tanulmányok 78/1978

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

PAPERS ON COMPUTER GRAPHICS

Darvas Péter
Gallai István
Hosszu Péter
Krammer Gergely

Tanulmányok 78/1978.

A kiadásért felel:

DR VÁMOS TIBOR

ISBN 963 311 063 7

ISSN 0324-2951

789686 MTA KESZ Sokszorosító. F. v.: dr. Héczey Lászlóné

TARTALOMJEGYZÉK

Jelen dolgozat a 7. 3-G
témaszám keretében készült.

SUMMARY

The GTU subroutine package was designed to meet the requirements of different types of applications while being modest in core requirement and computing time. The paper describes the structure of GTU and explains some basic decisions in its design.

KEY WORDS

computer, graphics, graphic subroutines

FOREWORD

In this volume we publish three papers from the recent past of the Computer Graphics Department, they have been laying around in a few copies.

Since the thoughts described serve as sources for our further development we decided to publish them as a volume of "INTÉZETI TANULMÁNYOK"

Paul Verebély

THE DESIGN OF ANOTHER GRAPHIC PACKAGE

P. DARVAS, P. HOSSZU and G. KRAMMER

INTRODUCTION

A refresh type line drawing graphic system called GD'71 was designed at the Institute. The screen has 1024 x 1024 addressable rasterpoints and the picture is refreshed from the memory of a 16 bit word small computer using hardware character, line and arc generation.

After the implementation of a set of basic i/o handling routines which were used to write applications at a rather low level the design and implementation of a run time subroutine package became necessary.

The subroutine package described here aims at far less as Ivan Sutherland's SKETCHPAD /Reference 4/ did in 1963. The latter is an early summary of features required in interactive graphic programs.

Graphic software developments first tended to achieve sophisticated graphic information handling using clever data structures. These data structures are mostly oriented towards one or other applications or, alternatively they are "general purpose" enough to fill the bigest available computer of the era.

The knot of Gordius was solved again: it was cut into two fields. The graphic software per se only deals with the generation of pictures and the input information concerning these picutres, while the storage and handling of the graphic properties of the whole modell are left to the model data structure, (see *Figure 1* and Reference 1).

The main problems are different for the two fields indeed. The first deals with the general aspects of picture structuring, generation and users intervention, while the other is very much dependent on the type of application. (Certainly, both has to deal with graphic data and require a common interface.)

GPGS and GINO are fine examples of graphic subroutine packages developped according to these lines. (See References 2 and 3 respectively.)

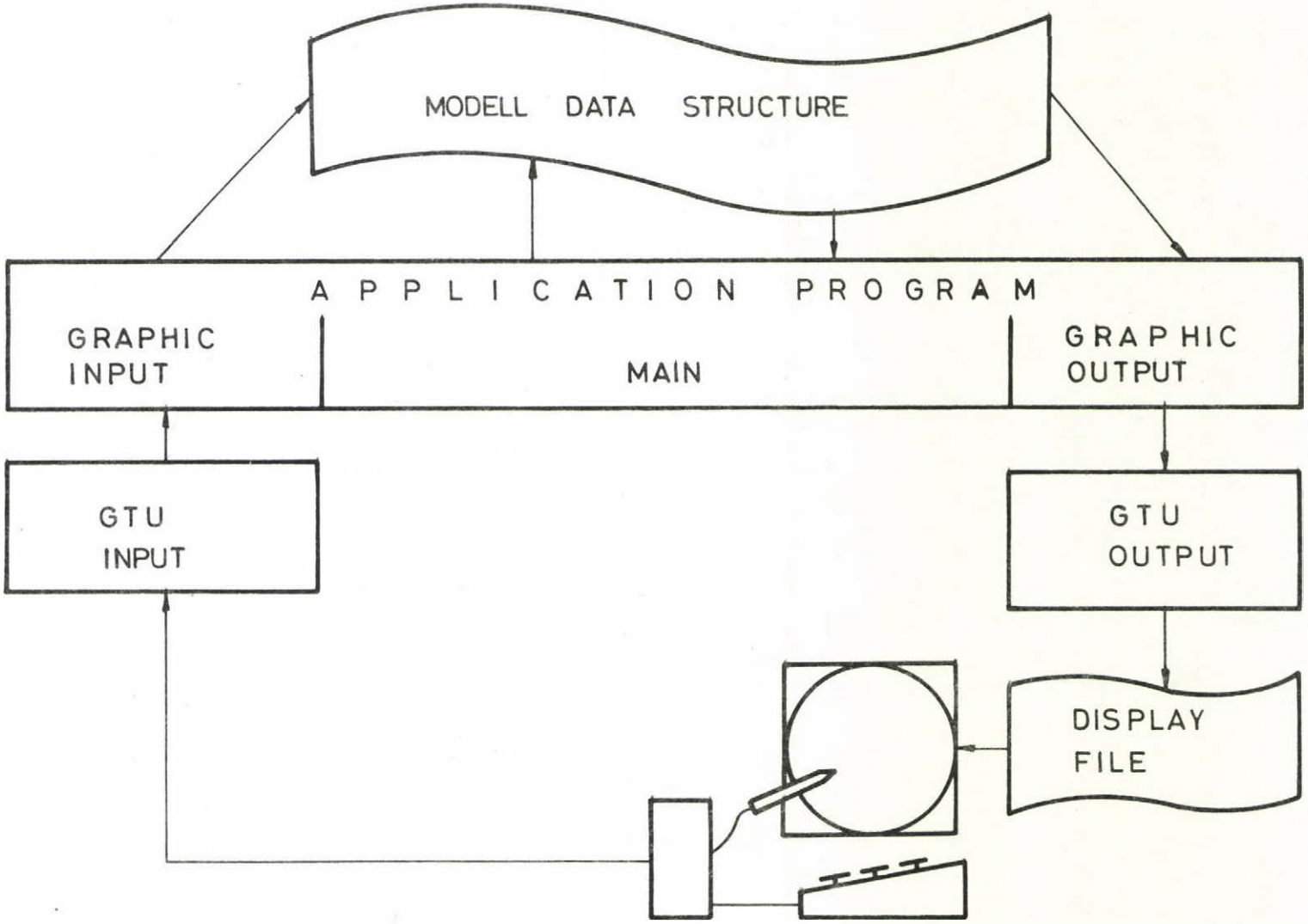


Figure 1. Application program and graphic subroutine package

Certainly the modell data structure handling is still a task of current research (see e.g. Referencee 6). If a general purpose programming language is used for graphics, it has to take care both for graphic input-output and modell data structure handling, either as auxiliary subroutine packages, or as language extensions.

