

A NETWORK COMPATIBLE DISPLAY PROTOCOL

Bernhard Schweeger

February 1979
PP-79-1

Professional Papers do not report on work of the International Institute for Applied Systems Analysis, but are produced and distributed by the Institute as an aid to staff members in furthering their professional activities. Views or opinions expressed are those of the author(s) and should not be interpreted as representing the view of either the Institute or its National Member Organizations.

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

1. INTRODUCTION

At present IIASA offers to its scientific community a set of basic, but limited computer graphics capabilities. Recent computer-program development in the graphics area has created greater demand and more awareness of the possibilities of computer graphics as a visualisation tool in modelling. Some other application fields, like the use of graphics as a tool for program development, were still not touched at IIASA.

In response to this demand, IIASA decided to order additional graphics hardware (display terminals and graphics compatible printers) and to invest a certain amount of resources into the generation of appropriate software.

It is not the aim of this paper to discuss the benefits of graphics, but rather to identify the general graphics software requirements for IIASA and to propose a possible way of meeting them.

Thanks to J. Kulp, whose comments were very valuable during the preparation of this paper.

2. "MINI-GLOSSARY"

device: part of a computer or computer peripheral

display: any device where visual information of any type can be output; includes printer, plotter, COM, terminals...

display protocol: set of rules which enable a program to transfer visual information to a display device

addressing: specifying locations on a display; can be done in two ways: through a coordinate system or through symbol-box-units (or character positions).

communications protocol: set of rules which enable the communication of data between a group of remotely located devices.

computer graphics: special case of visual output, where the display information is generally described by coordinates and vectors; in this sense a historical term.

data flow (path): any type of information transfer from a device or program to another device or program. Better term than "data communication"

writing device: pen, electron beam, ink jet, typeball, typewheel etc. used to produce visual information on a display device.

picture: image produced on the display

cursor: optical marker of a certain position on the display. Some times indicates the position of the writing device. It is different from the target used in GIN (see below). Some devices support multiple cursors.

home: name of the upper left corner of the display or window.

pixel: shorthand of "picture element". Special term for a dot on a raster scan device. Each pixel can be set or erased selectively without altering others.

raster scan device: special type of display. The picture is made up of pixels. Each pixel has its corresponding memory bit. The memory is associated (bit-mapped) with a TV-like monitor and read at video-rate. The increasing market position of this device is due to the fact that memory becomes cheaper and cheaper. Currently monitors are available from 256x256 (low resolution) to 4096x4096 (very high resolution). Advanced devices not only allow bi-level display (black/white; one intensity), but also colour, different intensities, stereo view etc.

XOR-mode: short for "eXclusive OR"-mode. A term borrowed from Boolean logic, where an expression is true if one and only one of the operands is true. It is a mode in which raster scan devices can work. If a memory bit (or pixel) is set (=true), an attempt to set it again in XOR-mode will erase (=false) it. In this mode drawing and erasing are the same operation.

COM: abbreviation for "Computer Output on Microfilm"

font element: representation of a symbol in a standardised format either as a set of vectors (lines) or as dot matrix pattern. In this paper, a font will be considered as a sort of display macro (a small, non-recursive picture segment)

font: collection of font elements corresponding to potentially all ASCII characters.

kern: amount by which the intersymbol space is to be increased (or decreased) to achieve proper spacing with symbols that are "broader" above or below the base line.

coordinate system: imaginary space mapped to the display used to specify locations on the display; can be done in absolute or relative units.

device independent: not dependent on the characteristics of any particular device; every display has a different instruction format; to be able to control more than just one device, an intermediate pseudo instruction format is selected which is converted (post-processed) to the format that an individual device understands. The intermediate format is normally not understood directly by any device.

plot, graph: restrictive words for visual output. This paper tries to take a more general view, for which "displayed output" is a better term.

symbol: any alphanumeric, special or user-defined character; will normally be treated as a unit of information differently from other display information.

symbol-box: area of the display where a symbol can be put; comprises the symbol and the space to the adjacent symbol to the right and to the top. They can be of variable size.

picture segment, subpicture: a unit of display information defined separately

and which can be displayed with actual parameters (comparable to a sub-program). The implementation of this concept requires a high level of "local intelligence" (programmability) of the display device.

Compared to font elements, subsets are large and potentially recursive. Each subset has its individual name.

register: small unit of memory allocated to store a single value

window: area of the display limited by logical boundaries. It is associated with the concept of an environment (context). More than one window can be defined at a time, but only one window will be the current (active) window. Windows can overlap, but they are hierarchical in the sense that a display command can only have an effect within the current window, and a change of window can only be made to a "parent" or "child" window, i.e. to the level directly above or below it.

environment, context: set of values of different parameters which affect the behaviour of the writing device or the functioning of different commands. Only one context can be current, but different context can be stored, stacked or associated with different windows or different subsets.

stack: storage concept working by the LIFO (last in, first out) principle. Information can be pushed onto a stack or popped off a stack.

macro: set of commands stored locally and which can be executed by calling it.

clipping: suppressing display information outside of defined boundaries

rectangle: portion of the display whose boundaries are parallel to the x, y (and z) coordinates

display primitives: set of high level program facilities called by a user program to specify his picture. This is the lowest program level a normal application programmer is confronted with.

2D/3D: Two and Three dimensional display. Real 3D hardware does currently not exist and is projected for visualisation into a 2D space.

parallel processing: prerequisite for fast real-time display modification (e.g. for flight simulation). Only available on special hardware. A support of this facility in a later implementation in a display protocol should be possible.

GIN: abbreviation for "Graphical INput". There is a variety of devices (eg. light pen, track ball, tablet, joy-stick, "mouse" ...) which essentially only can change the current location of a target. GIN can to a certain extent be simulated with a regular keyboard. Real-time drawing requires software for interrupt handling and sampling. Since the target is often used to position the writing device, GIN can be an output concept for a display protocol.

target: location on the display directly associated with a GIN-device. It is not to be confused with a cursor.

3. IDENTIFICATION OF THE LOGICAL PARTS OF A DISPLAY SYSTEM

Description of operation (see fig. 1): An application program resides on some ("host"-) computer; it creates, with the help of user specifications and essentially non-graphical application data a device independent display file. This file is not necessarily a disk-file but rather a logical file. The normal user is never confronted directly with the creation of this file, but rather calls display primitives of reasonably high level (eg. draw lines, grids, labels, erase lines, select parameters).

The device independent display file is transformed by a device dependent display code generator into device dependent display code, which is now understood by the device driver, whose presence is not always obvious, since it is often partially configured in hardware and/or incorporated in the graphics output device (it does things like video-signal generation, plotter motor control and the like).

