KERNFORSCHUNGSZENTRUM

KARLSRUHE

Februar 1977 KFK 2394

Institut für Reaktorentwicklung

A Recommendation on Methodology in Computer Graphics

J. Encarnaçao, Technische Hochschule Darmstadt
B. Fink, Philips GmbH, Forschungslabor Hamburg
E. Hörbst, Siemens AG., Zentralforschungslabor München
R. Konkart, AEG-Telefunken, Konstanz
G. Nees, Siemens AG., Erlangen
D. L. Parnas, Technische Hochschule Darmstadt, now University of North Carolina at Chapel Hill
E. G. Schlechtendahl, Gesellschaft für Kernforschung, Karlsruhe

GESELLSCHAFT
FÜR
KERNFORSCHUNG M.B.H.

KARLSRUHE

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.

KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

KFK 2394

Institut für Reaktorentwicklung

A Recommendation on Methodology in Computer Graphics

J. Encarnacao, B. Fink, E. Hörbst, R. Konkart,
G. Nees, D.L. Parnas, E.G. Schlechtendahl

Authors' Affiliations:

Technische Hochschule Darmstadt

Philips GmbH, Forschungslabor Hamburg

Siemens AG., Zentralforschungslabor München

AEG-Telefunken, Konstanz

Siemens AG., Erlangen

Technische Hochschule Darmstadt, now University of North Carolina at Chapel Hill

Gesellschaft für Kernforschung, Karlsruhe

Gesellschaft für Kernforschung m.b.H., Karlsruhe

Abstract

This report shall provide a design basis for device independent computer graphics. The concept of a portable graphics system is outlined, its interfaces to output and input devices, to the operating system and to the application program are described. A recommendation on development methodology starting from formal specification is given.

The report is the result of a cooperation of scientists from a number of institutions in the Federal Republic of Germany, all dealing with computer graphics. The initiative for forming the group was taken by Prof. Encarnacao. The members of the group represent computer scientists, developers of graphic systems and users of graphic systems. The goal was to establish a consistent approach to the methodology for defining, describing, designing und using graphical systems. The paper was presented to the IFIP GRAPHICS WORKSHOP, Chateau de Seillac France, May 23-26, 1976

Zusammenfassung

Eine Empfehlung zur Methodologie in der graphischen Datenverarbeitung

Dieser Bericht bietet eine Entwurfsgrundlage für geräteunabhängige graphische Datenverarbeitung. Das Konzept eines portablen graphischen Systems wird dargestellt; die Schnittstellen zu Ein- und Ausgabegeräten, zum Betriebssystem und zum Anwendungsprogramm werden beschrieben. Eine Vorgehensweise für die Entwicklung, basierend auf einer formalen Spezifikation wird gegeben.

Der Bericht ist das Ergebnis einer Zusammenarbeit von Wissenschaftlern mehrerer Institutionen der Bundesrepublik Deutschland, die alle mit graphischer Datenverarbeitung befaßt sind. Die Initiative zu dieser Gruppe ging von Prof. Encarnacao aus. Die Mitglieder der Gruppe repräsentieren Informatiker, Entwickler und Anwender graphischer Systeme. Das Ziel war ein in sich konsistenter Ansatz zur Methodologie hinsichtlich Definition, Beschreibung, Aufbau und Verwendung graphischer Systeme. Die Arbeit wurde dem IFIP GRAPHICS WORKSHOP, Chateau de Seillac, Frankreich, 23.-26.Mai 1976 vorgelegt.

CONTENTS

1. Introduction

The motivation and objectives for a standard graphical software are in addition to uniform representation of graphical problems, portability or machine independence and the device independence.

Demands that should be fulfilled by a standard graphical system are:

a) It should be defined in a way general enough to support a wide variety of users,
b) The definition should be easily understood for programmers (users).

To be able to reach the desired goal of having such a device independent, general-purpose, easy understood system with a good chance of success, we propose to make the following restrictions:

a) The graphics functions to be defined include the construction and manipulation of pictures (no picture analysis)

b) A picture is generated from vector and/or text elements (no gray-scale pictures)

c) Only those graphics peripheral devices that are suitable for a general-purpose system will be considered. These include:

   Text displays
   Plotters, Digitizers, COM
   Storage tube devices
   Refresh displays
   Refresh displays with transformation hardware

d) The definition of the graphics functions, that in the final analysis represent the user interface, should reflect the expected frequency of occurrence of the operational device types e.g. the hardware of the most frequent used device types should be as effectively as possible utilized. One may accept a loss of efficiency for more unusual device types.

Device independence (e.g. independence from graphic peripherals) is the point on which we want to concentrate. We define device independence as the ability to use an application program with a wide variety of devices.

Standarization in this area should contribute to our ability to produce graphical data in one computer and then output it successively or simultaneously in different graphics output devices (also using external intermediate storage). Simultaneous output on different devices is important when it is necessary to work with different picture sections, for example a general view and some detail view of the same picture. It is very important to have in addition to output device independence, also input device independence. This will be realized in this concept by using logical input devices and having them assigned to physical input devices.

1.1 Review of graphic systems

We can distinguish between the following four types of graphic systems:

1) Terminal systems - these systems were in both hardware and software manufacturer dependent (for example IBM 2250, CDC 1700-Digigraphics, Adage, Vector General). They were developed mainly for users coming from the military area, from the car or airplane industry, etc. and also mainly financed from there.

2) Type specific systems - these systems were primarily designed for special device combinations (Plotter/Tektronix for example), and used mostly in computing center environments. For the input there were only special solutions, since systems had no convenient structure. The output was generally solved and the corresponding routines were imbedded in a high level language (mainly FORTRAN).

3) User specific systems - these systems were designed for a special application and were optimized for that application (for example Computervision). Even for slight changes of the applications or on the basic techniques one would be dependent on the manufacturer, since the systems were not transparent for the user.

4) Device independent systems - these are new systems, which try to solve the problem of device independence based on a stepwise development.

A methodological concept for device independent systemswill be topic of this position paper. Therefore we will first give some constraints and then discuss the basic interfaces in such a system, as well as its implementation. We will also shortly consider system aspects such as the environment provided by the operating system and the users' influence on simulation routines.

1.2 Constraints for a general-purpose concept

Today in the commerc ıl and scientific-technical data processing it is not significant which external storage device is just in use, provided that the devices make the same basic functions (for example direct access) possible. The same should be valid for graphical peripherals in the future.

With interactive systems it should also be possible to process different graphic dialog modules in parallel.

We have to proceed on the assumption that more and more applications will deal with a virtually unlimited amount of data (for example 60000 vectors).

In order to make an adjustment from the user side feasible, it has to be possible to determine device parameters (for example the set up of drawing area) in the application program by calling subroutines.

2. Interfaces in a graphic system

The basic idea in the design of a device-independent general purpose graphic system is to split the system in a device-dependent and a device-independent part and to have so the graphic input and output completely separated from the user program. This concept follows /COT 72/, was discussed in/ENC,ECK 76/ and is shown in Fig. 1. The interesting interfaces for our purposes are there:

a) User interface - it should be completely independent from the application

b) Code generator interface - this is the interface between pre-processor and output processor. The pre-processor interprets the graphic output functions of the application program and generates device independent data; this data will be translated by the output processor into a device

dependent form. Relative to the output the pre-processor has therefore picture compiler characteristics, the output processor on the other hand has picture assembler characteristics.

c) Logical input interface (only by interactive graphics)

- this is the interface between input and pre-processor. The input processor processes the physical input events and transfers the logical input results to the pre-processor.

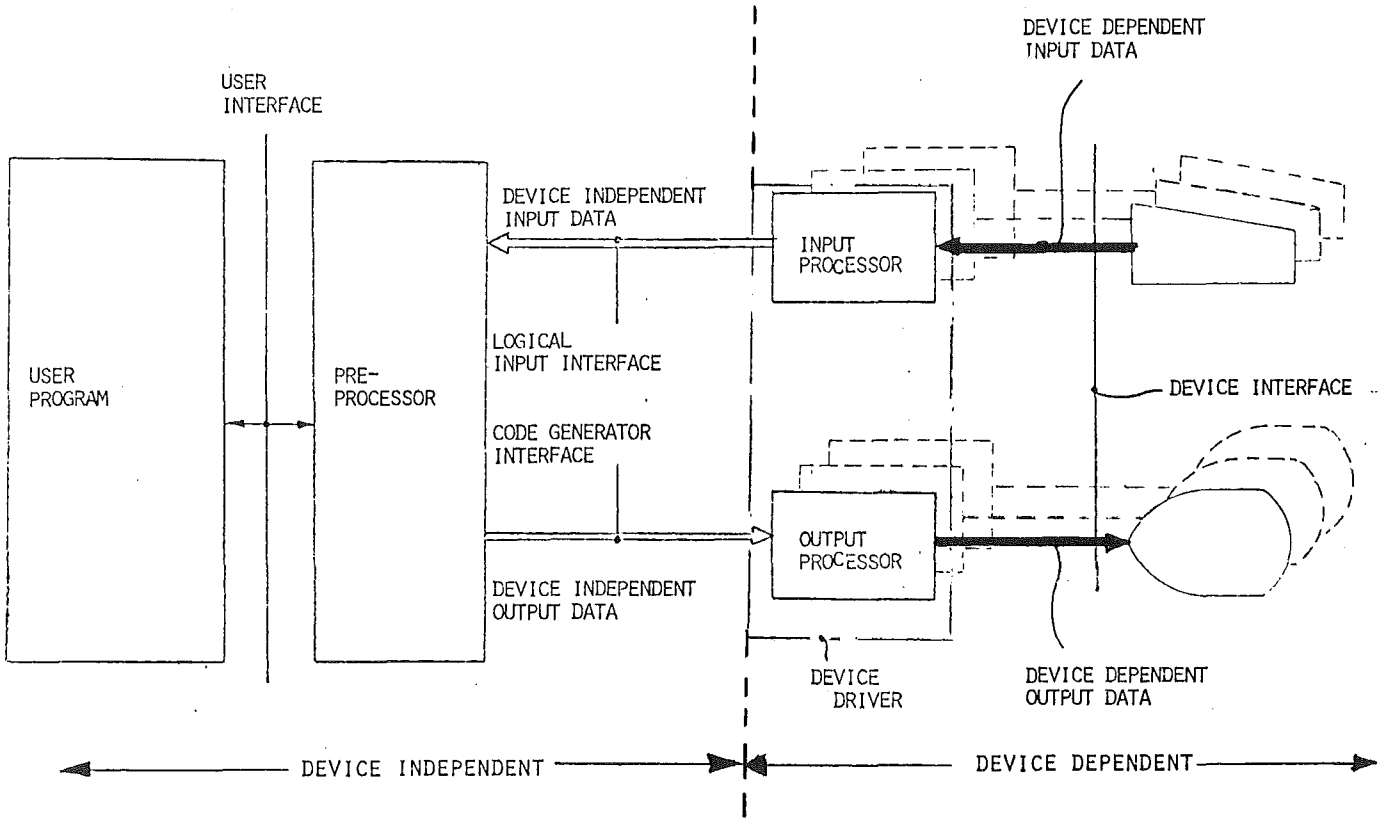


Fig. 1: Separation between user program and graphical input and output

In order to be able to discuss in detail these interfaces, we have first to deal with graphical i/o-devices, its classification and their inference in the graphic system.

3. Graphical output devices and corresponding formats

Regarding the classification of output devices we might characterize them according to their operating behaviour or according to the graphic elements which they accept for display. The first criterion would distinguish between "random" or "incremental" devices on one side (XY-plotters, random scan tubes, etc.) on which graphic objects may be displayed one at a time and "sequential" or "batch" devices on the other side which can display only whole pictures (electrostatic plotters, colour jet plotters, etc.). In the sequel we consider the second criterion and propose following /PHIL 75/ and/ENC,ECK 76/ an upwards compatible classification for these devices, which means that the characteristics of the lowest device class should be a subset of the characteristics of the next higher class. This means that the characteristic graphic basic functions of a certain class may always be implemented in higher classes. We classify the devices according to the formats which the corresponding output processors will accept.

