Integral Personal Computer

Alpha/Graphics Window Reference Manual

HEWLETT PACKARD

Edition 1 October 1985

82865-90005

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Portable Computer Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330 U.S.A.

Printing History

Edition 1 October 1985 Mfg. No. 82865-90005

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Contents

1. Introduction

The *Alpha/Graphics Window Reference Manual* for the Integral Personal Computer describes the programmatic interface of the alpha/graphics window driver.

Built into Read-Only Memory (ROM) of Release 5.0 of the operating system, the AG window driver provides primitives to support high-performance alpha and graphics functions in the same window.

Alpha/graphics windows are designed to emulate UNIX™ device types. That is, alpha/graphics windows are manipulated by means of **open()**, **ioctl()**, **read()**, and **close()** system calls. Furthermore, as **tty** devices, alpha/graphics windows can be thrown into *raw mode* in order to intercept keystrokes as they happen.

This manual documents the low-level **ioctl()** requests that your application can use to draw vectors, write rasters, read rasters, and fill rasters in alpha/graphics windows.

Note that the functions described in this manual are specific to the Integral Personal Computer. Using these **ioctl()** requests means that your code will run only on the Integral Personal Computer. However, the resulting performance gains and added functionality may well be worth the cost in portability.

2. Terminology

AG	An abbreviation for alpha/graphics. Sometimes used to refer to the alpha/graphics window or window type.
block	A rectangular area of the display or working surface. Usually defined by the *x,y* coordinates of the upper left corner and a height and width.
pixel	A single dot on the display, usually represented as a single *bit* of character information. Pixels on the Integral

™ UNIX is a trademark of AT&T.

Personal Computer have an aspect ratio of 1:1.

raster — A rectangular area of the display or working surface. Used interchangeably with *block*.

raster surface — The size of the display buffer used to hold alpha/graphics information, set by the `w_gen1` and `w_gen2` fields of the `windio` structure. The size of the alpha/graphics window may be *smaller* than the size of the raster surface, so that only a portion of the raster surface will appear at a time under the window.

vector — A line drawn from one point to another in an alpha/graphics window.

window driver — On the Integral Personal Computer a body of code residing in the operating system ROM that handles one of three window types: alpha (or Term0) windows, graphics (or plotter emulator) windows, and alpha/graphics (or AG) windows.

3. Alpha/Graphics Window Description

The purpose of the alpha/graphics windows type is to provide alpha and graphics functions at a higher performance level than that provided by the alpha window driver and graphics window driver. For example, the AGPOLYLINEW request allows an unlimited number of vectors to be drawn with a single **ioctl()** call, a highly efficient way to draw many lines in a window.

3.1 Limitations

The increased graphics performance means that some features are not available:

1. Unlike the graphics window driver, the AG window driver provides no support for HP-GL, the language understood by Hewlett-Packard plotters.
2. Unlike other **tty** devices, alpha/graphics windows do not support the **write()** system call. All **write()**s issued to an alpha/graphics window will be ignored. (There are alternate **ioctl()** requests provided to do the same work.)

3. No scaling or user-coordinate system mapping is provided. Coordinates are the natural system of the device being used (dots). The origin is in the lower left corner of the raster with positive x and y values to the right and up respectively.

4. Moves and stretches (from the [Move] and [Stretch] keys on the System menu) of alpha/graphics windows may produce unexpected results.

3.2 Moves and Stretches

The bottom line regarding the last limitation above is that your application must either:

- Disallow a move or stretch by the user, or
- Allow a move or stretch only under the direct control of your application.

The idea is that your application should have the NO_MOVESTRETCH bit of the `windio` structure set so that the user cannot move an alpha/graphics window off the display or change the size of an alpha/graphics window.

If you do want something to happen on a stretch, you need to catch the EVENT_STRETCH signal and do what you want to do for yourself.

For example, assume that the user has activated a stretch by pressing [Stretch], moving the display pointer to the desired location, and pressing [Select]. The NO_MOVESTRETCH bit setting ensures that no stretching occurs; however, your application will receive the stretch (or in this case, the attempted stretch) signal. In its signal handling routine, your application can read the display pointer location and can explicitly change the size of the window and redraw the display.

