ОБЪЕДИНЕННЫЙ
ИНСТИТУТ
ЯДЕРНЫХ
ИССЛЕДОВАНИЙ

ДУБНА

E11 - 10883

A.A.Karlov

SOFTWARE FOR GRAPHIC DISPLAY SYSTEMS

1977

E11 - 10883

A.A.Karlov

SOFTWARE FOR GRAPHIC DISPLAY SYSTEMS

Submitted to the Second Summer School on Computing Techniques in Physics (20 – 28 August, 1977, ČSSR).

Карлов А.А. **E11 - 10883**

Математическое обеспечение графических дисплейных систем

Рассматриваются вопросы математического обеспечения графических диалоговых систем: выбор способа реализации, структура библиотеки дисплейных подпрограмм, организация диалога с многоуровневой структурой, аппарат диалоговых (глобальных) переменных и т.п.

Приведены примеры программирования на языке ФОРТРАН.

Работа выполнена в Лаборатории вычислительной техники и автоматизации ОИЯИ.

Препринт Объединенного института ядерных исследований. Дубна 1977

Karlov A. **E11 - 10883**

Software for Graphic Display Systems

The software problems for graphical display systems are considered: the implementation, the structure of display subroutine library, the organization of a dialogue with multilevel structure and the extension of a high-level programming language (FORTRAN) by the dialogue variables.

Some examples of programming technique for FORTRAN are given.

Preprint of the Joint Institute for Nuclear Research. Dubna 1977

1. Introduction

A display software can be classified into a general (base) software, which is independent of the particular problem and specifies general-purpose logic capabilities of a display system, and a special (problem-oriented) one to be developed for the solution of some special problem or some class of problems.

Difficulties arising in the programming for graphic diplays are one of the reasons preventing the introduction of these devices. The difficulties are relating to the development of just the base software, which has to provide the user with flexible and natural for usage facilities for the graphic data handling and the interaction with a picture on a display screen and user program.

2. Implementation

To develop a general-purpose software for graphic displays three methods can be used:

I. The development of a special graphic language.

2. The extension of the existing languages.

3. The creation of a display subroutine library.

A special graphic language. The development of a graphic language (e.g. refs./1,2,3,4/) like any special development has the advantage of high efficiency in the solution of problems of a graphic data processing. Along with this important advantage, however, there are essential disadvantages, namely, quite heavy expenses for the development of the language itself, those for the user training and the addition of a new language into the running operating system and the difficulties of the graphic language extensions to be required in the future.

The development of a special graphic language is justified, where the expenses for the solution of the main problem are sufficiently large, so that the difficulties related to the development of a special graphic language and to its introduction, are compensated by the efficiency to be achieved.

The extension of existing high-level languages. In this case the special graphic operators are added to the language /5,6/. Therefore, in practice the user has only "to improve his knowledge" of a known language. It might be well to point out the difficulties to be caused by some modification or extension of the language. This is, firstly, the modification of a translator, which results in the increase of the translator size, that appears to be unjustified, when not all the graphic functions are used, or the development of a preprocessor to transform the graphic operators into the operators of the main language before the program translation.

A certain difficulty of both these approaches is a considerable amount of work of the system programmers.

Display subroutine library. The creation of a display subroutine library available of high-level languages allows one to avoid a lot of problems, arising in the development of a special graphic language or in the extension of high-level languages. The obvious advantages of such an approach are as follows:

- the simplicity of a user "training" (a known mechanism for calling subroutines and functions, e.g. in FORTRAN by the operator CALL);
- the development of a large part of the library can be performed by non-system programmers;
- the simplicity of the extension if necessary by the addition of new subroutines to the library;
- time required for the software development is essentially reduced;
- the possibility of a rather simple adoptation of the software created for various computers and displays.

Therefore, the development of the display library is now the basic approach for the designing of the display software (e.g. refs. /7,8,10,11,12/).

3. Some principles of the designing of the display library

The basic principles of the designing of the display library are selected from the point of view of the maximum conveniences for users and must provide for the following possibilities.

