NAME

gp—a simple 2d graphics package

SYNOPSIS

```
#include "gp.h"
```

DESCRIPTION

gp is a simple, device independent, two-dimensional vector graphics package. It is mainly a tool for research in portable graphics, but has been used for "serious" work.

The package is implemented in two layers:

- gp, which handles the viewing transformation, and
- dv, which handles the device.

The layer gp is device independent and merely takes care of mapping user coordinates to device coordinates before calling dv primitives, which do the actual drawing.

The layer dv implements the drawing metaphor defined in gp for a particular device. A layer dv must be written for each device.

The philosophy is that the devices are "intelligent", in the sense that they "know" how to perform the required drawings. In most cases, the device is actually another graphics package. The main function of dv is to map the gp drawing metaphor onto the native drawing metaphor of the device, thus proving a common API for using different devices.

FUNCTIONS

The functions available in gp can be divided into the following categories:

- control: `gpopen`, `gpclose`, `gpclear`, `gpflush`, `gpwait`;
- viewing: `gpunview`, `gpviewport`, `gpview`, `gpwindow`;
- primitives: `gpbegin`, `gpbox`, `gpline`, `gpplot`, `gppoint`, `gptext`, `gptri`;
- attributes: `gpcolor`, `gpfont`, `gpmark`, `gppalette`, `gprgb`;
- input: `gpevent`.

SUMMARY

`void gpbegin(int` *c*`)`

begin poly primitive with code *c*.

`void gpbox(real` *xmin*`, real` *xmax*`, real` *ymin*`, real` *ymax*`)`

draw solid rectangular box with diagonal (*xmin*, *ymin*)—(*xmax*, *ymax*).

`void gpclear(int` *wait*`)`

clear device, possibily waiting for user input before acting.

`void gpclose(int` *wait*`)`

close graphics package, possibily waiting for user input before acting.

`int gpcolor(int` c`)`
select color number c as the current drawing color.

`void gpend(void)`
end last open poly primitive.

`char* gpevent(int` *wait*`, real*` x`, real*` y`)`
get next event in queue.

`void gpflush(void)`
flush graphics buffer.

`int gpfont(char*` *name*`)`
select font *name* as the current text font.

`void gpline(real` x_1`, real` y_1`, real` x_2`, real` y_2`)`
draw line segment from (x_1, y_1) to (x_2, y_2).

`void gpmake(void)`
compute viewing transformation.

`void gpmark(int` *size*`, char*` *mark*`)`
set size and type of marker.

`real gpopen(char*` *title*`)`
open graphics package.

`int gppalette(int` c`, char*` *name*`)`
bind color *name* to color number c.

`void gpplot(real` x`, real` y`)`
plot a marker at (x, y).

`int gppoint(real` x`, real` y`)`
add point (x, y) to current poly primitive.

`int gprgb(int` c`, real` r`, real` g`, real` b`)`
bind rgb color (r, g, b) to color number c.

`void gptext(real` x`, real` y`, char*` s`, char*` *mode*`)`
draw text s at (x, y) with alignment defined by *mode*.

`void gptri(real` x_1`, real` y_1`, real` x_2`, real` y_2`, real` x_3`, real` y_3`)`
draw solid triangle with vertices (x_1, y_1), (x_2, y_2), (x_3, y_3).

`void gpunview(real*` x`, real*` y`)`
perform inverse viewing transformation on device point (x, y).

`void gpview(real*` x`, real*` y`)`
perform viewing transformation on user point (x, y).

`real gpviewport(real` *xmin*`, real` *xmax*`, real` *ymin*`, real` *ymax*`)` define viewport in normalized device coordinates.

`void gpwait(int` *t*`)`

sleep *t* miliseconds or wait for user input.

`real gpwindow(real` *xmin*`, real` *xmax*`, real` *ymin*`, real` *ymax*`)`

define window in user coordinates.

SEE ALSO

`dv`, CORE, GKS, SRGP, and several others.

BUGS

`gp` is yet another graphics package.

Several `gp` functions are actually `#define`'d as the corresponding `dv` functions.

