M E M O

To:	Graphics Interest	Date: April 9, 1974
From:	R.G.Shoup, P.Baudelaire	Location: Palo Alto
Subject:	Graphics System 0.6	Organization: PARC/CSL

Archive category: Graphics

A simple graphics package, using the run-length picture memory hardware, has been written for the color graphics system. It is similar in flavor to Bob Flegal's "System 0", and is basically Point Array oriented. (A Point Array is the name used for a rectangular array of adjacent cells in a Cartesian space.) It should be easy to maintain and easy to use, and therefore it will provide a common basis for writing application and user programs for the color graphics system. Please report complaints and suggestions to the authors.

For a clearer definition of terms and conventions, see Appendix A. In any case, consult the appended definition file G06DEF which defines useful globals, constants and BCPL structures. Documentation on the BCPL floating point package (courtesy of Bob Sproull) is also appended.

Document convention: in the definition of procedures, optional arguments are enclosed inside square bracket pairs, possibly nested (but always with a common right bracket). For example, the notation:

```
        P( A [, B, C [,D ])
implies that the following calling sequences are possible:
        P( A )
        P( A, B, C)
        P( A, B, C, D).
```

An illegal number of arguments will generally cause an error message.

- [illegible] (
- [illegible]
- erio ([illegible], [illegible])
- vdisk
- default in [illegible]

STRUCTURES

Some procedures take an argument specifying a pair of coordinates (a picture memory or tablet position). Such an argument is defined by the BCPL structure POS:

POS.X:	x coordinate,
POS.Y:	y coordinate.

A POS vector has length POSLEN. In this document, this type of argument will be designated as a POINT.

Calls on the point array routines deal with three types of tables: WINDOW, RUN_TABLE, XREL_TABLE, structured as follows.

WINDOW:

The information about the rectangular window framing the point array is provided in a vector (of length WINLEN) defined by the BCPL structure WIN:

WIN.YORIG	y coordinate of the point array ORIGIN,
WIN.XORIG	x coordinate of the point array ORIGIN,
WIN.H	height H of window,
WIN.W	width W of window,
WIN.YOFF	relative y coordinate of the point array ORIGIN,
WIN.XOFF	relative x coordinate of the point array ORIGIN.

The ORIGIN of a point array is an arbitrary reference point used for defining the position of the point array. The coordinates of the lower left corner of the rectangular window are:

XORIG - XOFF, YORIG - YOFF.

A standard window which matches the picture memory is defined by the global vector PMWINDOW. It has H=480, W=640, and its ORIGIN is in the center of the rectangle.

RUN_TABLE:

For input to and output from the pictuer memory, point arrays are X-runlength encoded. The *VALUE* and runlength data are stored in RUN_TABLEs of length WIN.H words. There are NTABS of them (stored consecutively), where NTABS is the maximum number of runs on any scan line in the point array. NTABS is obviously a function of the size and the complexity of the point array. A RUN_TABLE is a table of RUNCODEs, one per scan line, indexed by Y line number relative to the window. Each word of the RUN_TABLE contains a *VALUE* and a ***RUN*** which are accessed using RUNCODE.VALUE and RUNCODE.RUN. Entries having RUN=0 are ignored.

XREL_TABLE:

This is a single table of length WIN.H which specifies the beginning x coordinate of each scan line relative to the left side of the rectangular window. Therefore a point array can have an arbitrary left contour. This table is normally used only for cursors (see FILLXTAB below).

THE PICTURE MEMORY

The Picture Memory (PM) consists of an array of points (480 by 640) which is addressable from the Nova and displayable on CRT video monitors. Coordinate values range from (y,x) = (0,0) at the lower left corner to (479,639) at the upper right. The PM dimensions are available as manifest constants in the system: PMBOT, PMTOP, PMLEFT, PMRIGHT, PMHEIGHT, PMWIDTH. Clipping is automatically provided to a rectangular area called the PMVIEWPORT. This viewport is by default the PMWINDOW. The PMVIEWPORT set by the following calls:

SETPMVIEWPORT(WINDOW):	set the PMVIEWPORT to be the given window;
SETPMVIEWPORT():	set the PMVIEWPORT to be the PMWINDOW.

Low-level routines are also provided which bypass this clipping operation.