While the group was aware of the major trends in present day graphic subroutine systems, a new, simple, 2D package was nevertheless designed and implemented both as a working system (with early availability) and as a practical study of problems in architecture and implementation.

In the next part we formally describe the most important subroutines. The third part describes some internal matters and takes a critical view of the design. The fourth and fifth parts describe two applications and some extensions.

THE GTU GRAPHIC SUBROUTINE PACKAGE

The GTU subroutine are described in four groups. The subroutines are callable either form FORTRAN or assembly language programs. They may be stored in the memory all at once or, alternatively, an overlay version was prepared where GTU overlays itself rather than letting the programmer bother about the overlays.

DISPLAY, BUFFER AND PICTURE SEGMENT HANDLING

GDINIT — Checks the availability of the display and sets initial values.

GDREL — Closing the system.

GBDEF — A memory field with given name (starting address) and length is defined (and reserved) as a display buffer.

GBREL — The named buffer is "forgotten" by GTU. The contents of the buffer is not changed, thus the user may save it for later use. The program can use the memory field for other purposes.

GBCONN Similar to GBDEF, but the buffer is supposed to have a correct buffer head and pictures descriptions created earlier (possibly saved on backing store and now being restored).

GBSEL Selects a previously defined buffer to store all subsequent picture segments.

GSEGM A new picture segment is defined with the given name (a 16 bit integer).

GSEXT A segment is opened for extension: to accomodate all subsequent picture elements.

GSCLOS Marks the end of a segment.

GSDISP Makes a defined segment visible, while

GSDARK makes it invisible.

GSREL The segment is destroyed, the memory occupied is freed.

GSLPEN The elements of the segment are enabled for light pen hits.

GSLPDA The elements of the segment are disabled for lightpen hits.

PICTURE DEFINITION USING USER SPACE COORDINATES

The three GX subroutines set some geometric parameters which are used by the GF subroutines to generate screen coordinates from the coordinates given by the user.

GXWNDW Specifies the corners of a window in the user space.

GVPRT Specifies the corners of a viewport on the screen.

GXTRAF Specifies a field which contains the coefficients of a linear transformation.

GXCLIP Is used to switch the clipping on/off.

The first group of subroutines accept absolute coordinates:

GFPNT
GFVECT
GFCRIC — Define a point, a vector and a circle respectively.

GFTITL — Puts a string of characters into the specified point of the user space.

GFPOS — Specifies a point in the user space (usually to introduce relative elements).

The following subroutines accept coordinates relative to the end position of the last element

GFDRAW — Define a vector and a circle.

GFARC — Respectively.

GFMOVE — Specifies a new position with its relative coordinates.

GFTEXT — Writes a text from the current position.

The subroutine GFNAME is used to assign a 16 bit integer as a name to all subsequent picture elements in this segment, until the next GFNAME call.

The GF subroutines perform both the transformation specified by the GXTRAF and by the last GXWNDW, GXVPRT pair. All elements are clipped unless clipping is swithed off.

PICTURE DEFINITION USING INTEGER SCREEN COORDINATES

The subroutines GPOINT, GPVEC, GPOS, GPDRAW, GPMOVE and GPTEXT are similar to the appropriate GF routines, however integer numbers are used as parameters. The subroutines do not change these values nor do they perform any check on their correctness; the parameters are assumed to be correct screen coordinates.

A set of subroutines are provided to the user to aid in clipping: GYPNT, GYVEC and GYTEXT are used to clip points, vectors and texts respectively. They result in a flag and the clipped element.

It is the user's responsibility to call the appropriate GY - if necessary - and GP routines one after the other.

PICTURE SUBROUTINES

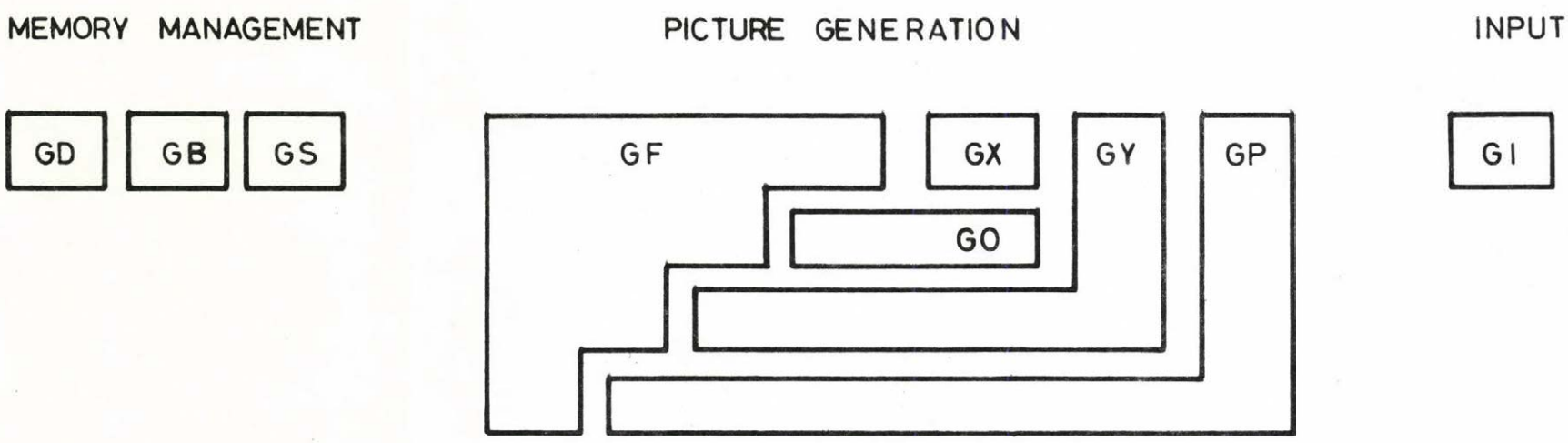
Picture subroutines may be defined similarly to picture segments. A separate subroutine buffer has to be defined, (GBSUBR), one only of these may be alive at a time. A subroutine is started with GSUBR and its picture created with GF or GP routines using relative coordinates only.

In between the elements of a picture segment a picture subroutine may be referenced via a GFREF or GPREF call.

The applications programmer is warned that the clipping process checks only the begining of a subroutine but not the subroutine body. If it comes close to the window boundary the subroutine may cross over. Sorry.