The elements for the classification are:

class 1 : symbol
class 2 : symbol, point, line, (circle, arc)
class 3 : segment
class 4 : oriented graph

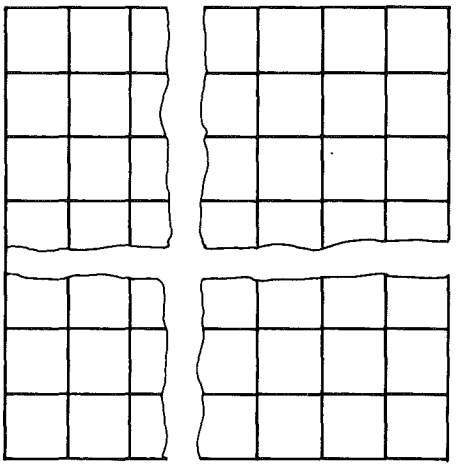
In Fig. 2 for the four classes the corresponding format of its representation in the computer is shown.
A typical device of class 1, which would accept the line-column format would be an A/N-Display. If we include the vector graphic supported by class 2 devices, we have the unstructured format.

A segment is a linear list of graphic primitives and represents the simplest graphics operand (display group), that can be manipulated as an unit and be used to build bigger units. These are called collections, they are a linear list of segments.

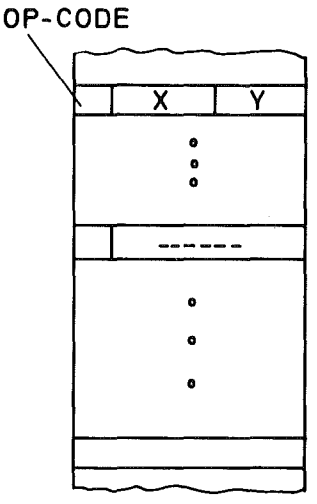
A segment is the characteristic element for the device class 3, since the selective picture manipulation is the device characteristic. The corresponding format is the segmented format. This is the usual picture format, that we find in the types of graphic system discussed in the chapter 1.1.

The oriented graph is the characteristic element for the device class 4; the corresponding format is the structured format. This format is not suitable for devices without transformation hardware.

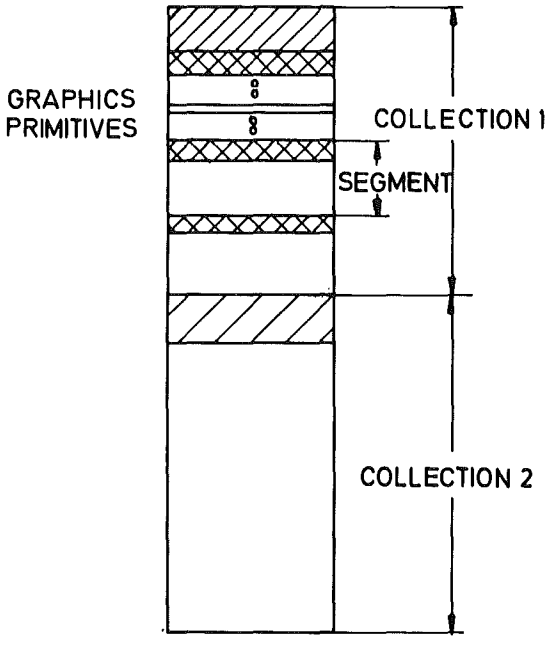
These different classes of output devices and some typical examples /ENC, ECK 76/ are shown in Fig.3.



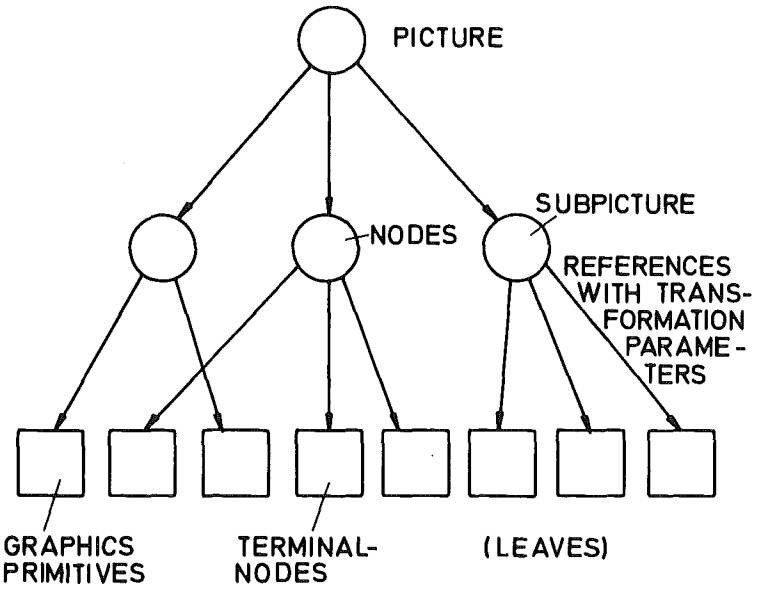
CLASS 1: LINE-COLUMN FORMAT



CLASS 2: UNSTRUCTURED FORMAT

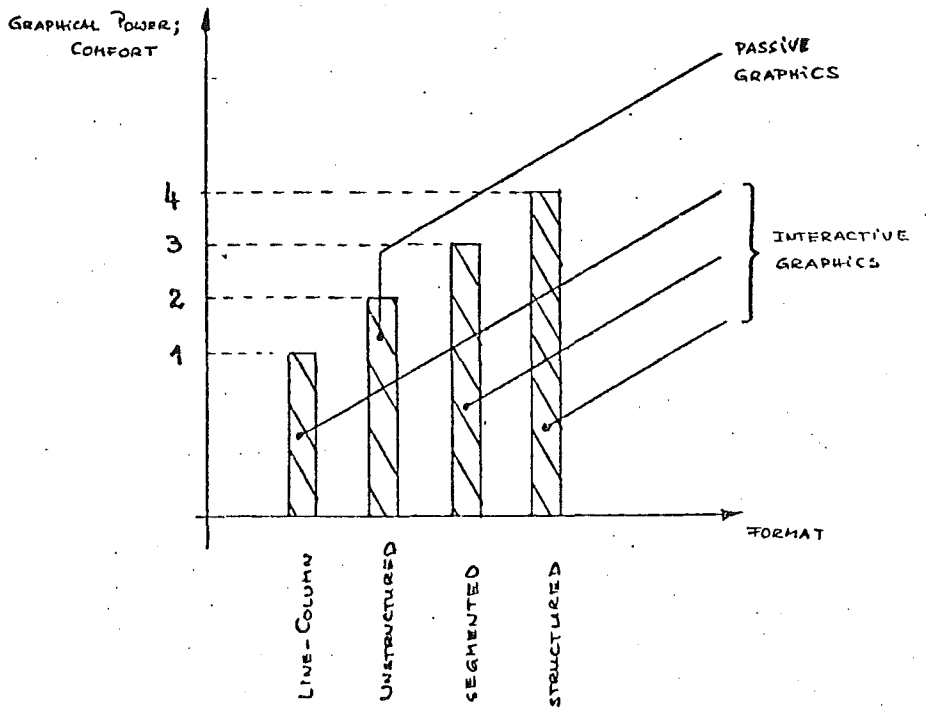


CLASS 3: SEGMENTED FORMAT



CLASS 4: STRUCTURED FORMAT

FIG. 2: FORMATS OF THE FOUR CLASSES OF GRAPHICAL OUTPUT DEVICES



DEVICE CLASS	CHARACTERISTIC	TYPICAL DEVICES
1	ONLY TEXT GRAPHIC	A/N - DISPLAYS
2	AS CLASS 1 PLUS VECTOR GRAPHIC	• PLOTTERS • STORAGE DEVICES • RASTER DEVICES
3	AS CLASS 2 PLUS SELECTIVE PICTURE MANIPULATION	DISPLAYS WITH PICTURE REGENERATION
4	AS CLASS 3 PLUS REAL TIME GRAPHICS	DISPLAYS WITH TRANSFORMATION HARDWARE

FIG. 3: CLASSES OF GRAPHICAL OUTPUT DEVICES

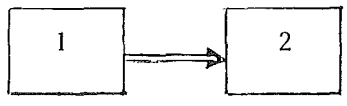
The following tables proposed by R. ECKERT give a detailled information about these four formats. They contain the basic functions for the generation and manipulation of the different formats. We consider a graphical language with operands (display group) and operators. The operands are built out of primitives, which may have attributes /ENC,ECK 76; ECK 76/.

In the first row we give the characteristic elements of the four formats; the second row contains the corresponding localization data. The possibilities of grouping the characteristic elements and the corresponding identifying data are presented in the subsequent two rows. Finally the different attributes and the different kinds of graphic operators are listed.

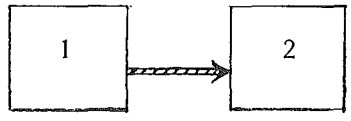
4. Basic design of a device-independent graphic system

The basic design of a device-independent graphics system is shown in fig. 4.

Two types of connectors are used to represent control and data flow in these figures in accordance with the methodology to be described in chapter 13.



Block 1 sends information to block 2, thus changing the state in block 2 and/or subsequent blocks (O-function). Data and control flow is in the direction of the arrow.

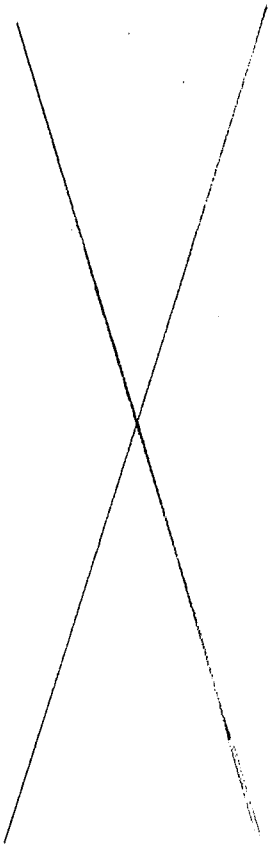


Block 1 requests information from block 2 and/or subsequent blocks (V-function). Control flow is in the direction of the arrow, data flow is opposite to the arrow direction.

In the following we shall distinguish between the hardware which is used to produce a picture, an interrupt or an input message on one side and the identification on the other side, which is used in the users' program to specify the hardware piece to which a picture is to be sent or from which message is being expected. We call the hardware pieces themselves "devices", while we call their identification in the users' program and in the graphics system "units". We do this in analogy to standard I/O practice. The I/O unit number in FORTRAN or the file in PL 1 corresponds to the "unit", while the printer, card reader, tape etc. corresponds to the "device". Thus the application program will be written in terms of units only and will not refer directly to devices.

The different possible ways on generating pictures lead to the definition of two different kinds of picture buffers. In Fig. 4 these buffers are called pseudo picture code and device dependent picture code. The Pseudo-Picture-Code (PPC) is a device independent structured description of the picture to be displayed and it contains all the picture data in user specific form. The pseudo-picture code serves as a source representation, that can be used to control several output devices.

	Line column format	Unstructured format	Segmented format	Structured format
Characteristic element(s)	Symbol • A/N-character • special Symbols (these are the graphics primitives)	Graphics Primitives • symbol • point • line segment	Segment (see Fig. 2)	Oriented graph (see Fig. 2)
Data for the localization of the characteristic elements	• line-number • column-number	• character (4 numbers) position coordinates character area • point 2D/3D absolute relative • line segment 2D/3D absolute relative	• static segments all coordinates are absolute no positioning data • segments, which may be modified absolute positioning coordinates relative coordinates in the graphic primitives	localization data is part of the calling entities
Display groups	character line vector line column vector column matrix picture		graphics primitives segment collection	leaf (terminal nodes) calling entities (nodes)
Identifying data for the display groups	line, column line, begin and end of string line number column, begin and end of string column number start line, start column, end line, end column		graphics primitives IC:collection name IS:segment name IG:name of the graphics primitives in a segment (IS) of a collection (IC) segment IC IS collection IC	name of the leaf name of the calling entity

	Line-column format	Unstructured format	Segmented format	Structured format
Attributes	blink mode safe mode	character size write directions brightness	character size write directions brightness selectability blink mode positioning mode	character size write directions brightness selectability blink mode positioning mode 4X4-Matrix
Graphics operators	define graphics operands define attributes change attributes scrolling		begin segment end segment begin collection end collection omit segment omit collection delete segment delete collection extend segment extend collection translate segment translate collection scale segment scale collection rotate segment rotate collection define attributes change attributes	define leaf define node omit leaf omit node delete leaf delete node translate (node) zoom (node) rotate (node) define attributes change attributes

The Device-dependent Picture Code (DDPC) is used for picture refresh and is generated from the pseudo picture code in such a form, that in each case the selected device can be used in an almost optimal way. That means, if the output device is able to interpret subpictures or subpictures out of the structured description, then this ability will be utilized; the same is valid for transformations and graphical primary elements (primitives) as circle and other curve generators. If when designing a graphics system, we want to take into consideration the whole spectrum of existing I/O-devices and we want to be able to use them effectively, then it should in principle be possible to omit both internal picture definitions. Of course in a specific case, for example plotter output, the one or the other or both of these may be dropped. In Fig. 4 hardware output, input and interrupts producing devices of the types listed in chapter 1 have been considered. Other hardware units of similar characteristics may be included.