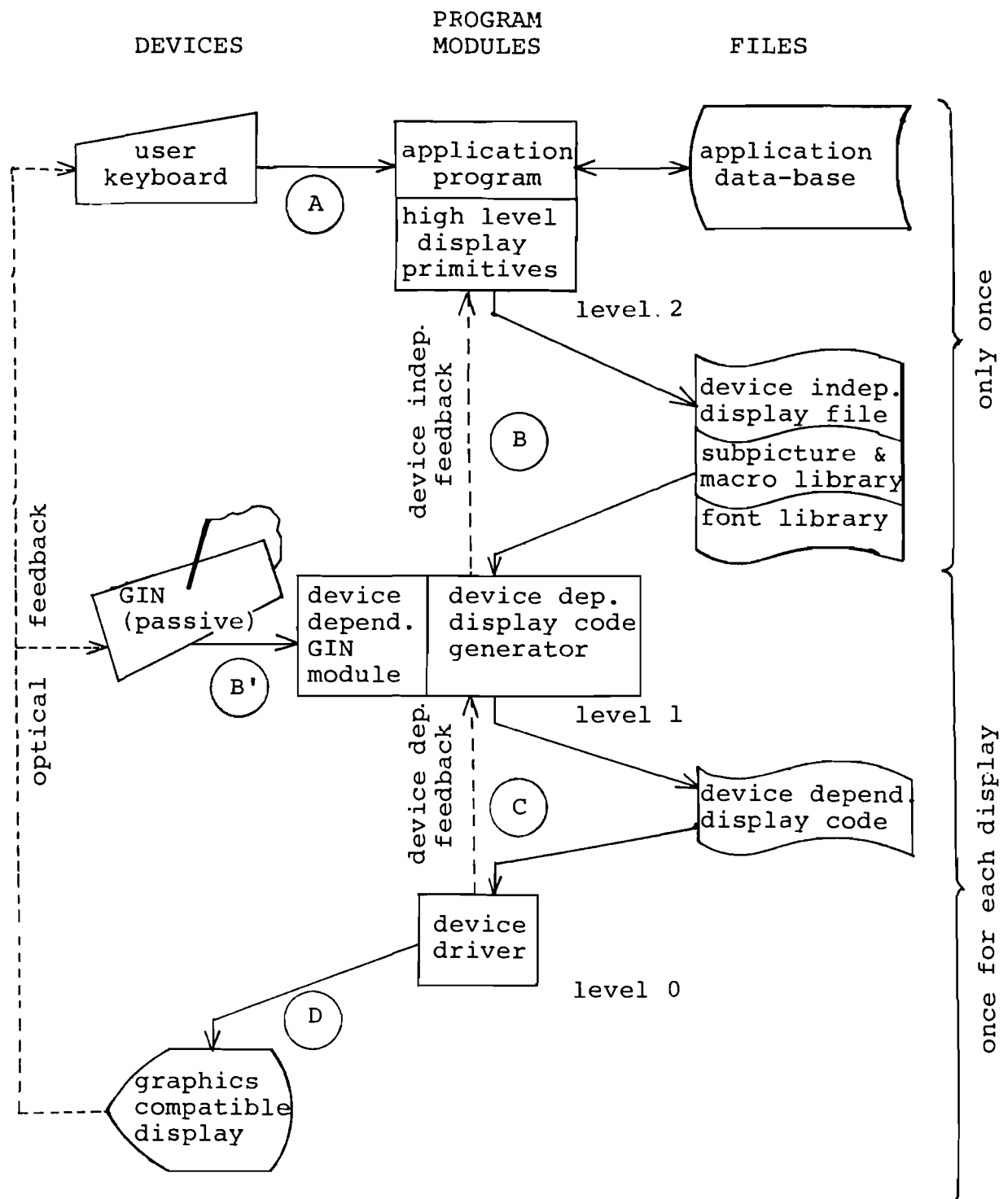
In this model of a display system, graphics input (GIN) is configured at the device dependent level. Only passive GIN is foreseen, i.e. GIN-mode is triggered by the application program; no interrupt handling or sampling, no unsolicited GIN is possible. Only the coordinates in device independent units are transmitted.

In figure (1), A, B, B', C and D are the communication links between the different program modules of the system. The required bandwidth (volume of data transferred per logical unit of information) is increasing from A to D; the content in structural information is decreasing (with the exception of B' being a special case). The device independent part of the system is configured only once, the device dependent part is replicated at least once for every device type. Information flow of B and C in the opposite (upward) direction is difficult to implement. It would nevertheless be a handsome facility to be able to sense certain device dependent conditions and do more sophisticated processing with it.

This model does offer limited interactivity since the modules can be executed in a synchronous manner, thereby offering "optical feedback"; changes to the picture are done at the application program level (possibly with the assistance of a GIN device, which can forward the writing device to the target location). A keyboard can replace real GIN to a certain extent.

The reader will realise the fact that in existing graphics installations not all parts of this system are broken apart, but rather that one module takes care of more than one logical function. Mainly the distinction between level 0 and level 1, the device dependent levels, is often very unclear.

To be able to share labor between a local and a remote computer, or with more or less intelligent terminals, it is mandatory to break the described tasks apart; to achieve device independent graphics, it is essential; for networking applications it is desirable.

MODEL OF A DISPLAY SYSTEM (fig. 1)

This model offers a clear logical data flow, which also facilitates later changes and upgrades of this system. To post-process a device dependent display code file for another device (or even worth, to cascade this process!) is a bad design. Later changes to such a design are nearly impossible to implement.

Achieving device independence also implies that the picture definition contains comprehensive enough (and even redundant) information to drive all device dependent display code generators. Information not relevant to a specific device is simply ignored.

4. GENERAL CONCEPTS AND CONSTRAINTS FOR A DISPLAY SYSTEM AT IIASA

This paper does not try to reinvent the wheel. A lot of literature exists about design and actual implementation of graphics systems (see bibliography). The proposal contained in this paper tries to consolidate ideas here and there and to merge them with our own experience with in-house requirements.

It is crucial that various kinds of display hardware (pen-plotters, storage tube displays, raster scan devices, graphic-compatible printers and hardcopy terminals, phototypesetters, vector storage devices, Computer-Output-Microfilm (COM) devices) are considered; later addition of new display hardware should be easily feasible.

Graphical output can be considered as a special case of "displayable" output. In view of IIASA's network ambitions it should be possible to send to a remote user normal text possibly intermixed with graphical data (if the destination is aware of the display protocol). Anything not being interpreted as display commands in interpreted as normal text.