Use "small" subroutines and accept this inadequancy. (Our do not use subroutines at all.)

Picture macros would help, but his would require having macro storage and expansion incorporated into GTU. User program subroutines can produce similar results with no substantial excess effort.



GD: display initialisation and release

GB: buffer Randling

GS: seßment handling

GF: picture generation; user space coordinates

GX: parameter definition

GO: common field for GX and GF

GP: picture generation: integer screen coordinates

GY: clipping

GI: input

Figure 2: GTU subroutine groups

INPUT

The user program gets reports on input events and status reports on input processes. Reports are generated for events from devices enabled and processes started. Devices may be enabled for one event only or, alternatively until disabled by the user program.

Event reports are stored in the input queue which is interrogated by the program independently from the flow of events.

A status report on a process may be read by the user program at any time directly or, alternatively, an event may be assigned to an input process in order to store an event report and status report together in the event queue whenever this event occurs.

Typical events are: keyboard hits, clock interrupts, and light pen identifications. Three processes are implemented at present: a tracking cross following the light pen or the tracking ball, and the tracking ball on its own.

User program subroutines may be assigned to events. In this case no event report is generated into the queue, but rather the user subroutine is called whenever the event occurs. Since these user subroutines are to handle interrupt-like events they are supposed to be "short" and malfunction may kill the program. Therefore the novice user is adviced not to make use of this facility.

An alphanumeric dialogue was implemented with all special key functions using facility and driven by single keyboard events.

A CRITICAL VIEW IN DEPTH

The set of subroutines described above is quite conventional.

In this part we summarise the inside of GTU which is thought to have something new - at least it did for the designers.

Display buffers are allocated by the user. Memory management within a buffer is done automatically. Segments can be extended in any order and the space occupied by a segment is freed after deletion.

The buffer connect mechanism was implemented mainly to aid using overlays. At certain points in the dialogue some pictures may become temporarily obsolete. The buffer which contains them may be released and saved on disc. Later they can be retrieved and connected again either to the same part of the memory or to any other place.

Most programs use "constant" pictures e.g. menus, pages of information, etc. These can be prepared within a separate program in a similar way. In order to avoid the dull work of composing these simple preparatory programs, a dialogue version of GTU is provided which enables the user to type in consecutive GTU calls and the effect can be seen immediately on the screen.

Linear transformations are performed automatically by the GF subroutines. The user sets up in a table the coefficients of the transformations using the set of subroutines which set up a rotation, a translation, scaling or mirroring and any combination of these. GTU stores but one transformation at a time, the user may store, stack, etc others in his model.

Whenever the window, viewport or transformation is redefined, GTU recalculates the superposition of the window-viewport transformation and the user defined transformation.

All coordinates are transformed only once, the resutls of which are screen coordinates in floating point form. These are then truncated to integers and clipped using integer arithemitc. Integer arithmetic is used mainly because of the lack of floating point hardware.

During truncation an overflow may occur if the floating point number to be truncated is greater than 2^{15} in magnitude. Since the viewport coordinates are less than $1023 = 2^{10}$ the above condition may happen only if the distance of a point from the user window is more than 32 times the size of the window itself. The number space of the "transformation box" does not allow this to happen.

At a first glance this figure seemed frightening, however in most cases an application program deals with fine details of an object or with larger scale outlines only. The feeling was that the application model itself should be different for the two, thus the graphic package may not have to face this problem. (One may think here, that 18 bit words are better for graphics than 16 bit words as 36 bit are better for scientific calculations than 32 bits in general.)

This restricted number space relates to the GF subroutines only, the alternative set, for the GP subroutines, has no similar restrictions at all.

The two alternative sets of subroutines for picture generation were defined after many hours of discussions.
Certainly there are applications which use real numbers as coordinates and there are certain "raster-type" applications which use only integers. For the latter group of applications clipping and transformations are usually required in a less general form.

The two levels were built on top of each other: the GF routines first perform the transformation then call a GY routine for clipping and finally store viewport coordinate data using GP subroutines.

This structure makes GTU quite transparent and furthermore it is up to the user's decision if he wants to keep control over the time consuming calculations or rely on GTU. This decision depends highly on the type of application. While all graphic subroutine packages aim at a certain group of applications their efficiency varies through the whole spectrum. The whole GTU package and its subset opened at the GP level tries to be efficient in two ranges.