3.3 Accessing an Alpha/Graphics Window

Accessing an alpha/graphics window from your application requires that you:

- Create an alpha/graphics window using a low-level window manager routine.
- Open the alpha/graphics window using an **open()** system call.
- Use the *file descriptor* of the alpha/graphics window returned from the **open()** to perform subsequent vector and raster operations.

For code samples that show how to create, open, and manipulate alpha/graphics windows, refer to Chapter 7, *Alpha/Graphics Windows*, of the programmer's guide.

3.4 Sequence of Events

Normally, output to an alpha/graphics window proceeds as follows:

1. The operating system intercepts the user **ioctl()** request and hands it to the AG window driver.
2. Based on the pass parameters, the AG window driver sets up the appropriate data fields and calls the display driver at a lower level.
3. The display driver channels the data to the Graphics Processor Unit (GPU), which finally blasts it to the display.

3.5 Coordinate System

In the coordinate system used by the AG window driver, the units are dots, and the origin is in the lower left corner with positive x and y values to the right and up respectively.

3.6 Clipping

Because all operations affect only the window, clipping is performed only on the window area. Thus the resulting clipping area is the intersection of the effective window boundary, the user-defined clipping rectangle, and the physical display.

The effective window boundary is the window boundary clipped by the display. Thus, if the window extends off of the display, the effective window boundary is that part of the window that fits on the display.

The user-defined clipping rectangle is initialized to the size of the display. The clipping area changes any time the window changes size or the user-defined clipping rectangle is changed via the ioctl call or the window's position on the display is changed.

The positioning of the user-defined clipping rectangle is relative to the raster surface. Thus, if the window is moved on the raster surface, the user-defined clipping rectangle will stay in the same position on the raster surface and the resulting clipping area on the window may change. Thus, it's best to reset the user-defined clipping rectangle after moving the window on the raster surface.

Clipping affects the various operations as follows:

- Vectors are clipped to the clipping rectangle and the part of the vector that is outside the clipping rectangle is discarded.
- Block write is clipped. Thus, only the unclipped portion of the block will be written to the window.

- Block read is clipped. Thus, only the unclipped portion of the block will be returned to the application.
- Block fill will be clipped and the fill pattern will be shifted in both the x and y dimensions to make it relative to the upper left corner of the window rather than relative to the upper left corner of the display.
- Block copy, which can be considered as block read followed by a block write, will follow the same rules as block read and write--that is, clipped on the read and clipped on the write.
- Character placement will be clipped to partial characters.

---

4. Keyboard Handling

There are three types of keystrokes that the user can press:

1. *Typing keystrokes* generate 8-bit ASCII codes.
2. *System keystrokes* are handled by the window manager. These keys include:
   - [Shift][Stop].
   - [Shift][Reset] and [Shift][Ctrl][Reset].
   - [Menu] and [User/System].
   - The [CTRL] arrow keys.
   - [Select] and [Shift][Select].
   - [Caps].
   - [Shift][Print].
3. *Other keystrokes* are seen as a high-escape (hexadecimal 0x9B, or an escape with the high bit set) character followed by the appropriate ASCII 8-bit code.

The codes returned from the third category are detailed below:

Keystroke	Character Codes
right-arrow	0x9b 0x00
down-arrow	0x9b 0x01
left-arrow	0x9b 0x02
up-arrow	0x9b 0x03 (but see below)
up-arrow+right-arrow	0x9b 0x04
up-arrow+left-arrow	0x9b 0x05
down-arrow+left-arrow	0x9b 0x06
down-arrow+right-arrow	0x9b 0x07
[Shift] right-arrow	0x9b 0x08
[Shift] down-arrow	0x9b 0x09
[Shift] left-arrow	0x9b 0x0a
[Shift] up-arrow	0x9b 0x0b
[Shift] up-arrow+right-arrow	0x9b 0x0c
[Shift] up-arrow+left-arrow	0x9b 0x0d
[Shift] down-arrow+left-arrow	0x9b 0x0e
[Shift] down-arrow+right-arrow	0x9b 0x0f
[Tab]	0x9b 0x10
[Shift][Tab]	0x9b 0x11
[Shift][Next]	0x9b 0x12
[Shift][Prev]	0x9b 0x13
home up	0x9b 0x14
home down	0x9b 0x15
[Shift][Delete char]	0x9b 0x16
[Shift][Insert char]	0x9b 0x17
[[Shift][Delete line]	0x9b 0x18
[Shift][Insert line]	0x9b 0x19
[Clear display]	0x9b 0x1a
[Clear line]	0x9b 0x1b
[Shift][Clear display]	0x9b 0x1c
[Shift][Clear line]	0x9b 0x1d
[Back space]	0x9b 0x1e
[Enter]	0x9b 0x3e
[Return]	0x9b 0x3f
[f1]	0x9b 0x40
[f2]	0x9b 0x41
[f3]	0x9b 0x42
[f4]	0x9b 0x43
[f5]	0x9b 0x44

Keystroke	Character Codes
[f6]	0x9b 0x45
[f7]	0x9b 0x46
[f8]	0x9b 0x47
[Shift][f1]	0x9b 0x48
[Shift][f2]	0x9b 0x49
[Shift][f3]	0x9b 0x4a
[Shift][f4]	0x9b 0x4b
[Shift][f5]	0x9b 0x4c
[Shift][f6]	0x9b 0x4d
[Shift][f7]	0x9b 0x4e
[Shift][f8]	0x9b 0x4f

Note that the second byte of the up-arrow keycode is 0x03 which will appear to the **tty** front end of the AG window driver as if the user typed a [CTRL][C], thus sending an interrupt signal to the process. To avoid this, the ISIG bit should be disabled during program initialization.

The [Stop] and [Break] keys return the **tty** defined VQUIT and FRERROR values respectively. Keys such as function keys and arrow keys are simply passed to the application for interpretation.

---

5. Alpha/Graphics Window Functions

Listed below are the descriptions of the functions, or **ioctl()** requests, provided by the AG window.

Any error codes are returned by storing the code in the global variable **errno**. Any request not recognized by the AG window driver will be passed to the window manager. If the window manager doesn't recognize the request, the value **EINVAL** will be returned.

Note

The following routines are specific to the Integral Personal Computer operating system and will not port to other HP-UX systems.

Be sure that you isolate these system-dependent routines in a separate module of your software.

Throughout these pages, the header file **<scrn/ag.h>** is included in code listings. The **<scrn/ag.h>** file is shipped on the C language preprocessor disc and includes the data definitions and structures recognized by the alpha/graphics window driver.

It's *not* necessary to link any but the standard C runtime library in order to use any of the following functions.

5.1 AGLINEW - Draw a Single Line

The **AGLINEW** request draws a single vector to the window using the window coordinate system.

The vector is clipped to a rectangle that is the intersection of the window boundary, the display, and the user-defined clipping rectangle. Any part of the vector that lies outside of this clipping rectangle will be discarded.

```
#include <scrn/ag.h>

ioctl (filedes, AGLINEW, &vec)

struct vect_info {
    short    x;
    short    y;
    short    pen_down;
} vec;
```

Starting from the current position, the pen is moved to the x,y position indicated. If pen_down is TRUE (non-zero), a line will be drawn. Any value for x and y that can fit in an short (-32768 to 32767) is legal. The values will be clipped as described above.

5.2 AGPOLYLINEW - Draw Multiple Lines

The **AGPOLYLINEW** request draws multiple vectors to the window using the window coordinate system.

The vectors are clipped to a rectangle that is the intersection of the window boundary, the display, and the user-defined clipping rectangle. Any part of a vector that lies outside of this clipping rectangle will be discarded.

```
#include <scrn/ag.h>

ioctl (filedes, AGPOLYLINEW, &vlist)

struct poly_info {
    unsigned int count;
    char  *cp;
} vlist;
```

--where cp is a pointer to an array of vect_info structures as shown below:

```
struct vect_info {
    short    x;
    short    y;
    short    pen_down;
};
```

The argument is a pointer to a poly_info structure. This structure contains a count of the number of vectors to be plotted and a pointer to an array of plot_info vector structures.