This very specific requirement could be generalised in the following way: it seems reasonable to define a display protocol, able to handle graphical and non-graphical data as well, based on character transmission (8-bit-bytes) for which cheap communications hardware (TTY-interfaces, modems...) exists and is universally available. Special characters could indicate the begin of "graphics mode", and the subsequent text until an end-of-graphics-mode-character would not be interpreted as normal ASCII characters, but as picture definition codes. Even if such a protocol contains enough redundant information not to leave a device indefinitely confused in case of transmission errors, it should allow information to be communicated in a reasonably condensed form.

To fulfill its dual function as "graphics"- and "character"-protocol, the display protocol must have facilities to perform the typical plot operations with high resolution as well as text manipulations and various screen editing tasks.

The protocol must provide a limited interactivity, i.e. the user must be able to force buffers to be flushed, and after optical feedback to respecify the picture through his application program and possibly through GIN.

In addition the protocol should be defined in a way to be flexible and open to future enhancements (both in hardware and software).

These various constraints lead to the conclusion that the best place to implement such a protocol is the link between level 1 and level 2 (see fig. 1), i.e. on the device independent level. The protocol defines the communication link "B". Graphics data produced by an application program must conform to the protocol.

It is obvious that such a protocol is network-compatible, even if networking is not its first aim. It is also open to distributed processing; display

code generators and/or device driver can operate locally as well as remotely. This paper will not talk about network communication protocols. The only requirement for the communication protocol is that it is transparent to the display protocol (8 bit transparent transmission).

Provision should be made for several facilities, even if the features are not implementable on all devices, or not implemented in a first stage

- a. Multiple cursor control: each window (context or environment) will "remember" the position of the cursor when the environment is left. Only the active cursor will be associated with a marker visible on the display. The protocol must allow to change the position of the active cursor without altering the remembered location of the inactive ones (useful for split-screen technique in interactive graphics). The cursors should be positionable in different units (two different addressing types): in standard device independent units or in character units (horizontal and vertical character spacing or symbol box size).
- b. The protocol must have a 2D-3D switch, which would tell all the commands expecting coordinates to be prepared to three instead of two operands.
- c. Every graph or subgraph should be able to be headed by a command specifying the limits of the picture (or subpicture), i.e. xlow, xhigh, ylow, yhigh, (zlow, zhigh), in device independent units; coordinate values outside the limits will automatically be clipped. This restriction will allow synchronous execution of the different level modules. The main graph should also have a command to specify the intended real size in centimeters on the hardware device. In case size commands are missing, defaults apply.
- d. Subpictures (picture subsets or picture segments). To make efficient use of picture segments, these segments must be stored locally to the device dependent display code generator (level 1) and accessible from there. If retransmission for every execution is necessary, the concept becomes less valuable. A stack must be foreseen to handle the parameters attached to each subpicture level.
- e. Labels and loops can be very useful, but imply (i) buffering like for subpictures and (ii) some "registers" (or counters) which can be set, modified and tested for exit conditions.
- f. "Rectangle"-operations: these are operations applying to a whole area of the display whose limits are parallel to the x and y (and z) axes; different operations should be possible, like blanking (erasing), highlighting, outlining, inverting, hatching such rectangle portions of the display. Operations like rectangle blinking of a screen are hard to achieve without hardware prevision (on raster-scan devices only in XOR-mode).

Local storage and processing power in the device will be inevitable.

- g. "Windows" or "Viewports": compared to subsets or subpictures, which are

procedural concepts, a window is a static concept describing an environment in which operation takes place. Several windows can exist simultaneously, only one window will be "active" or "current" at any time.

The default window is always the whole display (or a part thereof for hardcopy devices). Like subsets, windows are considered strictly hierarchical, i.e. it is impossible for a display command to affect anything outside the current window, without leaving the window and going back to the level above. This is no restriction as to the creation of overlapping windows, created (and controlled) by the same level. Since for windows a tree structure is possible, a stack concept as for subsets is insufficient. What is needed is some kind of hierarchical data-base structure, where the different environments are stored (including a marker for the cursor position). To ensure proper communication between windows, a clear distinction between "global" and "local" variables is required. Creating a window only initialises its parameters to certain defaults and locates it within the current window. Nothing happens on the display unless it is "entered".

- h. Macros are a sequence of commands that can be executed at any time. They do not change the current context. They can be very useful to set up a context in a window.

The concepts of macros, labels, windows, rectangles and subsets which are introduced here should not be confused.

In the next two sections a device independent display protocol is proposed; the last section is devoted to some aspects of high level picture definition.

5. PROPOSED DEVICE INDEPENDENT DISPLAY PROTOCOL

It is essentially a one-directional protocol consisting of commands followed by a number of operands which is fixed for each command, and whose meaning depends on the command. Inquiry about hardware features or states of the display device ("feature handshake") is conceived as a separate graphics channel, which is "filled" by default values unless the destination device is known at execution time.

A command always consists of one character (8 bit bytes), operands can consist of one or more characters. Commands and operands are distinguished by their high-order bit: it is "1" for commands and "0" for operands (this is already a redundant information, but it guarantees that a device dependent driver will not "forget" that it is in "graphics mode"; it also ensures that transmission errors do not necessarily affect the whole picture, because synchronisation is reestablished with the next command).

ASCII characters not preceded by a protocol command are treated as normal text. Thus an ASCII text file is a valid display file.

Comments to the format of the protocol elements (cf fig. 2)

- a. Commands: with seven bits available to code a command, $2^7 = 128$ different graphics commands are possible. I can think of only roughly sixty to eighty reasonable graphics commands at present, so there is largely room for later additions. A command not "understood" by a specific device dependent display code generator is simply ignored (together with subsequent operands).
- b. Operands: one byte operand also enable $2^7 = 128$ different specifications; two byte operand offer $2^{(7+7)} = 2^{14} = 16384$ possibilities (since the high-order bit of all bytes of a multibyte operand must be "0"); three byte operands offer $2^{21} = 2097152$ possibilities; four byte operands 268435456 etc.

Most commands will not need any operands at all, and another substantial group will go along with one byte operands. It is mostly the coordinate values that require a very large number of bits to achieve a resolution that is sufficient for all device types.

If the decision is taken to use two bytes to specify absolute coordinates, the lower left corner being defined as $(0,0)$, $(0,0,0)$ for 3D, than x and y (and z) axes can run up to 16383 ($2^{14}-1$). For relative addressing one byte appears to be sufficient. The convention could be adopted to have the coordinates run from -64 to 63 (2^6-1), negative numbers represented as twos-complement. Especially for subpictures, character fonts (in vector representation, not as dot-matrix), one-byte relative coordinates seem to be sufficient. Twos-complement notation slightly facilitates arithmetic.