The question of simulation of certain units in devices which do not readily provide the required capabilities, will be discussed later in chapter 11. Also the question of separated computers will be trated in chapter 12.

Each users' application program communicates with the graphics system by means of an interface. Within the scope of a "Begin" and "End of the graphics system" as described in chapter 11 this interface consists of:

a) identification of graphic units (e.g. names, numbers)
b) identification of graphic objects (e.g. names, numbers)
c) routines, callable in the users' program language
d) possible working space for the graphic system.

The graphics supervisor communicates with

a) the PPC,
b) service routines for picture transformations and
c) device dependent routines.

In order to select the appropriate device dependent routine, the graphics system will have to read the device description table. When an interrupt is generated from a unit, the flow of control is reversed, it now goes up from the unit to the graphics system.

The overall structure of the system reflects the fact that the whole spectrum of available graphics units has been taken into account. In particular the consideration of refresh type displays and the necessary high speed interaction is responsible for a certain redundancy of information. Namely: the information, which is

necessary to produce a picture of various objects on an output unit, is certainly available in the users data base. In order to avoid costly operations in case of simple modifications of the picture (e.g. rotation) the PPC and the archive contain a redundant but more suitable set of information.

5. The structure of the pseudo picture code (PPC)

This list contains the graphical information in user coordinates; besides that there is an administration list.

The primitives, that may be included in the PPC-list are:

	Primitives	Attributes of the collection, that are effective on the primitives
2 D	Point: X, Y	blink, intensity, selectability(*) and colour
	Line: X, Y or X_1, Y_1, X_2, Y_2	blink, intensity, selectability colour and line modus
	Circle: centre X_M, Y_M and radius	- " -
	Arc: centre X_M, Y_M	- " -
	starting point X_A, Y_A	- " -
	another free point X_F, Y_F	- " -
	direction	
	Symbols: Code or Code string	- " - in addition: size and direction of line drawing
3 D	Point: X, Y, Z Line: X, Y, Z, or X_1, Y_1, Z_1 X_2, Y_2, Z_2	as in 2 D

(*) Selectability means that the object can be indicated from the device

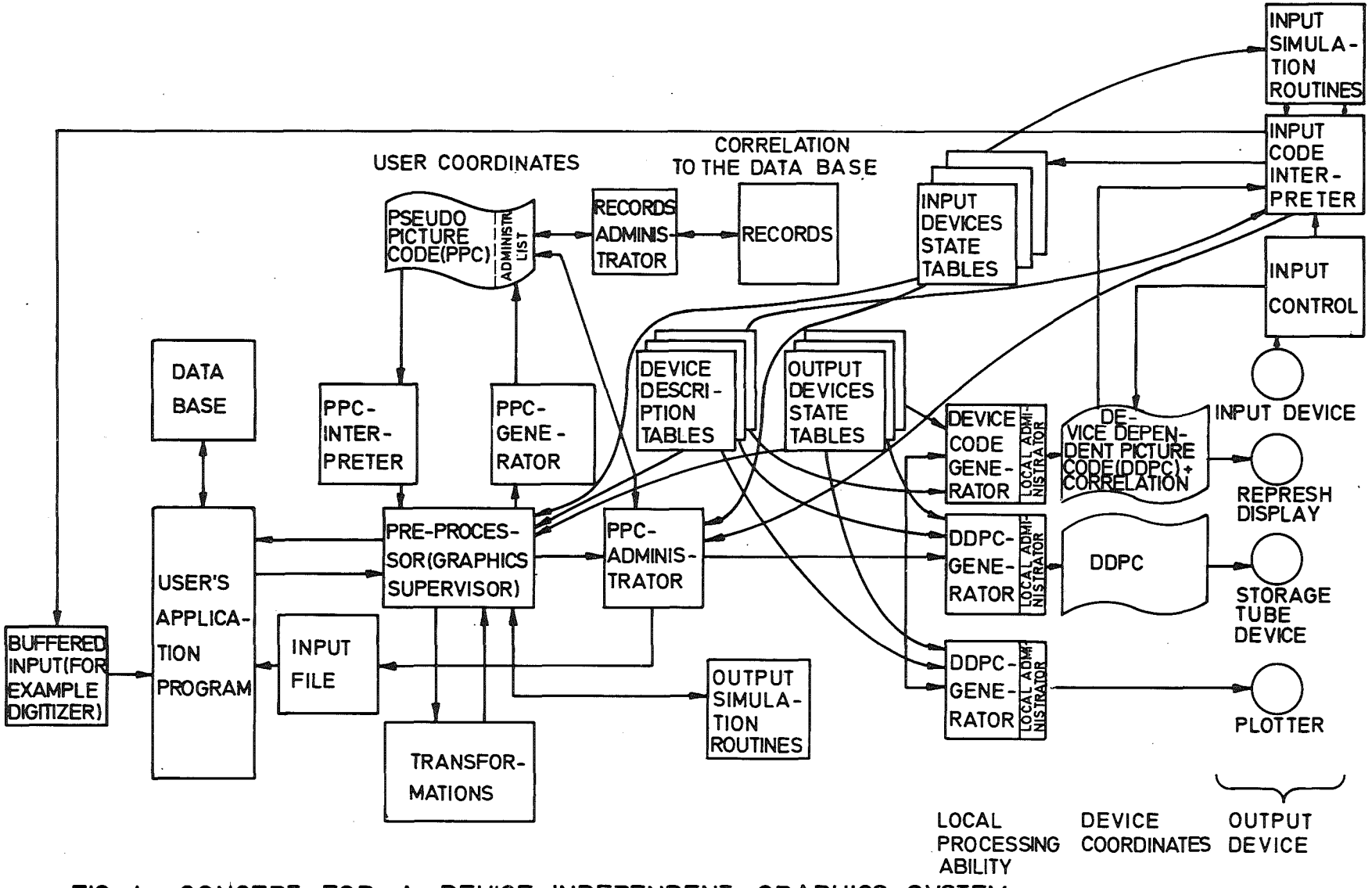


FIG. 4: CONCEPT FOR A DEVICE INDEPENDENT GRAPHICS SYSTEM

Primitives which represent for example a square surface and an acoustic signal are included among the symbols; characters are also viewed as symbols.

The coordinates can be absolute or relative. Absolute coordinates refer to the zero of the users coordinate system; relative coordinates on the other hand refer to an actual reference point, that may be stored in a stack.

The PPC-list is structured as follows: We distinguish between "picture" and "subpictures". The "subpictures" have to be "called" in order to become effective. The following is a rough definition of the syntax of "picture" and "subpicture". (We underlined those terms which we did not define in more detail at this state).

< picture >:: = picture header < sequence of entities > end mark
< sequence of entities >:: = < sequence of entities >< entity > | < entity >
< entity >:: = < collection > | < general primitive >
< collection >:: = collection header < sequence of entities > end mark
< general primitive >:: = point | line | circle | arc | symbol | subpicture call

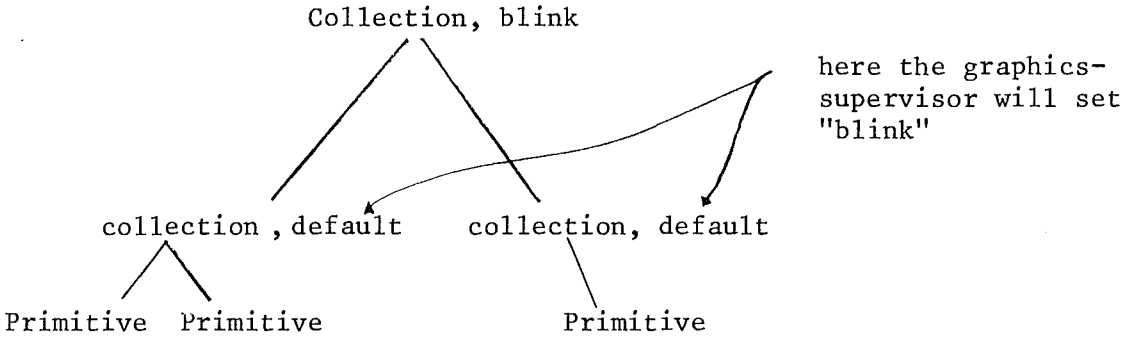
< subpicture >:: = subpicture header < sequence of primitives > end mark
< sequence of primitives >:: = < sequence of primitives >< general primitive > |
< general primitive >

Collections and subpictures can be modified (i.e. delete, insert, move, ..) but primitives - and thus subpicture calls - can not be modified. The latter restriction allows to implement in the DDPC subpicture calls as efficient as possible, namely by in-line generation (copy of the subpicture) or by using hardware subpicturing (Return-Jumps). Normally hardware subpicturing is restricted to a maximum allowed subpicture nesting depth also in the device independent part.

The headers of the various items (picture, collection, subpicture) contain information relevant to the entire item. Among others a header contains the corresponding ID and the attributes valid for that item. In case of ambiguity (which results from nesting of collections and/or subpictures) the attributes of the lower-level item suspend meanwhile those of the superior item.

Considering the application program, the attributes can be defined explicitly or implicitly. When they are defined implicitly, then the graphics supervisor will automatically take the values from the superior item or, if there is none, default values.

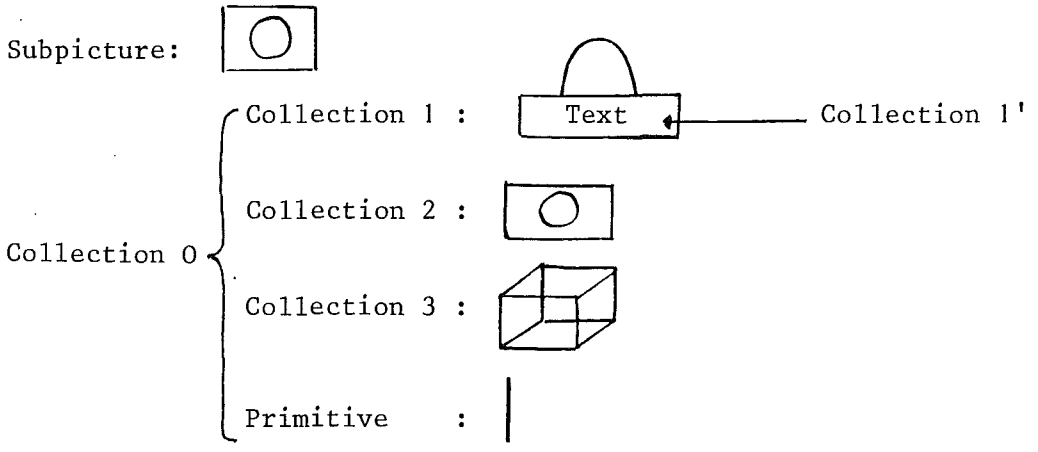
Example:



6. Production of the output

To describe the output we will assume, that for a simple example describing the generation of a graphical object the corresponding program (written consistent to the BNF of chapter 5 but otherwise arbitrary syntax) based on Fig. 4 has to be processed.