REFERENCES

J. R. Rankin, *Computer Graphics Software Construction*, Prentice Hall, 1989.

W. M. Newman and R. F. Sproull, Principles of Interactive Computer Graphics, McGraw–Hill, 1979.

J. D. Foley, A. van Dam, S. K. Freiner and J. F. Hughes, *Computer Graphics: Principles and Practice*, Addison–Wesley, 1990.

AVAILABILITY

`gp` is available free of charge by anonymous `ftp` from `ftp.rnp.impa.br` (`147.65.1.11`). The file is `~ftp/pub/graphics/gp.tar.Z`.

AUTHOR

Please send your comments, bug reports, suggestions, etc ... to

Luiz Henrique de Figueiredo (`lhf@visgraf.impa.br`)

NAME

`gpbegin`—begin poly primitive

SYNOPSIS

`void gpbegin(char` *c*`)`

DESCRIPTION

All parameters in `gp` functions are scalars. Some primitives need a list of points as input; these are called *poly primitives.*

A common solution for passing lists as parameters is by using arrays. In the case of points, these arrays are either arrays of structures or separate arrays for each coordinates. (Some graphics packages even mix coordinates in the same array.) The problem with arrays is that it is very likely that the data structure used by the application is not exactly the one needed by the graphics packages. This problem is solved in `gp` by listing the points inside a `gpbegin`–`gpend` pair:

```
gpbegin(c);
 gppoint(x1,y1);
 gppoint(x2,y2);
 ...
 gppoint(xn,yn);
gpend();
```

The parameter *c* selects one of the following poly primitives:

`'l'`	open polygonal line
`'p'`	closed polygonal line
`'f'`	filled polygon
`'m'`	polymarker

Nothing is drawn before `gpend` is called. The attributes used for drawing, such as color, are the ones which are current by the time `gpend` is called. Thus, there are no multi-color poly primitives.

Although it is not required, it is recommended that only `gppoint` occurs inside a `gpbegin`–`gpend` pair.

SEE ALSO

`gpend`, `gppoint`, `dvbegin`

BUGS

`gpbegin` is actually `#define`'d as `dvbegin`.

NAME

gpbox—draw solid rectangular box

SYNOPSIS

`void gpbox(real` *xmin*, `real` *xmax*, `real` *ymin*, `real` *ymax*)

DESCRIPTION

`gpbox` draws a rectangle, filled with the the color selected by the last call to `gpcolor`.

This rectangle is aligned with the coordinate axes: x ranges between *xmin* and *xmax*, and y ranges between *ymin* and *ymax*.

It is not required that $xmin < xmax$, or that $ymin < ymax$; the output is the same in any case.

SEE ALSO

`gpbegin`, `gptri`, `dvbox`

BUGS

Fill style is always solid and cannot be changed.

Some graphics systems specify rectangles by the coordinates of two diagonal vertices. In these systems, the argument order for a box function is *xmin*, *ymin*, *xmax*, *ymax*.

NAME

gpclear—clear device

SYNOPSIS

`void gpclear(int wait)`

DESCRIPTION

`gpclear` erases everything drawn on the device, leaving it empty. An empty device has all its pixels set to color 0. After `gpopen`, the device is empty.

If *wait* is non-zero, then `gpclear` waits for user input before clearing.

If *wait* is zero, then the device is cleared at once.

In hardcopy devices, clearing implies the beginning of a new page.

SEE ALSO

`gpcolor`, `gpopen`, `gpwait`, `dvclear`

BUGS

`gpclear` is actually `#define`'d as `dvclear`.

NAME

gpclose—close graphics package

SYNOPSIS

`void gpclose(int` *wait*`)`

DESCRIPTION

After everything is drawn, `gp` must be closed by calling `gpclose`. Closing means restoring the state before `gpopen`.

If *wait* is non-zero, then `gpclose` waits for user input before closing.

If *wait* is zero, then `gp` is closed at once.

SEE ALSO

`gpopen`, `gpwait`, `dvclose`

BUGS

`gpclose` is actually `#define`'d as `dvclose`.

NAME

`gpcolor`—select color

SYNOPSIS

`int gpcolor(int` c`)`

DESCRIPTION

Color selection in `gp` is based on a *colormap*: colors are bound to color numbers, by which they are later accessed.

The binding of color numbers to real colors is done with `gppalette` and `gprgb`.

