March 13, 1997

PostScript Scientific Graphics Library for C

with Screen Previewing

Version 2.02

Wayne C. Crawford
Izumi Ohzawa
Geoff M. Ghose
Bruce MccReady
William Grosso

University of California, Berkeley.

For current information, see:
http://totoro.berkeley.edu/software/A_Cgraph.html
ftp://pinoko.berkeley.edu/pub/software/

Introduction

This graphics function library has been created for producing "publication quality" scientific graphics on laser printers. Subroutines (functions) in this library are meant to be linked into scientific application programs to produce graphs directly from within these programs. It has been written to address our need to produce many graphs of a similar format for a large number of data sets quickly and with minimum human interventions. For this type of situation, the standard approach of transferring data from the analysis program to spread sheet, and then to menu driven commercial graphics packages is too clumsy and time-consuming. Besides, most of these commercial packages are primarily for those silly pie-charts, 3-dimensional bar graphs, etc., and they never can do what we want in the way we want.

The library functions produce output suitable *only* for **PostScript** graphics devices. No attempt has been and will be made to make this library compatible with any other graphics device. PostScript is a *page description language*, developed by Adobe Systems, Inc., capable of describing in detail how text and graphics are placed on a page. Apple LaserWriter is the first and best-known laser printer which accepts PostScript.

PostScript is also used also for computer displays. **Display PostScript** on now included in various platforms can take the same PostScript description intended for a PostScript printer and show it on the screen instead of a piece of paper. This can be done because PostScript descriptions are device independent, and each device, be it a screen or printer, tries its best to represent the image within its device resolution. These charecteristics of PostScript allow us to achieve so called "what-you-see-is-what-you-get" (WYSIWYG) most painlessly. In the current version (2.0) of Cgraph, screen previewing is achieved by passing the generated PostScript to a PostScript viewer application automatically. For NEXTSTEP/OPENSTEP platform, Preview.app is used as the viewer. On other platforms, freely available viewer called Ghostscript may be used (my small changes to **bgshow.c**).

In addition to the scope of the library described above, we have developed the library with the following design goals:

[1] Versatile X and Y axis routines are provided for linear as well as logarithmic axes.

[2] Two-dimentional coordinate system will be defined by the latest calls to X and Y axis routines such that (X, Y) values can be used directly to specify a point in the domain without additional scaling or logarithmic operation. Any combination of linear and logarithmic axes may used to define the XY coordinate system. Prior to calls to axis routines, coordinate system is in *inches* with the origin at the bottom-left corner of the page. Even after a domain is defined by axis routines, coordinate system may be switched to and from the inch-based coordinate system without losing the current axis coordinate system.

[3] Markers (20 types) and pointers are provided to allow marking of data points. These markers may be scaled to any size. Markers drawn later occlude previous lines and other objects. For example, open circle drawn over a line will have a white disk inside the circle. This is difficult with a pen plotter, but is trivial in PostScript devices because of their capability to lay white as well as black pixels.

[4] Background mesh may be drawn to give the apperance of a graph paper and to aid in locating data point easily.

[5] Lines can be of any thickness, can be dashed, and made into different shades of gray.

[6] (NEW) Flexible color specification is now possible in which the decision to use B/W or color may be based on whether the imaging device supports color or not. Alternatively, a simple editing of just one line in the PS file can toggle B/W and color output. The purpose of this is to support plots for B/W prints and color slides and prints with a single code.

Installation/Usage

[1] Compile and build the library file:

```
cd source
make
```

[2] Copy files to appropriate locations:

```
make install
```

OR

```
cp cgraph.h /usr/local/include
cp libcgraph.a /usr/local/lib
ranlib -s /usr/local/lib/libcgraph.a
```

[3] Specify the library name at compile/link time, e.g.,

```
cc -O -Wall -o CGminimal CGminimal.c -lcgraph
```

[4] To send the output to standard output, use **cg_use_stdout().**

cg_aorigin

Summary

int cg_aorigin(float xpos, float ypos);

Description

Places the origin of the graph xpos inches to the right and ypos inches above the lower left corner of the page. The axis coordinate system becomes identical to the inch coordinate system.

See Also

cg_rorigin, cg_coord_select

Example

```
cg_init(1, 1, 1.0);
cg_text( 0., 0., 0., "example" );      /* Prints at currentorigin */
cg_aorigin( 1.0, 1.0 );
cg_text( 0., 0., 0., "example" );      /* 1 inch to the right and up from */
                                       /* the bottom left corner.*/
cg_showpage();
```

cg_axis_enable

Summary

int cg_axis_enable (int axisEnable, int numberEnable)

Description

This function is used to disable plotting of axes and numbers on them, while retaining the ability to define scaled domain via axis calls. The flag value of zero for the two arguments will disable axis and numbers, respectively. The flags apply to all axis calls, linear, logarithmic, X and Y, and remain in effect until another call is made with new flag values. The domain mesh may still be drawn by a separate call to cg_mesh() even if the axes are disabled.

See Also

cg_xaxis, cg_yaxis, cg_xlog, cg_ylog

cg_centerlabel

•Summary

int cg_centerlabel (char *textstring, float x, float y, float rotate, int flag**)**

•Description

Cg_centerlabel is a function to display a string of text on a graph. The parameters x and y are the coordinates of the point on the graph that the text is to be centered around (the flag determines the coordinate system: 1=domain, 0=inches). The rotate parameter lets the user determine the angle that the text is printed at (0="flat").

•Example

```
cg_centerlabel ("Bill is Great", 5.0, 5.0, 90.0, 0);
```

cg_closepath

Summary

int cg_closepath(void);

Description

Closes the current defined *path* by drawing a line to the last **cg_move** or **cg_rmove** position. PostScript devices will bevel connections between the starting and end points when **cg_closepath** is used. If it is not used and the line is wide, a notch will be apparent at this vertex.