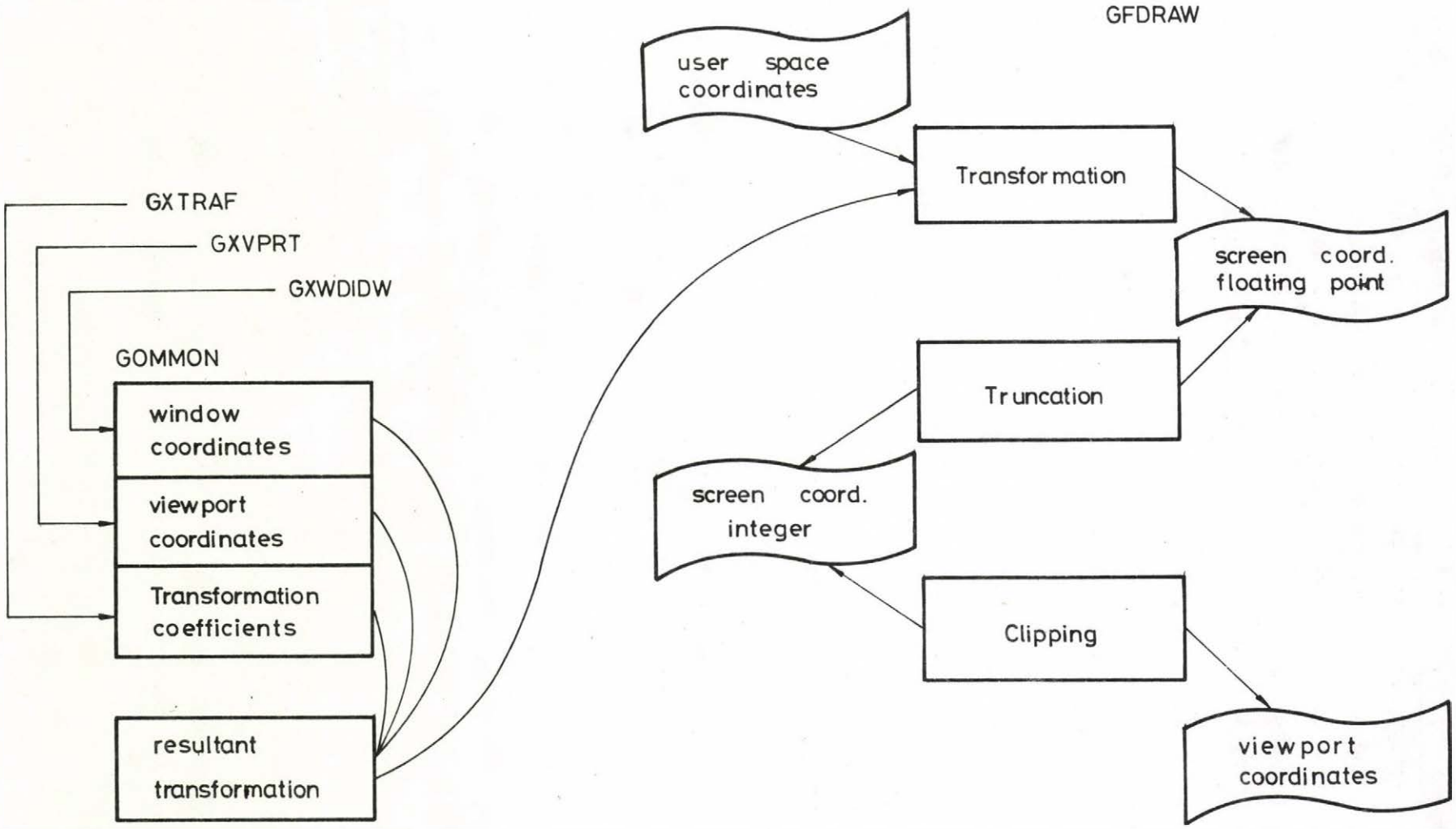


Figura 3. Definition and use of display parameters

The input branch of GTU can be divided into three parts: light pen identification, moving picture parts between discrete points and all others. Function keys and clock pulses are not real graphic devices. Their basic actions are quite simple and their more sophisticated use is easily programmable using the subroutine assignment facility. We did not deal with very special input possibilities like free-hand sketching, etc.

Moving picture parts is used mostly in "raster type" applications and GTU provides us with adequate tools to do it. (When Sketchpad pulled together two close points, this was similar to what is done in raster type problems.)

Light pen identification is possibly the tool which can provide the most information to the machine - if correctly used.

The necessary steps are: define the gorup of entities subject to identification both for the program and the operator, accept identification (more that one if necessary), refuse identifications outside the group, make sure if program and operator are thinking of the same entity and finally (or beforehand) learn action to be done on the entity.

The composition of a drawing from points, lines and circles is a simple task for schoolchildren, and for more sophisticated geometric objects mathematical algorithms are provided.
The good organization of a picture to facilitate good identification facilities is not a trivial task.

In GTU names are assigned to buffers (the buffer's starting adress, or array name), to segments and to sequences of elements.

When pointing to a line on the screen these three names are passed to the user program. Note the delicate problem all application programmers may come across: identify a point as the required element which lies on a line or where two lines intersect.

Since picture element names are implemented by the hardware "load namefield register" command, this steals buffer space and refresh time from "useful" commands.

In applications where the screen is full of elements which need separate identification, special techniques have to be used.

A graphic subroutine package is usually called device - independent if it has

(i) a central, device independent part,

(ii) device drivers for all different devices, and

(iii) data is passed between the above two in device - independent form.

GTU is not device-independent in this sense and the internal 16 bit integer representation prevents this in general.

The limited power of the small computer (up to 32 K words of core, no hardware floating point) may justify our decision to build the interactive GTU as fast possible.

An alternative set of subroutines with identical user interface is designed for use with plotters. The user interface still remains device independent if the "device select" action is performed via "library select".

The GTU package was used on a 16 K word machine, however the dialogue was held up sometimes by overlay changes. The same application runs more smoothly on 24 K words. GTU together with the basic graphic peripheral support occupies 5 K words. An overlay version with 30% less core requirement was compiled, where GTU overlays itself rather than maxing with the user program.

On output much core can be saved if no circles are used (circle clipping!).

TWO APPLICATIONS

Two pilot applications were used to test our ideas and influence the final design. Both interactive programs are parts of a larger design program suite.

The first, an interactive 2D workpiece definition and manipulation program, made extensive use of the whole GTU. The model stored canonical forms of plane geometric elements and "contours" composed of finite line segments and points. The canonical elements and all contours can be extended, displayed and extinguished separetely, contours can be moved and transformed.

The main viewport is left unchanged, thus it is only a change in the window which results in the regeneration of the whole picture. The new window is requested by the operator on rare occassions and the 2 - 3 second's delay is acceptable.

Canonic elements and contour elements can be deleted, only the canonic elements or one contour respectively have to be regenerated.

Names of picture elements used as displacements are pointers of the appropriate element in the model.

The other application is printed circuit board checking and editing. The GP subroutines were used to generate pictures.

The whole PCB can be divided into square fields (as ordinary maps are) and a window may be composed of 1, 4, 9, etc neighbouring squares. Clipping is performed in two steps: when changing a window a submodel is generated from elements which may fall into the window according to a rough criterion. While the window is not changed, only the elements of the submodel are clipped accurately. The screen is usually full of elements, and the operator may want to delete any tiny piece there!

The printed circuit board can not be moved in real time, or at least it is not clipped while moving.

These are well known restrictions in this class of applications (for this category of displays).

EXTENSIONS

GD'71 is used in three configuration types: standalone system with local disc store, simple graphic terminal and intelligent graphic terminal.

GTU was developed : for the standalone configuration.

A set of FORTRAN callable subroutines were implemented for a remote host machine interfaced with the GD'71 configuration through telephone line. The subroutines represent the GP level of GTU. Once GF is rewritten in FORTRAN it may provide a compatible host version of GTU.

The intelligent terminal software is developed to provide application programmer control over the task division between host and terminal. On the input branch, the user subroutines assigned to input events may be exploited. For the output branch the jig-saw puzzle of subroutines should be composed in different configurations for the two machines. For both branches the protocol of graphic information transmission has to be sufficiently flexible.

ACKNOWLEDGEMENTS

We are grateful for the friendly atmosphere at our department - led by Joe Hatvany - in which GTU has evolved. Special debt is due to G. Gerhardt and A. Lábadi for many fruitful discussions and for implementing, together with P. Darvas, the Basic Graphic Terminal software. G. Hermann has helped us with his fierce critisism. G. Pikler as the designer of the first application, T. Tolnai and D. Keresztély those for the other have helped by formulating the needs of their applications. Miss J. Ságí has a worked hard in the implementation.