The availability of all the facilities to handle graphic data and to organize a dialogue through a high-level language. It should not be necessary for the user to have the detailed information about the operation of the display equipment, computer channels and communication circuits. All the facilities for the construction and modification of the pictures and also for the operation with a light pen and a keyboard have to be available for the user through a high-level language (or through an assembly language, when he wants to use it for the purpose of effective programming).

The possibility of the execution of the same functions by various means and a minimum dependence of the display subroutine from each other. Such an opportunity allows the programmer, in any particular case, to select the optimal, from his point of view, set of display subroutines. There must be the basic set of subroutines, which is always used and a large set of additional subroutines, which can be used, depending on the requirements of the application program and the skill of the programmer.

A possibility of constructing hierarchy of graphic objects. Due to this possibility, the creation of composed objects is assumed, when an object can be included as a part of a more complex object and the latter, in its turn, be a part of another object and so on. An object being formed at relative coordinates can be included into several composed objects. For this case, to display several identical objects, it is sufficient to have only one copy in the computer memory.

A flexible mechanism of memory allocation. To allocate graphic and auxiliary information, i.e. to organize the graphic data structure, the memory is required, whose size depends on the amount of graphic data in a user program. Therefore, if there is no the automatic memory allocation for a given operating computer system, the user should himself be responsible for the memory allocation. It is important, that this task would be maximum easy for him. For example, it would be sufficient to define the buffer to be used for the graphic data and auxiliary information. Service display subroutines have to be responsible for the utilization of the allocated memory.

4. Classification of display subroutines

From the functional point of view all the display subroutines can be divided into three classes:

I.Subroutines-generators of the graphic command and data for the display unit. The picture on the screen can consist of both the simplest graphic components, such as dots, vectors, texts and others (generated by low-level subroutines-generators), and more complex pictures, for instance, coordinate axes, equipotential lines and so on (generated by high-level subroutines-generators)

2.Administrative subroutines providing the memory allocation for graphic data and also executing different operations at the picture components (adding, removal, replacement and others). This class may include also the auxiliary subroutines, which allow one to estimate the memory allocation, to print output listings of the graphic data, internal display variables and the system control tables, which is quite useful for the debugging of new display subroutines.

3. Control subroutines providing independently of the execution of the user routine, the output of a picture on the display screen and the operation of input devices provided for the man-machine-interaction (a light pen, a keyboard and others) and the dialogue organization as well.

5. The example of the display library

Consider the example of the implementation of the principles mentioned above.

A software/11/ for the graphic display SIGDA on the minicomputer M-6000, has been developed at JINR. It has been realized as a subroutine package available for a user through the M-6000 assembler or FORTRAN and added as the extension for a standard subroutine library. At present the display library consists of about 100 subroutines.

Graphic objects on the display screen are constructed of the primitive items (dots, vectors, a text, circles and arcs), which are generated by the display hardware, when graphic instructions are entered from the computer. Any combination of the primitive items from a dot to a complex picture can be defined as an object.

In a computer memory the object occupies a set of subsequent words with a graphic and control information which have been assigned a name. To establish a correspondence between the names of the object and the actual storage location, the graphic object table (GOT) is provided (Fig. 1). The size of this table is selected by the user and depends on an assumed maximum number of objects. For each currently existing object, there is an entry of three words which contains the initial address of the object in the storage and also the length and the name of the object. Besides, a special mask bit determines if a current object should be displayed on the screen (active object) or not (masked object).

When a new object has to be added, service subroutines search for a free place in the GOT, form a new entry and insert it into the table. The removal of the object is performed by clearing of the appropriate entry in the object table.

A regeneration subroutine (a display driver) is activated, when the first active object appears in the GOT and is then called with regeneration frequency (normally 50 Hz) via a computer interrupt system. It is responsible for scanning of the object table and the organization of output for the active objects on the screen.

The objects for SIGDA display can be primitive and composed.

The primitive and composed objects have the same representation in the GOT. A reference to the subobject consists of two words: the first is a special instruction "control", the second is the name of subobject. Such a structure of the reference permits to refer to the subobjects which do not yet exist and will be generated some later.

The object location in the memory is determined by the fact, which area was ordered by the user as a so-called current buffer before the procedure of the object generation. When ordering the current buffer, the user specifies the initial address and the length of the area, which he allocates for this purpose.