See Also

cg_line, cg_move, cg_rline, cg_rmove, cg_stroke

cg_coord_select

Summary

int cg_coord_select(int marks, int text);

Description

Choose whether the coordinates given to various functions which place lines, pointers, markers and text should be treated as inches or as part of the axis defined coordinate system. **[0 = inches, 1 = axis coordinate system]**. Text is the boolean for all coordinates used within a text function, marks handles the rest of the figures. Text default is 0, marks default is 1.

cg_dash

Summary

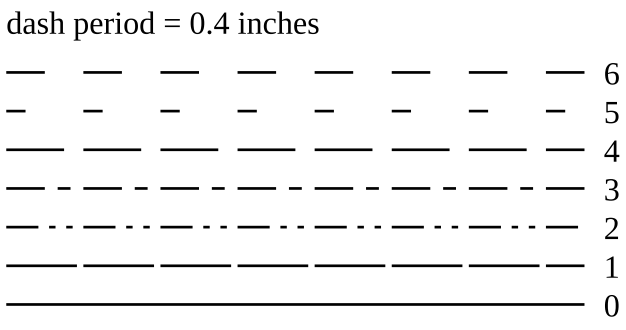
int cg_dash(int type, *float dashperiod*);

Description

Sets the type of line to be printed using the **cg_line**s. The different type s are shown on page A-2. The dashperiod represents the "period of the dash (how many inches that one period of the dash will occupy) in inches. **cg_stroke** resets to a solid black line as does **cg_dash**(0, 1.0).

See Also

cg_line, cg_reset, cg_rline, cg_stroke



cg_fill, cg_eofill

Summary

int cg_fill(void);
int cg_eofill(void);

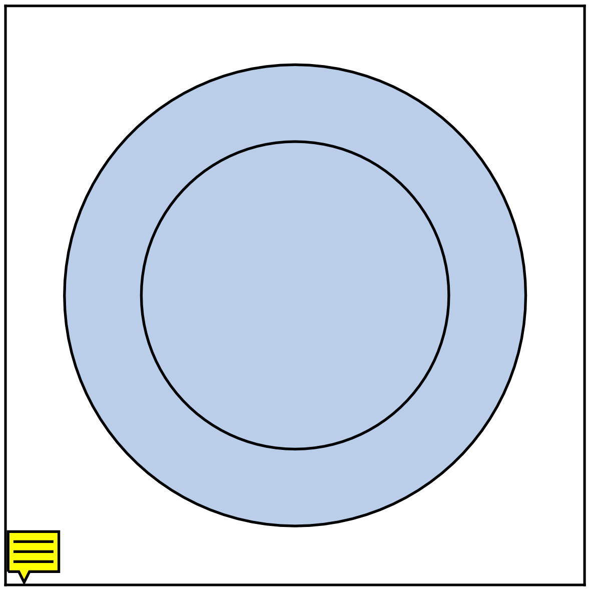
Description

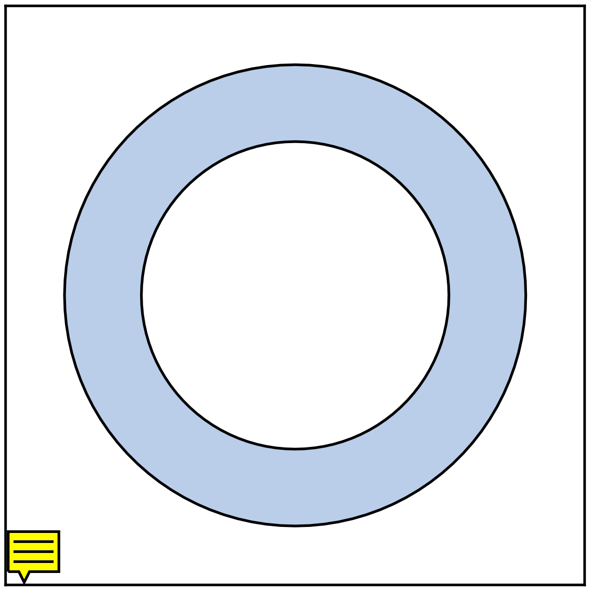
Fills the area enclosed by the current path with current color. The inside of the current path is determined by the normal non-zero winding rule. Fill operation will remove the path after the operation. If you need to retain the path, bracket cg_fill() with cg_gsave() and cg_grestore().

Cg_eofill() is similar to cg_fill() but it fills every other region of complex paths using the even-odd rule defined by *winding numbers*. See the PostScript Red Book for detailed explanation of rules that define inside and outside of complex paths.

See Also

cg_closepath, cg_postscrip, cg_gsave, cg_grestore


fill


eofill

cg_font

Summary

int cg_font(float *fontsize*, int type);

Description

Select the text size and type to use for whatever is to be printed until the next **cg_font** call . Fontsize is in "points" which is 1/72 of an inch. (the font you see here is 12-point Times-Roman). Type = **n** * 10 + **m**, where **n**=

1: Times-Roman
2: Courier
3: Helvetica

and **m**=

0: normal
1: **bold** (not available on the screen)
2: *italicized*
3: ***bold and italicized***

The type of the numbers automatically printed on the axes are selectable using this function. The startup font is 12-point Times-Roman.