Each point in the PM has a *VALUE* which represents the color in that cell. This *VALUE* is currently stored as and 8-bit number. Eventually, the hardware will allow the programmer to specify a *COLOR* to be associated with each *VALUE*. At present, however, the (fixed) mapping of a picture memory *VALUE* into RGB signals (red, blue and green) is as follows;

```
bit      7 6 5 4 3 2 1 0
goes to  R R R B G G G B .
```

Manifest constants are defined for the commonly used colors: BLACK, RED, GREEN, BLUE, YELLOW, MAGENTA, CYAN, WHITE.

Procedures which write into the PM look up each *VALUE* in a global *VALUE* translation table VALTRANSTABLE (256 entries) to get the actual *VALUE* to be sent to the picture memory. Therefore any arbitrary mapping of 256 *VALUES* into 256 *VALUES* can be made at writing time by apropriately setting the VALTRANSTABLE. Furthermore, any *VALUE* marked as *NULL* (a reversible operation) will be ignored and will not be sent to the picture memory. Thus holes or permanently transparent areas can exist in point arrays.

The following procedures operate on the VALTRANSTABLE:

SETVALTRANSTAB(*VALUE*, -1):	mark *VALUE NULL*
SETVALTRANSTAB(*VALUE*):	reset a *VALUE* to its original non-*NULL VALUE*
SETVALTRANSTAB(*VALUE*, *NEW_VALUE*):	assign a new *VALUE* to a *VALUE*.

The VALTRANSTABLE must be initialized (normally done by INITGSYS):

INITVALTRANSTABLE():	initialize to the identity mapping
INITVALTRANSTABLE(SOME_TABLE):	initialize to the given 256 entry table.

As a matter of policy, the VALTRANSTAB should be used only locally to rearrange the color space. Routines which use it should reset it to the identity mapping before exiting.

POINT ARRAY ROUTINES

PM CORE

Two basic procedures are provided for writing a point array from RUN_TABLEs into the picture memory, and for reading a point array from the picture memory into RUN_TABLEs. These two operations are only operative inside the rectangular viewport mentioned above. The MODE parameter indicates which PM quadrants to operate on or overlay and must have one of the following values (manifest constants):

Q0,Q1,BOTHQ:	write/read quadrant 0, quadrant 1, both quadrants;
OVQ0,OVQ1,OVBOTHQ:	overlay quadrant 0, quadrant 1, both quadrants;
Q0OVQ1:	write quadrant 0, overlay quadrant 1;
Q1OVQ0:	write quadrant 1, overlay quadrant 0.

About the interlaced TV fields: the global variable FIELDMODE specifies which fields are to be written or read. FIELDMODE must be set to one of the following values: FIELD0ONLY, FIELD1ONLY, BOTHFIELDS (default value).

<u>RUNTABSTOPM</u>(WINDOW, RUN_TABLES, NTABS, MODE [, XREL_TABLE [, UPDATE_XREL])

This routine writes RUN_TABLEs to the picture memory. The argument XREL_TABLE is not needed in the normal case (i.e., the left contour of the point array coincides with the left side of the window and thus all XRELs are equal to 0). If an XREL_TABLE is given, its values will be used as starting X coordinates (relative to the window) on each line. The table is not modified. If, however, the XREL_TABLE is given <u>and</u> the boolean argument UPDATE_XREL is *TRUE* , the given XREL_TABLE will be updated to contain the coordinates of the endpoint of the last run written or read on each scan line (again, relative to the window).

NTABS=<u>PMTORUNTABS</u>(WINDOW,RUN_TABLES,MAXNTABS,MODE [,XREL_TABLE [,UPDATE_XREL])

Here also, the arguments XREL_TABLE and UPDATE_XREL are optional, with the same conventions as above. This procedure returns NTABS --the number of tables actually filled of the MAXNTABS tables provided by the caller. If MAXNTABS is too small, the window will not be read in its entirety, but nothing bad will happen. The calling program can test NTABS against MAXNTABS to detect this. If PMTORUNTABS is called when the window has been completely read (all XRELS are at right boundary) then NTABS will be 0.

FILLXTAB(WINDOW, RUN_TABLES, NTABS, XREL_TABLE)

This is a somewhat ad-hoc procedure for "eliminating" the background color at the beginning and end of every scan line in a point array. Using the VALTRANSTABLE, it replaces any *NULL* runs at the beginning of each scan line by the appropriate XREL value, and it erases any *NULL* runs at the end. The RUN_TABLE is correspondingly reorganized.