February 13, 1979

display protocol - 11

FORMAT OF THE ELEMENTS OF THE
DISPLAY PROTOCOL

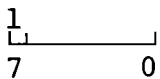
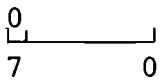
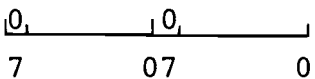
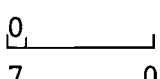
COMMANDS	1 byte	 $2^7 = 128$ commands possible range 200 to 377 (octal) high order bit 1
NORMAL OPERAND	1 byte	 $2^7 = 128$ operands possible range 0 to 177 (octal) high order bit 0
ABSOLUTE COORDINATES	2 bytes	 $2^{7+7} = 16384$ possible addresses range 0 to 16383 (decimal) high order bit 0 for both bytes
RELATIVE COORDINATES	1 byte	 $2^7 = 128$ possible addresses range -64 to 63 (decimal) negative numbers in 2's complement high order bit 0

figure 2

- c. Special ASCII codes: several ASCII control codes, some of which affect the cursor position, keep their special meaning; all of them are duplicated by display commands.

<CR>	carriage return and line feed (same as CRLF)
<LF>	line feed (same as CDN)
<BEL>	acoustic signal (same as BELL)
<BS>	backspace (same as CLF)
<HT>	horizontal tab (same as TAB)
<VT>	vertical tab (same as CUP)
<FF>	form feed (same as CLRW)

TABLE OF COMMANDS (fig. 3)

command	dec. code	short description	# operands
basic commands:			
NOP	128	no operation	-
GBEG	129	enable graphics	-
GEND	130	disable graphics	-
LIM	131	addressing space	4(6)
SIZE	132	real picture size	2(3)
TSJ%	133	real size (text units)	2
2D	134	2-D switch	-
3D	135	3-D switch	-
CLR	136	clear device	-
PLTA	137	plot absolute	2(3)
PLTR	138	plot relative	2(3)
MOVA	139	move absolute	2(3)
MOVR	140	move relative	2(3)
ERSA	141	erase absolute	2(3)
ERSR	142	erase relative	2(3)
SYMS	143	set symbol size	2(3)
SYMA	144	set symbol orientation	1
FONT	145	set character font	1
SYMB	146	vector font-element follows	1 (+n)
SYME	147	vector font-element end	-
RSTH	148	set raster font height	1
BASL	149	set font base line	1
RSSP	150	set inter-symbol space	1
RSTF	151	raster font-element follows	4+m
FLSH	152	flush buffer	-
SLEP	153	sleep	1
COLF	154	set foreground colour	1
COLB	155	set background colour	1
CTRT	156	set contrast	1
INTN	157	set intensity	1
INV	158	invert window	-
DATA	159	pass without processing	1 (+n)
ERR	160	error handling	1
MSG	161	error message	n
RECT	162	rectangle processing	5(7)
TREC	163	rect. proc. (text units)	5
BELL	164	acoustic signal	-

command to manipulate text:

WRAP	165	wrap mode	-
NWRP	166	nowrap mode	-
SCRU	167	scroll up mode	1
SCRD	168	scroll down mode	1
PAGE	169	page mode	-
TSCU	170	top scroll up	-
TSCD	171	top scroll down	-
BSCU	172	bottom scroll up	-
BSCD	173	bottom scroll down	-
LSHL	174	left shift left	-
LSHR	175	left shift right	-
RSHL	176	right shift left	-
RSHR	177	right shift right	-
HOME	178	cursor home	-
CUP	179	cursor up	-
CDN	180	cursor down	-
CLF	181	cursor left	-
CRI	182	cursor right	-
CRLF	183	carriage return/line feed	-
SETT	184	set tabulator stop	-
CLTB	185	clear tabulator stop	-
TAB	186	move to next tab stop	-
CLRW	187	clear to end of window	-
CLEL	188	delete to end of line	-
CLBL	189	delete to begin of line	-
ERSC	190	erase character	-
RMCH	191	delete character	-
INSC	192	insert character	-
TXTC	193	set cursor (text units)	2

advanced commands:

MULT	194	multiplier	1
GIN	195	graphics input	-
BLNK	196	set blink (highlight) on	-
NOBL	197	set blink (highlight) off	-
COPY	198	make hard copy	1
CHAN	199	set channel	1
XOR	200	set "exclusive or" mode	-
OR	201	set "or" mode (normal)	-
WIDT	202	line width	1
ENQ	203	enquire features	1

commands that require local programmability:

OPMA	204	open macro	1
CLMA	205	close macro	-
EXMA	206	execute macro	1
WIND	207	window creation	5(7)
TWND	208	window creation (text units)	5
RMWI	209	remove a window	1
ENTR	210	enter window	1
RETN	211	return from window	-
LABL	212	label	1
JUMP	213	jump	1
SET	214	set register	2
INCR	215	increment register	2
TRFR	216	transfer register	2
TEST	217	test register	3
OPSS	218	open subset	1+4(6)
CLSS	219	close subset	-
CAT	220	catalog subset or macro	1
EXEC	221	execute subset	1+5(7)

222 - 255 unused and free for future use

6. DISCUSSION OF SPECIFIC COMMANDS (see overview fig. 3)

There is of course freedom to select the codes for a command as long as they are unique; I would suggest to assign ascending values because it is easier to keep track of available codes for later upgrades.

We select a "left-handed" coordinate system: the x axis runs from left to right, the y axis from bottom to top, the z axis from the display away from the viewer. By convention the viewer is at -f (the "focal length") on the z-axis. Absolute (0,0,0) is defined as the lower left corner on the display plane.

The following order is always followed in the description of the commands: mnemonic name, decimal code, name, number of operands and description.

a. Basic commands (certainly to be implemented in a first stage)

NOP 128 no operation -

In graphics mode this is a null operation; as heading of a file, a series of a NOP and a GBEG ("magic number") will tell the device dependent modules that display commands can be expected. NOP can also serve for "bracketting" (terminate an argument list for certain commands). Special bracketting commands will nevertheless be foreseen, to provide correct resynchronisation in case of transmission errors while executing nested commands that require bracketting of parameters.

GBEG 129 enable graphics -

Enable graphics mode; this possibly redundant code allows a device dependent initialisation as well as an "early warning" that display commands will follow. Recommended at the begin of a display file (see NOP).

GEND 130 disable graphics -

Disable graphics mode; all buffers are flushed and released.

LIM 131 addressing space 4(6)

this command sets up a virtual addressing space in which the user will specify his display coordinates in the current window or subset. The operands specify these limits in device independent absolute units ($0 \leq x, y, z \leq 16383$). Every window as well as every picture subset can have its own virtual space defined. The four (six) operands are xlow, xhigh, ylow, yhigh, (zlow, zhigh). Any value outside this range is automatically clipped; LIM expects 2-byte operands. Should this command be missing, the assumption is that the coordinates will cover the full 0 to 16k range.