Example:



Programm:

```
BEGIN SUBPICTURE (NAME)   ┐
ATTRIBUTE ( INTENSITY)    │
DRAW (4, ARRAY)           ├ Subpicture
CIRCLE ( X, Y, RADIUS )   │
END SUBPICTURE (NAME)     ┘

BEGIN (IDo)                                                          ┐
COMMENT (TEXT)                                                       │
    BEGIN (ID1)                                                      │
    DRAW (4, ARRAY)                                                  │
    SHAPE (3, ARRAY)                                                 │
      BEGIN (ID1')                                                   │
      ATTRIBUTE(INTENSITY)                                           ├ collection 1
      TEXT (ARRAY)                                                   │
      END (ID1')                                                     │
    END (ID1)                                                        │
    BEGIN (ID2)                                                      │
    SUBPICTURE (NAME)                                                │
    END (ID2)                                                        │
    BEGIN (ID3)                                                      │
    DRAW (12, ARRAY)                                                 │
    END (ID3)                                                        │
    LINE (IX, IY)                                                    │
END (IDo)                                                            ┘
```

collection 1' = BEGIN (ID1') … END (ID1')

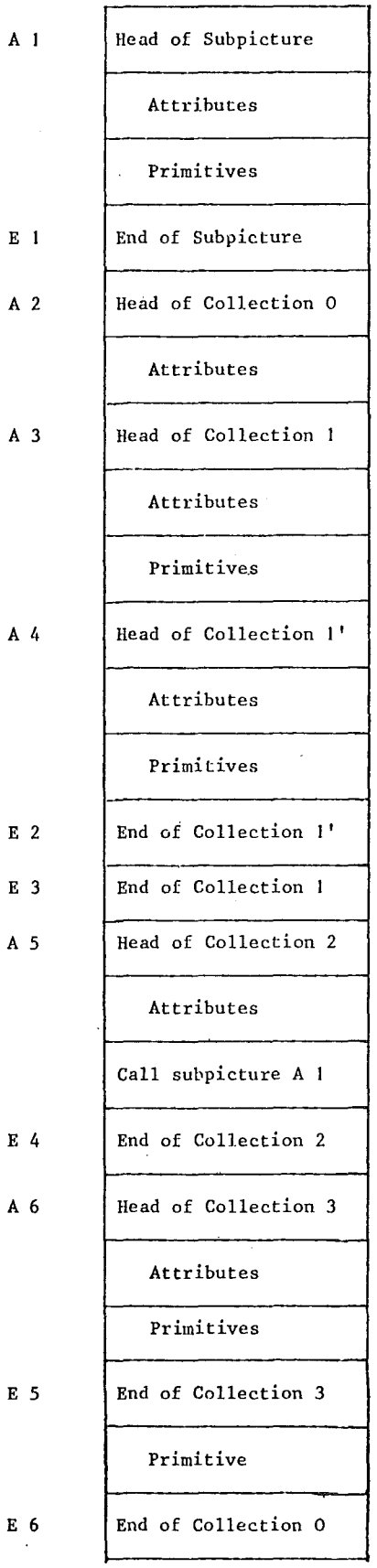
collection 2 = BEGIN (ID2) … END (ID2)

collection 3 = BEGIN (ID3) … END (ID3)

Primitive = LINE (IX, IY)

collection o = BEGIN (IDo) … END (IDo)

The pseudo-picture code generator will produce if we suppose the segmented format discussed in chapter 3 the following PPC-Code from the given program. If, however, the graphics system is processed on a special-purpose hardware, then the structured format (see chapter 3) may also be used as the basis for the implementation.


A 1
Head of Subpicture
Attributes
Primitives
E 1
End of Subpicture
A 2
Head of Collection 0
Attributes
A 3
Head of Collection 1
Attributes
Primitives
A 4
Head of Collection 1'
Attributes
Primitives
E 2
End of Collection 1'
E 3
End of Collection 1
A 5
Head of Collection 2
Attributes
Call subpicture A 1
E 4
End of Collection 2
A 6
Head of Collection 3
Attributes
Primitives
E 5
End of Collection 3
Primitive
E 6
End of Collection 0

Default attributes are inserted in the above PPC-Code, whenever they are not explicitly defind in the sample program.

An is the address of the corresponding collection. Since collections may be nested, an end mark En for each collection is stored. By means of a given end mark it is possible to find primitives after an END (ID).

The PPC-administrator builds the two following tables:

ID-list:

IDs of the collection levels				Address (A.. Begin; E.. End)	
IDo	0	0	...	A2	E6
IDo	ID1	0	...	A3	E3
IDo	ID1	ID1'	...	A4	E2
IDo	ID2	0	....	A5	E4
IDo	ID3	0	...	A6	E5

Subroutine-list:

Name	Address	
Name	A1	E1
–	–	–

The graphics-supervisor can now (using the PPC interpreter) interpret the PPC-Code in normalized system coordinates. It can (by use of the device description tables) determine whether a simulation (for example for the circle by non-existence of a circle-generator) has to be performed or not. The graphical information will then be available in the following form

PPC-Address	PPC-Code	X, Y or similar

The primitives are produced and output in device coordinates by the device code generator with the support of the device description tables and device state tables. The following correlation table is produced by the device code generator. It will be used later on when processing the input or making changes for the mapping between PPC and DDPC codes.

Correlation table:

PPC-Address	DDPC-Address
A2	AD 1
A3	AD 2
A4	AD 3
E2	AD 4
E3	AD 4
A5	AD 4
E4	AD 5
A6	AD 5
E5	AD 6
E6	AD 7

Between E2 and A5 as well as E4 and A6 no code will be produced.

Let us now consider some output functions. If an attribute is to be changed, for example the collection in our example shall blink, then a list, in which the ID hierarchy is stored will be accepted as input parameters. The PPC-administrator must then find all heads and internal end marks of the ID (collection) and will set the blink attribute; through the device code generator and the DDPC-administrator, the corresponding addresses will be set up. When executing a delete the PPC-administrator must delete all the corresponding ID-addresses out of the ID-list. Having refresh-displays this must also happen for the correlation table, which must be reorganized afterwards. If we have storage tube devices, then the

collection which is desired may be shown, because it is not always desirable to erase the picture and to make a new display of the changed picture.
Let us have a look at the insert function; the PPC-administrator has to find the corresponding ID, to cut off the PPC-list at that position and by means of the administrator to reorganize all lists. Another possibility is to delete and to build the collection "from scratch". A third possibility is the use of JUMPS or similar commands, when they exist in the graphics-supervisor.

If a picture is dispayed on different devices, in which different functions are processed with the same picture data, then we have to produce several PPC-copies - one for each output device. This is shown in Fig. 5.

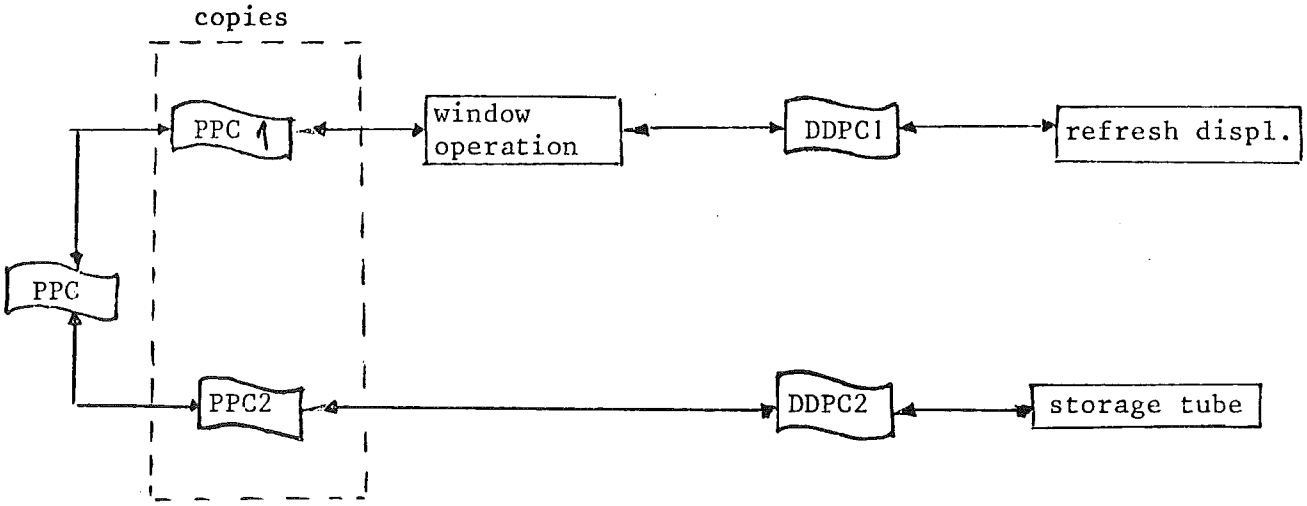


Fig. 5: PPC-Output of different devices

Fig. 4a represents the output part of the graphics system from Fig. 4, as discussed in detail in this chapter.

7. Input process and input code interpreter

We assume the use of logical input devices (input units). As classes of input characteristics we shall consider

I1 : text input

I2 : 1:N - choice (for example menu, function keyboard)

I3 : input of scalar, analog values (for example dials)

I4 : position input

I5 : identification

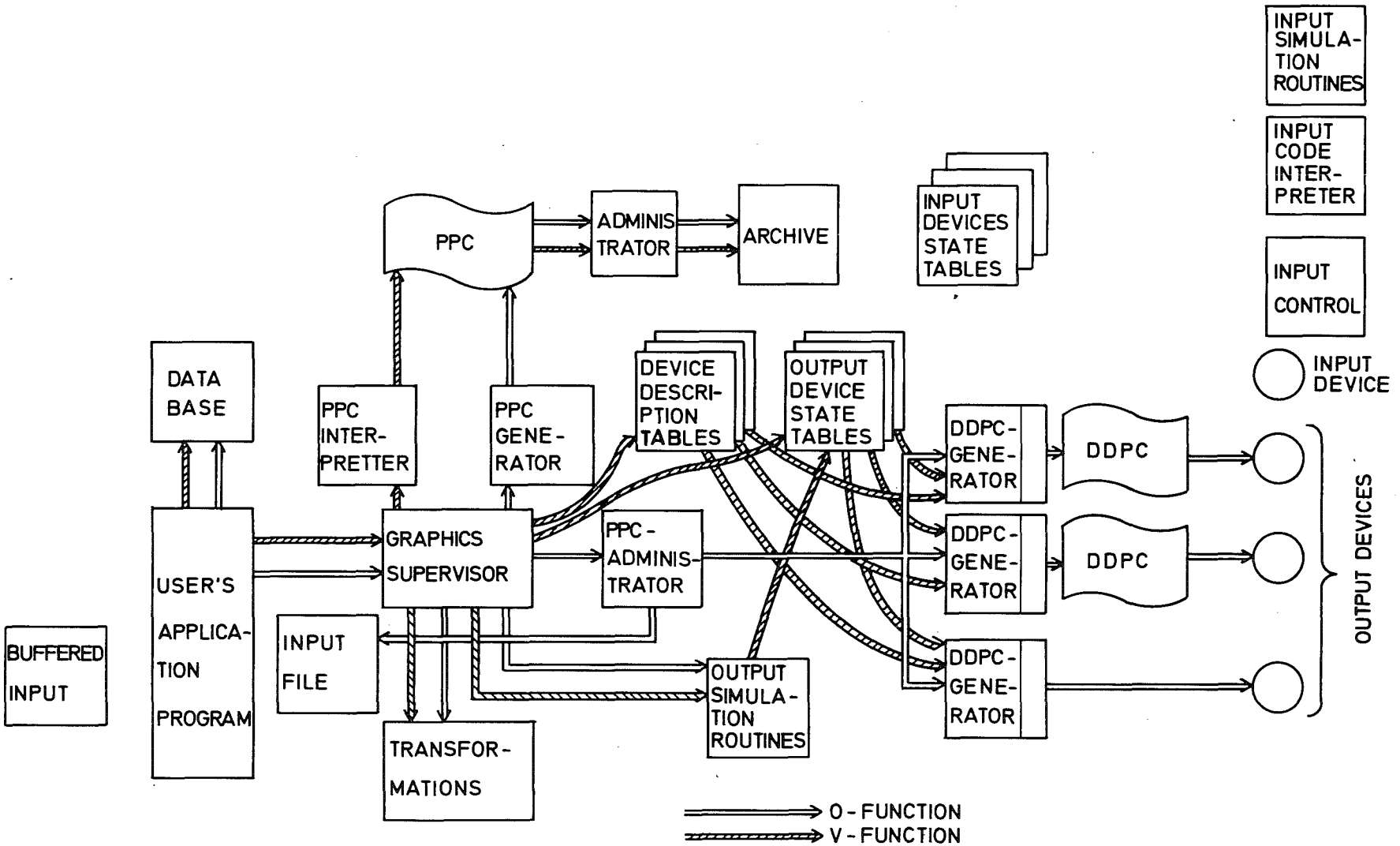


FIG. 4A: OUTPUT FLOW IN THE GRAPHICS SYSTEM

These logical functions may be realized by certain input devices; if they are not available, they can be simulated by pseudo-devices. The corresponding simulation routines are administrated by the input code interpreter.