See Also

cg_fontbyname, cg_text, cg_textalign

Example

```
cg_init( 1, 1, 1.0 );
cg_text( 1.0, 1.0, 0.0, "12pt Times-Roman [default]" );
         /* Print in the default font( 12-point Times-Roman). */
cg_font(15.0, 30);
cg_text( 1.0, 2.0, 0.0, "15pt Helvetica" );
         /* The letters are now 15-point, Helvetica. */
cg_font(12.0, 22 );
cg_text( 1.0, 3.0, 0.0, "12-point Courier-Oblique" );
         /* The letters are now 12-point Courier, italicized.*/
cg_showpage();
```

cg_fontbyname

Summary

int cg_fontbyname(float *fontsize*, char *fontname);

Description

Fontsize is in the unit of points. Instead of coded font specification used by cg_font(), this function allows you to specify font by name. Exact fontname with correct capitalization must be used.

The type of the numbers automatically printed on the axes are selectable using this function. The startup font is 12-point Times-Roman.

See Also

cg_font, cg_text

Example

```
cg_init( 1, 1, 1.0 );

cg_text( 1.0, 1.0, 0.0, "12pt Times-Roman [default]" );
         /* Print in the default font( 12-point Times-Roman). */
cg_fontbyname(20.0, "Helvetica-Oblique");
cg_text( 1.0, 2.0, 0.0, "20pt Helvetica-Oblique" );
         /* The letters are now 15-point, Helvetica Oblique. */
cg_fontbyname(12.0, "Courier-Bold" );
cg_text( 1.0, 3.0, 0.0, "12-point Courier-Bold" );
         /* The letters are now 12-point Courier, Bold.*/
cg_showpage();
```

PostScript fonts available in the base NEXTSTEP, and the original Apple LaserWriter
(Font names are case-sensitive.)

Helvetica
Helvetica-Bold
Helvetica-Oblique
Helvetica-BoldOblique
Times-Roman
Times-Bold
Times-Italic
Times-BoldItalic
Courier
Courier-Bold
Courier-Oblique
Courier-BoldOblique
Symbol

Additional PostScript fonts available in most PostScript printers.

AvantGarde-Book
AvantGarde-BookOblique
AvantGarde-Demi
AvantGarde-DemiOblique
Bookman-Demi
Bookman-DemiItalic
Bookman-Light
Bookman-LightItalic
Helvetica-Narrow
Helvetica-Narrow-Bold
Helvetica-Narrow-Oblique
Helvetica-Narrow-BoldOblique
NewCenturySchlbk-Roman
NewCenturySchlbk-Italic
NewCenturySchlbk-Bold
NewCenturySchlbk-BoldItalic
Palatino-Roman
Palatino-Italic
Palatino-Bold
Palatino-BoldItalic
ZapfChancery-MediumItalic
ZapfDingbats

cg_get_output_filename

Summary

char *cg_get_output_filename(void);

Description

Gets the current output filename which is generated automatically unless it is set explicitly by a call to cg_set_output_filename().

See Also

cg_set_output_filename

Example

```
char strbuf[1024];
char *outfile;

   cg_launch_preview(0);         /* do not launch preview */
   cg_init(rot, expand, scale);

   ... lots of drawing here

   cg_showpage();

   outfile = cg_get_output_filename();
   sprintf(strbuf, "lpr %s", outfile); /* automatic printing */
   system(strbuf);
   remove(outfile);
```

cg_gray

Summary

int cg_gray(*float lightness*);

Description

Sets the gray level of objects (lines, polygonfill) to be printed, with "0.0" meaning black, "0.5" gray, and "1.0" meaning white.

See Also

cg_grayrgbcolor, cg_rgbcolor, cg_markergray, cg_markercolor, cg_markergraycolor, cg_fill, cg_line, cg_linewidth, cg_rline, cg_stroke

Example

```
cg_move( 0.0, 0.0);
cg_line( 4.0, 4.0);
cg_stroke();               /* Shows the current gray level.*/
cg_gray(0.6);
cg_move( 1.0, 0.0);
cg_line( 5.0, 4.0);
cg_stroke();               /* different gray level on printout.*/
```

cg_grayrgbcolor

Summary

int cg_grayrgbcolor(*float gray, float red, float green, float blue*);

Description

Sets the current gray level and rgb color at the same time. This generates an PS file with a flexible color specification in which the decision to use B/W or color may be based on whether the imaging device supports color or not. Alternatively, a simple editing of just one line in the PS file can toggle B/W and color output. The purpose of this function is to support plots for B/W prints and color slides and prints with a single code.

The gray level of objects (lines, polygonfill) is defined as: "0.0" meaning black, "0.5" gray, and "1.0" meaning white.

See Also

cg_gray, cg_rgbcolor, cg_markergray, cg_markercolor, cg_markergraycolor, cg_fill, cg_line, cg_linewidth, cg_rline, cg_stroke

Example

```
cg_move( 0.0, 0.0);
cg_line( 4.0, 4.0);
cg_stroke();               /* Shows the current gray level.*/
cg_grayrgbcolor(0.6, 1.0, 0.0, 0.0);   /* gray or red */
cg_move( 1.0, 0.0);
cg_line( 5.0, 4.0);
cg_stroke();               /* different gray level on printout.*/
```

cg_grestore, cg_gsave

Summary

int cg_gsave(void);
int cg_grestore(void);

Description

Saves the current graphics state to be restored later by cg_grestore(). The graphics state saved includes, current color, path, linewidth, ...
This call is useful for retaining a path for fill and stroke, because both of these operators deletes the path.

See Also

cg_fill, cg_stroke

Example

```
 ... we now have a closed path ...

cg_grayrgbcolor(0.0, 0.0, 0.0, 1.0);   /* black or blue */
cg_gsave();
cg_grayrgbcolor(0.5, 1.0, 1.0, 0.7);         /* gray or pink for fill */
cg_fill();
cg_restore();              /* grestore also restores color */
cg_stroke();               /* Stroke with current color */
```

cg_init

Summary

int cg_init(int rot, int expand, float scale)

Description

Cg_init() initializes the drawing environment for other functions, including opening the output stream and generating the PS prolog of the output.