Last but not least we cannot forget the influence of other graphic systems and their designers. We are grateful to them in particular to Andries van Dam, William M. Newmann, the Cambridge (UK) and Delft graphics groups for discussions.

The mistakes made are due to us, not to the persons mentioned above.

REFERENCES

[1] W.M., Newmann and R.F., Sproull: An approach to Graphics System Design
Proceedings of the IEE, April 1974

[2] General Purpose Graphic System Reference Manual
Technische Hogeschuool Delft,
Publication No: RC-GFX-75002-0

[3] P.A., Woodsford: The design and implementation of the GINO 3D graphics software package
Software Practice and Experience,
Vol 1. No 4. 1971) pp 335-365

[4] I., Sutherland SKETCHPAD - A Man Machine Graphical Communication System
AFIPS Conference Proceedings,
SJCC, 23, pp 329-346 (1963)

[5] R., Williams: A survey of Data Structures for Computer Graphics Systems
ACM Computing Surveys,
Vol 3. No 1, March 1971, pp 1-21

[6] R., Williams: On the Applications of Relational Data Structures in Computer Graphics
IFIP Congress 1974

A GRAPHIC PROTOCOL (FIRST DRAFT)

ISTVÁN GALLAI

1. INTRODUCTION

The main purpose of this graphic protocol is to serve as a "user level" protocol (/1/) for graphic work in terminal networks or higher level computer networks. However the format in the line drawing protocol (which is the first part of this graphic protocol) may be used to store picture information as well.

While other, "lower level" protocols,cater for the error-free transmission of information blocks (including all necessary actions, like: connect, disconnect, error-handling, routing, ect.), this graphic protocol deals with the information contents of these blocks and thus describes their interpretation by graphic terminals and/or graphic programs.

A protocol in general is a set of agreements on the format and relative timing of information transmitted over a transmission line. The first part of this protocol deals exclusively with the format of line "interaction information" thus not only with the format of blocks, but furthermore with the relationship of information sent in both directions at different times.

The main tasks of the graphic protocol is to define

- Device independent description of line drawing pictures (line drawing protocol)
- The mode and the format of different data in connection with the graphic devices (interaction protocol)

2. GENERAL DESCRIPTION

The protocol is designed to be used between two "processes" at two "sites" in a network - most commonly between a graphic program and a graphic terminal equipment.

The protocol is divided into two main parts:

- the line drawing picture protocol and
- the interaction protocol

Different process may interpret different subsets of this protocol. (Simple graphic terminals, plotters, intelligent terminals, picture file store, etc). There are no agreements described here for feature interrogation or error messages for not interpreted sentences.

The line drawing information transmitted in one direction is related to "pictures". A picture consists of "segments". Every segment has its own identifier.

A picture is built up from primitivies:
points, vectors, circles and texts

Groups of primitivies may be grouped into brackets. (blocks) within blocks new pairs of brackets may be defined. Every block level may have an identifier, and all primitivies may have their own identifier. (Different implementation may restrict the depth of bracketing. In worst case as 1 or 0.)

After the opening bracket we may give a numben This tells how many times the primitives inthe brackets have to be drawn.

The parameters of primitives are given in the application coordinate system chosen by the user. Clipping and transformation will be done accordingly the window, viewport and the transformation matrix defined.

The coordinates may be given as integer numbers or floating point numbers. The type and length of the coordinates is defined by the user.

Primitives may have 2D or 3D coordinates. These can be defined by mode modifiers.

Following a block-end the values of the modal parameters beeing valid at the block-start will be valid again.

The user can build segments in parallel.

Macros can be used in the description of pictures.

For the interaction protocol graphic devices are divided into two groups:

- Event peripherals (light pen, keyboard, ...)
- Status peripherals (track ball, ...)

Event peripherals have to be enabled before used. If an event occurs the user will get a report. The status of the status peripherals may be asked at any time or in case of a chosen event.

The protocol consists of sentences, the sentences consist of words.

The structure of the sentences:

- 1st word — op code
- 2nd word — op code extension
- 3rd and following words — parameters

Sentence types are sescribed by giving their mnemonic, parameters and the hexadecimal and decimal value of the op code.

3. LINE DRAWING PROTOCOL

3.1 Picture primitives

<MOVA>	AX,AY[,AZ] ABS, MOVE	CODE== :0 (0)
<VECA>	AX,AY[,AZ] ABS, VECTOR	CODE= :1 (1)
<CRCA>	AX,AY[,AZ],AXM,AYM[AZM],AS ABS, CIRCULAR ARC (ENDPOINT,MIDDLEPOINT,SIGN)	CODE= :2 (3)
<PNTA>	AX,AY[,AZ] ABS, POINT	CODE= :3 (3)
<MOVD>	AX,AY[AZ] REL. POS	CODE= :8 (8)
<VECD>	AX,AY[,AZ] REL, VECTOR	CODE= :9 (9)
<CRCD>	AX,AY[,AZ],AXM,AYM[,AZM],AS REL. CIRCULAR ARC	CODE= :A (10)
<PNTD>	AX,AY[,AZ] REL. POINT	CODE= :B (11)
<TXT>	I, 'TEXT' TEXT I - NUMBER OF CHARACTERS	CODE= :7 (7)
<TXTC>	I,'TEXT' TEXT CONTINUE I- NUMBER OF CHARACTERS	CODE= :F (15) CODE= :F (15)

3.2 Picture hierarchy

<SDEF> NAME CODE= :40 (64)
SEGMENT DEFINITION

<SOPEN> NAME CODE= :41 (65)
SEGMENT OPEN

<SCLOSE> NAME CODE= :42 (66)
SEGMENT CLOSE

<BEGIN> I,NAME CODE= :20 (32)
OPEN BRACKET
(I=0 --> I=1)

<END> CLOSE BRACKET CODE= :21 (33)

3.3 Modal modifiers

<INT> I CODE= :10 (16)
ACTUAL INTENSITY LEVEL
0 <= I < 16

<TYPE> I CODE= :11 (17)
ACTUAL LINE TYPE

<CHSI> I CODE= :12 (18)
ACTUAL CHARACTER SIZE

<CHOT> I CODE= :13 (19)
ACTUAL CHARACTER ORIENTATION

<DIM> I CODE= :14 (20)
ACTUAL DIMENSION
I=2,3

<DATA> I CODE= :15 (21)