All output after a call to `gpcolor` uses color number c. If c is not a valid color number, then the current color is not changed.

Color is 0 used for the background. Thus, drawing with color 0 can be used for erasing without clearing.

After `gpopen`, the current color is color 1, which is initially bound to black.

RETURN VALUE

`gpcolor` returns the color previously in use. This can be useful for local color changes.

If c is not a valid color number, then `gpcolor` returns $-n$, where n is the number of colors in the colormap. This can be useful for palette selection.

SEE ALSO

`gpopen`, `gppalette`, `gprgb`, `dvcolor`

BUGS

The only non-destructive way of knowing the current color is by artificially changing the color and then restoring it: `gpcolor(c=gpcolor(0))`.

The only way of knowing the size of the colormap is by selcting an invalid color: `n=-gpcolor(-1)`.

`gpcolor` is actually `#define`'d as `dvcolor`.

NAME

gpend—end poly primitive

SYNOPSIS

```
void gpend(void)
```

DESCRIPTION

All parameters in gp functions are scalars. Some primitives need a list of points as input; these are called *poly primitives*. This problem is solved in gp by listing the points inside a gpend–gpend pair.

gpend ends the last open poly primitive; this makes the drawing appear on the device.

The attributes used for drawing, such as color, are the ones which are current by the time gpend is called. Thus, there are no multi-color poly primitives.

SEE ALSO

gpbegin, gppoint, dvend

BUGS

gpend is actually #define'd as dvend.

NAME

gpevent—get event

SYNOPSIS

`char* gpevent(int` *wait*`, real*` x`, real*` y`)`

DESCRIPTION

gpevent is the only input function in gp. It reports key presses, button presses, button releases and pointer motion. In window systems, it also reports redraw requests and size changes.

The gp input model is based on a event queue. gpevent reports the next event in the queue.

If *wait* is non-zero, then gpevent waits until the next event occurs, if the queue is empty.

If *wait* is zero, then gpevent returns at once, if the queue is empty,

RETURN VALUES

gpevent returns a textual report of the event. It also reports the user coordinates (x, y) of the position of the pointer when the event occured.

The format of the report is the following:

"*k*+"	key k has been pressed;
"b*i*+"	button i has been pressed;
"b*i*-"	button i has been released;
"m*i*+"	pointer motion with buttons i pressed;
"i*i*+"	pointer is idle with buttons i pressed;
"r+"	redraw request;
"s+"	the window has new size (x, y).

After + or -, the report contains the state of the modifiers SHIFT, CONTROL and META, in this order.

Thus, "m13+CM" means “pointer motion with buttons 1 and 3 pressed, and CONTROL and META down”.

SEE ALSO

dvevent

BUGS

Not all devices are interactive.

There is no way of selecting events.

There is no way of detecting double clicks (add time stamp?).

Key releases are not reported, due to autorepeat.

The string report is a static buffer.

Parsing event reports is a bit boring.

NAME

`gpflush`—flush graphics buffer

SYNOPSIS

```
void gpflush(void)
```

DESCRIPTION

Some devices buffer graphics requests for efficiency. `gpflush` flushes the request buffer, forcing all pending drawings to appear.

Flushing is automatically done in `gpwait` and `gpevent`. Thus, there is no need for explicit flushing before interacting with the user. Nevertheless, flushing may be necessary for animation.

SEE ALSO

`gpevent`, `gpwait`, `dvflush`

BUGS

`gpflush` is actually `#define`'d as `dvflush`.

NAME

`gpfont`—select font

SYNOPSIS

`int gpfont(char*` *name*`)`

DESCRIPTION

After a call to `gpfont`, all text drawn with `gptext` uses the font described by the string *name*.

The possible values for *name* are device dependent; it is usually the name of a font, but can contain additional information, such as size.

`gpopen` selects a default font.

RETURN VALUE

`gpfont` returns 1 if the change of font was done, 0 otherwise.

If `gpfont` fails, then the current font is unchanged.

SEE ALSO

`gpopen`, `gptext`, `dvfont`

BUGS

There is no way of knowing the current font. Thus, there is no way to change the font locally.

`gpfont` is actually `#define`'d as `dvfont`.