The first and last elements of the vector list are checked to make sure that valid memory is being referenced; otherwise, no data integrity checks are made. An attempt to reference non-existent memory will cause **EFAULT** to be returned.

Starting from the current position, the pen is moved to each x,y position indicated in order. If pen_down for an x,y pair is TRUE (non-zero), a line will be drawn.

This moving/drawing will continue until count vectors have been processed. Except for the multiple vector capability, this function performs like the single vector function AGLINEW.

5.3 AGBLKRDW - Read a Block of Data

The AGBLKRDW request reads and returns a block of data representing a rectangular area of the window.

Only that portion of the block that lies within the clipping area will be returned.

```
#include <scrn/ag.h>

ioctl (filedes, AGBLKRDW, &blkrd)

struct  agblk {
    int     x;
    int     y;
    int     height;
    int     width;
    int     boffset;
    int     aoffset;
    char    sysflag;
    unsigned char *data;
} blkrd;
```

The x and y parameters are the coordinates of the upper left corner of the block to be read. The height and width parameters define the size of the block in dots. The boffset parameter is the bit offset from the word boundary to the left-hand edge of the block. The aoffset parameter is the address offset from the word boundary at or beyond the right edge of the block to the word boundary at or before the left edge of the block. The sysflag parameter is reserved for use by the operating system. The information is placed in the buffer pointer to by data.

The AGBLKRDW request is affected by the current clipping rectangle, which is the intersection of the display boundary, the window and the user-defined clipping rectangle. Only that portion of the described block that is not clipped will be returned.

If boffset is non-zero the first 'boffset' bits of each row will be read from the applications data space, masked and ANDed with the data so that the applications data will not be damaged by window data outside of the area being requested.

The buffer is assumed to be large enough to hold the data. The first and last bytes are checked to determine is they are in valid memory; otherwise, no data integrity checks are performed.

5.4 AGBLKWRW - Write a Block of Data

The AGBLKWRW request writes a bit-map to an area on the alpha/graphics window.

The block sent is clipped to the intersection of the window, the display and the user-defined clipping rectangle. If the block is entirely outside of the clipping rectangle, the block will be clipped to nothing, but no error will be returned.

```
#include <scrn/ag.h>

ioctl (filedes, AGBLKWRW, &blkwr)

struct    agblk {
    int    x;
    int    y;
    int    height;
    int    width;
    int    boffset;
    int    aoffset;
    char   sysflag;
    unsigned char *data;
} blkwr;
```

The x and y parameters are the coordinates of the upper left corner of the block to be written. The height and width parameters define the size of the block in dots. The boffset parameter is the bit offset from the word boundary to the left-hand edge of the block. The aoffset parameter is the address offset from the word boundary at or beyond the right edge of the block to the word boundary at or before the left edge of the next row in the block. The sysflag parameter is reserved for use by the operating system.

The write takes place according to the current replacement rule, which may be set by means of the AGWRRR request prior to using this function. The information is in a buffer that is pointed to by the data parameter.

The information for the line should start at 'boffset' bits from the left hand edge of the word. If the width of the data block is not the same as the width of the buffer, then the aoffset parameter should be set to the number of words from the end of one row of data to the start of the next row. The buffer is assumed to contain all the information needed and in the proper format.

5.5 AGPOLYBLKWRW - Write Multiple Blocks of Data

The AGPOLYBLKRW request is similar to the AGBLKWRW call except that the argument is a pointer to a structure containing the number of block writes and a pointer to an array of block write structures.

```
#include <scrn/ag.h>

ioctl(filedes, AGPOLYBLKWRW, &blist)

struct    poly_info {
    unsigned int    count;
    char    *cp;
} blist;
```

--where cp is actually a pointer to an array of agblk structures as shown below:

```
struct    agblk {
    int     x;
    int     y;
    int     height;
    int     width;
    int     boffset;
    int     aoffset;
    char    sysflag;
    unsigned char *data;
};
```

The argument is a pointer to a poly_info structure. This structure contains a count of the number of blocks to be written and a pointer to an array of agblk structures.