- SIZE 132 real picture size 2(3)
given in centimeters. This command specifies into which real size the top level window (default 16k x 16k, if not specified otherwise) is to be mapped. The command is only allowed in the top level window, ignored elsewhere. The picture is centered on the hardware device; defaults to full screen on CRT or to a certain maximum for hard copy devices. A faulty operand is replaced by default values.
- TSIZ 133 real size (text units) 2
similar to SIZE. Specifies how many columns and rows have to be allocated to map the virtual space (either the default 16k x 16k space or the space defined with LIM) of the highest level window. The first 2 byte operand specifies how many columns the picture should be wide, the second how many lines it should have. The picture will be centered on the display.
- 2D 134 2-D switch -
default. Turns two-dimensional mode on. Only x and y operands are expected by addressing commands.
- 3D 135 3-D switch -
Turns three-dimensional mode on. Commands expecting coordinates expect x, y and z operands.
- CLR 136 clear device -
Clear device completely; reinitialise it. For hardcopy devices move to next frame. If encountered in picture segments or windows, clear only the segment or window area.
- PLTA 137 plot absolute 2(3)
the writing device is moved from its current location to the position specified in absolute coordinates (2-byte operands). A line is drawn.
- PLTR 138 plot relative 2(3)
the writing device is moved relatively to its current position as specified in the relative coordinates (1-byte operands). A line is drawn.
- MOVA 139 move absolute 2(3)
moves the writing device, but will not produce a line (2-byte operands)
- MOVR 140 move relative 2(3)
the writing device is moved relatively to its current position as specified in the relative coordinates (1-byte operands). No line is drawn.

- ERSA 141 erase absolute 2(3)
same as PLTA, only erases instead of drawing.
- ERSR 142 erase relative 2(3)
same as PLTR, only erases instead of drawing.
- SYMS 143 set symbol size 2(3)
defines the symbol size in x and y (and z) directions: the range is 0 to 16383 (2-byte operands). If a 0 size is specified, a default, transformation independent size is taken.
- SYMA 144 set symbol orientation 1
angle with the x axis in degrees (number is taken mod 360) counted counterclockwise (2-byte operand)
- FONT 145 set character font 1
The one byte operand indicates which font is to be taken for symbol generation. The fonts will be taken from the font library if the code designates a software font. One or several codes are foreseen for hardware fonts. If this command is omitted, hardware font (code 0) is default.
Vector-fonts are described in relative coordinates ranging from -64 to 63. Font description may only consist of commands PLTR, MOVR, SYMB and SYME. Start point for font processing is always assumed to be (-64, -64, -64), the default base line is assumed to be y=0.
Raster fonts are only meaningful on raster-scan-devices and matrix printers or other display devices that permit point addressing (COM, Diablo, Phototypesetter).
- SYMB 146 vector font-element follows 1 (+n)
tells the system to begin vector font processing. The first bytes until the first command represent the table code for the font. One byte will be the rule, several bytes only for ligatures. Optional subsequent text has the same format as symbol fonts in the font library (range -64 to 63 for the 1-byte relative coordinates, default base line is y=0). This enables sending of fonts in standard units for devices which have not enough memory to store fonts, but enough to process them.
- SYME 147 vector font-element end -
is mandatory to terminate font processing. Is required even if no font description text follows SYMB, i.e. the minimum sequence for the description of one symbol is three bytes long (SYMB code SYME).
- RSTH 148 set raster font height 1

Number of rows the raster matrix of the font-elements is made of. The command is needed for sending of raster font elements.

BASL 149 set font base line 1

For raster fonts the parameter designates the row number of the raster matrix which is to be considered as base-line for the positioning of the symbol on the text line. For vector fonts the parameter designates the y coordinate which is to be considered as base line. This command is needed for sending of font elements.

RSSP 150 set inter-symbol space 1

Space to be left free between two consecutive symbols on a line. For raster fonts the parameter corresponds to the number of column, for vector fonts to the number of standard units on the x axis, to be left free. This distance will be modified by the "left kern" of the individual characters. Needed for sending of font elements. Optional with library fonts and then overwrites the font specification.

RSTF 151 raster font-element follows 4+m

Raster fonts are stored (and sent) columnwise. The first bytes until NOP represent the table code of the font. One byte will be the rule, several bytes for ligatures. The second parameter (1 byte) specifies the significant width (i.e. the number of significant columns) of the raster symbol. The third 1 byte operand specifies the "left kern", and the fourth the "right kern", which are the increments to the horizontal inter-symbol-gap (RSSP). A kern can be negative or positive. The following m bytes contain the raster, columnwise; the number of bytes is given by the formula $(RSTH * \text{significant width} / 7)$ rounded to the next integer. The reason for the existence of this command is the same as for SYMB.

FLSH 152 flush buffer -

process all unprocessed text in the buffers; implies no change in the display context.

SLEP 153 sleep 1

No action is taken for the number of seconds specified in the 1-byte operand ($0 \leq n \leq 127$); enables manipulation on the device like changing paper... It will generally be a good practice to precede this command with a FLSH command. SLEP 127 will pause until a signal or interrupt is sent to the executing process; only then execution resumes.

COLF 154 set foreground colour 1

select the colour for the writing device

- COLB 155 set background colour 1
select the background colour for the current window
- CTRT 156 set contrast 1
select contrast grade (saturation) for the current window
- INTN 157 set intensity 1
select intensity for the writing device. Intensity "0" means erase, intensity "1" is default.
- INV 158 invert window -
Foreground and background reversed for the whole current window
- DATA 159 pass without processing 1 (+n)
The 1-byte operand is a transparent count specifying that the next n bytes (8 bits) are to be passed to the device without processing (transparent mode).
- ERR 160 error handling 1
the 1 byte operand specifies different error handling options. The value of the operand ranges from -64 to 63. Negative values indicate the number of protocol errors allowed before aborting. -64 means ignore errors. 0 means execute a GIN command on error. 1 enables error messages to be sent by the MSG command, 2 disables these messages. Other codes are free for future additions. An example could be to send a "bell" tone in case of illegal addressing.
- MSG 161 error message n
The following text is to be displayed in some device dependent way if error messages are enabled. No terminator is required since the next protocol command (or a NOP, if no subsequent commands are foreseen) terminates the message text.
- RECT 162 rectangle processing 5(7)
The second to fifth (or seventh) operands of this command define a rectangle (2D) or cube (3D). Since a rectangle can be considered as a temporary window, the same conventions for the specification of its limits (operands 2-5 or 2-7) apply as for the WIND (see there) command. If no arguments follow the first operand, the whole window is taken to be the current rectangle, and the operation executed for the window. The first one byte operand specifies a limited set of operations which are defined for the whole area: INV (invert), CLR (clear), COLF (foreground colour), COLB (background colour), BLNK (blink), NOBL (noblink), BOX (frame around the rectangle), different hatching pattern: vertical, horizontal,

diagonal left, diagonal right, crossed horizontal, crossed diagonal, dotted, dashed etc.