Meaningful and implemented input functions are - stored in an device allocation table - part of the device description tables. An example of such a description (which can be different for each installation), is the following table:

logical input functions	physical input devices								simulated input devices			
	light-pen	tablet	function keyboard	dial	A/N-keyboard	tele-type	joy-stick	menu	pseudo-dial	tracking	mouse	pseudo-lightpen
I 1					X	X						
I 2			X		X	X		X				
I 3				X					X			
I 4		X					X			X	X	
I 5	X									X		

If the installation provides more than one tablet e.g., then this column in the table above would have to be subdivided. Corresponding if the application program requests the use of more then one input unit of type I4 e.g., then the corresponding row would have to be subdivided.

The input code interpreter transforms the information taken from the input device into the following format

Device identification	Type of information	Information Code

The PPC-administrator uses it for PPC.manipulations and/or for passing it to the input file.

The information codes of the single logical input functions are:

I 1 - An Array with ASCII-Code
I 2 - A number
I 3 - A scalar
I 4 - Three scalars (normalized system coordinates) as identification input
I 5 - ID search information

The input code interpreter receives (under control of the local DDPC-administrator) as information the correlation between the physical address in the DDPC-list and the physical address of the same segment in the PPC-code interpreter and can herewith determine the corresponding collection by the aid of the ID search information. It supplies then the users' program with the corresponding ID's of all superior collections to which it belongs or waits until this information is requested by the user's program. Also the device identification is replaced by the corresponding unit identification, when the message is passed to the application program.

The user will then acknowledge the input; by the message acceptance by the users' program out of the input file this acknowledgement will be deleted. The interrupt handler gives each device its own priority in the operating system (pre-processor).

Fig. 4b represents the input part of the graphics system from Fig. 4, as discussed in detail in this chapter.

8. Device tables

In the device state table (both for input and output devices) is stored whether the device is on or off and in which form it may actually be addressed (that means which device type is at the moment simulated on it).

The device description table contains the hardware parameters of the I/O-devices, so that the device code generators are able to produce the corresponding DDPC for output or PPC for input. Besides that it contains pointers to the device allocation table.

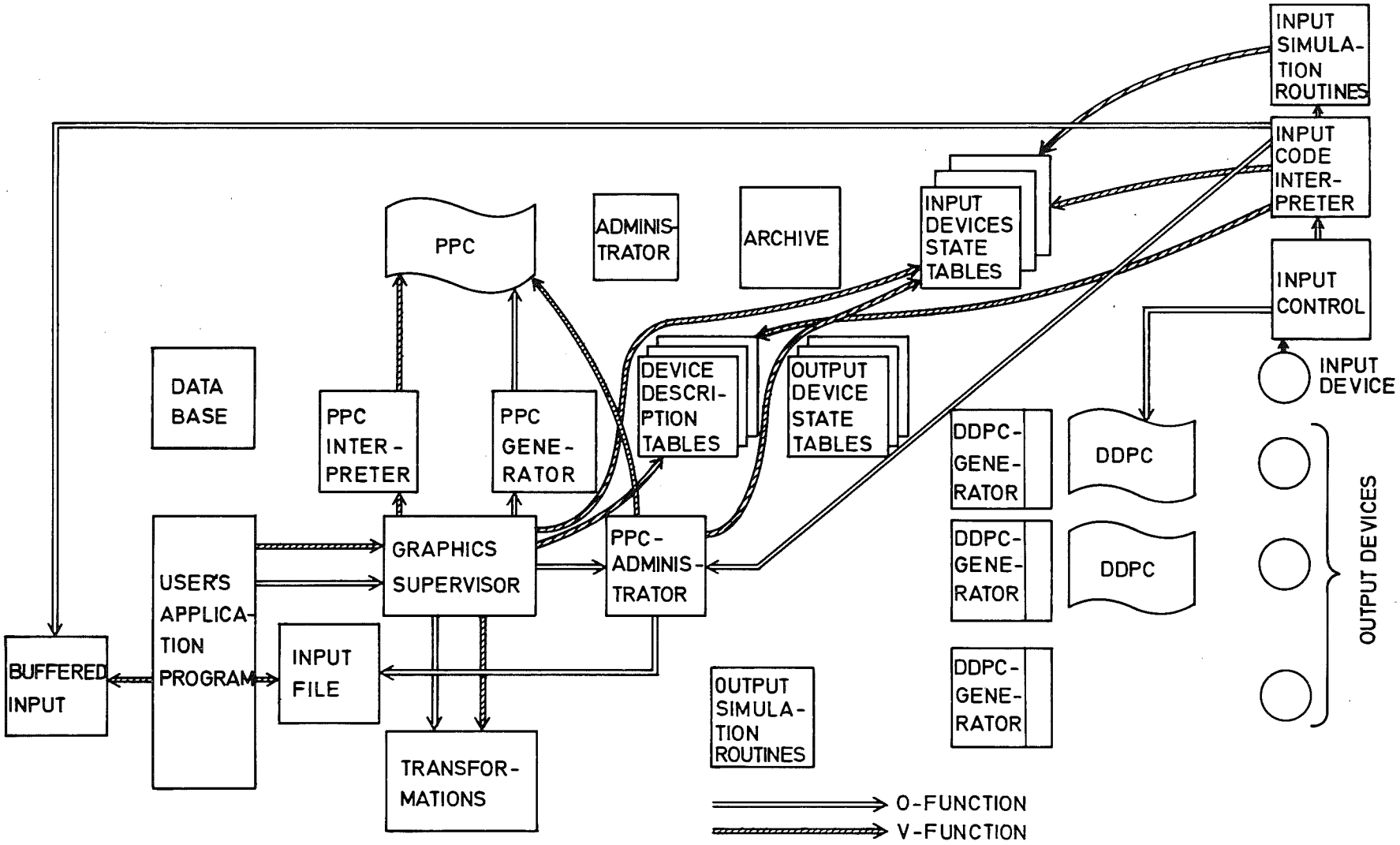


FIG. 4B: INPUT FLOW IN THE GRAPHICS SYSTEM

9. Remarks on implementation

The modules existing in the global system shown in Fig. 4 have different implementations. We want to distinguish in the following table between tasks for the operating system of the graphics system, drivers, parts of the program package (that probably in near future also can be partly realized in hardware or firmware), data, archive and data bases. Everything except the drivers should be written in a high level programming language. In order to permit certain multiprogramming and time-sharing applications, it may be advantageous to implement all parts reentrant.

Tasks for the operating system (OS-Data)	operating system (Drivers)	Program package (or Hardware or Firmware)	Data	Archive	Data base system
device dependent picture code - I/O-device description tables - I/O-device state tables	device code generator - local device administrator - input control - input code	pseudo picture code generator - pseudo picture code interpreter - graphics-supervisor - transformations - I/O-simulation routines PPC-administrator	input file (under the users' program) - buffered input	archive administrator - records	users' data base

10. Input file

For each device there is an input file, in which is stored by

I 1 ... text
I 2 ... a number
I 3 ... a scalar
I 4 ... three scalars (in user coordinates)
I 5 ... lenght, Array (ID)

From the point of view of the users' program we have to distinguish between

(1) interruptable devices

and

(2) permanent readable devices

In the case of interruptable devices the input file, that in most cases is built as a queue, contains in its head the list length and then the corresponding input information. The users' program waits until the length of the queue is greater then zero or it gives the control back immediately if it doesn't contain any message (input information). In the case of permanent readable devices the users' program can always read this input information.

11. Interface to the application program

11.1 Job Control Language

The concept as described in this paper should be realisable not only in a single user environment but also in a time sharing system. In order to permit this without running into deadlock problems all devices which may be used during execution of a perticular users' program may have to be allocated to this program prior to its initiation by means of the job control language. We will illustrate this using an example written in an arbitrary job control language:

... jobstep
... allocate refresh display # 1 to be referred to in the program as DISPLAY 1
... allocate storage tube # 1 to be referred to in the program as DISPLAY 2
... allocate the lightpen of refresh display # 1 to be referred to in the program as KEY 1
... allocate teletype # 18 to be referred to in the program as KEY 2

... allocate the keyboard of refresh display # 2 to be referred to in the program as TEXT 1

... allocate teletype # 18 to be referred to in the program as TEXT 2

Within the users' program the reference names thus defined may be used as device identifications. Using an arbitrary programming language we write an example:

```
DECLARE  (TEXTIN 1, TEXTIN 2) GRAPHIC TEXT INPUT UNITS,
DECLARE  (PLOTTER 1, PLOTTER 2) GRAPHIC OUTPUT UNITS;
DECLARE READ GRAPHIC UNIT VARIABLE;

ALLOCATE PLOTTER 1 TO DISPLAY 1;
ALLOCATE PLOTTER 2 TO DISPLAY 2;
ALLOCATE TEXTIN 1  TO TEXT 1 INTERRUPT (KEY 1);
ALLOCATE TEXTIN 2  TO TEXT 2 INTERRUPT (KEY 2);
GENERATE OUTPUT ON PLOTTER 1 AND PLOTTER 2;
ACTIVATE INTERRUPT FOR TEXTIN 1, TEXTIN 2;
WAIT FOR INTERRUPT SET DEVICE (READ);
DEACTIVATE INTERRUPT FOR TEXTIN 1, TEXTIN 2;

GET MESSAGE FROM DEVICE (READ) INTO (TEXTSTRING);
```

In this example output is produced on two output units, then a text input is expected from one of the two input units. The program waits for the interrupt and then gets the message from the corresponding unit.

11.2 Device simulation

As an example a program may have been written to operate with

a choice input unit,
a identification input unit and
a scalar input unit.

The program was operating in an installation where the devices

a function-keyboard,
a refresh display with lightpen and
an analog signal input

were available.

If we now assume that the program should be implemented in a different installation, the following situations may arise

a) the hardware is equivalent (of the same class as defined in chapter 3)
b) the hardware is more powerful (of a higher class)
c) the hardware is less powerful (of a lower class)

Cases a) and b) do not make problems, since they can at least be handled by simulation routines within the graphics system. Case c) could also in some cases be handled by simulation routines, in other cases, however, it may be necessary or desirable from an users' point of view to give the application program some control of the simulation. For such a technique we propose the following.

We assume then that sample program should operate either

1) with one refresh display only (no scalar input, no keyboard) or
2) with only a storage tube with joystick (tracking cross) and keyboard (no identification device).

The following solutions might be chosen:

a) case 1: a menu is to be set up in a reserved area of the refresh display for simulating the function keyboard and a pseudo dial in an other display area for simulating the analog signal input.

b) case 2: a text input may be used to simulate the choice device and the scalar input, the joystick together with the keyboard interrupt may be used to locate a point on the screen such that the nearest displayed element can be identified to simulate the identification device.

However, other ways of simulation may be more suitable. It may be difficult to provide a standard simulation package for such situation without regarding the actual users' program. It would be better that the simulation be done for each users' program. However, this simulation should be such, that the part of the program which already exists must not be changed, only a prologue and epilogue should be used. This may be realized by the concept of pseudo units. We will illustrate this by using case 2 as an example. The original program may look as follows:

```
DECLARE PICK GRAPHIC PICK UNIT;
        KEY GRAPHIC CHOICE INPUT UNIT,
        SPEED GRAPHIC SCALAR INPUT UNIT;

ALLOCATE PICK TO DISPLAY;
ALLOCATE KEY TO F-BOARD;
ALLOCATE SPEED TO DIAL;
```

In case 2 the units which are available may be

TELETYPE - to which the keyboard is allocated
STORAGE - to which the storage tube is allocated
JOYSTICK - to which the joystick is allocated

The prologue to the above program may take the form:

```
DECLARE (DISPLAY, F-BOARD, DIAL) GRAPHIC UNIT;
DEFINE DISPLAY SIMULATION (JOYSTICK, TELETYPE);
DEFINE F-BOARD SIMULATION (TELETYPE, characters simulating the keys);
DEFINE DIAL    SIMULATION (TELETYPE, data format specification);
```

These simulation routines would have to perform the following tasks:

a) Allocation of pseudo device description tables for use in the subsequent program in place of the original device description tables.
b) Insertion of addresses of special simulation routines with references to the actual device description tables of TELETYPE, STORAGE, JOYSTICK and of specified parameters in the pseudo device description tables.