ACTUAL DATATYPE END DATASIZE

I < 128

PARAMETERS WILL BE I BYTE INTEGERS

I > 128

PARAMETERS WILL BE I-128 BYTE

FLOATING POINT NUMBERS

<WNDW> AX1,AX2,AY1,AY2[,AZ1,AZ2] CODE= :22 (34)

ACTUAL WINDOW

<TRANS> I,A1,...,AN CODE= :23 (35)

ACTUAL TRANSFORMATION MATRIX

I - TRANSFORMATION CODE

A1,...,AN ELEMENTS OF THE MATRIX

<NAME> N CODE= :25 (37)

UNIQUE IDENTIFIER

IDENTIFIER OF A PRIMITIVE

<VPRT> I,AX1,AX2,AY1,AY2[,AZ1,AZ2] CODE= :26 (38)

ACTUAL VIEWPORT

PARAMETERS ARE IN

I=0 DEVICE COORDINATE SYSTEM

I=1 PER CENT OF THE MAX OF THE DEVICE COORD.

I=2 0.01 MM

I=3 1MM

3.4 Macros

<MACRO> N CODE= :30 (48)
MACRODEFINITION

<MACEND> CODE= :31 (49)
END OF MACRO

<CALL> N,NK1,B1,B2,... CODE= :32 (50)
CALL MACRO
N - NAME OF THE MACRO
NK1 - NUMBER AND TYPE OF PARAMETERS
B1 .. PARAMETERS

4. INTERACTION PROTOCOL

4.1 Asking

<EABP> N,K,S,E,I,AX1,AY1,AX2,AY2 CODE= :50 (80)

ENABLE EVENT PERIPHERAL

N - PERIPHERAL IDENTIFIER

K - MAX NUMBER OF EVENTD ENABLED
(0=INFINITE; UNTIL DISABLE)

S - STOP SIGN (=0 IF THERE IS NO STOP SIGN)

E - FLAG (0 WITHOUT ECHO, 1 WITH ECHO)

I - (IF E=1) CODE TO THE INTERPRETATION OF THE VIEWPORT COORDINATES

AX1,AY1,AX2,AY2 (IF E=1) VIEWPORT COORDINATES

THE LAST 6 PARAMETERS ARE USED ONLY FOR TEXT PERIPHERALS.

<DABP> N CODE= :51 (81)

DISABLE PERIPHERAL

N - IDENTIFIER OF THE PERIPHERAL

<DABPC> N CODE= :52 (82)

DISABLE PERIPHERAL AND CLEAR QUEUE

N - IDENTIFIER OF THE PERIPHERAL

<ASK> N CODE= :53 (83)

ASK STATUS OF STATUS PERIPHERAL

N - PERIPHERAL IDENTIFIER

<ASKE> N,NE,K,C CODE= :54 (84)

ASK STATUS IN CASE OF EVENT

N - IDENTIFIER OF THE STATUS PERIPHERAL

NE- IDENTIFIER OF THE ENEVT PERIPHERAL

K - MAX NUMBER OF EVENTS (=0 INFINITE)

C - VALID CODE COMING FROM AN EVENT PERIPHERALS (=0 ALL CODE ARE VALID)

<CONN> N,S,AX,AY CODE= :55 (85)

CONNECT A PERIPHERAL DEVICE AND A DATA OF A PRIMITIVE

N - PERIPHERAL IDENTIFIER

S - SEGMENT IDENTIFIER

AX- SCALING FACTOR TO X COORDINATE

AY- " " " Y "

<RES> CODE= :56 (85)

RESET PERIPHERALS

<RESC> CODE= :57 (87)

RESET PHERIPHERALS AND CLEAR QUEUE

<SERV> CODE= :58 (88)

ASK FOR A SERVICE REPORT

4.2 Reports

<REPT> N,IN,A1 ,... CODE= :60 (96)

REPORT FROM PERIPHERAL

N - IDENTIFIER OF THE PERIPHERAL

IN- NUMBER A DATABYTES IN THIS REPORT

A1,..... - DATA FROM THE PERIPHERAL

<REPTE> N,IN,A1,...,NE,INE,AE1,... CODE= :61 (97)

REPORT FROM A STATUS PERIPHERAL CONNECTED TO AN EVENT: TWO REPORTS IN ONE SENTENCE

N - STATUS PERIPHERAL IDENTIFIER

IN - NUMBER OF DATABYTES FROM STATUS PERIPHERAL

A1,..., - DATA FROM STATUS PERIPHERAL

NE - EVENT PERIPHERAL IDENTIFIER

INE NUMBER OF DATABYTES FROM THE EVENT PERIPHERAL

AE1,... - DATA FROM THE EVENT PERIPHERAL

<SERVB> CODE= :62 (98)
SERVICE REPORT 1

<SERVE> CODE= :63 (99)
SERVICE REPORT 2

4.3 Actions with segments

<SDISP> N CODE= :43 (67)
DISPLAY SEGMENT

<SDARK> N CODE= :44 (68)
DARK SEGMENT

<SLPEN> N CODE= :45 (69)
ENABLE SEGMENT TO LIGHT PEN

<SLPDA> N CODE= :46 (70)
DISABLE SEGMENT TO LIGHT PEN

<SPOS> N,AX,AY CODE= :47 (71)
SET THE POSITION OF THE SEGM

<SREL> N CODE= :48 (72)
RELASE SEGMENT

<DINIT> CODE= :49 (73)
INIT DEVICE

<DREL> CODE= :4A (74)
RELEASE DEVICE

<PEND> END OF PROTOCOL CODE= :4B (75)

4.4 Input device numbers for GD'71/T

Functional keyboard	1
Alphanumerical keyboard	2
Light pen	10
Tracking ball	5
Tracking cross with LP	6
Tracking cross with track ball	7

AN ANALYSIS OF GRAPHIC TERMINAL FUNCTIONS

GERGELY KRAMMER

1. INTRODUCTION

This paper is intended mainly to deal with the analysis of problems in the design, construction and operation of intelligent terminals rather than to propose a unique solution for them. Furthermore, we do not specialize on any existing mainframe or graphic display manufacturer.

The term: "intelligent graphic terminal" is used in a somewhat liberal sense. When used, a family (or a member of a family) of configurations is understood which is based on a mini (or similar) computer, with local storage and processing power, with an interactive graphics facility and terminal access to a mainframe computer or a network. These components may be augmented with documentary graphic devices, digitizers, and other, more specialized hardware and software components; all based on a uniform, modular architecture.