The following calls to subroutines-generators will place graphic information in the current buffer consequently beginning from the first free word.

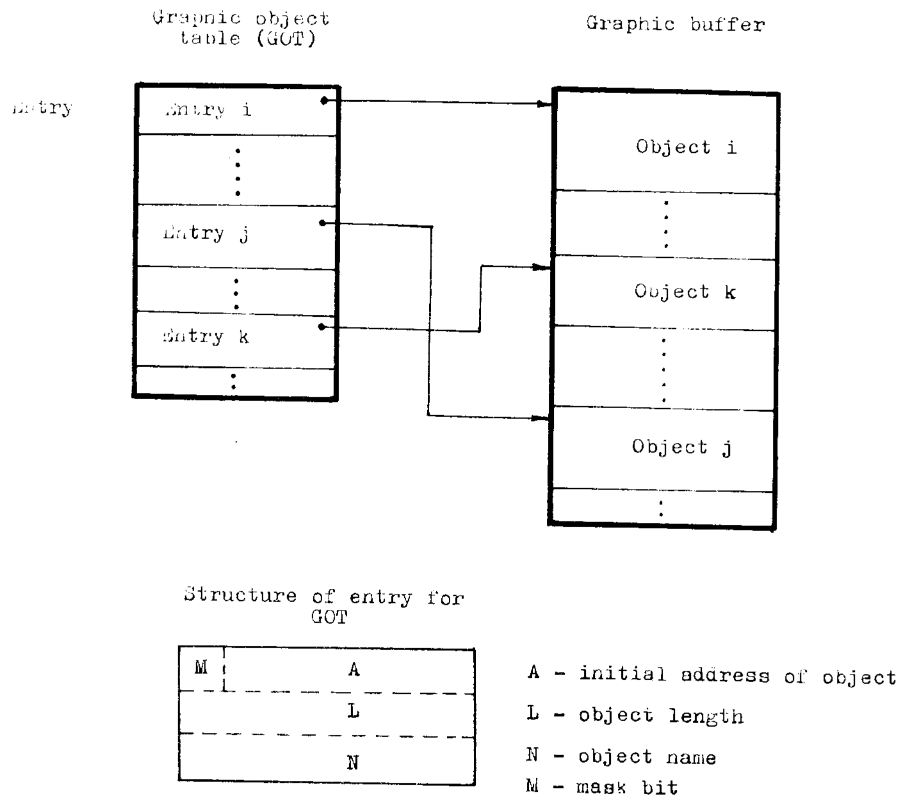

Fig. 1. The structure of the graphic object table.

When designing the library for the SIGDA graphic display, a great attention was paid to the software for the man-machine interaction facilities: a light pen and a display keyboard. A light pen can be used both in the indication mode, i.e. for the identification of objects existing on the screen, and in the tracing mode to construct new objects. In the indication mode the user has an access to the coordinates of a point indicated by a light pen and the name of the object, which this point belongs to.

The process of tracing the light pen is performed by means of a special marker simultaneously and independently on the user program execution. At any time the user program has the access to the coordinates of the current position of the marker on the screen via variables specified by the user. Minimum user efforts are required to specify the tracing type (without drawing or with that). When drawing the user has to indicate only a minimum step of registrating the coordinates of the intermediate marker position and the type of approximation (by dots, vectors etc.). A possibility to move the objects existing on the screen by means of a tracing marker is also provided.

A set of keyboard subroutines, which independently on the execution of the main program can input and display characters from the keyboard and perform some editing procedures, is developed.

Thus, all the possibilities of the display library are available for the user both through the M-6000 assembler and FORTRAN.