Specific patterns can be obtained by issuing consecutive RECT commands for rectangles contained in, or overlapping other rectangles.

TREC 163 rect. proc. (text units) 5

Same as for RECT, except that the rectangle is defined in terms of character units and only in 2D. Since the rectangle can be considered as a sort of temporary window, the same conventions as for TWND (see there) apply to the parameters. If no arguments follow the first operand, the whole window is the default rectangle.

BELL 164 acoustic signal -

issues an acoustic signal at the display device. Could be configured as an optical signal on some devices.

b. commands for text oriented manipulation of the cursor

In this set for most of the commands positioning and addressing is done in character size (symbol-box-units, i.e. character spacing and line spacing) units, rather than in standard coordinate system units.

The protocol assumes that the displays always work in overstrike mode, i.e. text issued at the cursor location overlays existing text information without destroying it. It is recalled that ASCII-characters not preceded by commands are simply interpreted as text, so that no special text or graphics-escape commands are required.

WRAP 165 wrap mode -

Commands that would position the cursor outside the edges of the window continue their relative movements on the opposite side of the display; text exceeding the right margin continues on the next line at the left margin.

NWRP 166 nowrap mode -

Commands that would position the cursor outside the edges of the window are ignored and leave the cursor at the edges; text that exceeds the window will not be displayed; the last character will not be overwritten.

SCRU 167 scroll up mode 1

scrolling up of the top n lines; a <CR> or CRLF in the n-th line of the window will move the top n lines of the window up one line; the top line of the window will be lost; the cursor will be left in the first column of the n-th line of the window.

SCRD 168 scroll down mode 1

scrolling down of the bottom n lines; a <CR> or CRLF in the n-th line of the window counted from bottom will move the bottom n lines of the window down one line; the bottom line will be lost; the cursor will be left in the first column of the n-th line of the window counted from bottom.

PAGE 169 page mode -

no scrolling; <CR> or CRLF in last line of the window will put cursor on first line of the window; subsequent characters will overwrite existing information. Relative cursor movements from the bottom line of the window will cause wrap to top.

TSCU 170 top scroll up -

From the current cursor position to top of window scroll up one line; a blank line is inserted at the cursor; the top line is deleted; the cursor position is left unchanged.

TSCD 171 top scroll down - From the current cursor position to top of window scroll down one line; the current line is deleted; a blank line is inserted at the top; the cursor position is left unchanged.

BSCU 172 bottom scroll up -

From the bottom of window to the current line scroll up one line; the current line is deleted; a blank line is inserted at the bottom. the cursor position is left unchanged (typically "delete line").

BSCD 173 bottom scroll down -

From the current line to bottom of window scroll down one line; the bottom line is deleted; a blank line is inserted at the cursor; the cursor position is left unchanged (typically "insert line").

LSHL 174 left shift left -

From the current cursor position to the left margin shift the window left a column; the first (leftmost) column is deleted; a blank column is inserted at the cursor; the cursor remains unchanged.

LSHR 175 left shift right -

From the current cursor position to the left margin shift the window right a column; the current column is deleted; a blank column is insert-

ed at the left margin; the cursor remains unchanged.

RSHL 176 right shift left -

From the current cursor position to the right margin shift the window left a column; the current column is deleted; a blank column is inserted at the right margin; the cursor remains unchanged.

RSHR 177 right shift right -

From the current cursor position to the right margin shift the window right a column; the last (rightmost) column is deleted; a blank column is inserted at the cursor; the cursor remains unchanged.

HOME 178 cursor home -

cursor moved to upper left corner of the current active window.

CUP 179 cursor up -

cursor up one line; in wrap mode a cursor move across a window boundary is continued on the opposite side; in nowrap mode, the cursor remains at the boundary.

CDN 180 cursor down -

cursor down one line; in wrap mode a cursor move across a window boundary is continued on the opposite side; in nowrap mode, the cursor remains at the boundary.

CLF 181 cursor left -

cursor left one column; in wrap mode a cursor move across a window boundary is continued on the opposite side; in nowrap mode, the cursor remains at the boundary.

CRI 182 cursor right -

cursor right one column; in wrap mode a cursor move across a window boundary is continued on the opposite side; in nowrap mode, the cursor remains at the boundary.

CRLF 183 carriage return/line feed - cursor moved to first column of next line; in page mode, a CRLF in the last line of the window will put the cursor to window home; in scroll mode, a CRLF on the input line will cause a one line scroll up or down of the top or bottom n lines (see SCRU and SCRD).

SETT 184 set tabulator stop - a tabulator stop is set at the current position of the cursor; the tab is valid for the whole column of the current window, not just for the current line. Default tab settings are

8, 16, 24, 32 etc.

- CLTB 185 clear tabulator stop - the tab stop at the current position of the cursor is cleared for the whole column of the window. No action is taken if no tab was set at the cursor position.
- TAB 186 move to next tab stop - the cursor is right moved to the next tab stop. If no tabs are set, puts cursor at the rightmost column of the current window.
- CLRW 187 clear to end of window -
clear to end of window
- CLEL 188 delete to end of line -
delete to end of line. The cursor remains next to the last symbol of the line.
- CLBL 189 delete to begin of line -
delete to begin of line; the text remaining on the line is moved to the leftmost column. The cursor is positioned at the beginning of the line.
- ERSC 190 erase character -
blank current character
- RMCH 191 delete character - removes the current character; the remainder of the line is left shifted one column.
- INSC 192 insert character -
insert a space at the current position of the cursor and move leave the cursor unchanged. The remainder of the line is right shifted one column.
- TXTC 193 set cursor (text units) 2
This command enables setting the cursor to some location inside the current active window, relative to the lower left corner of the window. The 2-byte operands are the coordinates in column and line units.
Example: TXTC 50 12
will position the cursor on the 50th character position of the 12th line of the current window. Exceeding values will leave the cursor at the edges of the window.

c. advanced commands (second stage of implementation)

