τ	υ	ψ	ω
120	ξ	ψ	ζ	{	|	}	~	°		

2.3.2 Symbols

Symbols can optionally be drawn at every data point by setting `set( PLOTTYPE , SYMBOLS);` or `set( PLOTTYPE , SYM_LINES);`. The actual symbol used is set using `set( CURSYMBOL , sym_number );` Symbols are just a special font and much of the above discussion on how to change a letter's appearance applies also to symbols. There is, however, no complex version of the symbols. Additionally, the size of the symbols can be adjusted relative to the current font size using `set( SYMMULT ,multiplier);`. Symbols can also be drawn independent of `drawdata( );` at an arbitrary location by using `symbol( x , y , sym_number);`. For an example of the symbol capabilities consult the last example of the previous chapter. Symbols may also be included in text strings drawn with `text();` by enclosing the symbol number with '$' in the text string.

2.3.3 Gsave/Grestore

The pair of functions `gsave();` and `grestore();` are useful for a variety of purposes. `gsave();` stores the complete graphics state including the currently defined path while `grestore();` restores the state to the last saved state. Each `grestore();` must have a matching `gsave();` preceding it.

The simplest use for this pair is to restore the initial state after translations and scaling. This often comes up in composite figures that are constructed from other existing picture. The composite is built by setting translations and scale factors for each component and then drawing the component. If a `gsave();` was done at the beginning then a `grestore();` can be done after drawing the components to restore the initial scale and translation. This can be useful for adding annotations to the final composite figure.

It is also possible to use gsave/grestore to repeatedly draw a given path since the currently defined path is also saved. In the following example a path is defined, saved (using `gsave();`), filled, restored (using `grestore();`) and then finally stroked. Filling and stroking the same path causes the filled region to have a border drawn in the current line colour and style.

```
#include <splot.h>
/* This is a region filling example */
int i;
main()
   {
   set(LINEWIDTH,0.1);
   translate(10,15);
   for (i=0;i < 8;i++)
      {
      set(LINECOLOUR,i);
      leaf();
      rotate(45);
      }
   }

int leaf()
   {
   arc(3.1,3.38896,0.4,0,360);
   moveto(0,0);
   curveto(3.38924,0.598051,3.18987,3.85411,5.91456,6.11341);
   curveto(2.06013,5.44891,2.52532,1.5948,0,0);
   closepath();
   gsave();
   fill();
```