The first and last bytes of the array are checked to make sure that valid memory is being referenced; otherwise, no data integrity checks are made. An attempt to reference non-existent memory will cause EFAULT to be returned.

Besides performing multiple block writes, this request behaves like the single block write function, AGBLKWRW.

5.6 AGBLKFILLW - Fill a Block With Data

The AGBLKFILLW request fills a rectangular area (block) with a given fill pattern in an alpha/graphics window.

The operation is clipped; that is, only that portion of the block that is contained within the clipping area will be filled.

The fill pattern will be shifted in both dimensions if necessary to make the fill pattern relative to the top left corner of the window.

```
#include <scrn/ag.h>

ioctl(filedes, AGBLKFILLW, &blkfl)

struct  blkfill {
    int     x;
    int     y;
    int     width;
    int     height;
    unsigned char *fillp;
} blkfl;
```

The x and y parameters are the coordinates of the upper left hand corner of the block to be filled. The height and width parameters define the size of the block to be filled. The fillp parameter is a pointer to a 32 byte array that contains a 16 by 16 bit pattern of 1's and 0's to be used in the block fill.

The origin of the fill pattern is in the upper left corner of the window. Thus, the first line of the block will not necessarily have the first line of the fill pattern or start with the left edge of the pattern.

The current replacement rule is used for the fill. The replacement rule may be set by means of the AGWRRR request prior to invoking this function.

The fill pattern will be repeated inside the clipped block until the whole clipped block is covered with the pattern.

5.7 AGBLKCPW - Copy a Block of Data

The AGBLKCPW request copies data from one rectangular area (block) to another rectangular area in the alpha/graphics windows.

The source and destination rectangles for the copy are both clipped.

```
#include <scrn/ag.h>

ioctl(filedes, AGBLKCPW, &blkcp)

struct    blkcopy {
    int     x1;
    int     y1;
    int     x2;
    int     y2;
    int     height;
    int     width;
} blkcp;
```

The x1 and y1 parameters are the x and y coordinates of the upper left hand corner of the source block. The x2 and y2 parameters are the x and y coordinates of the destination block. The height and width parameters define the size of the block to be copied.

Overlapping blocks are allowed to be copied. They are copied in such a manner so that the destination block contains all the information of the source block.

The copy is made using the current replacement rule.

5.8 AGWRRR - Set the Replacement Rule

The **AGWRRR** request sets the replacement rule to be used for subsequent operations.

```
#include <scrn/disp.h>
#include <scrn/ag.h>

ioctl(filedes, AGWRRR, rule)
int rule;
```

The rule parameter is an integer that specifies the replacement rule. The replacement rule constants are defined below (and in **<scrn/disp.h>**):

Rule	Value
FORCE_ZERO	0
AND	1
AND_NOT_OLD	2

Rule	Value
NEW	3
AND_NOT_NEW	4
OLD	5
EXOR	6
OR	7
NOR	8
EXNOR	9
NOT_OLD	10
OR_NOT_OLD	11
NOT_NEW	12
OR_NOT_NEW	13
NAND	14
FORCE_ONE	15

5.9 AGSETCLIPW - Set User-Defined Clipping Rectangle

The **AGSETCLIPW** request allows the application to specify a clipping rectangle in the window coordinate space.

```
#include <scrn/ag.h>

ioctl(filedes, AGSETCLIPW, &cliparea)

struct  area {
    int     minx;
    int     maxx;
    int     miny;
    int     maxy;
} cliparea;
```

The resulting clipping rectangle is the intersection of the user-defined clipping rectangle and the window boundary (which may be affected by its location on the display). This clipping rectangle is recalculated whenever the window changes size or the user-defined clipping rectangle is changed.