- MULT 194 multiplier 1
This is a count to be applied to the next command. The following command is to be executed n times (if possible). For commands where a multiple execution does not change anything, the count is ignored.
- GIN 195 graphics input -
flush buffers; enter graphics input mode; move the writing device to the target (position specified by the coordinates returned by the GIN-device). It is considered as an error to place the target outside the current window.
- BLNK 196 set blink (highlight) on -
subsequent display elements will be highlighted (e.g. blink)
- NOBL 197 set blink (highlight) off -
return to normal mode
- COPY 198 make hard copy 1
make hardcopy on device n
- CHAN 199 set channel 1
divert display output to channel (subdevice) n. The subdevice is assumed to have exactly the same specifications than the master device.
- XOR 200 set "exclusive or" mode -
This mode of operation is particularly useful for raster scan devices. If the same picture element (pixel) is redrawn with the same intensity, it is erased. Repeating the same drawing several times produces a "software-blinking", provided the CPU is fast enough. Redrawing with other intensity enables different blinking contrasts
- OR 201 set "or" mode (normal) -
normal operation (overwrite mode)
- WIDT 202 line width 1
Thickness of lines drawn in standard coordinates (for raster scan devices the operand specifies the average number of pixels defining a point)
- ENQ 203 enquire features 1
provision for the possibility to return certain display device parameters or characteristics to the higher level application program on request. The one byte operand specifies which information is requested; the communication is done through a graphics input channel. Example of

parameters: hardware character size, selective blinking, monitor resolution, current cursor positions, current target position, real display size.

d. commands that require local programmability

- OPMA 204 open macro 1
opens a macro whose name is specified on the 2 byte operand. The subsequent commands until CLMA are diverted (stored, but not processed). No syntax checking is performed at creation time. All commands can be put into a macro, including other macros. A good application will be the initialisation of windows.
- CLMA 205 close macro -
closes (terminates) a macro.
- EXMA 206 execute macro 1
execute the macro named on the 2 byte operand. If an EXMA command is found, the system looks if the macro was defined previously in the protocol, if not, it looks in the subpicture/macro library. If a macro of this name is not found, the command is ignored. If it is found, the sequence of commands is executed. EXMA does by itself not change the current window, nothing is pushed to a stack.
- WIND 207 window creation 5(7)
This command creates a new window; the first 2-byte operand specifies its name which is local to the current window. The next four (or six) 2-byte operands define the position of this window in the current window. The parameters are in the same order as for the command LIM. The default window is the whole display. Windows created on the same level can overlap. No action except initialisation of the window to "outer modes" is taken. Nothing happens on the display unless the window is actually entered. A window can have its own local limits (LIM), parameters, modes, a cursor, local and global registers.
- TWIND 208 window creation (text units) 5
This command performs the same task than the command WIND, but the window is defined (i) only 2D and (ii) the limits are characterized in terms of the current symbol-box width and height (lines and columns). The first parameter is again the window name. The next four 2 byte parameters specify the limits in the following order: left margin

(starting column of the window on the left side), the right margin (end column on the right side), the bottom margin (starting line from the bottom) and the top margin (ending line at the top). No difference exists between windows created with WIND and those created with TWND.

RMWI 209 remove a window 1

The window created with WIND or TWND is removed (disabled). After this command, the window cannot be entered any more. A window can only be removed from a active window of higher level. No action is performed on the display which remains unaffected by this command.

ENTR 210 enter window 1

the current context (including the current cursor position) is saved. The window named on the 2 byte parameter is entered and becomes thereby the "active" window. The context (environment) associated with the window entered overwrites the old context. The cursor is moved to the location which it last had in this window (only thing that happens on the display). If the window is entered for the first time, the cursor is put into home position of the window.

RETN 211 return from window -

the current context (including the current cursor position) is saved. The context from the parent window is restored; the only direct effect on the display is the change of the cursor position to its last position in the previous (parent) environment. If a window is left and entered again without changing global parameters, the switching of environment will be totally transparent.

LABL 212 label 1

is a no-operation followed by a 1-byte label to which control can be passed with JUMP (128 labels possible)

JUMP 213 jump 1

transfer control to the location in the text defined by the appropriate LABL number. If this label was not yet defined, the device dependent code generator looks in forward direction (terminated by end of subpicture or GEND)

SET 214 set register 2

The first 1-byte operand, the register, is set to the value specified in the second (2-byte) operand. Only registers 13 to 127 can be set with SET.

INCR 215 increment register 2

The first 1-byte operand, the register, is incremented with the value of the second 1-byte operand ranging from -64 to 63.

- TRFR 216 transfer register 2
copy content of register designated by operand one (1-byte) to register designated by operand two (1-byte)
- TEST 217 test register 3
the register content designated with the first 1-byte operand is tested against the content of the third operand (1-byte: register number; 2-byte: actual content); the test relation is defined by the second 1-byte operand (0 EQ; 1 NE; 2 GT; 3 GE; 4 LT; 5 LE). If < op1 rel op2 > is true, the next command is skipped.
- OPSS 218 open subset 1+4(6)
opens a picture subset (picture segment) whose name is specified on the first 2-byte operand. Subsequent display commands until CLSS are diverted (stored, but not processed).
Operands two to five (seven) are as for LIM.
- CLSS 219 close subset -
closes the subset
- CAT 220 catalog subset or macro 1
catalog and store the subset or macro whose name is specified on the 2-byte operand in the subpicture library under its name.
- EXEC 221 execute subset 1+5(7)
executes a picture segment. The parameters have the following meaning:

2D	op 1	name	2 bytes
	op 2-5	limits as for LIM	4x2 bytes
	op 6	angle as for SYMA	2 bytes

3D	op 1	name	2 bytes
	op 2-7	limits as for LIM	6x2 bytes
	op 8	angle with x-axis	2 bytes
	op 9	angle with y-axis	2 bytes
	op 10	angle with z-axis	2 bytes

A 2D EXEC-command is followed by a total of 12 bytes, a 3D call by 20 bytes. 2D segments can be executed in 3D mode and vice versa. Execution depends on the EXEC sequence. If an EXEC command is found, the system looks if the segment was defined previously in the protocol; if

not it looks in the subpicture library. If a segment of this name is not found, the command is ignored.

If it is found, the current context is saved to a stack, the register 0 (the level register) is incremented; if a command in the subset does not explicitly change some parameter (e.g. background colour), the parameter of the higher level is default. EXECs can be nested to any depth (stack size limitation). The limits specified in the EXEC command serve to recalculate all coordinates of the subset. Example: given the sequence

OPSS name 0 100 0 100 and

EXEC name 20 25 800 850 45,

the system will position the lower left corner of the picture segment name at location (20,800); the unrotated upper right corner would be at (25,850), but the segment is rotated 45 degrees around the point (20,800).

Changes of parameters in segments of lower level (higher level number) never affect the higher level. At the end of the segment processing, the original parameter set is restored from the stack and the level register decremented.