11.3 Logical functions

These functions form (part of) the interface to the users' program. We want to distinguish between the following classes of logical functions:

(1) Administration routines - Functions supporting type, assignments and identification for the device

(2) Graphics functions - Functions for the definition of graphical objects, for the set up of the coordinate transformations and of the display mode

(3) Dialog functions - Functions for the dialog process, for structur and attribute manipulations.

Some examples of these functions will now be given in a table form.

Administration routines	. Begin of the graphics system . Set graphical device on (with or without simulation) . Reset of a graphical device . Begin of a subpicture . End of a subpicture . Begin of a collection . End of a collection . Begin of a picture . End of a picture . Functions for the definition, administration and call of a choice 1:N . Set the graphical device off . End of the graphics system
Graphics functions	. Scale and coordinate system . Windowing . Definition of attributes . Display . Read data out of the device description tables
Dialog functions	. Delete of a collection . Move of a collection . Real-time move . Zooming . Detail scaling the defined window . Read permanent readable devices . Read interruptable devices

All logical functions together with the necessary conventions have to be made available at the language level of the users program in a suitable syntactical form.

12. Configurations of graphic systems

We shall distinguish between four typical types of configurations of graphic systems /DUNN 73/:

1) Simple I/O-System
2) Buffered I/O-System
3) "Intelligent" Terminal
4) "Intelligent" Satellites

By simple I/O-system we mean the graphics I/O-system that is seen as a remote device, independent of its location. All functions have to be processed by the computer components, to which the I/O-system is connected. The I/O-devices are only loaded and activated to respond to user requests or other actions. The graphics system need only the display of the results of the output function as pictures. This involves the decoding and execution of the graphics commands, the recording and coding of the user input. All the other functions will be performed by the computer.

In a buffered I/O-system the graphics system also functions as an external device. It is mainly used to refresh displays. The graphics system contains a picture buffer, some registers and its own (limited) instruction set. With this equipment it is possible to directly and completely process the I/O-functions in the graphics system. The load on the computer is therefore smaller and the performance, from the users' point of view, can be substantially increased. In this configuration we must consider two different response times. The graphics system, because of the local hardware support responds quickly; the computer may be slow and will depend on the computer load as well as the transmission time. In some applications this may lead to some synchronisation problems.

A third kind of configuration is commonly referred to as "intelligent" terminals. By "intelligent" we mean a certain degree of autonomy or processing ability, which allows some execution of some classes of process without interrupting the computer, to which it is connected. If the "intelligence" can be extended to the storage and user function (data base and users' application program) then we have the so called "intelligent" satellites.

As a graphics control block (sometimes called protocol) we shall mean an information block (commands, programs and data) that makes a communication between computer (host) and the graphics system possible. This protocol defines the interface; on each side of the interface there must be a corresponding protocol interpreter and generator (see Fig. 6).

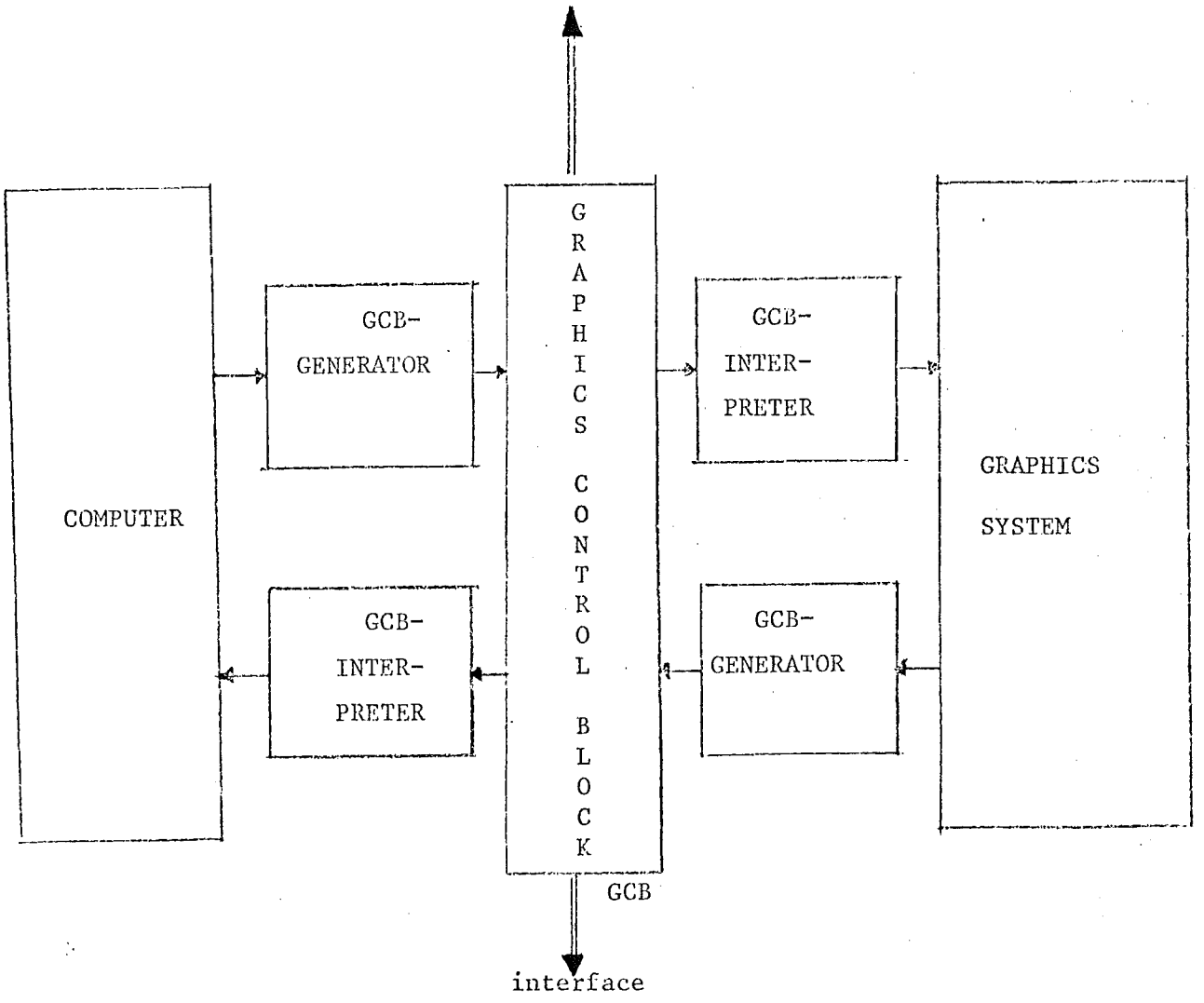


Fig. 6: The graphics control block

The interfaces of the four configurations in our concept are indicated in Fig. 7.

13. Development Methodology

Successful development of graphic systems of the sort discussed in this paper requires a methodological approach to the development of computer systems. Experience with an ad-hoc approach to the development of large systems has shown a need for a more systematic or step by step approach. Such a step-by-step approach requires an ability to precisely specify the interfaces between components and to precisely document the design decisions made at each stage of the development. If we are not able to precisely document the decisions and inter-

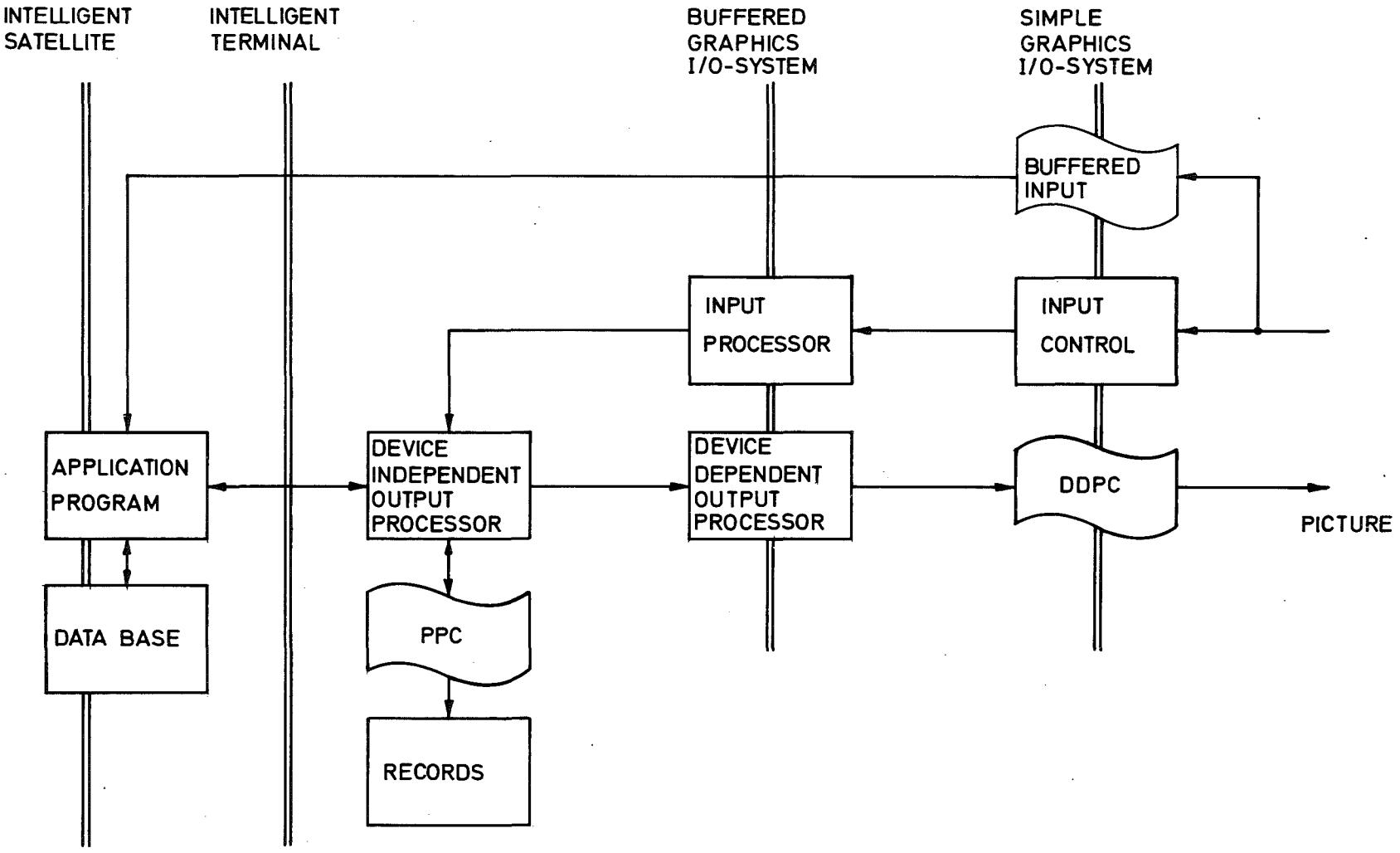


FIG. 7: INTERFACES OF THE DIFFERENT CONFIGURATIONS FOR GRAPHICS SYSTEMS

faces made along the way, our final product may be just another one indistinguishable from those developed with moderate success in a less systematic way.

For these reasons we need a specification language in which to write this documentation. However, "Specification" is used in two quite different ways in the computer system literature. Engineers tend to use it in the sense of "SPECS" meaning a statement of the requirements which a product to produce must meet. Mathematically trained persons often use the word in a more general sense meaning simply any statement which makes the description of an object more specific. For example the specifications for operating systems given in /HOARE 73/ and /BREDT 75/ are specifications in the more general sense, but would not be accepted as specifications in the narrow sense used in /PAR 72/. While both of these papers provide more specific information about the systems being described, they do not provide a statement of the requirements which the systems must fulfill.

In order to avoid terminological confusion we shall in the sequel use "specification" in the engineering (SPECS) sense, and use the phrase "abstract implementation" instead of the more general use of the word "specification". An implementation of a function or component is a program which is written in terms if existing hardware or in a programming language which can and will be translated automatically into a machine level program. In an abstract implementation, we write the program in terms of a language or machine for which there may not exist a practical implementation. By writing this program we make and document certain decisions, but writing this in terms of an abstractly defined machine or programming language we leave certain other design decisions open. Thus the concept of an abstract implementation supports the concept of step-by-step development and provides documentation of the intermediate design decisions. Such an abstract implementation is not however, a specification in the narrow sense because it is not a requirement that the system be implemented in terms if such a machine. The abstract implementation goes beyond the requirement stage as a step towards the implementation.