This type of configuration can be used as a member of a more complex system with substantial local intelligence, and it is sometimes called: the engineers' workstation.

We consider mainly line drawing graphics, however, the looser term "graphics" will be used throughout this paper.

The subject of this paper is a part of collaborative work which has been conducted in recent years on Graphics, Computer Aided Design, and Computer Networking.

2. TYPES OF TERMINAL ACCESS

The methods applied for terminal access can be classified from many different points of view.

The first question considered here will be the difference between terminals connected directly to host computers and the connection through a more general network (*Fig. 1*). The first type of connection is in many respects more simple and may be considered as more traditional.

Physically this connections involve a multiplexor type equipment only. Further levels are: the line control procedure (BSC, SDLC, UT 200, DDCMP, etc.), block transfer level macros and user program.

Since the operation of packet switching networks raised a host of new problems in comparison with terminal networks, their solution has led to a new understanding and new solutions in the simple terminal access too.

The most important parts here are the following: line control procedure, packet protokol, end-end protocol, user process protocols and "special purpose" protocols, e.g.: graphic.

The end-end protocol level provides for errorfree, sequential transmission of blocks. The packet protocol may be considered as the internal matter of this level but still, considerably differs if the terminal is connected to the network directly or through a Network Terminal Concentrator. The latter case is more similar to the traditional host-terminal connection, the concentrator provides for all the specialities of the Data Transmission Subnetwork and provides unique attention to each terminal connected to it.

The evolution of line control procedures is but a partial problem. It is worth noticing that the new protocols (HDLC, BDLC, SDLC, DDCMP, etc.) are more suitable for processor to processor sommunications than their predecessors.

The terminal's access type may be typified as: interactive network control, interactive job control, interactive program control, remote batch and file transfer. At least two of these are available for all terminals. Most terminals allow one at a time only while others may allow more in parallel.

This situation is best understood via the "process" concept of networks: after the Initial Connection procedure two processes communicate through the line: a terminal process with a host process.

A host computer should allow more unit addresses within one line address to run parallel processes at one terminal. If the terminal can run one process only some error messages and especially unexpected process terminations hang always over the terminal's head as a second process. Thus all messages have to be identified for the sending process. This problem is analysed further in a later paragraph.

If a terminal with parallel process capacibilities is connected directly to a network, it may communicate with more than one host at a time. This does not bring in new problems, the terminal deals with processes in the network, irrespectively of their geographical situation.

3. TERMINAL OPERATION MODES

Prophets in academic circles often (and justly) say, that the only way of using a computer worthy of mankind is fast interaction. Also it is often (and justly) emphasized, that designers can not bear and would not use other ways. Further, similar requirements call for man oriented, application oriented, situation oriented and individual oriented languages and other interfaces.

The author does not belong to any of the above two respectable classes and will venture to discuss a few methods somewhere between manual methods with two weeks turn around time and the only worthy solution. These methods may be justified in different fields of applications by the limited memory and limited processing power available, by our poor understanding of the design process itself, by our poor methods, languages, etc., derived from this understanding and, finally by the strong demand to use existing techniques now, even though our present knowledge might condemn them.

Most of the design process deals with geometry, positioning and other problems which can be visualized graphically. Thus much of the designer's communication can be performed through graphic dialogue. Still the alpha--numeric dialogue is a substantial part of his work and I expect it to remain so.

The terminal may perform as a *simple graphic terminal* with a local buffer and limited intervention handling only (IBM 2250, and others) or an *intelligent graphic terminal* with more processing power at the terminal in the expansion of the output information and/or the processing of user intervention. There is no upper limit for this category and preferably this intelligence is programmable.

A third method of terminal operation is the *cyclical batch* operation. The terminal is used interactively to collect, edit, check and correct substantial amounts of input data, which, after this preparation are transmitted as files to a host with appropriate processing power (this transmission is sometimes done via magnetic tape). The processing may take some time, and is often scheduled in a batch queue. The results

- transmitted to the terminal as files again - are interpreted interactively at the terminal, and the input data corrected accordingly. While the data are processed at the host, the terminal may be used for other purposes.

An application program using an intelligent terminal works on two processors with a certain *task division*. This task division is mostly defined when writing the application program and involves data division aswell. This latter is the more difficult: the separation of model data structure, store cross references, doubling some information and ensuring the influence of new born data at the "other" end in time.

This subdivision of data and work at present is decided by the application programmer, however the time may come for dynamic task shifting between host and terminal for better utilization of resources according to the workload.

The choice of appropriate operation mode depends mainly on the stages of the design process. Ergonomic and psychological facts have to be considered.

If the host is working under an application oriented monitor, this may behave in a familiar way in any mode towards the user.

Otherwise he has to learn about the manufacture's operating system, all about job quences, spooling, job restarts, parallel tasks, foreground and background, an similar things.

Virtual memory gives much help in processing big programs and if tuned to a certain user society may fulfill all their needs. If outside the horizon of the system tuning staff, one has to care for the internal parameters of the virtual system.

4. INTELLIGENT TERMINAL ARCHITECTURE

The basic configuration of an intelligent graphic terminal consists of:

- a mini (or micro) computer,
- an interactive graphic display,
- local peripherals, e.g.: card reader, printer, etc.,
- local backing store, and
- communications equipment.

The requirements for communications were discussed in paragraph 2. The communication software deals with line interrupts, the line control procedure and block transfer on an existing line.

LOGIN and LOGOUT for direct host terminal communication and the "virtual circuit" building is provided by an Initial Connection Module which has to be prepared for failure messages during terminal operation.

The different acces types - interactive job control, interactive program control, remote batch and file transfer - are dealt with by different processes. In a multi console configuration more than one process may run at a time with interactive program control.

These processes may run in parallel, if the configuration allows for it, or may run alternatively, overlayed from the local backing store.

A switch is incorporated between the Basic Transfer Module and the Process Modules to route the ariving data to their destination module.

The terminal software should run under a Disc Operating System (called by other names if other backing store is used), with the terminal software library, a progamming system and user file handling facilities.

As expressed earlier, the configuration described so far provides but a part of the needs of an "engineers' workstation". A modular, expandable design has to allow for further options: plotters, digitizers, archiving facilities may be required.

The DOS may be of the "single-shot" type or a multiprogramming system. There is a definite need for "dual programming" or "foreground-background" system which fall inbetween both in complexity and in processing power.