Cg_init() function must be called before all others cg_*() functions **except for** cg_setboundingbox, cg_settitle , cg_setcreator, cg_setprolog(), cg_set_top_comments(), and cg_useflexcolor().

rot =

0: the page is printed with the x-axis the long way ("Landscape" style).
1: the page is printed with the x-axis the short way ("Portrait" style").

expand = OBSOLETE, just plug any integer.

scale =

Use 1.0 for the initial unit to be inches. Using any other number scales the unit accordingly. For example, using 1/2.54 will allow the use of centimeters as the initial unit.

Example

```
if(use_stdout)
    cg_use_stdout(use_stdout);   /* decide where to send output */
cg_useflexcolor(2);        /* device capability dictates color/monochrome */

cg_setboundingbox("0 0 216 216");
cg_settitle(prog_version);
cg_setcreator(progname);
cg_set_top_comments(comment);    /* set the comments for EPS header */

cg_init(rot, expand, scale);     /* expand is not used but for compatibility */
```

cg_launch_preview

Summary

void cg_launch_preview (int flag);

Description

Controls whether a PostScript previewer application is launched automatically on the generated output file automatically upon execution of cg_showpage(). If flag is non-zero, it will be launched. Default flag is non-zero.

See Also

cg_init, cg_set_output_filename

Example

```
char strbuf[1024];
char *outfile;

   cg_launch_preview(0);         /* do not launch preview */
   cg_init(rot, expand, scale);

   ... lots of drawing here

   cg_showpage();

   outfile = cg_get_output_filename();
   sprintf(strbuf, "lpr %s", outfile); /* automatic printing */
   system(strbuf);
   remove(outfile);
```

cg_linax_style

Summary

int cg_linax_style(precision, numoff, numdist, tnposition, ticlen);
int precision, numoff, numdist, tnposition, ticlen;

Description

Routine to modify the style of all subsequent linear axes. If you want to use standard axes (described as defaults below) you don't need to call this function.

Precision specifies the maximum possible number of significant digits printed on the axes. Setting it high gives you the ability to do very fine scales (.9999976 - 1.0000000, for example). Setting it low prevents floating point deviations from showing up on your graphs. A value of 4 (the startup value) is probably the lowest value you will have to choose, and is a good compromise for standard graphs.

Numoff controls at which tick the numbering begins. To start axis numbering at the first tick, set it to 1.

Numdist controls the distance of the numbers from the end of the ticks. The numbers are placed (fontsize/10)*numdist points away from the ends of the ticks. Numdist=5 (the default value) is usually satisfactory.

Tnposition is used to control which side of the axis the tick marks protrude, and which side of the axis the numbers are on. The current choices are:

1: ticks left of [below] the axis,numbers left [below].
2: ticks through the axis, numbers left [below] -- [STARTUP value]
3: ticks right [above] the axis, numbers left [below]
4: ticks right of [above] the axis, numbers right [above]
5: ticks through, numbers right [above]
6: ticks left [below], numbers right [above]

Ticlen allows you to choose the length of the tickmarks on either side of the axis in units of 1/72". A good standard is 5 (the startup value), creating ticks of length 5 for tickmarks on each side of the axis.

See Also

cg_xaxis, cg_yaxis, cg_logax_style

Example

```
cg_init( 1, 1, 1.0 );
cg_rorigin( 2.0, 2.0 );
```

```
      /* Draws at the middle of the screen.*/
cg_xaxis( 5.0, 0.0, 400.0 ,0.0 ,50.0 ,2);
/* 1. Draws a 5-inch-long axis in the X-dir.
   2. The axis ranges from 0.0 to 400.0.
   3. The axis is offset from the the origin of the
      y-axis by 0.5 inches.
   4. The ticks are placed at the multiples of 50.
   5. The ticks are numbered every 2 ticks.
   6. Both the numbers and the ticks are BELOW the
     axis. [DEFAULT]
   7. The numbers are 2 points away from ends of the
     ticks. [DEFAULT]
   8. The numbering starts at the 1-st tick. [DEFAULT]
   9. The tick length is 5*1/72 inches. [DEFAULT]
                                        */
cg_linax_style(4, 3, 5, 1, 5);
cg_xaxis( 5.0, 0.0, 400.0, 0.0, 50.0, 2);
      /* Same as above except that numbers are
         now 5 points away from the end of the
         ticks, and numbering starts on the third
         tick.                  */
cg_linax_style(4, 3, 5, 2, 5 );
cg_xaxis( 5.0, 0.0, 400.0, 0.0, 50.0, 2);
      /* Same as above except that ticks are now
         THROUGH the axis.                  */
cg_linax_style(4, 3, 5, 3, 5 );
cg_xaxis( 5.0, 0.0, 400.0, 0.0, 50.0, 2);
      /* Ticks are now ABOVE the axis, and numbers
         are still BELOW the axis.                */
cg_linax_style(-1, -1, -1, 4, -1 );
cg_xaxis( 5.0, 0.0, 400.0, 0.0, 50.0, 2);
      /* Ticks are still ABOVE the axis, but numbers
         are now also ABOVE the axis.       */
cg_linax_style(-1, -1, -1, 5, -1 );
cg_xaxis( 5.0, 0.0, 400.0, 0.5, 5.0, 2);
      /* Ticks are THROUGH, and numbers ABOVE.*/
cg_showpage();
```

cg_line

Summary

int cg_line(*xpos*, *ypos*);
float *xpos*, *ypos*;

Description

Draws an invisible line from the last pen location [the "currentpoint"] to xpos, ypos in units of the x- and y-axes, and redefines the currentpoint as xpos, ypos. To make the line visible you have to **cg_stroke()** it, but this shouldn't be done until you have made your complete curve, because **cg_stroke** gets rid of the currentpoint.