In summary then, we see two fundamental problems involved in the development of the systems described. (1) We require precise descriptions of the various components so that interface problems can be avoided. The description of the properties of the components visible at the interface are considered requirements that those components can meet and will be called specifications. (2) When one begins to implement such components, it is necessary that internal design decisions be precisely documented. This topic will be handled in the paragraph on abstract implementation. Naturally, larger components will be subdivided into smaller

components. The way that these subcomponents will be used to obtain the larger, is expressed in form of an abstract implementation in terms if those components.

13.1 Component Specification

A methodology and system structure such as that pictured on the previous pages is only realizable if each component is precisely and abstractly specified. The need for a precise specification should be clear to all. Every arrow in Fig. 4 represents an interface between two components which will be developed independently and perhaps changed later. Without a precise description of the interface, major difficulties can appear at the time of system integration and whenever maintenance or improvement is needed. The need for abstract specification is perhaps not as obvious; it causes from two factors:

(1) The structure shown is expected to be shared by many different systems implemented by different manufacturers using somewhat different technologies.

(2) Technological advances and/or environmental demands will lead to the replacement of single components with improved, and/or less expensive versions using new techniques. For example a software pseudo-code-interpreter might be replaced with hardware or micro-programming. For a smooth execution of such changes, it is essential that the interface of the new component be the same as that used for the old one. Thus the specification of the interface, (which states the assumptions, which each side may make about the other) must abstract from any possible differences between various implementations.

It is also important that the specification of the components be parameterized. This is necessary because one cannot expect that all systems will have the same capacity. Were we attempt to require that all the systems have the same capacity, deviations will be made anyway and will be unconstrained. The parameters in an abstract specification indicate the freedom which individual system designers enjoy, but they also indicate the borders within which those systems should stay. Each parameter corresponds to an observable property of a component. It's value will be specified at a later time. For example, it is necessary to include in the specifications a parameter which indicates how many device description tables can be supported by the system. The need for both precision and abstraction means that the specification must be written in a standard formalism developed for this purpose. Because we wish to abstract from implementation details, the ALGOL-like programming languages and hardware description languages often cannot be used.

These reveal exactly the implementation details which we wish to abstract from. For example when one writes two assignment statements, one implies a sequence of events which in some cases one could violate without violating the requirements (e.g. A : = B; C : = D when A,B,C,D are simple variables).

In this paper we will not attempt to specify the exact notation to be used in describing the components. We will, however, describe the basic principles of abstract formal specifications, and outline a methodology for coming to such specifications.

The basic philosophy behind most current approaches to abstract specification is to be found in /PAR 71/ and PAR 72/. A substantially improved notation for specifications has been developed by Guttag /GUT / for use when many objects of the same specification will be created and functions for creating and deleting such objects are available. Both of these methods are discussed in a survey by Liskov and Zilles /ZI,LIS 75/.

13.2 Methodology

The first step in preparation of a formal abstract specification is the identification of all channels of communication between the component and its environment. This means to identify all ways in which the component can give information to its users and all ways in which it can ge information from its users and from those components which it uses. In doing this it must be remembered that these communication channels should be those which will be present for all conceivable implementations. All ways by which one can obtain information from the component will be called V-functions (Value delivering functions); all ways by which one can give information to the component are called O-functions (they Operate on the state of the component).

For example for a simple display there would be a V-function which should indicate part of the state of the screen at a given point (light on, light off, blink, etc.). There would be O-functions corresponding to each key, and input connection. For each of the V-functions an initial value may be specified.

The next step is to describe the effects of all of the O-functions exclusively in terms of the immediate and delayed effects in the values of the V-functions. This is critical: if the effects are described in terms of something other than the V-functions (which include everything available outside of the components) then the danger exists that one would be providing or suggesting information that was biased towards a particular implementation. Thus, we ought not to describe the

effect of pushing a key by indicating that it causes a change in an internal buffer used for refreshing the screen. This buffer might not be present if a plasma screen technology were applied. Instead, we must describe the effect in terms of the V-functions that describe the screen. If the effect on the screen is delayed, (e.g. until a REWRITE-KEY is depressed), then the effect of the change is described in terms of the change that will occur in the future when that action occurs. In cases where information is stored away for future use, and a new picture is put in the display, we can describe the effect of this storage instruction by describing the possible ways (sequences of actions) that will result in the display being restored to its previous state.

If, for example, a picture stack is available, then the effect of the commands STACK and RESTORE can be described by indicating that

(1) After executing STACK the screen is blank, and
(2) The sequence of commands STACK; RESTORE leaves the state of the system unchanged.

It is important to note that no mention of the internal memory (which must be present) is made. It is implied by the existence of a sequence that will restore the screen, but it is never mentioned explicitly. This indirect or abstract form of specification is the only certain way of avoiding a bias towards a certain design or technology.

13.3 Notations

In the original work by Parnas a very direct notation was used. Each function was described in terms of its possible values (for V-functions), and its effects on other observable functions (for O-functions). In order to describe delayed effects, "hidden functions" were introduced. These hidden functions contained those aspects of the device's state which would influence future behavior. It is now felt that this reliance on hidden functions was an error. Although it is theoretically possible to avoid bias towards a certain implementation, it is not easy.

In a more recent working report, Parnas and Handzel /PA,HA 75/ have extended the notation in order to remove the hidden functions. Two notational tricks were introduced:

a) History characterizing sets

In the specification, functions which described the history of the object (which actions have been executed on it) are defined. The effect of each O-function on the history set is defined and the values of V-functions are defined in terms of the history sets. The history sets are always defined so that only the minimum (or strictly relevant) history about the object is maintained.

b) Canonical sequences

Instead of the history sets, a set of identity preserving sequences is given. Each of these sequences preserves the state of the device, thus any combination of them preserves the state of the device. As shown in the example above, this implies the information that must maintained internally, but does not provide any suggestions of its form or representation.

John Guttag in his dissertation (apparently working on the basis of a proposal by Zilles) has extended the canonial sequences by introducing an algebraic approach to the specification. The various components are assumed to define types of variables. Every O-function transforms a variable to a new value in the space appropriate to that type. The notation assumes that O-functions have values which are variables of that type. This allows one to refer to the whole object without referring to its individual components and/or possible internal information.

In our example above, one would regard the functions STACK and RESTORE as having values which are displays. (Thus "RESTORE(B)" (where B is a display) is itself a display, but not just the visible part of the information - the complete state. The canonical sequence used in our example could then be written: RESTORE(STACK(B)) = B. For simple examples, such as stacks, the advantages of this notation are minor, but as shown in Guttag's theses, it becomes quite advantageous for more complex examples. (Note: in comparing the size of Parnas's specifications with those of Guttag, it is important to note, that Parnas's specifications include some information about error treatment and initial values which is not present in the Guttag specifications).

13.4 Abstract implementation

We now sketch a solution to the second of the two problems mentioned in the introduction to paragraph 13: The adequate documentation of the internal design decisions for modules /NEES 76,72/. One may say that to specify a module means

to solidify its exostructure (input/output constraints), at the same time leaving its endostructure (the algorithm) malleable. The boundary between exo- and endostructure coincides roughly with the line which is commonly drawn between declarative and procedural definitions of programs. Hence one would reason, that specification being available, the next step should directly lead to compilable code. The distance between specification and machine code may however be so grear, that the insertion of intermediate steps is recommendable. This will be realized, if one admits that to design the endostructure of a module means constructively to define the transformation of one class of data structures (representing a type of variable) into another, where the one class is given by the O-function, the other one by the V-function of the specification. It will be useful in many cases, to state the cyclic or recursive steps of that transformation, without to drag along e.g. a doublelinking of records. The coding of such a transformation in a programming language still to be characterized, we will call an abstract implementation /GUT 75/.

For such a purpose almost all existing programming languages are either too specialized (LISP) or too voluminous (ALGOL 68). One language which qualifies itself however, is GEDANKEN, which Reynolds has introduced in two papers /REY 69,70/. In GEDANKEN assignment and indirect addressing are formalized by the concept of reference. The values a reference can possess are reference, integers, booleans, characters, functions, and label values (the latter essentially being states of the module considered). Every data structure is a function. Some data structures may be implicit, i.e. they are to be defined by an algorithm for computing or accessing their components. In this way hardware modules can be directly modelled into GEDANKEN-data-structures.

The usefulness of GEDANKEN for abstract implementations in computer graphics shall now be demonstrated by coding an algorithm for the transformation of user-oriented picture-structures into pseudo-picture-code-sequences, i.e. device-independent structured display files. In GEDANKEN any sequences $s = (x_1,\ldots,x_n)$ of values is a function, where s LL = 1, s UL = n for two special atoms LL (Lower Limit) and UL (Upper Limit) and $s\ i = x_i$ for $1 \leqq i \leqq n$. A record (in the sense of Hoare) is given by a function, which is defined in a set of atoms, the field names of the record. The set of records is subdivided into record classes, where a record class C is given by an expression $(CLASS,C,(f_1,v_1),\ldots,(f_n,v_n))$ where $f_1,\ldots,f_n$ are field names of the records of C. Every record r owns a special name TYPE such

that r TYPE = C. If v_i denotes a set s, then r f_i s. If v_j is of the form SEQ, v where v denotes a set s, then r f_i is a sequence the components of which are members of s. Another record class definition (UNION,v_o,v_1,...v_n) denotes by v_o the union of the sets denoted by v_1,...v_n. Several record classes can be combined into a sequence which is called an abstract syntax /REY 69/.

Fig. 8 gives an abstract syntax for picture structures. This syntax is roughly equivalent to the BNF-syntax given in chapter 5. With the definition of the concept of a record class in mind, the meaning of an abstract syntax can very easily be translated into common language:
"A picture has Subpictures (which is sequence of Subpicture) and Collections (which is Collection). A Collection has ID (which is Number and Attributes ...."
This abstract syntax APPC is now used to abstractly-implement a function PPC.Sequence (the ".." being used as a delimiter) which is exactly a pseudo code generator in the sense of chapter 6 (fig. 9). The structure of fig. 10 e.g. will by PPC.Sequence be mapped onto the PPC-code shown by fig. 11. The abstract syntax APPC does not point directly into the body of the function PPC.Sequence because it is bound to a free variable in the body of the function Test (top of Fig. 8 /REY 69/).

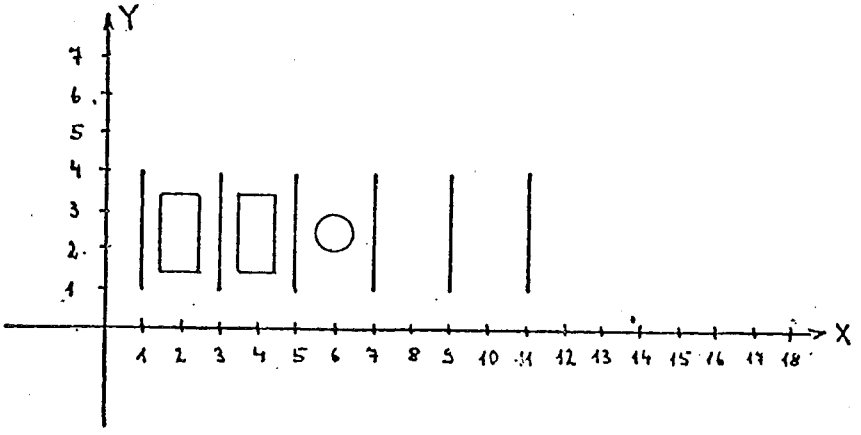


Fig. 10: Picture described by Fig. 8

The abstract syntax APPC defines the structure of any datum which represents a certain picture out of a given class of pictures. Thus e.g. lines 2 to 4 say, that a Picture is given by declarations of Subpictures and by a structure called Collections which is a Collection. If one row considers lines 5 and 8, one will realize, that a Collection includes a sequence of Entities, where an Entity may be a Primitive or a Collection again (notize line 9). Because a Primitive can have a UNIVERSAL which can be everything, a Primitive can for example be a Subpicture.

```
Def. of the function Test(x,C):
If x is a primitive datum (e.g. a number)
or a record, and C is a record class,
then T(x,C) = x∈C


An abstract syntax for the pseudo picture code (PPC)

 1  APPC = (
 2  (CLASS Picture,
 3     (Subpictures, SEQ, Subpicture),
 4     (Collections, Collection)),

 5  (CLASS, Collection,
 6     (Id, Number),
 7     (Attributes, SEQ, Attribute),
 8     (Entities, SEQ, Entity)),

 9  (UNION, Entity, Primitive, Collection),

10  (CLASS, Primitive,
11     (Prim, UNIVERSAL),
12     (Params, SEQ, CHARCLASS)),

13  (CLASS, Subpicture,
14     (Id, SEQ, CHARCLASS),
15     (Attributes, SEQ, Attribute),
16     (Things, SEQ, Thing)),

17  (CLASS, Thing,
18     (Prim, UNIVERSAL),
19     (Params, SEQ, NUMBERCLASS)),
20  )
```