NAME

gpline—draw line segment

SYNOPSIS

```
void gpline(real x1, real y1, real x2, real y2)
```

DESCRIPTION

gpline draws a solid line segment joining the points (x_1, y_1) and (x_2, y_2). The color of this segment is the color selected by the last call to gpcolor.

SEE ALSO

gpcolor, dvline

BUGS

Line style and width cannot be changed.

NAME

`gpmake`—compute viewing transformation

SYNOPSIS

`void gpmake(void)`

DESCRIPTION

The coordinates used in `gp` primitives are user coordinates.

The mapping from user coordinates to device coordinates is called the *viewing transformation*, and is defined by giving a rectangle in user coordinates, called the *window*, and a rectangle in normalized device coordinates, called the *viewport*. The viewing transformation is computed by mapping the window onto the viewport. This computation takes the aspect ratio of the device into account, and is performed by `gpmake`.

SEE ALSO

`gpopen`, `gpunview`, `gpviewport`, `gpview`, `gpwindow`

BUGS

There is no need to explicitly call `gpmake`: it is called automatically every time the window or the viewport is changed.

NAME

`gpmark`—set size and type of marker

SYNOPSIS

`void gpmark(int` *size*`, char*` *mark*)

DESCRIPTION

SEE ALSO

`gpplot`, `dvmark`

BUGS

`gpmark` is actually `#define`'d as `dvmark`.

NAME

`gpopen`—open graphics package

SYNOPSIS

`real gpopen(char*` *title*`)`

DESCRIPTION

Before anything can be drawn, `gp` must be opened by calling `gpopen`.

`gpopen` sets the following minimal defaults:

- the window and the viewport are set to the unit square;
- color 0 is set to be white and color 1 is set to be black;
- color 1 is selected as the current color.

The string *title* is used for error messages and for window titles in window systems.

RETURN VALUE

`gpopen` returns the aspect ratio of the device, represented by the quotient x/y.

SEE ALSO

`gpclose`, `gpcolor`, `gpviewport`, `gpwindow`, `dvopen`

BUGS

There is no way to open several windows in window systems: calling `gpopen` many times does *not* work the way it should in this case.

There is no way to change window titles in window systems.

NAME

`gppalette`—bind color name to color number

SYNOPSIS

`int gppalette(int` *c*`, char*` *name*`)`

DESCRIPTION

Color selection in `gp` is based on a *colormap*: colors are bound to color numbers, by which they are later accessed.

The binding of color numbers to real colors is done with `gppalette` and `gprgb`.

`gppalette` binds the color named *name* to the number *c*. On some devices, re-binding a color number to a different real color may immediately affect existing pixels. This behaviour, if present, should not be relied upon. Ideally, the colormap should be set before any output is drawn.

Color is 0 used for the background. When color number 0 is rebound, the background is only guaranteed to change after the next `gpclear`.

After `gpopen`, color 0 is bound to white and color 1 is bound to black. All other color numbers are unbound.

RETURN VALUE

`gppalette` returns 1 if the binding was done, 0 otherwise.

SEE ALSO

`gpcolor`, `gpopen`, `gprgb`, `dvpalette`

BUGS

Color names are device dependent. There is no way of knowing which names are valid, but common names should be ok. In particular, the following names are safe: black, blue, brown, cyan, green, grey, grey0, . . . , grey100, magenta, red, white, yellow.

There is no way of knowing the current binding. Thus, there is no way to change the binding locally.

Re-binding the current color may not work immediately.

`gppalette` is actually `#define`'d as `dvpalette`.

NAME

gpplot—plot a marker

SYNOPSIS

`void gpplot(real` x`, real` y`)`

DESCRIPTION

to be written.

SEE ALSO

gpmark, dvplot

BUGS

gpplot is actually #define'd as dvplot.

NAME

gppoint—add point to poly primitive

SYNOPSIS

`int gppoint(real x, real y)`

DESCRIPTION

All parameters in gp functions are scalars. Some primitives need a list of points as input; these are called *poly primitives.* This problem is solved in gp by listing the points inside a gpend–gpend pair.

gppoint adds the point (x, y) to the current poly primitive.

RETURN VALUES

gppoint returns the number of points in the current poly primitive.

