AD662008

(14)

IGS--THE INTEGRATED GRAPHICS SYSTEM FOR THE S-C 4060

Gary D. Brown

November 1967

DDC
RECEIVED
NOV 29 1967
C

P-3722

Reproduced by the
CLEARINGHOUSE
for Federal Scientific & Technical
Information Springfield Va. 22151

This document has been approved for public release and sale; its distribution is unlimited.

ABSTRACT

IGS, the Integrated Graphics System, is a machine independent group of subroutines which may be called to produce graphic output on the S-C 4060. The subroutines may be called from FORTRAN, COBOL, machine language, or PL/I.

ACKNOWLEDGMENTS

Several people deserve credit for the IGS package. Chuck Bush of RAND did much of the design and programming. Marv Kaitz of Visual Computing Corporation and Tracy Rumford of RAND were instrumental in the design of IGS. Bob Berman and John Rieber of RAND have also contributed to the implementation of IGS.

IGS--THE INTEGRATED GRAPHICS SYSTEM FOR THE S-C 4060

Gary D. Brown*

The RAND Corporation, Santa Monica, California

INTRODUCTION

About two years ago, the passive CRT graphics world was very orderly. There was one display device--the S-C 4020 [1], one main scientific computer line, the IBM 7000 series, one scientific programming language, FORTRAN, and one graphics language, SCORS [2] (a subroutine package used for producing output on the S-C 4020).

All this has now changed. First came the third generation computers. The 7000 series was no longer the scientific computer. More surprising, IBM could no longer be considered the company. PL/I, a new programming language, was introduced. There were even rumors about a remarkable thing called time-sharing. Next, Stromberg-Carlson introduced the S-C 4060 as the successor to the S-C 4020. This Paper will introduce IGS, the Integrated Graphics System, as a follow-on to the SCORS package.

The S-C 4060 is a high-precision graphics output device which produces output on both film and hardcopy. It features a 116-hardware character set in four sizes and two orientations, an unlimited software stroke character

---

*Any views expressed in this Paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

This Paper was presented at UAIDE, Washington, D.C., October 16, 1967.

set, four line weights with four dash sizes, and four times the resolution of the S-C 4020.* It includes many new hardware features as well as a small peripheral computer called the Product Control Unit (PCU). This computer is called the PCU to avoid confusion with the host computer, the installation's general-purpose computer that generates input for the S-C 4060. The PCU is roughly twice as fast as an IBM 360/50 allowing it to take some of the load off the host computer.

Data was the first consideration. Normally, the installation's host computer generates a tape which is read by the PCU and then converted to hardware commands needed to actually draw the display. The byte was picked to be the basic unit of input for the S-C 4060, and the PCU is programmed to handle both six and eight bit bytes. This byte-oriented input is called the meta-language.

Still remaining was the problem of deciding on a software package for the host computer producing input for the S-C 4060. An obvious candidate was the SCORS package. But however excellent, SCORS has several limitations. It cannot be called from PL/I, it does not lend itself to the time-sharing environment, it is not readily adaptable to other computers, and it has no provisions for the expanded S-C 4060 hardware features. These limitations indicate the progress of the computing industry and not any failings in the SCORS design.

A new software package was designed called IGS (Integrated Graphics System). It was heavily influenced by the present SCORS package and a SHARE committee report on Standard Graphic Output Subroutines, GRAFPAC [4].

---

*A full list of features can be obtained from the hardware specifications [3].

THE DESIGN OF IGS

The IGS design was dictated by several goals. First, we had to decide which languages our system was to support. FORTRAN, COBOL, and machine language were obvious choices. In addition, we decided to make it compatible with PL/I.* PL/I and FORTRAN are incompatible in several respects. Since PL/I can call on FORTRAN subroutines but FORTRAN cannot call PL/I routines, the system had to be written in FORTRAN and not PL/I. PL/I calls must have a fixed-length calling sequence so we eliminated variable length calls. Finally, PL/I addresses a character string with a dope vector rather than by a direct address as is done in FORTRAN. An important benefit for the FORTRAN programmer in providing PL/I compatibility is better character manipluation capability than available in ordinary FORTRAN. In the graphic system, individual characters may be addressed effectively. This removes the restriction usual in most FORTRAN written systems in which character strings must lie on word boundaries and be left-justified.