As an example, the procedures of the memory allocation and the graphic object generation are given in Fig. 2.

```
       :
CALL DSTO(TAB,LT)                  Definition of the graphic object
                                   table
CALL DSBUF(BUF,LB)                 Definition of the graphic buffer
       :
CALL DSPTA (IX,IY)                 Point generation
CALL DSMVA(IX1,IY1,IX2,IY2 MODE)   Vector generation
CALL DSTXA(KX,KY,TEXT,LT)           Text generation
CALL DSADD(NAME)                    Graphic object declaration
       :

       Generation
    of the other graphic
         objects
       (if any )
       :

CALL DSLPN(LPF, LNAME,LX,LY)       Light pen call
5 LF(LPF)7,5,7                     Waiting of the light pen hit
       :                           (if required)
```

Fig. 2. The example of the graphic object generation

6. The dialogue organization

6.1. Terminology

To organize the man-machine dialogue various devices can be used by the user: a keyboard, a light pen, functional keys and so on. By means of them he enters symbolic and graphic information or some special codes and influences, in such a way, on his program and the computer system.

For simplicity, a block of information entered by the user as an inquiry answer from a computer, or on his own initiative, is called as a message. Thus, in this case, the line of symbols from the keyboard, the information from a light pen when hiting the graphic object on the screen and the code of a function button can be considered as a message. In the general case, the message consists of an order, which indicates the action to be performed and parameters which have to be used.

The state of the dialogue is described by many factors: an operational situation in the computing system, the state of a user program and so on. Here, the state of a dialogue at some moment is considered in the limited sense as a set of messages allowed at this moment. The combination of all the states, which are available when running the user program together with the conditions of jumping from one state to another, forms the structure of the dialogue of a current program. From the point of view of programming, the stay in some state is a loop of waiting a user message. If the jump from one state to another is completed by return to the previous loop, we say, that we stay at the same level. If a jump to another loop has a place, we say about the jump to another level (that may be the jump to a more "deep" level with respect to original one or the return to the previous level).

6.2. Message processing

In special dialogue systems (for example, when designing a printed board, analyzing the electric circuits and so on) the structure of a dialogue and the appropriate set of commands (messages) available for the user are selected by the system designer and are fixed. In the general case, the user of the dialogue system should be allowed to select by himself the set of messages convenient from his point of view to contact his program.

In principle, to organize the processing of messages it would be sufficient to provide the user with some set of subroutines of input and unpacking of messages and to rest for him the analysis of these messages in accordance with the algorithm of his program. Besides, programming required for such an analysis, in this case the dialogue structure becomes difficult to observe and to correct.

For example, the part of the user program in FORTRAN, which is responsible for the organization of operation with a light pen (light buttons) is given in Fig. 3.

```
                    :
      CALL DSLPN(LPF,LNAME,LX,LY)    Light pen call
                    :
    5 IF(LPF) 7,5,7                  Loop to wait
      GO TO 7, IF LIGHT PEN HIT      a light pen hit
                    :
    7 IF(LNAME.EQ4HEDIT) GO TO 10    Analysis of
      IF(LNAME.EQ.3HRUN) GO TO 20    the light button
      IF(LNAME.EQ.6HRESULT) GO TO 30 selected
                    :
C
C     GO TO 10 TO EDIT INITIAL DATA  Section
C                                    of initial
   10 CONTINUE                       data editing
C
C     GO TO 20 TO CALCULATE          Section
C
   20 CONTINUE                       of calculation
                    :
C
C     GO TO 30 TO DISPLAY RESULTS    Section
C                                    of the analysis of
   30 CONTINUE                       the results calculated
                    :
```

Fig. 3. The example of decision making for light buttons in the user program.

The user has to analyze what graphic object was selected, and he has to program the jump to the proper section of his program or to the separate subroutine. When the algorithm of the dialogue is complete enough, the programming of the dialogue structure is a a labour-consuming procedure, including a lot of tests of various conditions in many subroutines of the user program. In other words, the structure of a dialogue is distributed to a number of subroutines and in order to implement it, a great accuracy and patience, which are not related to the solution of an application program, are required from the user.

Therefore, programming means, which allow the user to be free from the programming of standard procedures of the message processing (the request for input, analysis of the message, the organization of jumps to executive subroutines and so on.) must be provided in a modern dialogue system (for example, /9,13/). It is natural only to demand from the user the description of the

dialogue structure and the presence of executive program related to the specific algorithm of his job.