See Also

cg_closepath, cg_fill, cg_move, cg_rmove, cg_rline, cg_stroke

Example

```
cg_init( 1, 1, 1.0 )
cg_move( 1.0, 1.0);
cg_line( 2.0, 2.0);
      /* Draws an invisible line from (1.0, 1.0)
        to (2.0, 2.0)*/
cg_stroke();
      /* Shows the line*/
cg_showpage();
```

cg_linewidth

Summary

int cg_linewidth(*width*);
float *width*;

Description

Sets the width of the lines in units of "points" (1/72 inch) to be used in creating the axes and graph lines. The start-up value is 1.0 (point). This function has no effect on markers, letters, numbers, the mesh.

See Also

cg_dash, cg_gray, cg_reset, cg_stroke

cg_logax_style

Summary

int cg_logax_style
(int numsel,int ticsel,int numdist,int tnposition,int ticlen, int ticlen10, int numformat);

Description

Routine to modify the "style" of all subsequent logarithmic axes. If you want to use standard axes (described as defaults below) you don't need to call this function.

Numsel and ticsel allow you to choose the spacing of the numbers and of the tickmarks. Setting bits corresponding to each position will turn on the number or tick according to the following table. Numsel default is 2 (1's only as in 0.1, 1.0, 10.0, 100.0...), and ticsel default is 1022 (all tick marks).

ALL MULTIPLES OF TEN OF THE "ON" (=1) DIGITS ARE MARKED

pos|	min	1	2	3	4	5	6	7	8	9	max
bit |	0	1	2	3	4	5	6	7	8	9	10

Numdist controls the distance of the numbers from the end of the ticks. The numbers are placed (fontsize/10)*numdist points away from the ends of the ticks. Numdist=5 (the startup value) is usually satisfactory.

Tnposition is used to control the type of ticks created and which side of the axis the numbers are on. The current choices are:

1: ticks left of [below] the axis,numbers left [below].
2: ticks through the axis, numbers left [below] -- [STARTUP value]
3: ticks right [above] the axis, numbers left [below]
4: ticks right of [above] the axis, numbers right [above]
5: ticks through, numbers right [above]
6: ticks left [below], numbers right [above]

Ticlen, and ticlen10 allow you to choose the length of the tickmarks on either side of the axis in units of 1/72". Ticlen10 is for tick marks for powers of 10. A good standard for ticlen is 5 (the startup value), creating ticks of length 5 for tickmarks on each side of the axis.

Numformat should be set to **0**, but it currently does not do anything. It's intended use is for supporting power notations such as 10^3.

See Also

cg_xlog, cg_ylog, cg_linax_style

cg_marker

Summary

int cg_marker(*xpos*, *ypos*, type, *size*);
int type;
float *xpos, ypos, size*;

Description

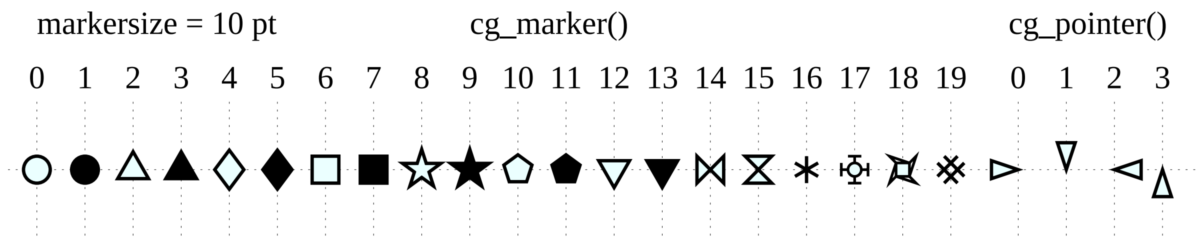
Places a marker at the designated position. There are 20 different markers, selectable by the type parameter [see page A-1]. The size of the marker [in points] is also selectable.

See Also

cg_pointer

Example

```
cg_init( 1, 1, 0 );
cg_marker( 3.0, 2.0, 2, 2.4 );
cg_marker( 5.0, 5.0, 6, 2.4 );   /* Two different markers of the same size */
cg_showpage();
```



cg_markergray, cg_makercolor, cg_markergraycolor

Summary

void cg_markergray (float peri_gray, float inside_gray);
void cg_markercolor (float peri_r, float peri_g, float peri_b,
float inside_r, float inside_g, float inside_b);
void cg_markergraycolor (float peri_gray, float peri_r, float peri_g, float peri_b,
float inside_gray, float inside_r, float inside_g, float inside_b);

Description

Specifies colors for markers. Marker colors are independent of colors of other objects. To use flexible color, use cg_markergraycolor(). Flex color will not be used if you call either cg_markergray() or cg_markercolor(). Color of the perimeter of markers is specified by the arguments peri_*, and that of inside (fill) is specified by the arguments inside_*.

See Also

cg_gray, cg_rgbcolor, cg_grayrgbcolor, cg_pointer, cg_useflexcolor

Example

```
cg_init( 1, 1, 0 );
cg_marker( 3.0, 2.0, 0, 2.4 );   /* using default color: white in black */

/* Marker color is specified by a different function. */
cg_markergraycolor(0.0, 1.0, 1.0, 1.0,
           1.0, 1.0, 0.0, 0.0);  /* white in black, or red in white */
cg_marker( 5.0, 5.0, 0, 2.4 );   /* using flex color */
cg_showpage();
```

cg_mesh

Summary

int cg_mesh();

Description

Draws a mesh corresponding to the ticks on the current graph. This gives the appearance of a graph paper and aids the localization of data points.