5.10 AGCREATEFONT - Create a Font for Use With the AG Window

The **AGCREATEFONT** request directs the AG window driver to make a font available for use. If necessary--and if possible--the specified font will be downloaded from disc into the system font pool.

```
#include <scrn/disp.h>
#include <scrn/ag.h>

ioctl(filedes, AGCREATEFONT, &font)

struct font_data {
    int    esc_num;
    unsigned char esc_char;
    unsigned char width;
    unsigned char height;
    unsigned char cursval;
    unsigned char curs_top
    unsigned char curs_bot;
    long  id;
    unsigned char filenm[FILENAME_SIZE];
} font;
```

Only the filenm[] parameter must be supplied. The driver will fill in the rest of the information. If the designated font is already in the font pool it will not be loaded again; instead, the usage count will simply be incremented.

5.11 AGREMFONT - Remove Font

The **AGREMFONT** request informs the AG window driver that a font will no longer be needed.

The usage count for this font is decremented. If the usage count becomes zero, the font is removed from the font pool and the system reclaims the memory formerly occupied by the font.

```
#include <scrn/disp.h>
#include <scrn/ag.h>

ioctl(filedes, AGREMFONT, &font)

struct font_data {
    int    esc_num;
    unsigned char esc_char;
    unsigned char width;
    unsigned char height;
    unsigned char cursval;
    unsigned char curs_top
    unsigned char curs_bot;
    long  id;
    unsigned char filenm[FILENAME_SIZE];
} font;
```

The only element of the font structure that must be supplied is the id.

5.12 AGSELFONT - Select Font for Use

The AGSELFONT request informs the AG window driver that a previously created font is desired for use.

This capability allows an application to create several fonts and then select between them without the overhead of creating the font several times.

```
#include <scrn/disp.h>
#include <scrn/ag.h>

ioctl(filedes, AGSELFONT, font_id)

int font_id;
```

The font_id parameter is the id value returned by a previous AGCREATEFONT call. Note that this function must be called after an AGCREATEFONT before the created font may be used.

After this request, characters received by the character placing **ioctl()** call will be displayed according to the selected font.

5.13 AGWINPOS - Set Window Position on Raster

The AGWINPOS request is used to position the window on the raster surface.

```
#include <scrn/ag.h>

ioctl(filedes, AGWINPOS, &corner_location)

struct  scroll_xy {
    int     x;
    int     y;
} corner_location;
```

The scroll_xy structure is set to contain the x,y coordinates of the location on the raster surface where the *upper left corner* of the window will appear.

Coordinates that would place any part of the window off of the raster will result in the **errno** variable being set to **EINVAL**.

The coordinate system has 0,0 in the lower left corner of the raster with positive x and y values to the right and up respectively.

5.14 AGSETENHMASK - Set Enhancement Mask

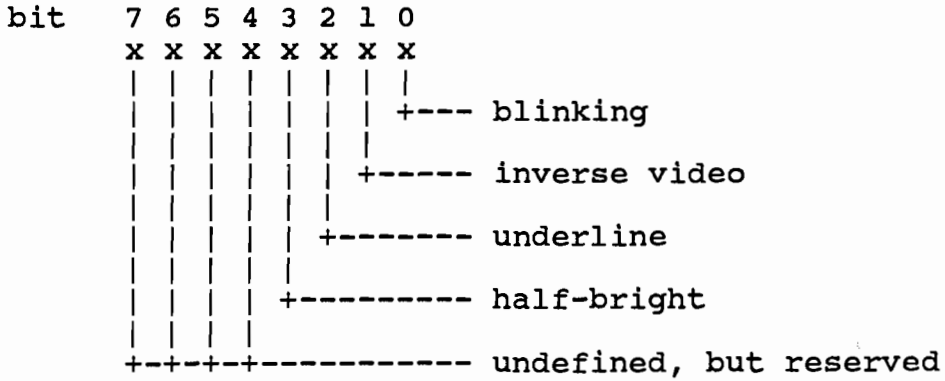
The **AGSETENHMASK** request allows an application to select enhancements to be used when character codes are received via the **write()** call.

```
#include <scrn/ag.h>

ioctl(filedes, AGSETENHMASK, mask)

int mask;
```

The enhancement mask is defined as follows:



The upper 24 bits of the mask are undefined. After receipt of this mask, all characters received will be displayed via this mask.