SEE ALSO

gpbegin, gpend, dvpoint

NAME

gprgb—bind rgb color to color number

SYNOPSIS

`int gprgb(int` c`, real` r`, real` g`, real` b`)`

DESCRIPTION

Color selection in gp is based on a *colormap*: colors are bound to color numbers, by which they are later accessed.

The binding of color numbers to real colors is done with gppalette and gprgb.

gprgb binds the color with RGB components (r, g, b) to the number c. The components r, g and b are real numbers between 0 and 1.

On some devices, re-binding a color number to a different real color may immediately affect existing pixels. This behaviour, if present, should not be relied upon. Ideally, the colormap should be set before any output is drawn.

Color is 0 used for the background. When color number 0 is rebound, the background is only guaranteed to change after the next gpclear.

After gpopen, color 0 is bound to white and color 1 is bound to black. All other color numbers are unbound.

RETURN VALUE

gprgb returns 1 if the binding was done, 0 otherwise.

SEE ALSO

gpcolor, gpopen, gppalette, dvrgb

BUGS

There is no way of knowing the current binding. Thus, there is no way to change the binding locally.

Re-binding the current color may not work immediately.

gprgb is actually #define'd as dvrgb.

NAME

`gptext`—draw text

SYNOPSIS

void gptext(real *x*, real *y*, char* *s*, char* *mode*)

DESCRIPTION

`gptext` draws the text string s at point $p = (x, y)$. The position of the text with respect to this point is given by the string *mode*, using the bounding box b of the text:

"n"	p is the midpoint of the top horizontal size of b;
"s"	p is the midpoint of the bottom horizontal size of b;
"e"	p is the midpoint of the right vertical size of b;
"w"	p is the midpoint of the left vertical size of b;
"c"	p is the center of b;
"ne"	p is top right corner of b;
"se"	p is bottom right corner of b;
"nw"	p is top left corner of b;
"sw"	p is bottom left corner of b; this is the default.

Moreover, if the first character of *mode* is `!`, then the text is drawn in opaque mode: the bounding box b is cleared to the background color before drawing the text.

The color of the text is the color selected by the last call to `gpcolor`.

The font of the text is the font selected by the last sucessful call to `gpfont`.

SEE ALSO

`gpcolor`, `gpfont`, `dvtext`

BUGS

Text cannot be draw vertically or rotated, unless you can ask for this with `gpfont`. The “correct” way of drawing rotated text is by using stroke text from a client library.

There is no way of knowing the extent of a text string.

NAME

`gptri`—draw solid triangle

SYNOPSIS

`void gptri(real` x_1`, real` y_1`, real` x_2`, real` y_2`, real` x_3`, real` y_3`)`

DESCRIPTION

`gptri` draws a solid triangle with vertices (x_1, y_1), (x_2, y_2) and (x_3, y_3).

The color of this triangle is the color selected by the last call to `gpcolor`.

SEE ALSO

`gpbegin`, `gpcolor`, `dvtri`

BUGS

Fill style is always solid and cannot be changed.

This functions is kept for historical reasons. It has been rendered obsolete by `gpbegin`. `gptri(x1,y1,x2,y2,x3,y3)` is exactly the same as

```
gpbegin('f');
 gppoint(x1,y1);
 gppoint(x2,y2);
 gppoint(x3,y3);
gpend();
```

NAME

gpunview—perform inverse viewing transformation

SYNOPSIS

```
void gpunview(real* x, real* y)
```

DESCRIPTION

The coordinates used in gp primitives are user coordinates. They must be converted to device coordinates before being used in device primitives.

The mapping from user coordinates to device coordinates is called the *viewing transformation*. The viewing transformation is computed by mapping the window onto the viewport.

gpunview takes a point (x, y) in device coordinates and maps it to user coordinates. Note that gpunview returns the result in the same variables, which must be passed by reference. Note also that these variables are real and not int, even though device coordinates are int.

gpunview is typically used for special event handling.

SEE ALSO

gpview, gpviewport, gpwindow

NAME

gpview—perform viewing transformation

SYNOPSIS

`void gpview(real* x, real* y)`

DESCRIPTION

The coordinates used in gp primitives are user coordinates. They must be converted to device coordinates before being used in device primitives.