See Also

cg_logax_style, cg_xaxis, cg_yaxis, cg_xlog, cg_ylog

Example

```
cg_init( 1,1,1.0 );
cg_xaxis( 5.0, 0.0, 40.0, 0.0, 5.0, 2 );
                           /* Make an axis.*/
cg_yaxis( 4.0, 0.0, 10.0, 0.0, 2.0, 1);
cg_mesh();                 /* Draw a mesh on the axis.*/
cg_showpage();
```

cg_move

Summary

int cg_move(*xpos, ypos*);
float *xpos, ypos;*

Description

Moves the "pen" [without writing] to xpos, ypos relative to the origin. This function must always be used before the first **cg_line** in order to define a "currentpoint" for the line to go from.

See Also

cg_closepath, cg_fill, cg_line, cg_rline, cg_rmove, cg_stroke

Example

```
cg_init( 1, 1, 1.0 );
cg_move(1.0, 1.0);
      /* Defines the current point at (1.0, 1.0).*/
cg_line ( 2.0, 2.0 );
cg_stroke();
      /* Draws the line.*/
cg_showpage();
```

cg_pointer

Summary

int cg_pointer(*xpos*, *ypos*, type, *size*);
int type;
float *xpos*, *ypos*, *size*;

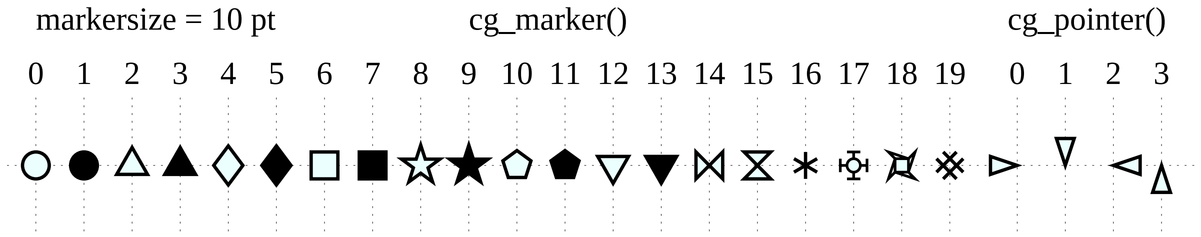
Description

Places a pointer (open arrow head) at the designated position. There are 4 different markers, selectable by the type parameter [see page A-1 and the list below]. The size of the pointer [in points] is also selectable.

type = 0:   point right
            1:   point down
            2:   point left
            3:   point up

See Also

cg_marker



cg_postscript

Summary

int cg_postscript(char *string);

Description

Output the input string verbatim to the output file. This is useful in injecting a little bit of PS code directly into the PS stream, and inserting useful comments in the PS file.

Example

```
cg_init( 1, 1, 1.0 );
cg_move( 3.0, 2.0 );
cg_line( 4.0, 2.0 );
cg_line( 6.0, 6.0 );
cg_closepath();
cg_postscript(" gsave ");     /* Save current path */
cg_gray(0.6);              /* Set output to light gray */
cg_fill();              /* Erases current path */
cg_postscript(" grestore ");  /* Restore current path */
cg_gray(0.0);                 /* Black output "gray level" */
cg_stroke();                  /* Black outline of gray shape*/
cg_showpage();
```

cg_reset

Summary

int cg_reset()

Description

Moves the origin back to the bottom left corner of the page. Makes the axis coordinate system identical to the inch coordinate system. Resets lines to solid black with thickness 1.

Example

```
cg_init( 1, 1, 1.0 );
cg_aorigin( 10.0, 3.0 );
cg_linewidth( 2.5 );
cg_move( 0.0, 0.0 );
cg_line ( 3.0, 4.0 );
cg_stroke();               /* Shows a line starting from (1.0, 3.0) inches from
*/
                     /* the bottom-left corner, with thickness of 2.5.*/
cg_reset();
cg_move( 0.0, 0.0 );
cg_line( 3.0, 4.0 )
cg_stroke();               /* Shows a solid, black line starting from the */
                     /* bottom-left corner, with a thickness of 1. */
cg_showpage();
```

cg_rgbcolor

Summary

int cg_rgbcolor(float red, float green, float blue);

Description

Sets the color of objects drawn after this call.   Use cg_grayrgbcolor() to specify gray and color at the same time for B/W prints and slides or color prints.

See Also

cg_gray, cg_grayrgbcolor, cg_useflexcolor, cg_makercolor, cg_markergraycolor, cg_markergray

cg_rmove, cg_rline

Summary

int cg_rmove, cg_rline(*xpos, ypos*);
float *xpos, ypos*;

Description

Like **cg_move** and **cg_line** except that xpos and ypos are relative to the currentpoint rather than to the origin. This is very confusing when one of the axes is logarithmic, so only use it for linear axes or in the inch coordinate system.

See Also

cg_line, cg_move, cg_stroke

Example

```
cg_init( 1, 1, 1.0 );
cg_move( 1.0, 1.0 );       /* Defines a current point*/
cg_rline( 2.0, 2.0 );      /* Draws a line from (1, 1) to (3, 3). */
cg_stroke();
cg_showpage();
```

cg_rorigin

Summary

int cg_rorigin(*xpos*, *ypos*);
float *xpos*, *ypos*;

Description

Places the origin of the graph xpos inches to the right and ypos inches above the current origin (the initial origin is at the exact lower left corner of the page). Also sets the axis coordinate system to inches.