The sophistication of the operating system is limited by the terminal computer's processing power rather than the skills of software designers. Beyond these limits one can use two interconnected mini computers, and share the tasks between them. On the other hand, "end-functions" can be pushed out into device controllers, like communication drives, plotter drivers, etc. The latter solution is given much help by the fast development of microprocessors.

Again, as with dual programming operating systems, the first logical steps lead to dual processor configurations (one central processor with more device controller processors can still be regarded as such).

Beyond these lesser improvements experiments with the new technology suggest that we rethink the whole terminal architecture as it is established for both graphic processors and the mainframe computers themselves.

The structure of terminal graphics software is illustrated in *Fig.3*. A simple graphic terminal runs the graphic driver only controlled by a slightly separable protocol interpreter.

Intelligent terminals require a graphic package at the terminal which is compatible with the host package. Model building tools at both ends may (or may not) be regarded as graphic software. An important tool for graphic terminal operation is the macro definition, expansion or interpretation facility at the terminal.

The host and terminal computers have to be "cross programmable".

5. ON GRAPHIC PROTOCOLS

A graphic protocol is an agreement on the format and relative timing of data transmitted between two processes.

The agreements for the relative timing are quite simple and relate to the interpretation of input data in connection with output pictures.

Before going into the discussion of format agreements it is important to note, that agreed graphic protocols up to now were designed to provide a common format for conversion, rather than to serve as a standard hardware code of devices.

Also, the devices used in connection with these protocols differ substantially from each other.

The latter fact leads to the need for device categorization.

An important part of the protocol is the "initial dialogue" between the host process and graphic terminal. This initial dialogue is intended for the host process to establish the category and parameters of the graphic terminal and decide if the terminal is appropriate for the program's purposes, and adjust parameters at both ends for proper understanding through the protocol.

The host process has to decide e.g. if the terminal offered is an interactive device or a passive plotter, or if a plotter, is the accuracy and size acceptable for the program.

For simple terminals usually the host process asks questions and the terminal answers with its parameters. Alternatively, for an intelligent terminal the host process could perhaps make suggestions and the terminal could load the appropriate interpreter package if available and answer with an acknowledge or reject message as appropriate.

My feeling is that a very narrow group of devices can provide the parameters which an application program expects of one of its graphic devices. Thus the host process may have very definite suggestions. On the other hand intelligent graphic terminals - or any terminal - with present day technology can be programmed to accept a protocol which is an abstraction of the codes of a very narrow range of devices. This feeling is expressed in the lines below, where we try to give the outlines of a protocol designed for the GD'71/T intelligent terminal.

The first part of the protocol is called: simple graphic output. This part describes 16 bit word command codes and itneger data for line drawings.

The second part is the extended output. Codewords are identical to the simple output, but the format of data words can be choosen via a modal command word (single integer, double integer, floating point). Another model command word decides if 2D or 3D representation is used, while the third modal parameter describes the viewport convention used.

Three ways of viewport definition may be applied:

- data defining the viewport edges in device coordinates,
- in metric units,
- and as the percentage of the maximum possible viewport on the device.

The third part of the protocol allows macro definition, macro referencing, conditional expression and macro variables. This group of conventions help the construction and use of picture macro libraries and plot previewing on terminals with both plotter and graphic display. For the latter purpose values to macro variables may be given from the terminal console aswell.

These tools, for a plot-previewing facility were incorporated with the cyclical batch regime in mind.

Unfortunately FORTRAN which can be used to produce pictures via subroutine calls, is quite inadequate to describe macros and conditionals.

The input protocol deals with input events and processes associated with the display. The terminal accepts event enable/disable, process start/stop commands and a command for the association of a termination event to a process. Inputs are queued, and event reports and status reports are sent back in protocol format.

A framework for programmable intelligence at present is provided by an "ENTER SECTION" command, which invokes a new set of code interpretation rules. Two sections may have command codes with the same meaning beyong those few for the communication between sections.

Timing conventions ensure the arrival of all updates to a picture part before its use at the terminal and still allow certain data blocking for better line utilization. The host and terminal input queues are logically linked together, thus sequentially read input is sent to the host only if the host queue becomes empty, but for search manipulations the whole terminal queue has to be transmitted.

If programmed actions are defined at the terminal - and activated via codes in a special section - these actions have priority over the input queue and only the reports they leave in the queue are transmitted to the host.

APPENDIX 1.

A SUMMARY OF THE GD'71/T

GD'71/T includes a refresh type display unit with a 60 cms (24") diameter screen and 1024 x 1024 addressable raster points. The unit includes hardware character, vector and circular are generation.

Attention handling devices are: tracking ball, function keyboard, alphanumeric keyboard and the light pen.

The picture description resides in the core memory of a general purpose mini computer and is read via a DMA facility. It does not necessarily form a consecutive display buffer in the core store, the display control unit can follow picture structuring commands.

The mini computer core memory is expandable from 8K 16 bit words up to 32K words. The BUS is used to connect the basic peripherals: the alphanumeric console, paper tape i/o, line printer and the synchronous modem interface.

The 5 Mbyte disc store uses its own controller.

The Disc Operating System incorporates the usual components for programming the systems library, the user program library and data library handling tools.

The CDC 200 UT, CDC 734, IBM 3708 terminal emulators are system programs and do disc file transfer rather than using the slow peripherals.

The graphic software includes the basic i/o handling routines and graphic package with a FORTRAN based compatible version on host computers.

There is a graphic protocol interpreter (which at present interprets the basic output commands, extended output commands and a few macro commands).

The configuration used at the Institute has 16 Kwords of core store and beyond peripherals already mentioned, there is a CALCOMP 936 plotter connected to it.

APPENDIX 2.

COMPUTING FACILITIES FOR THE HUNGARIAN ACADEMY OF SCIENCES

1. Mainframes

A *CDC 3300* with 4 batch terminals and 6 alphanumeric consoles.
A 512 KByte *R-40* in Szeged.
A few smaller configurations.

2. Mainframes planned

A 1 MByte *R-series* machine planned for 1977.
One new central computing facility with substantial remote access, local storage and processing power for HAS planned for 1979-1980.

3. Networking

Terminal network for the CDC 3300.
Terminal network for all R-series machines with own front-end-processor design, envisaged for 1977.
In-house temporary symbioses of small computers.
The LITER terminals: a batch terminal emulator embedded into the disc operating system of a mini (at present working with CDC 220 UT and IBM 3708/2708 modes).
The GD'71/T refresh graphic display and Intelligent Graphic Terminal.
The preparation of the HAS packet switching network. Participation in the IIASA packet switching network project.