The mapping from user coordinates to device coordinates is called the *viewing transformation.* The viewing transformation is computed by mapping the window onto the viewport.

gpview takes a point (x, y) in user coordinates and maps it to device coordinates. Note that gpview returns the result in the same variables, which must be passed by reference. Note also that these variables are real and not int, even though device coordinates are int.

gpview is only needed for special applications, since the main function of gp is precisely to perform the viewing transformation before calling dv primitives.

SEE ALSO

gpunview, gpviewport, gpwindow

NAME

gpviewport—define viewport

SYNOPSIS

real gpviewport(real *xmin*, real *xmax*, real *ymin*, real *ymax*)

DESCRIPTION

The coordinates used in gp primitives are user coordinates.

In the viewport rectangle defined by gpviewport, x ranges between *xmin* and *xmax*, and y ranges between *ymin* and *ymax*. It is not required that $xmin < xmax$ or that $ymin < ymax$. This is can be used for special coordinate systems.

The viewport defines where the output is to appear in the device. All output is clipped to the viewport.

Normalized device coordinates are used to specify the viewport to avoid having to know the size of the device. In these normalized coordinates, the bottom left point is $(0, 0)$. The top right point is (x, y), where $x \leq 1$ or $y \leq 1$, with at least one equality (two, for square devices). The aspect ratio x/y of the device is returned by gpopen.

After gpopen, the viewport is set to the unit square. Thus, only the largest square in the device is initially used for output. For using all available display space, the following code can be used:

```
real a=gpopen("use it all");
if (a<1) gpviewport(0,a,0,1); else gpviewport(0,1,0,1/a);
```

More frequently, what is desired is to fit a drawing into the available space *without distortion*. This means that we need to find the largest viewport that is similar to the window. The following code can be used for this:

```
real d=gpopen("use it all");
real w=gpwindow(xmin,xmax,ymin,ymax);
real vx=1;
real vy=1;
if (d>1) vx*=d; else vy*=d;
if (w>1) vx*=w; else vy*=w;
gpviewport(0,vx,0,vy);
```

RETURN VALUE

gpviewport returns the aspect ratio of the viewport, represented by the quotient x/y.

SEE ALSO

gpopen, gpunview, gpview, gpwindow

BUGS

Some graphics systems specify rectangles by the coordinates of two diagonal vertices. In these systems, the argument order for a viewport function is *xmin*, *ymin*, *xmax*, *ymax*.

NAME

gpwait—sleep or wait for user input

SYNOPSIS

`void gpwait(int` *t*`)`

DESCRIPTION

If t is positive, then gp sleeps for t miliseconds.

If t is negative, then gp waits for user input. In this context, user input is a key press or a button release.

Automatic flushing is done before gpwait.

The common cases of waiting before clearing the device and waiting before closing the package are handled by arguments to the respective functions.

SEE ALSO

gpclear, gpclose, gpevent, gpflush, dvwait

BUGS

gpwait is actually #define'd as dvwait.

NAME

`gpwindow`—define window

SYNOPSIS

`real gpwindow(real` *xmin*`, real` *xmax*`, real` *ymin*`, real` *ymax*`)`

DESCRIPTION

The coordinates used in `gp` primitives are user coordinates.

The mapping from user coordinates to device coordinates is called the *viewing transformation*, and is defined by giving a rectangle in user coordinates, called the *window*, and a rectangle in normalized device coordinates, called the *viewport*. The viewing transformation is computed by mapping the window onto the viewport. This computation takes the aspect ratio of the device into account.

In user coordinates, x increases from left to right, and y increases from bottom to top, as in mathematics.

In the window rectangle defined by `gpwindow`, x ranges between *xmin* and *xmax*, and y ranges between *ymin* and *ymax*. It is not required that *xmin* < *xmax* or that *ymin* < *ymax*. This is can be used for special coordinate systems.

After `gpopen`, the window is set to the unit square.

RETURN VALUE

`gpwindow` returns the aspect ratio of the window, represented by the quotient x/y.

SEE ALSO

`gpopen`, `gpunview`, `gpview`, `gpviewport`

BUGS

Some graphics systems specify rectangles by the coordinates of two diagonal vertices. In these systems, the argument order for a window function is *xmin*, *ymin*, *xmax*, *ymax*.