See Also

cg_aorigin

Example

```
cg_init( 1, 1, 1.0 );
for(i=0; i<4; i++) {
   cg_text( 0.0, 0.0, 0.0, "example" ); /* Print at default origin.*/
   cg_rorigin(1.0, 1.0 );
}
cg_showpage();
```

cg_setboundingbox, cg_settitle , cg_setcreator, cg_setprolog, cg_set_top_comments

Summary

int cg_setboundingbox (char *bbox);
int cg_settitle (char *title);
int cg_setcreator (char *creator);

int cg_setprolog (char *prolog);

int cg_set_top_comments (char *comments);

Description

Cg_setboundingbox(), cg_settitle(), and cg_setcreator() sets %%BoundingBox:, %%Title:, and %%Creator: lines of the header, respectively. If you are creating an EPS file, use cg_setboundingbox(). Otherwise, a letter-size bounding box will be specified. Note that the argument for cg_setboundingbox() is a pointer to a string.

Alternatively, in one call, Cg_setprolog() allows you to set multiple lines of header comments including but not limited to those settable by the above three individual functions. Don't forget to include "%!PS-Adobe-3.0" or "%!PS-Adobe-3.0 EPSF-3.0" as the first line if this function is used.

Cg_set_top_comments() allows you to insert comments that describe the plots such as the original data source and plot parameters. These comments are inserted just after the prolog. These comments are also made into an Acrobat annotation box if the PS file is later converted into a PDF file via an Acrobat Distiller. These comments make the plot PS/EPS files traceable, therefore its use is highly recommended.

All of these functions [and cg_use_stdout() and cg_useflexcolor()] must be called **before cg_init()**.

Example

```
char *comment[8192];       /* enough space to contain all comments */

cg_settitle("CGminimal - example of Cgraph usage");
cg_setboundingbox("0 0 270 270");
cg_setcreator(argv[0]);

/* PostScript comment lines must start with a % character. */
sprintf(comment, "%% %s\n", prog_version);
sprintf(comment+strlen(comment), "%% Original data file: %s\n", filename);
sprintf(comment+strlen(comment), "%% Analysis parameter-1: %g\n", radius1);
/* ... more stuff to record as needed .. */
cg_set_top_comments(comment);    /* Comments that go directly after prolog. */

cg_init(rot, expand, scale);
```

cg_set_output_filename

Summary

void cg_set_output_filename(char *filename);

Description

Sets the current output filename to a string passed as an argument. It must be called before cg_init(). If it is not called, a temporary filename is automatically assigned, which may be obtained by cg_get_output_filename(). If you have a definite path for saving the output, use this. In a typical mode of operation, a PS preview application generally allows saving the document being viewed via the "Save As" menu item. If cg_init() is called multiple times in a loop, this function must be called before cg_init() ***every time***.

See Also

cg_get_output_filename

Example

```
cg_launch_preview(0);         /* do not launch preview */
cg_set_output_filename("/Users/me/PS/myoutput.ps");
cg_init(rot, expand, scale);

... lots of drawing here

cg_showpage();
```

cg_showpage

Summary

int cg_showpage();

Description

Prints the created page on the laserwriter. ALWAYS necessary. ALWAYS the last command. This function closes the output PS file and pass the file to a PostScript viewer application, from which you can generally save and print the PS file. To print more pages, you must reinitialize the system with cg_init() call.

Example

```
   .     .
   .     .     /* old page info */
   .     .
   .     .
cg_showpage(); /* Prints out the old page */

cg_init( 1, 1, 1.0 );
cg_text( 5.0, 5.0, 45.0, "example");
cg_showpage(); /* Prints out the new page. */
```

cg_stroke

Summary

int cg_stroke();

Description

Strokes the path created by **cg_line**s and **cg_rline**s. Resets dashtype to a solid, black (on the laserwriter) line.

See Also

cg_closepath, cg_dash, cg_gray, cg_line, cg_linewidth, cg_rline

Example

```
cg_init( 1, 1, 1.0 );
cg_dash( 2, 3.0 );
cg_move( 0.0, 0.0 );
cg_line( 5.0, 5.0 );
cg_stroke();         /* Draws a line to (5.0, 5.0), */
cg_move( 0.0, 0.0 );
cg_line(3.0, 4.0);
cg_stroke();
cg_showpage();
```

cg_text

Summary

int cg_text(*xpos*, *ypos*, *dir*, text);
float *xpos*, *ypos*, *dir*;
char *text;   /* pointer to a character string */

Description

Prints the text given to it, starting at the position given by xpos and ypos in inches from the current origin. Dir chooses the direction [in degrees counterclockwise from the horizontal] in which to print.

See Also

cg_textalign, cg_font, cg_wraptext, cg_xlabel, cg_ylabel, cg_centerlabel

Example

```
int number = 987;
float fnumber = 1.23456789;
char strbuf[128];

cg_init( 1, 1, 1.0 );
cg_text( 1.0, 1.0, 0.0, "example1");         /* Print at current origin.*/
cg_text( 1.0, 2.0, 10.0, "example2");        /* Print at 1 inch to the right */
                     /* and up from the origin, at 10 degrees from the */
                     /* horizontal.*/
sprintf(strbuf,"Do this to print #s: %d %f", number, fnumber);
cg_text(1.0, 5.0, 0.0, strbuf);
cg_showpage();
```

cg_textalign

•Summary

int cg_textalign (float x, float y, float rotate, char *textstring,
int xalign, int yalign, int flag)

•Description

This is perhaps the most general text alignment function in the cgraph library.
The basic idea is this: One can draw a "box" around a textstring and attach a coordinate system for alignment to the box. In our case, the coordinate system looks like.
(xalign, yalign) =

(0,2)	(1,2)	(2,2)
(0,1)	(1,1)	(2,1)
(0,0)	(1,0)	(2,0)

(the text itself is centered at the (1,1) spot).

The location of the text string "box" specified by (xalign, yalign) is positioned at the coordinate (x, y). For example, choosing (xalign, yalign) = (0, 1) will position the midpoint of the left boarder of the text string "box" at the coordinate (x, y).