The second goal was that IGS be machine independent as much as possible. It had to run equally well on six- or eight-bit machines, and not be dependent upon the word size of the computer. As one step towards machine independence, it was programmed in ASA Standard FORTRAN as much as possible.

The next goal was to isolate the user from the intricacies of the S-C 4060 hardware features. There is full access to all the hardware features, but IGS always acts as a intermediary between the user and the S-C 4060. This is because there may be a successor to the S-C 4060 with different features and the division of labor between the

---

*Whether or not PL/I will become an established language remains to be seen; but since IBM is backing it at the expense of FORTRAN and COBOL, the probability is high.

host computer and the PCU may change. Some things currently done by IGS in the host computer may be done in the CPU, but a change of this nature must not necessitate any reprogramming on the part of the user. Finally, the S-C 4060 itself has various hardware configurations. For example, it can produce 8 1/2 × 11 or 11 × 14 hardcopy. If the user insists upon addressing absolute raster units, he must reprogram if he wants to switch from one hardcopy mode to the other. IGS is designed to protect the user from this inconvenience.

Another consideration was introduced by the advent of time sharing. Because this mode of operation may become widely used, IGS has been designed to be adaptable to time-sharing systems.

We also felt that IGS should not force the user to modify his operating system. Some modifications may still be desirable (e.g., end-of-filing the output tape if the job fails), but these are at the option of the installation. This is important to those installations that buy computer time and are not free to modify the operating systems.

Nothing has been said about low core storage requirements, fast running speed, and flexibility since these are automatic considerations in the design of any software system.

DESCRIPTION OF THE IGS SYSTEM

IGS consists of a group of subroutines which are called by the user to perform whatever graphic functions needed. In their simplest form, these functions include drawing lines, plotting points, and displaying characters. These seem simple enough, but they become more complex when considering all the variations possible. For example, a line can be drawn in four line weights and four dash sizes. This raises the question of doing simple tasks without getting immersed in all possible variations.

We adopted the mode-set concept outlined in GRAFPAC, whereby mode sets are used to specify various parameters. The user is required to define a mode set array which must appear as an argument in each call to a graphics subroutine. When the system is initialized, all the default values are stored in this array. One reason for requiring reference to the mode-set array in each call is to hedge against the time-sharing environment. This allows multiple user programs to share a single public copy of the graphic system routines; each user holding more information in his private mode set array.

The use of the mode set array also adds flexibility to the graphics system. At some later date, users may find that they cannot live without the ability to draw lines with arrowheads. Previous systems had two ways to accomplish this. One way would be to add an additional argument to the calling sequence of the lines subroutine, and get the users to re-code their programs. This method has never had much success. A second way would be to write a separate subroutine, giving it another name. This results in a myriad of subroutines with slightly different calling sequences doing almost exactly the same things. By using the modes set concept, subroutines can be modified without changing their calling sequences. A vacant spot in the mode set array is used to specify that arrowheads be drawn at the end of lines. The documentation becomes simply a footnote at the bottom of the description of the lines subroutine telling the user how to set the mode. The coding is easy since the basic line subroutine need not be rewritten, but only modified to test to see if an arrowhead is wanted.

The use of the mode-set array, by limiting the number of arguments that are needed in the subroutine calls, also makes them much easier to use. Consider a subroutine to display characters. In its minimal form, the user might only

specify an x, y location of the first character, the number of characters, and the characters themselves. In a more complex form, he might want to specify the character size, character spacing, line spacing, margins, line orientation, and other items not at first apparent. It would be unreasonable to expect the user to specify all of these parameters each time he wanted to display a line of characters. Equally unreasonable would be provision of a specific subroutine for all the combinations which could be made from the possible arguments. There is not time to write special routines to provide for each combination of likely arguments, they could not be maintained, and certainly would not all fit in core at the same time.