6.3. Definition of the dialogue structure

To describe the dialogue structure, the user depending on the requirements of his job, selects the number of levels in the structure, the number of states at each level, indicates for each state the correspondence between messages (which he selects by himself) and executive subroutines. A logic correspondence between every message and the executive subroutine to be called is established by the user through a special table, subroutine link table (SLT). Thus, every state is specified by its own SLT.

The dialogue structure can be defined by the programmer statically or dynamically. In the statical assignment it is described at the beginning of the program run (for example, by means of a separate subroutine). In the dynamical assignment a new set of messages allowed can be formed within the executive subroutine. Due to this, in particular, the opportunity to vary the dialogue structure appears depending on the intermediate results. Besides, the possibility of dynamic description is necessary to connect to the user program special dialogue subsystems for data processing and analysis with its own dialogue structure. For example, the subsystem of the analysis of two variable functions with respect to the user program might have its own multilevel structure of the dialogue, which will be defined dynamically when entering to the subsystem.

To perform typical control procedures by running the user job (to restart the job, to terminate the program, to store the "history" of the dialogue and so on) the system messages have to be provided along with the user messages.

The dynamic organization of the dialogue structure can be realized by means of the following system subroutines:

BSAMES (P,SP,IT) - the setting of the logical correspondence between the order P and the executive subroutine SP; IT - the descriptor of a number and a type of message parameters.

BSLTB (IBUF, LB) - the definition of the subroutine link table where the information defined by the subroutine BSAMES is stored.