In 3D, rotation in all directions is possible ("roll", "pitch" and "yaw").

222 - 255 unused and free for future use

Proposal for the register allocation:

All 128 registers are 2 byte words.

registers 0 to 63 are global,

64 to 127 are local to current window or subset,

0 to 31, 64 to 95 are for system use,

32 to 63, 96 to 127 are user-programmable.

Register 0 contains the stack-level; register 1 to 3 contain the current x,y and z absolute coordinates of the writing device relative to the highest level window; registers 4 to 6 contain the x,y and z coordinates of the lower left corner, registers 7 to 9 the x,y and z coordinates of the upper right corner of the highest level window; Register 10 contains the current window name, register 11 the current subset name. The remaining global registers from 12 to 31 (system use) and 32 to 63 (user) can be allocated for specific tasks.

The allocation of the 12 first local registers reflects logically the global ones, but are local to a window or subset. 64 is the stack pointer to the previous level, 65 to 67 the cursor position, 68 to 70 lower left, 71 to 73 upper right, 74 the window name, 75 the subset name.

The remaining local registers from 76 to 95 (system use) and 96 to 127 (user) can be allocated for specific tasks.

7. SOME ASPECTS OF HIGH LEVEL PICTURE DEFINITION

When a graphics system is restructured and rebuilt from scratch, and if additional hardware is implemented, it would be a bad idea to use any of the graphics software on the "primitives" level currently available at IIASA. These programs will have to be completely rewritten. It is clear that a new subroutine package of graphics primitives must provide at least the same facilities than the present one. In addition a set of subprograms to define, update, handle and execute picture segments will be required.

It might be a good idea to implement these user level functions as a graphics language, hosted by FORTRAN, but run through a preprocessor before compilation. This facility will improve debugging of the graphics program, since part of the error checking will already be done in the preprocessor. The user would not have to worry about the correct number of arguments or the proper dimensioning of arrays.

The produced source code should be absolutely basic "PORT"able FORTRAN.

In a first stage of implementation, the existing graphic primitives should be rewritten (simulated), which will make existing higher level graphical software operational for a certain transition period. Their names should nevertheless be different from the old ones, even if the parameter list remains the same. Program "hits" like "PLOT" will run with minor enhancements in the first stage, a later "reincarnation" will nevertheless be unavoidable if the new available hardware is to be used accurately.

8. BIBLIOGRAPHY

- [1] Walker B. S., Gurd J. R., Drawneek E. A.: Interactive computer graphics, Arnold, London, 1975
- [2] Newman W. M., Sproull R. F.: Principles of interactive computer graphics, McGraw-Hill, 1973
- [3] Stallman R.: The SUPDUP graphics extension, NWG/RFC#746, RMS 17-Mar-1978
- [4] Crispin M.: SUPDUP Protocol, NWG/RFC#734, RMS 7-Apr-1978
- [5] Sproull R. F., Thomas E. L.: A network graphics protocol (ARPA), Computer Graphics, SIGGRAPH Quarterly, Aug. 1974
- [6] Schwenk R. M. et al.: INTERPOL - an interactive plotting package for off-line CALCOMP systems, US Army Ballistic Research Lab., Aberdeen, Mar-1978
- [7] McCann C.: graphic display interaction - part 1 - literature review, springfield, ntis, 1978
- [8] McCann C., Innes L.: graphic display interaction - part 2 - information structure and basic functions, springfield, ntis, 1978
- [9] Levine S. R.: Seminar notes "computer graphics state of the art techniques and applications", Lawrence Livermore Laboratories, Rev. 5, 1978
- [10] Integrated Computer Systems, Inc.: Manufacturers' Literature - Computer graphic components and systems, 1978
- [11] Cohen D., Taft E.: An interactive network graphics system, Comp & Graphics, Vol 1, Pergamon, 1975
- [12] Smith L. B.: An example of a pragmatic approach to portable interactive graphics, Comp & Graphics, Vol 1, Pergamon, 1975
- [13] Marovac N., Elliot W. S.: An network-oriented language - a new approach to network design, using interactive graphics, Comp & Graphics, Vol 2, Pergamon, 1977
- [14] Cotton I. W.: Standards for network graphics communication, Comp & Graphics, Vol 1, 1975

- [15] Feldmann R. J., Heller S. R., Bacon C. R. T.: An interactive, versatile, three-dimensional display, manipulation and plotting system for biomedical research, J. of Chemical Doc, Vol 12, No. 4, 1972
- [16] Le Lous Y.: Les besoins actuels en logiciels de visualisation (Present requirements for graphics software), L'onde Electrique, Vol 57, No. 12, 1977
- [17] Newman W. M., Sproull R. F.: An approach to graphics system design, Proc. of the IEEE, Vol 62, No. 4, 1974
- [18] Scranton D. G., Manchester E. G.: Use of SIMPLOTTER - a high level plotting system, Ames Lab., IOWA, Mar 1978
- [19] Chernoff H.: Graphical representation as a discipline, Techn. Rep. 11, MIT, Camb. Dept of Math, Apr 1978
- [20] Teitelman W.: A display oriented programmer's assistant, XEROX PARC Report CLS 77-3, Mar 1977
- [21] Bergeron R. D.: Standards for interactive computer graphics software, Proc Workshop Pict Data Descr & Management, Chicago, Apr 1977
- [22] Wexler J.: Minimizing traffic between two processors used for interactive graphics, DECUS Europe 8th Seminar Proc, Strasbourg, Sep 1972
- [23] Michel J., Van Dam A.: Experience with distributed processing on a host/satellite graphics system, Computer Graphics, Vol 10, No 2, 1976
- [24] Mudur S. P.: Device independent graphics software, J of the Comp Soc India, Vol 6, No. 1, Dec 1975
- [25] Garrett M. T.: An interpretive/interactive subroutine system for raster graphics, ACM Sigplan Notices, Vol 11, No. 6, Jun 1976
- [26] Cohen D., Taft E.: Fast interactive computer graphics over the ARPA network, 7th Hawaii Int. Conf on Systems Sciences, Jan 1974
- [27] Schweeger B.: PLOT Users Manual, IIASA Graphics Library, 2. Ed, 1978
- [28] Ryder B. G.: The PFORT verifier, Software Pract & Experience, No. 4, 1974
- [29] Sproull R. F.: OMNIGRAPH: simple terminal-independent graphics software,

XEROX PARC Report CLS 73-4, 1973, Reprint Apr 1977

- [30] Watson C. R.: An interactive computer graphics system applied to the life sciences, diss. Oregon St. U., 1974
- [31] Giloi W. K.: Interactive computer graphics - data structures, algorithms, languages, Prentice-Hall, 1978