This procedure is intended for use on cursor point arrays, for which it is important to replace initial *NULL* runs by XRELs, since displaying initial *NULL* runs wastes one frame time each.

VALUE=RPMPOINT(POINT)

WPMPOINT(POINT,VALUE)

RUNCODE=RPMRUN(POINT)

WPMRUN(POINT, RUNCODE)

These routines represent several flavors of low-level procedures for read/writing an individual value or run from/to the picture memory. No clipping is done (the picture memory will hang if improperly addressed) and no value translation operation takes place.

PM FILES

Two procedures are provided for direct input/output of point arrays to Nova DOS files. A point array file contains the following information: a WINDOW, a MODE, a COLOR_TABLE (not yet implemented), data in RUN_TABLE form, and the number of RUN_TABLEs. Details of the file format will be found in the definition file G06DEF. It is the responsibility of the user to open and close the files. The point array file procedures expect a channel number.

PMTOPAF(CHANNEL [, WINDOW [, MODE])

Reads a point array from picture memory to a file. The last two arguments are optional. The default values are:

```
        WINDOW:         the PMWINDOW (if omitted or 0)
        MODE:           BOTHQ
```

PAFTOPM(CHANNEL [, POINT [, MODE])

Writes a point array from a file to picture memory. The argument POINT specifies the picture memory position where the point array ORIGIN point is to be placed. The last two arguments are optional. The default values are:

```
        POINT:  the ORIGIN position specified in the file (if omitted or 0),
        MODE:   the MODE specified in the file.
```

(leaves the valtrainstate as used during PM writing.)

CORE FILES

READBEGPAF(CHANNEL, WINDOW)

This procedure reads a point array file header into the vector WINDOW. The byte pointer in the file is left pointing at the beginning of the RUN_TABLES.

RANDOM RUNS

A set of procedures is also provided for writing runs in random sequence into a given rectangular window in the picture memory. The runs are first entered in a set of user provided tables; then these tables are written into the picture memory, either at the request of the user, or when the number of runs for one scan line equals the capacity of the tables.

SETUPRANDRUNS(WINDOW, RUN_TABLES, NTABS, mode, XREL_TABLES)

This procedure sets up the given buffers used for storing *RUN*, *VALUE* and relative X coordinates. RUN_TABLES and XREL_TABLES must be two tables both of size H * NTABS, where H is the height of the window. NTABS will be the maximum number of runs per scan line which can be stored into the tables before their contents will be written into the picture memory.

FLUSHED=RANDRUN(POINT, VALUE, RUN)

This procedure stores appropriately in the tables, *RUN*, *VALUE* and an X coordinate. The argument POINT specifies the coordinates of the run relative to the window given in the most recent SETUPRANDRUNS call. If the number of runs stored in the table for the given Y value is equal to NTABS, the entire contents of the tables is written into the picture memory, the tables are cleared, and the procedure returns *TRUE*; it returns *FALSE* otherwise. Thus the value of FLUSHED tells whether the tables are empty.

FLUSHRANDRUNS()

This procedure writes the contents of the tables into the picture memory and clears them.

TABLET ROUTINES

PROX=READTABLET(POINT, PENSTAT)

This is the basic procedure which reads the (Scriptographics) tablet and returns the pen position and status information in two vectors defined respectively by the BCPL structures:

POS.X	x coordinate,
POS.Y	y coordinate;

and:

PENSTAT.DOWN	TRUE if pen is pressed down, FALSE otherwise,
PENSTAT.PROX	TRUE if pen is in proximity of the tablet,
PENSTAT.PUSH	TRUE when pen has just been pressed down,
PENSTAT.PULL	TRUE when pen has just been lifted up.

The x and y coordinates are scaled so that their range over the tablet is slightly larger than the PM coordinates range. Calling the routine PROX=RTABLET(POINT, PENSTAT) is equivalent except that unscaled coordinates will be returned. Note that DOWN is TRUE whenever the pen switch is closed, while PUSH is TRUE only for one call to READTABLET just after the pen switch has closed (and similarly for PULL).