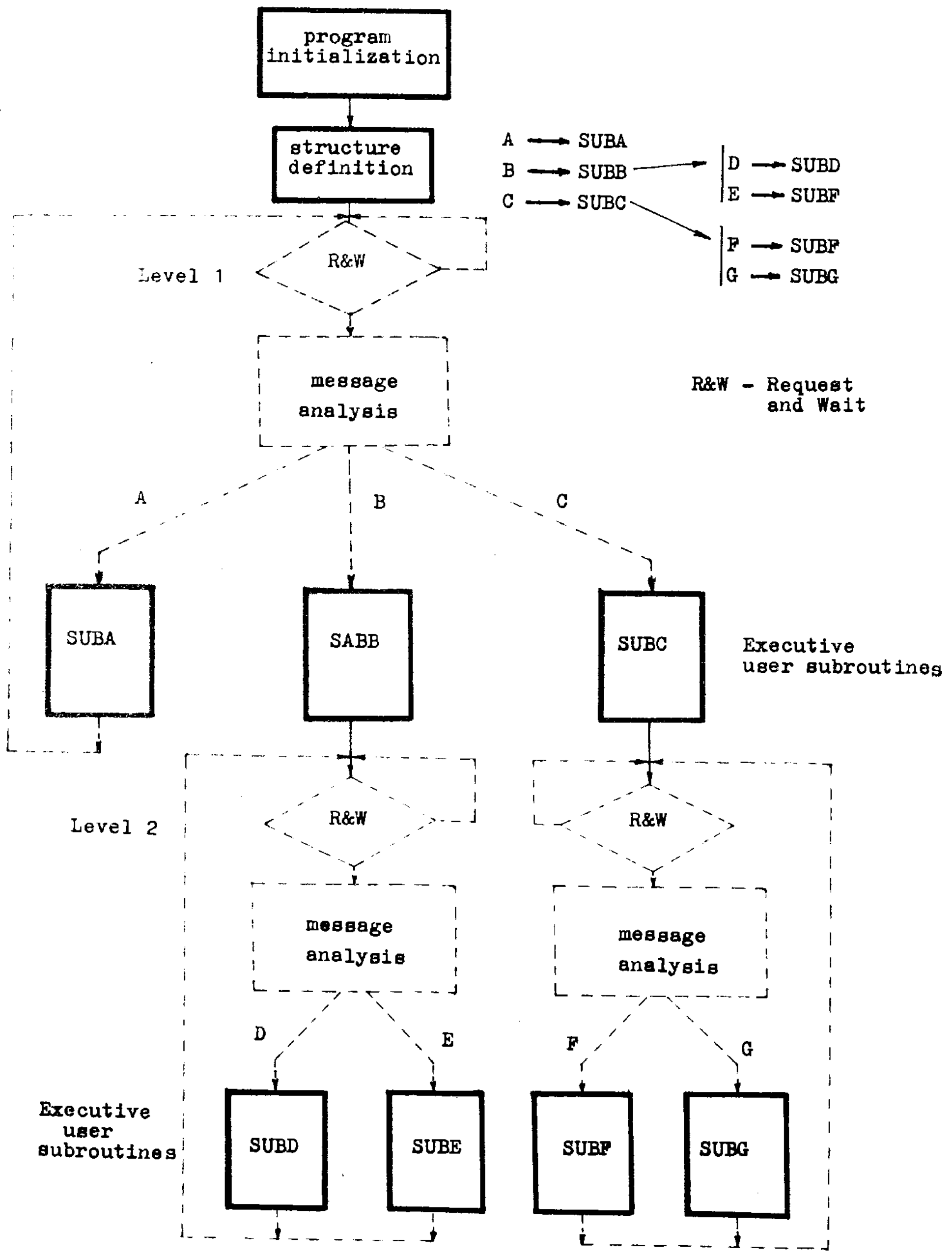

Fig. 4. Example of dialogue structure.

BSWAIT - the subroutine which generates the request for message input, organizes the loop to wait a message at the current level, the analysis of the message and the transfer of the control to the executive subroutine in accordance with a current SLT. With a statically defined structure the user must form some tables of connection depending on the number of states within the structure and besides to set the correspondence between every table and the order on which this table should become the current one. In this case, each table is assigned the name, coincident with the order and the service system subroutines have to declare on the order of the user, the current SLT from the number of those formed previously. To save "named" SLT, the SLT catalog is formed.

These functions can be, for example, performed by means of the following system subroutines:

BSCAT(CAT,LC) - the catalog definition.

BSLTC (TAB, LT,P) - the table TAB with the length LT is registered in the catalog under the name P; on the user order P this table is declared as a current SLT.

Consider, for example, the structure of a dialogue given in Fig. 4. There are two levels. At the first level there is one state (State I) with an allowable set of user orders A,B,C which the executive subroutines SUBA, SUBB, SUBC correspond to. There are two states at the second level. A jump executed on the user order B to one of them (State 2); this state is defined by the set of user orders D and E, which the executive subroutines SUBD and SUBE correspond to. To another (State 3) a jump is done on the user order C. In this state, the user orders F and G, which the executive subroutines SUBF and SUBG correspond to, are allowed. Then the static description of the dialogue can be performed in FORTRAN, as it is shown in Fig. 5.

```
     :
CALL BSCAT(CAT,LC)                Catalog definition

CALL BSLTB(TAB3,LT3)              SLT3 declaration
 CALL BSAMES('F','SUBF',ITF)      for state 3 and its
 CALL BSAMES('G','SUBG',ITG)      registration in
CALL BSLTC(TAB3,LT3,'C')          catalog with name 'C'

CALL BSLTB(TAB2,LT2)              SLT2 declaration
 CALL BSAMES('D','SUBD',ITD)      for state 2 and
 CALL BSAMES('E','SUBE',ITE)      its registration in
CALL BSLTC(TAB2,LT2,'B')          catalog with name 'B'

CALL BSLTB(TAB1,LT1)              SLT1 declaration
 CALL BSAMES('A','SUBA',ITA)      for state 1
 CALL BSAMES('B','SUBB',ITB)
 CALL BSAMES('C','SUBC', ITC)
          :
```

Fig. 5. Static description of the dialogue structure.

The description of the structure is done "below-up", thus, the SLT of the initial level is described as the last one. This table becomes the current one at the beginning of the work.

The subsequent procedures of the SLT switching, when a jump from one state to another exists, and the correspondent calls are realized by the system.

7. Dialogue variables

The absence of a simple access of the user to his program variables from the terminal is one of the disadvantages of non-dialogue programming languages (for example, FORTRAN). Such an access is required in the process of the dialogue when editing initial data, analyzing the computation results, observing the variation of variables during the calculation and so on. For the dialogue system the access must be provided by the system facilities and require minimum programming from the user.