The coordinate system for the (x,y) coordinates is selected by flag : 1 is domain, 0 is inches.
Rotate specifies the angle at which text is printed.

•Examples

```
cg_textalign(5.,5.,0.,"Saruman was right!!!", 1,1, 0);    /* prints the textstring
        centered at 5 inches, 5 inches. No rotation */

cg_textalign(5.,5.,0.,"Saruman was right!!!", 0,2,0);     /* prints the textstring
        so that the upper left corner of "S" is at 5 inches, 5 inches
        No rotation */
```

cg_useflexcolor

•Summary

int cg_useflexcolor (int flag)

•Description

If you wish to generate PS/EPS file that automatically chooses B/W or color depending on whether the imaging device supports color, use flag = 2. This may not be desired in general. Therefore, the default flag value is 0, which forces the use of B/W on any device. However, regardless of the flag used, the PS/EPS file always contains dual specifications for gray and color. If you use this function, it must be called **before** all other cg_*() functions.

flag = 0	"/__UseColor false def" is made active in the output PS file **(DEFAULT)**
flag = 1	"/__UseColor true def" is made active in the output PS file
flag = 2	"/__UseColor {statusdict begin /processcolors where {pop processcolors}{1} ifelse end 2 ge} def" is made active in the output PS file

It is not generally necessary to call this function. A simple editing of just one line in the PS file will change the use of color flexibly, if the source code utilizes appropriate flex color functions such as cg_grayrgbcolor() and cg_markergraycolor().

•See Also

cg_grayrgbcolor, cg_markergraycolor()

•Examples

The output PS/EPS file contains the following if cg_useflexcolor() is **not called**.
Change **false** to **true** below if you wish to make the file to use color. Automatic detection of color support in choosing B/W or color, may be enabled by uncommenting the last /__UserColor... line shown below.

```
--- near the top of generate PS/EPS file ---
 ...
% @@@@ Define ’true’ to enable color [works only if cg_grayrgbcolor() is used].
/__UseColor false def

% Let the use of color or B/W dependent on imaging device
% by checking if the device supports color or is B/W.
% /__UseColor {statusdict begin /processcolors where {pop processcolors}{1}
                                                     ifelse end 2 ge} def
```

cg_use_stdout

•Summary

int cg_use_stdout (int flag)

•Description

If you wish to send output to the standard output, e.g,, for piping into Ghostscript togenerate bitmaps automatically, call this function with flat=1 before cg_init(). If you do not call (default), it will be sent to a temporary file (/tmp/.cg-xxxxx.eps) and a PostScript preview application will be lauched.

flag = 0	Output to temp file and open a previewer **(DEFAULT)**
flag = 1	Output to the standard output.

•See Also

cg_init()

cg_xaxis, cg_yaxis

Summary

int cg_xaxis(*size, min, max, offset, ticsep,* numsep);
int cg_yaxis(*size, min, max, offset, ticsep*, numsep);
float *size, min, max, offset, ticsep;*
int numsep;

Description

Creates the linear X or Y axis starting at the origin and extending size inches in the appropriate direction.

Min and max define the numbers you want at the origin and the end of the axis.

Offset controls how far [in inches] the axis will be from the origin of the other axis.

Ticsep defines how far apart [in the units of min and max] you want the tick marks on the axis to be.

Numsep defines at what intervals the ticks should be numbered. Use **cg_linax_style** to change axis style detail, such as the length of tickmarks.

See Also

cg_axis_enable, cg_linax_style, cg_mesh, cg_xlog, cg_ylog

Example

```
cg_init( 1, 1, 1.0 );
cg_rorigin( 3.0, 3.0 );          /* Draws at the middle of the screen.*/
cg_xaxis( 5.0, 0.0, 40.0, 0.0, 5.0, 2);
   /* 1. Draws an 5 inches long axis in the X-dir.
      2. The axis ranges from 0 to 40.
      3. The ticks are separated by 5.0.
      4. The ticks are numbered every 2 ticks.
   */
cg_showpage();
```

cg_xlabel, cg_ylabel

•Summary

int cg_xlabel (char *textstring)
int cg_ylabel (char *textstring)

•Description

These functions are designed for labelling the x and y axes. The text string will automatically be center-justified along the axis.

cg_xlog, cg_ylog

Summary

int cg_xlog(*size, min, max, offset*);
int cg_ylog(*size, min, max, offset*);
float *size, min, max, offset;*

Description

Creates a logarithmic axis, using the same arguments as **cg_xaxis** and **cg_yaxis**, but without ticsep or numsep. Of course, min must never be equal to or less than 0. Min and max will be truncated by the program to one significant digit [min rounded down, max rounded up]. Use **cg_logax_style** to change axis style, such as tick length.

See Also

cg_axis_enable, cg_logax_style, cg_mesh, cg_xaxis, cg_yaxis

Example

```
cg_init( 1, 1, 1.0 );
cg_rorigin( 1.0, 3.0 );
cg_xlog( 5.0, 0.01 , 100.0, -0.3 );    /* logarithmic X axis 5 inches long */
                           /* from 0.01 through 100.  X axis is offset */
                           /* 0.3 inches downward from the origin */
cg_showpage();
```

COMMON MISTAKES:

MOST COMMON: Sending an integer as an argument to a function when it should be a floating point [e.g. **cg_linewidth**(1) where it should be **cg_linewidth**(1.0)]. This problem is often difficult to detect and is very common.

OTHERS: Not **strok**ing a line, or stroking it before the entire path has been defined. Although **cg_line**(x,y,color) draws lines *on the screen* as soon as it is called, this is not so in a PostScript device. This function only defines invisible paths which may be used for many operations. The paths are painted into visible lines by **cg_stroke**().

If you get a strange fill that you did not intend, it is usally the result of left-over path definitions that you forgot to use up by cg_stroke().