Pointing and inking:

Two procedures are provided for pointing, cursor tracking and inking of the common sort, using an arbitrary point array as a cursor. If no point array is given, the default cursor will be used (a bitriangular bicolor cursor). The following list of arguments must be given for specifying *YOURCURSOR*:
WINDOW, RUN_TABLES, NTABS, XREL_TABLE.
Notice that the argument XREL_TABLE is not optional in this case. The previously mentioned procedure FILLXTAB can be useful in creating an XREL_TABLE. However, only one run per scan line can be displayed per frame time (1/30th of a second). Be aware that any cursor which contains more than one run per scan line (NTABS greater than one) will flicker "runwise".

Tracking and inking are specified by the mode arguments UP_MODE and DOWN_MODE. The default mode for both is tracking (OVBOTHQ). When the pen is down, the 2 colors of the default cursor are exchanged so that the state of the pen is made visible to the user.

PROX=TABLETDOWN(POINT_DOWN [, UP_MODE, DOWN_MODE [, *YOURCURSOR*])

This procedure returns whenever the pen is sensed down.

PROX=TABLETSTROKE(POINT_DOWN [, POINT_UP [, UP_MODE, DOWN_MODE[, *YOURCURSOR*])

This procedure is used primarily for indicating a single point or position. It first waits until the pen is up, without tracking. Then it returns after the pen has been pressed down and lifted up again. POINT_DOWN and POINT_UP (if provided) contain the coordinates of the positions where the pen was pressed down and lifted up. The cursor is tracked or inked, according to UP_MODE and DOWN_MODE.

Note that all of these routines return the boolean pen PROXIMITY as their value. This makes it easy for the program to use "pen-down-off-the-tablet" as a special case.

PROX=<u>GETRECT</u>(WINDOW)

This routine allows the user to define a rectangle via the tablet. Two corners are displayed as cursors to indicate the corners of the rectangle.

PROX=<u>GETWINDOW</u>(WINDOW)

This routine gets a rectangle definition plus an origin point from the tablet, thus defining a window. If the user presses the pen down off the tablet (PROX false) when either corner is being specified, the PMWINDOW values are returned as a default. If the pen is pressed down off the tablet area when the origin is expected, the origin will be taken as the center of the already-specified rectangle.

MISCELLANEOUS ROUTINES

WAITVERTINT()
Waits for next PM vertical interval to begin, then returns.

RESULT=EXIO(CONTENT_OF_AC0, IO_INSTRUCTION)

Low level procedure for executing individual NOVA I/O instructions. Useful manifest constants are defined in G06DEF.

LENGTH=FREESPACE()

VECTOR=DYNAVEC(LENGTH)

Ad-hoc low level procedures for pseudo-dynamic BCPL vector allocation. FREESPACE returns the amount of memory available, i.e. the amount of unused BCPL frame space. DYNAVEC expands the current frame space to allocate a local BCPL vector of size LENGTH. It will abort the calling program if the requirements are excessive.

CLEARPM([VALUE [, WINDOW])

A somewhat heterogeneous procedure:

CLEARPM():	clears the PM read/write logic
CLEARPM(VALUE):	set all the points in the PMWINDOW to the given value.
CLEARPM(VALUE, WINDOW):	set all the points in the given window to the given value (clipping done to the PMVIEWPORT as usual).

CHANGEPMVALUE(VALUE, NEW_VALUE)

Replace every occurrence of VALUE with NEW_VALUE in the picture memory. This operation takes one frame time. This is the gadget behind the flashing demo, thresholding, etc.

STUFFVEC(VECTOR, LENGTH, [, *UP TO 16 ARGUMENTS*])

This procedure stores the given arguments taken sequentially into all LENGTH words of VECTOR, repeating if necessary. If no arguments are given, the entire vector is set to zero.

SETPMCONFIG(ARGUMENT1, ARGUMENT2)

This procedure permits the user to set the picture memory configuration logic (not yet fully implemented). Currently, the only expected calls are:

```
        SETPMCONFIG():    reset the configuration to its standard state;
        SETPMCONFIG(0,1): connect A/D to picture memory input;
        SETPMCONFIG(0,0): reconnect Nova run-length interface
                          to the picture memory input.
```

The only current application is for input to the picture memory from the A/D converter, connected for instance to the camera or tuner through the video switches. Then, any procedure writing into the picture memory (CLEARPM, RUNTOPM, PAFTOPM, tablet procedures, etc.) will actually use input from the A/D. This routine will be more fully defined when the configuration logic is finished.