NAME

`dv`—`gp` device dependent layer

SYNOPSIS

```
#include "gp.h"
```

DESCRIPTION

`gp` is a simple device independent two-dimensional graphics package. It is implemented in two layers:

- `gp`, which handles the viewing transformation, and
- `dv`, which handles the device.

The layer `gp` is device independent and merely takes care of mapping user coordinates to device coordinates before calling `dv` primitives, which do the actual drawing.

The layer `dv` implements the drawing metaphor defined in `gp` for a particular device. A layer `dv` must be written for each device.

The philosophy is that the devices are "intelligent", in the sense that they "know" how to perform the required drawings. In most cases, the device is actually another graphics package. The main function of `dv` is to map the `gp` drawing metaphor onto the native drawing metaphor of the device, thus proving a common API for using different devices.

FUNCTIONS

The functions available in `dv` can be divided into the following categories:

- control: `dvopen`, `dvclose`, `dvclear`, `dvflush`, `dvwait`;
- primitives: `dvbegin`, `dvbox`, `dvline`, `dvplot`, `dvpoint`, `dvtext`, `dvtri`;
- attributes: `dvcolor`, `dvfont`, `dvmark`, `dvpalette`, `dvrgb`;
- input: `dvevent`.

SUPPORTED DEVICES

The following devices are currently supported:

X11	X window system
bgi	Borland graphics interface
gl	Silicon Graphics graphics library
idraw	Interviews graphics editor
mac	Macintosh QuickDraw
msc	Microsoft C graphics library
ps	Encapsulated PostScript
sunview	Sunview window system
windows	Microsoft Windows

SEE ALSO

`gp`, `dv/X11`, `dv/bgi`, `dv/gl`, `dv/idraw`, `dv/mac`, `dv/msc`, `dv/ps`, `dv/sunview`, `dv/windows`

NAME

`dv/X11`—details of `dv` driver for X window system

DESCRIPTION

The X driver was developed under X11.4, but should run in both X11.3 and X11.5.

The default window geometry used in `dvopen` is $512 \times 512 + 0 + 0$. This can be changed by setting the `GP` environment variable to a standard geometry string. The window title is the string given to `dvopen`.

The driver tries to minimize exposures by asking for backing store.

Colors are allocated in the default colormap. This implies that color allocation depends on what colors have been allocated by previous clients. This may create problems when running applications that need many private colors. It is not too difficult to modify the driver to use private colormaps. The driver colormap has as many entries as the default colormap.

The window is automatically cleared when color 0 is rebound.

The cursor changes to `XC_icon` in `dvclear(1)`, and to `XC_pirate` in `dvclose(1)`.

The color names understood by `dvpalette` depend on the server, but are the ones in `/usr/lib/X11/rgb.txt`. `dvpalette` also understands X RGB notation: `#RRRRGGGGBBBB`.

`dvfont` does not free the old font information.

The size of a poly primitive is limited by XMaxRequestSize.

Key press reporting in `dvevent` is incomplete. It is easy to modify the driver to report pointer motion without a button pressed.

NAME

`dv/bgi`—details of `dv` driver for Borland graphics interface

DESCRIPTION

The bgi driver was developed under Turbo C 2.0, but should run in Borland C.

The driver assumes that software for controlling a mouse has been loaded and handles interrupt 0x33. The driver dies if no mouse can be found.

The mouse cursor is only present during the execution of `dvevent`. When the application is drawing, the mouse cursor is not visible.

The driver tries to select a good color for the mouse cursor based on the background color. The color of the mouse cursor is color number 15.

The driver does automatic video card detection, but assumes that the appropriate `.BGI` files is somehow available (the driver calls `initgraph` with `pathtodriver==NULL`). The driver dies if `initgraph` fails. Despite automatic detection, it is assumed that the card is an EGA or a VGA.

Font selection does not work.

Opaque text does not work.

Clipping does not work the wayt it should because `setviewport` changes the origin too.

The cursor does not change in `dvclear(1)` or in `dvclose(1)`.

The color names understood by `dvpalette` are the ones listed in `gppalette`.

Key press reporting in `dvevent` is incomplete.