Fig. 8: Description of Fig. 10

The structure of the procedure PPC.Sequence just inverts the structure of the syntax APPC (Fig. 8): Where APPC defines the different components of the data type Picture top down, the corresponding procedure PPC.Sequence is a bottom-up-construction of subprocedures corresponding to the different components of Picture. Thus e.g. the subclass Thing at the bottom of Fig. 8 is mapped onto the leading subprocedure Do.Thing in Fig. 9. Besides declaration of subprocedures Fig. 9 contains just one line of code: A call to the top-level subprocedure Do.Picture (line 29).

```
A function which converts PPC-data-structures p into
PPC-sequences, where APPC is the abstract syntax of the p

 1  PPC.Sequence (p,APPC) is
 2  (Do.Thing(x) is (Test(x,Thing) ->
 3     Cons(x Prim, x Params), T -> Error);
 4   Do.Things(x) is (Is.Empty x -> x, T ->
 5     Cons(Do.Thing(x 1), Do.Things(Tail x)));
 6   Do.Subpicture(x) is (Test(x,Subpicture) ->
 7     Cons(x Id, Cons("ATTRIB",
 8       Conc(x Attributes,
 9         Aug(Do.Things(x Things),"ENDSUB")))), T -> Error);
10   Do.Subpictures(x) is (Is.Empty x -> x, T ->
11     Cons(Do.Subpicture(x 1), Do.Subpictures(Tail x)));
12   Do.Primitive(x) is
13     Cons((x = "SUBPICTURE" -> "PSHJMP", T -> "IMPLIC"),
14        Cons(x Sort, x Params));
15   Do.Entity(x) is (Test(x,Entity) ->
16     (Test(x,Primitive) -> Do. Primitive x,
17      T -> Do.Collection x));
18   Do.Entities(x) is (Is.Empty x -> x, T ->
19     Cons(Do.Entity(x 1), Do.Entities(Tail x)));
20   Do.Collection(x) is (Test(x,Collection) ->
21     Cons(x Id, Cons("COLLEC",
22        Conc(x Attributes, Aug(Do.Entities(x Entities),
23           "ENDCOL")))), T -> Error);
24   Do.Picture(x) is (Test(x,Picture) ->
25     Cons("PICTUR",
26        Conc(Do.Subpictures(x Subpictures),
27           Aug(Do.Collection(x Collection), "ENDPIC"))),
28     T -> Error);
29   Do.Picture (p))
```

Fig. 9: Convection von PPC-data-structures into PPC-sequences

This figure tries to explain the idea of abstract implementation by a compilation of the data structure corresponding to Fig. 10 into a PPC-sequence. This will easily be grasped, when one realizes, that the data structure for Fig. 10 will start with a record Picture, which has just one Subpicture, which is a rectangle. Hence the evaluation of Do.Picture in lines 24 to 28 of Fig. 9 can do nothing else as to generate the string "PICTURE" which appears as the head of the operations and data listed in the last column of Fig. 11. In this way the compilation will proceed, generating for example "RECTAN". The compilation finally stops after generating the closing symbol "ENDPIC" by line 27 of Fig. 9.

Comment	Begin- or End-point	Adress	Operation or datum
Head of picture	A 1	1	PICTUR
Head of subpicture	A 2	2	SUBPIC
Subpicture Id		3	RECTAN
Implicit		4	IMPLIC
reference		5	i_{Path}
Parameters		6	Ø,Ø,1,Ø,1,2,Ø,2,Ø,Ø
End of subpicture	E 2	16	ENDSUB
Head of coll. Ø	A 3	17	COLLEC
Id of collect. Ø		18	Ø
Attri-		19	ATTRIB
butes		2Ø	Ø
Implicit		21	IMPLIC
reference		22	i_{Grid}
Parameters		23	1,1,3,2,6
Head of coll. 1	A 4	28	COLLEC
Id of collect. 1		29	1
Subpicture		3Ø	PSHJMP
call		31	2
Parameters		32	1.5,1.5
Subpicture		34	PSHJMP
call		35	2
Parameters		36	3.5,1.5
End of coll. 1	E 4	38	ENDCOL
Implicit		39	IMPLIC
reference		4Ø	i_{Circle}
Parameters		41	6,3,Ø.5
End of coll. Ø	E 3	44	ENDCOL
End of picture	E 1	45	ENDPIC

Fig. 11: Compilation of the data structure corresponding to Fig. 10 into a PPC-sequence

14. Acknowledgement

The authors are gratefully indebted to R. Eckert and M. Gonauser who contributed with their work and many valuable discussions to the contents of this position paper.

15. Bibliography

/ BREDT 75 /

A.R. Saxena and T.H. Bredt

A structured specification of hierarchical operating system

SIGPLAN Notices, Vol. 10, No.6, June 1975, pp. 310-318

/ COT 72 /

Ira W. Cotton

Network graphic attention handling

Online 72 Conference Proceedings, Uxbridge, England, 4-7 Sept. 1972, Vol. 2, pp. 465-490

/ DUNN 73 /

R.M. Dunn

Computer Graphics: Capabilities, Costs and Usefulness;

Quarterly Report of SIGGRAPH-ACM
Vol.7, No.1, 1973, pp. 1-29

/ ECK 75 /

R. Eckert

Geräteunabhängige graphische Software; Probleme und Lösungsmöglichkeiten

(Device independent graphical software; Problems and possible solutions)

Bericht Nr. GDV 75 - 4
FB Informatik, TH Darmstadt (in German)

/ ECK 76 /

R. Eckert

Functional aspects and specification of graphics systems

Bericht Nr. GDV 76 - 2
FB Informatik, TH Darmstadt

/ENC, TRA 73/

J. Encarnacao and U. Trambacz

The design and organisation of a general-purpose display processor

Proceedings of the "Journées Graphiques 1973" Colloques AFCET/IRIA, Paris, Déc. 1973, pp. 37-50

/ ENC 74 /

J. Encarnacao

Möglichkeiten zur interaktiven graphischen Datenverarbeitung in Time-Sharing-Systemen und ihre Leistungsabschätzung

(Practically of interactive computer graphics in time sharing systems and performance estimation)

Lecture Notes of the German Chapter of the ACM I-1974, pp. 1-17 (in German)

/ENC, ECK 76/

J. Encarnacao, R. Eckert

Bemühungen und Möglichkeiten bei der Begriffsbildung und Normung graphischer Systeme

(Activities and possibilities for the standardization of graphics systems)

Lecture Notes of the German Chapter of the ACM II/März 76 (in German)

/ FNI 75 /

FNI ad hoc Komitee beim Beirat "Verarbeitung graphischer Daten" Unterausschuß "Graphische Software"

Abschlußpapier zur Erarbeitung von Vorgehensalternativen für die Normung graphischer Software;

(Final report on alternative procedures for the standardization of graphical software)

September 1975 (in German) (unpublished)

/GI,ENC,SAV 75/

W.K. Giloi, J. Encarnacao and S. Savitt

Interactive Graphics on Intelligent Terminals in a Time-Sharing-Environment

Acta-Informatica 5, 257-271 (1975)

/ GBM 75 /

Graphische Methodenbank - Benutzerhandbuch

(GMB Users' handbook)

Siemens AG, München, FL SYST 152; 1975 (in German)

/ GRA 75 /

Interaktives graphisches System GRAFSY Systembeschreibung

AEG-Telefunken, A 510.6.6/0775

/ GUT /

J.V. Guttag

The Specification and Application to Programming of Abstract Types

Technical Report CSR6-59, University of Toronto, Canada (Sept. 1975)

/HOARE 73/

C.A.R. Hoare

A structured paging system

Computer Journal 16,3 (Aug. 1973),pp. 209-215

/HO,GE,GO 75/

E. Hörbst, G. Geitz and M. Gonauser

An integrated System which provides the use of various graphic terminals

Interactive Systems, London, 1975, pp. 45-56

/ NEES 72 /

G. Nees

Strukturunterschiede bei graphischen Programmiersprachen

(Different structures in graphical programming languages)

Lecture Notes of the German Chapter of the ACM III/Dezember 1972, pp. 65-86 (in German)

/ NEES 75 /

G. Nees

Toward an unified approach to the specification of computer graphics software

Lecture Notes of the German Chapter of the ACM I/April 1976

/NEW,SPR 74 /

W.M. Newman and R.F. Sproul

An Approach to Graphics System Design

Proceedings of the IEEE (April 74),pp. 471-483

/ PAR 71 /

D.L. Parnas

Information Distribution Aspects of Design Methodology

Proceedings of 1971 IFIP Congress, Ljubljana, 23-28. Aug. 1971, North-Holland Publishing Co. (1972), pp. 339-344

/ PAR 72 /

D.L. Parnas

A Technique for Software Module Specification with Examples

Communications of the ACM, May 1972, pp.330-336

/ PA, HA 75 /

D.L. Parnas and G. Handzel

More on Specification Techniques for Software Modules

Forschungsbericht BS I 75/1, Technische Hochschule Darmstadt 1975

/ PHIL 75 /

PHILDIG - General Description

N.V. Philips, Eindhoven
The Netherlands
Publ.No: UDP-DSA-SCA/75/001/DE/CN

/ REY 69 /

J. Reynolds

GEDANKEN - a simple typeless language which permits functional data structures and coroutines.

ANL-7621, Argonne Nat. Lab., Argonne Ill., 1969

/ REY 70/

J. Reynolds

GEDANKEN - a simple typeless language based on the principle of completeness and the reference concept.

Communications of the ACM 13, 308-319, 1970

/ SAN 75 /

T.L. Sancha

Some Notes on Software Standardization

Computer Aided Design Centre
Cambridge, England (October 75)

/SCHL et.al. /

G.Edler, E.G.Schlechtendahl, U. Schumann, R.Schuster

Design Principles of the GRAPHIC System

Bericht KFK 1722 (1973)
Gesellschaft für Kernforschung, Karlsruhe

/ SCHU /

R. Schuster

System und Sprache zur Behandlung graphischer Information im rechnergestützten Entwurf

Bericht KFK 2305 (Aug. 1976), Gesellschaft für Kernforschung, Karlsruhe

/ SPR,THO 74 /

R.F. Sproull, E.L. Thomas

A Network Graphic Protocol (Aug. 74)

Xerox Palo Alto Research Centre

/ ZI, LIS 75 /

S. Zilles and B. Liskov

Specification Techniques for Data Abstractions

I.E.E.E. Transactions on Software Engineering, vol. 1, No. 1, 1975, pp. 7-19

Authors' addresses

(1) Encarnacao, J.
Fachbereich Informatik
FG Graphische Datenverarbeitung
Technische Hochschule Darmstadt
Steubenplatz 12

6100 Darmstadt, W.-Germany

(2) Fink, B.
Philips Forschungslaboratorium Hamburg GmbH
Vogt-Kölln-Straße 30

2000 Hamburg 54, W.-Germany

(3) Hörbst, E.
Siemens AG
Zentralforschungslaboratorium
Postfach 700077

8000 München 70, W.-Germany

(4) Konkart, R.
AEG-Telefunken
Bücklestraße 1-5

7750 Konstanz, W.-Germany

(5) Nees, G.
Siemens AG
E 549
Werner-von-Siemens-Straße 50

8520 Erlangen 2, W.-Germany

(6) Parnas, D.L.
Fachbereich Informatik
FG Betriebssysteme I
Technische Hochschule Darmstadt
Steubenplatz 12

6100 Darmstadt, W.-Germany

(7) Schlechtendahl, E.H.
Institut für Reaktorenentwicklung
Ges. für Kernforschung mbH
Weberstraße 5

7500 Karlsruhe, W.-Germany