SETVIDSW(SWITCH, SOURCE, DESTINATION)

This routine allows the user to configure the video switch. Using this procedure requires some knowledge of the cable configuration (see posted diagram). G06DEF contains specially declared manifests to use as arguments to this procedure. For instance, the usual lab configuration (RGB color picture from the picture memory on the large monitor) is set by:

```
        SETVIDSW(RGBSWITCH, PICMEMA, MON3).
```

TTYWAIT([MESSAGE])

Outputs MESSAGE (if present) to the TTY and waits for a carriage return to be typed.

INITGSYS()

Initialize the VALTRANSTABLE, the PMVIEWPORT, BCPL I/O and other things. This routine must be called as the first action of the user program.

SUMMARY OF PROCEDURE CALLS

```
RUNTABSTOPM(WINDOW, RUN_TABLES, NTABS, MODE [, XREL_TABLE [, UPDATE_XREL])
NTABS=PMTORUNTABS(WINDOW,RUN_TABLES,MAXNTABS,MODE [,XREL_TABLE [,UPDATE_XREL])

PAFTOPM(CHANNEL [, POINT [, MODE])
PMTOPAF(CHANNEL [, WINDOW [, MODE ])
FILLXTAB(WINDOW, RUN_TABLE, NTABS, XREL_TABLE)
READBEGPAF(CHANNEL, WINDOW)

SETUPRANRUNS(WINDOW, RUN_TABLE, NTABS, MODE, XREL_TABLE)
FLUSHED=RANDRUN(POINT, VALUE, RUN)
FLUSHRANDRUNS()

VALUE=RPMPOINT(POINT)
WPMPOINT(POINT,VALUE)
RUNCODE=RPMRUN(POINT)
WPMRUN(POINT)

INITVALTRANSTABLE([ SOME_TABLE ])
SETVALTRANSTAB(VALUE [, NEW_VALUE ])
SETPMVIEWPORT([ WINDOW ])
```

PROX=<u>RTABLET</u>(POINT, PENSTAT)

PROX=<u>READTABLET</u>(POINT, PENSTAT)

PROX=<u>TABLETDOWN</u>(POINT_DOWN [, UP_MODE, DOWN_MODE [, *CURSOR*])

PROX=<u>TABLETSTROKE</u>(POINT_DOWN [, POINT_UP [, UP_MODE, DOWN_MODE [,*CURSOR*])

where *CURSOR* is the following argument list:
WINDOW, RUN_TABLES, NTABS, XREL_TABLES

PROX=<u>GETRECT</u>(WINDOW)

PROX=<u>GETWINDOW</u>(WINDOW)

RESULT=<u>EXIO</u>(AC0, IO_CODE)

<u>WAITVERTINT</u>()

<u>CLEARPM</u>([VALUE [, WINDOW])

<u>CHANGEPMVALUE</u>(VALUE, NEW_VALUE)

<u>SETPMCONFIG</u>([ARGUMENT1, ARGUMENT2])

<u>SETVIDSW</u>(SWITCH, SOURCE, DESTINATION)

LENGTH=<u>FREESPACE</u>()

VECTOR=<u>DYNAVEC</u>(LENGTH)

<u>STUFFVEC</u>(VECTOR, LENGTH [, *UP TO 16 ARGUMENTS*])

<u>TTYWAIT</u>(MESSAGE)

<u>INITGSYS</u>()

PRACTICAL DETAILS

You will find on the GRAPHICS NOVA (DP0):
the graphics package:

G06DEF	BCPL definition file,
G06A.BR, G06B.BR	BCPL procedures,
G06AC.SV	Assembly code procedures, with floating point routines and debugger;

and Ben Laws' IO package:

IOX	definition file,
IO1.BR, IO2.BR	BCPL IO procedures.

Loading command:

```
        BLDR/U G06AC/I your programs G06A G06B IO1 IO2

    or  BLDR/U/L/V ----------  (if load maps desired).
```

Your code should start with the statement:

```
        GET "G06DEF"
```

~~(which in turn contains: GET "IOX")~~ .

Your main graphics procedure should start with the call:

```
        INITGSYS()
```

(which in turn (will call: INITBCPLIO(2)).