The access to variables in nondialogue programming languages can be provided by means of the special package of service subroutines. Simple variables and arrays declared as dialogue ones or in the user program, or on the order from the terminal become available for the subsequent reference by the names, which were assigned to them. This results in the possibility to organize a simple and effective interface between the application program and processing.

The simplicity means that minimum attempts are required from the user for programming (see the example in Fig. 6.)

```
CALL BSGBF(GTAB, LC)           Declaration of dialogue variable table
CALL BSGLI(K,'K')              Declaration of variable K as dialogue
                               one with name 'K'.

CALL BSGLA(A,L,'A')            Declaration of  array  A as dialogue
                               one with name 'A'

CALL BSGLAD(B,L1,L2,'B')       Declaration of two-dimension array B
                               as dialogue with name  'B'.
```

Fig. 6. Declaration of dialogue variables.

He should specify the memory for registration of the dialogue variables and declare the variables as dialogue ones.

The efficiency is achieved due to a free (i.e. nonprogramable) access to a large number of subroutines and subsystems (both library ones and developed by the user) to represent, to analyse and to edit data. This access is performed by means of the system or the user messages. The dialogue variables can be used as parameters of these messages. For example, a call of the subsystem to analyse and to edit one-dimensional array declared as a dialogue one with its representation on the screen can be performed by means of a special system message, whose parameter is the name of this array. No special need of programming such a call in the user program is required.

Thus, the problem of the dialogue organization at a high-level language for a wide range of problems can be essentially simplified.

8. Conclusion

The development of effective software for graphic display systems continues to remain one of the actual problems of modern programming, despite of great attempts expended in this field.

Here, only some aspects of this problem have been considered, mainly, from the point of view of the user, who is programming in a high-level language. For those interested in a more detailed computer graphics, the book by W.Newman and R.Sproll/14/ which is recommended as a manual at many Institutes, will be very useful.

References

1. H.E.Kurarud. A General Purpose Graphic Language. CACM, v.11, No.4, 1968, 247-254.
2. L.B.Smith, C.E.Vandoni. Graphical Man-Machine Interactive Systems for Numerical Problems: PEG, a Special Purpose System and GAMMA, a general purpose System, CERN, 70-23, August 1970.
3. C.D.O'Brien, H.G.Bown. IMAGE - a Language for Interactive Manipulation of a Graphics Environment. SIGGRAPH - ACM, v.9, No. 1, 1975, 53-60.
4. C.E.Vandoni. SIGMA, A System for Interactive Graphical Mathematical Applications. JINR D10, 11-8450, Dubna, 1974, 234-248.
5. A.Hurwitz, J.P.Citron, J.B.Yeaton. GRAF: Graphic Addition to FORTRAN, 1967, SJCC, 47-54.
6. D.N.Smith. GRL/I-APL/I Extension for Computer Graphics. 1971, SJCC, 511-528.
7. A.Yule, R.Miller, A.Teavons. GD3-Graphic Display System, CERN Computer 6000 Series Program Library, 1970.
8. P.A.Woodsford "GINO: Graphical Input/Output" University of Cambridge Computer Aided Design Group, June, 1969.
9. T.Fergacs, G.Hermann, G.Pickler. Software methods in developing CAD Programs. Computer Aided Design (Proc. of the IFIP Working Conf. on Principles of Computer-Aided Design), edited by J. Vlietstra and R.F.Wielinga. NHC-Amsterdam-London, 1973, 205-215.
10. Yu.M.Baiakovsky, T.N.Mikhailova, S.G.Mishakova. Preprint of Applied Mathematics Institute of the USSR Academy of Sciences, No.41,1972.

11. A.V.Kavchenko, A.A.Karlov, A.D.Polyntsev, T.F.Smoliakova. Journal YC and M, Kiev, I, 1974, 110-113.
12. S.V.Gorin, V.I.Dvorzhets, V.A.Debelov, A.Ia.Krutikov. "Computer Graphics and Applications". Novosibirsk. 1971,7-18.
13. A.A.Karlov, T.F.Smoliakova. JINR, P-II-10440, 1977.
14. W.M.Newman, R.F.Sproll. Principles of interactive Computer Graphics. McCRAW-HILL Book Company, 1973.

Received by Publishing Department
on June 21, 1977.