The solution was to have the call to the character-display subroutine require only the minimum information needed to display characters. All other parameters are obtained automatically from the mode-set array. The user may make changes in this array when he wants to change a value such as character size or page margins; the mode set will stay in effect until the system is reinitialized. A single subroutine is provided to do all the mode setting.

Another basic design concept of IGS is the distinction between subject and object space. Object space is that portion of the S-C 4060 screen that the user wants to use. This can be adjusted by a subroutine call to put a margin around the page or to subdivide the screen. Subject space bounds the user's data, and can also be changed by a subroutine call. This is extremely useful not only in plotting, where the data may be in physical units not corresponding ing to the raster addresses of the S-C 4060, but also in textual displays. For example, the x-axis can range from 1 to 132, the number of normal size characters per line, and the y-axis from 1 to 60, the number of lines per page. Now lines and character positions can be addressed directly as on a typewriter. IGS is intended to do all the scaling

from user coordinates to absolute rasters. The user can of course specify that no conversion be done, to allow him to address absolute rasters.

IGS subroutines are provided to display joined lines, line segments, and points. Numeric data can be displayed in either I, E, F, or scientific format. There is also a subroutine to draw multiple line segments. To do this, one defines two delimiting vectors and the number of lines to be drawn between them. This is extremely useful in drawing grids or doing crosshatching. There are also subroutines for displaying text. Page margins, tab sets, character case, character size and orientation, and line spacing and orientation can all be established with mode sets.

One problem faced in designing IGS was that the S-C 4060 has a 116-character set while most keypunches have far fewer keys. The expanded character set is not too useful if one cannot get the characters into the computer. This problem was solved by providing special control characters which can be punched directly in the character string. For example, if the user wants lower case characters displayed, he precedes them with a '$L'. There is also a special character for upper and special case, line eject, tab, superscript, subscript, and even a null. In addition to the CHARACTRON characters, the user may design his own vector character fonts. Vector characters may be any size or orientation, and they may be skewed. Since they are composed of lines, they may be drawn with any of the four line weights.

Built on top of these routines are subroutines to draw, label, and title a grid. Grids may be either linear or non-linear in either axis. Finally, there is a subroutine to compute appropriate arguments for the grid routine, and a subroutine to draw an entire graph with a single call.

Although the design of IGS is considerably different from that of SCORS, there is a degree of compatibility. A few of the lowest level SCORS subroutines have been rewritten to produce output for the S-C 4060. Although the S-C 4060 can process the SCORS input directly, there are two reasons why an installation might not want this. First, there is a marginal benefit in making the operation of the S-C 4060 easier by having only one type of input. More important, a large SCORS program need not be rewritten in IGS to access the full S-C 4060 features. IGS and SCORS calls can be intermixed to modify the output of the SCORS program. Also, separate parts of the program can be written in IGS and then added to an existing SCORS program.

Currently, IGS is being converted to a variety of machines, and will be available in the UAIDE library in the near future.

REFERENCES

1. S-C 4020 Computer Recorder Product Specification, Document No. 281001-241, Stromberg-Carlson Corporation, May 1964.

2. Programmer's Reference Manual S-C 4020 Computer Recorder, Document No. 9500056, Stromberg-Carlson Corporation, October 1964, revised August 1965.

3. S-C 4060 Stored Program Recording System, Description and Specifications, Document No. 9500209, Stromberg-Carlson Corporation, October 1966.

4. Graphic Specifications for Standard Graphic Output Subroutines, proposed by the SHARE Standard Graphic Output Language Committee, August 8, 1966.

5. American Standard FORTRAN, American Standards Association, Inc., New York, March 7, 1966.