5.15 AGSYNCRW - Synchronize Raster Surface to Window

The **AGSYNCRW** request causes that portion of the raster surface under the window to be refreshed on the display--that is, to be copied to the window on the display.

```
#include <scrn/ag.h>

ioctl(filedes, AGSYNCRW, NULL)
```

Note that if the window has been drawn on since the last AGSYNCWR call, data may be lost.

5.16 AGSYNCWR - Synchronize Window to Raster Surface

The **AGSYNCWR** request causes the contents of the window to be copied to the raster surface.

```
#include <scrn/ag.h>

ioctl(filedes, AGSYNCWR, NULL)
```

The two synchronize requests exist solely to ensure that what you *think* is in the alpha/graphics window really *is* in the alpha/graphics window. Normally, window synchronization happens automatically, as when the user shuffles windows on the display.

5.17 AGMVWRW - Move the Pen and Write Characters

The AGMVWRW request is used to place characters to an alpha/graphics window.

The font and enhancement mask most recently selected is used.

```
#include <scrn/ag.h>

ioctl(filedes, AGMVWRW, &mv_write)

struct  mv_wr {
    int     x;
    int     y;
    char    *data;
} mv_write;
```

The pen is moved to the indicated x,y position and characters are placed in the window. The pen position is updated after every character.

Characters are clipped to the clipping area. If the 'x' value is negative, no move takes place before placing characters.

Note that normal position-affecting character codes such as carriage return and line feed have no effect.

5.18 AGPOLYMVWRW - Multiply Move Pen and Write Characters

The AGPOLYMVWRW request is identical to the AGMVWR request except that multiple move-and-writes are performed.

```
#include <scrn/ag.h>

ioctl(filedes, AGPOLYMVWRW, &poly_mvwr)

struct  poly_info {
    unsigned int  count;
    char  *cp;
};
```

--where cp is actually a pointer to an array of mv_wr structures.

```
struct  mv_wr {
    int     x;
    int     y;
    char    *data;
};
```

The count parameter is the number of move-and-writes to be performed. The cp parameter is a pointer to an array of mv_wr structures. Each move-and-write will be performed in turn as if several AGMVWRW calls have been made.

5.19 AGCLEARW - Clear the Window

The AGCLEARW request clears the specified alpha/graphics window.

The window is cleared according to the current background color (inverted or non-inverted).

```
#include <scrn/ag.h>

ioctl(filedes, AGCLEARW, NULL)
```

5.20 AGGETINFO - Get Information About an AG Window

The AGGETINFO request returns information about a specified alpha/graphics window.

Given a pointer to an aginfo structure, information about the current state of an alpha/graphics windows is written to the structure.

```
#include <scrn/ag.h>

ioctl(filedes, AGGETINFO, &info)

struct aginfo {
    int    penx;      /* pen x coordinate */
    int    peny;      /* pen y coordinate */
    int    w_id;      /* raster id number */
    int    gwinx;     /* location on raster where left
                         edge of window lies */
    int    gwiny;     /* location on raster where top
                         edge of window lies */
    int    ucminx;    /* min x of user clip rectangle */
    int    ucmaxx;    /* max x of user clip rectangle */
    int    ucminy;    /* min y of user clip rectangle */
    int    ucmaxy;    /* max y of user clip rectangle */
    int    cur_rr;       /* current replacement rule */
    int    font_width;  /* current font width  */
    int    font_height; /* current font height */
    int    font_id;     /* current font id number */
    int    enhmask;     /* current enhancements mask */
} info;
```

6. Further Information

For a first-hand view of the data definitions and structures honored by the AG window driver, examine the **<scrn/ag.h>** header file that is shipped on the C language preprocessor disc.

For a variety of code samples that show how to create alpha/graphics windows and perform vector and raster operations, refer to Chapter 7, *Alpha/Graphics Windows*, of the programmer's guide.

For keyboard handling information, refer to the raw mode information on *graphics* windows, Chapter 6, of the programmer's guide. Keyboard handling is quite similar between graphics and alpha/graphics windows.

For information on downloading and using fonts in an alpha/graphics window, refer to Chapter 8, *User-Defined Fonts*, of the programmer's guide.