Tech

# PROGRESS REPORT

for the period

September 1 to December 31, 1968

of

A Research Contract

between

Brown University

and

The International Business Machines Corporation

(Data Processing Division)

9TH CONTRACT PERIOD

31 JANUARY, 1969

TABLE OF CONTENTS

## 360/1130 Communications: Stabler, Krafchin, Sedgewick

During the past contract period we have been engaged in a review of various systems (existing and planned) supporting communications between a S/360 and a remote 1130 over high speed (40,800 baud) BSC line. A proposal for our 1130-360 graphics system was presented at a meeting with Messrs. Matsa, Rulle, Johnson, and various other IBM personnel in December, 1968. The proposal was on the whole accepted, and priorities for the completion of its various phases were established.

Our first objective will be to provide support for existing 2250 MOD I users. This will require a conversion of MOD I to MOD IV graphic orders and communications between the 360 and the 1130. The second phase of our system will provide a general facility for all graphic applications from the 360. The third stage of our research project will integrate this generalized graphics capability with a symmetric call processing facility from either processor. Progress made since the December meeting is outlined below.

### 360 Communications Routines

The 360 side of the 360/1130 interface has been received from Cambridge and is in the process of being modified to fit our needs. Included in the changes being undertaken are:

(1) Modification of the message blocking and error handling routines. This is made necessary by the lack of intermediate transmission block (ITB) characters on the 1130 high speed RPQ.

(2) Allowance for multiple tasks in the 360 under either MFT or MVT (not presently supported by the Cambridge code).

(3) Provision for error statistics and a trace table for 360/1130 I/O.

Cambridge work is being monitored for changes generated by their tests of the system on their low speed line.

### 1130 Communications Routines

Coding for the 1130 I/O monitor, wait executor, and core management package has just recently been completed at Cambridge and will be received during the first week of February. A low-level high-speed communications routine for use by the I/O monitor which will be call-compatible with the low-speed SCAF4 is

now being developed from a high-speed SCAT2 routine that was written at Ohio State University. This will provide us with a general purpose high-speed communications routine and should require little or no change in the coding being received from Cambridge.

## Graphic Support for Stage 1

The basic conversion of MOD I orders to MOD IV orders has been coded and debugged on the 360. The fundamental problem of compatibility has been solved, in many instances by software support in the 1130. Before an extensive amount of time was invested in coding and debugging the 1130 side of graphics support, advantage was taken of the MOD IV facilities in White Plains. This debugging session proved to be an invaluable aid in solving certain difficult areas (e.g., attention handling and cursor manipulation).

## Sketchpad: Dan Bergeron

From October to December, 1968, the primary addition to the Virtual Sketchpad System was the on-line ability to specify (or change) which data sets should be retrieved.

Considerable effort has also been aimed at reducing the size of the system. Currently, program requirements have been reduced to 45K bytes, but access methods and data areas require another 45-55K bytes. With some limitations the system can now be run in a 98K partition. These limitations should soon be removed.

The Information Retrieval System is still being tested and further improved. The circle drawing routines have been rewritten to create equilength arcs and to allow erasure while retracing the circle. A link to a CALCOMP plotting routine has been added allowing hard copy output of the scope display.

A user's manual has been produced and will accompany distribution of the system.

## 3DPDP: Charles Strauss

During the past contract period I have been engaged in cleaning up 3DPDP, documenting 3DPDP both for my dissertation and

Hypertext Editing System: A. van Dam, S. Carmody, D. Rice

The Hypertext Editing System has been continuously evolving since June. In the package currently being made available (and in use at the New York Scientific Center) there exist a number of new editing and formatting options. These changes and additions are described in two detailed reference manuals (Edit Phase and Format Phase), and the system as a whole is described in a paper to be presented at the second annual Computer Graphics Conference at the University of Illinois in March, 1969 . A card input program provides an inexpensive means for inputting rough draft texts, and text from different data sets may also be combined and merged.

The current system is in use minimally 8 hours per day, and all our systems documentation, including the 135 page SOS manual, are being (or have been) produced with it.

Tablet-driven Editing System: Mark Pozefsky

This editing system is designed to accept character strings from a pattern recognition system (which is to be provided by the NYC Scientific Center) and incorporate these strings into the data structure for a given text. The editing system then displays the change in the manuscript that was just called for, allowing the user to accept or reject his proposed changes (a la Hypertext Editing System).

Plan Data Structures: Alan Blitzblau

In the past four months, I have been actively completing and debugging the Dynamic Storage Allocation (DSA) and Data Structure (DS) routines for PLAN. A preliminary version of DSA (maximum of 32K pages, and some undebugged features) was sent to White Plains for testing near the end of November. Since then I have completed the major debugging of the Data Structure subroutines, and they were sent to White Plains in December for testing along with DSA. I have added the code for 128K pages and an EOV trap to DSA, which are in the final stages of debugging. In addition, a bug in DSA (in one of the paging routines) which previously would not allow us to save DSA data sets has been fixed, and the updated routines were subsequently sent to White Plains. Until DSA is completed, however, White Plains can run realistic tests

of DS and DSA using the version of the system they currently have.

Flowchart Programming System: James Michener

During the past months the Flowchart Programming System has been switched over from using an incore workarea for a single flowchart to a multi-flowchart paged system, using BSAM paging under MFT (developed for DSPS). The various necessary and desirable additions to make full use of a paged environment have been added.

For instance, the system uses an "activity stack" to keep track of the flowcharts (and pages of text, see below) the user is currently interested in. Each entry in the stack identifies the flowchart, the viewpoint for displaying the flowchart, and the scaling of the view. The latter is also a new feature: four scales are available (1, 1/2, 1/4, 1/8). Unit scaling is the normal case, and is the only scale for which text inside flowchart symbols is displayed. The other scales let the user see more of his flowchart at once.

The user has two screen areas in which he can display flowcharts. (The user may vary the relative size of these screen areas, but the total display area devoted to them remains constant.) Any entry in the activity stack may be displayed in either of the screen areas; thus the user may view two different flowcharts (or pages of text, see below) or two different parts of the same flowchart or even the same part at different scales. The user can obtain a CALCOMP plot of any of his flowcharts directly from the data structure and thus the system is currently used to in conjunction with the Hypertext Editing Program to document all the graphics projects.

Another CALCOMP facility which simply translates buffer programs has been released to 3DPDP, to SKETCHPAD-I.R., and to George Stabler's FORTRAN Graphics Package.

Facilities for text editing will be added soon; these will enable the user to create pages of text without flowcharting. A user might want to do this for long sections of comments or for long computational formulae.

A data set back-up facility, patterned after that of the Hypertext Editing System, will be debugged soon for use with any DSPS type paged data set.

A users manual for the Flowchart Editor has been prepared under HYPERTEXT and is available.

TABLE OF CONTENTS

## INTRODUCTION

The following pages describe a set of FORTRAN-callable graphic subroutines whose purpose is to provide simple, high level access to the facilities offered by the IBM 2250-MOD 1 scope. The design of the subroutines and their calling sequences has been motivated by the following considerations:

1. Many mathematical problems are emminently suitable for interactive graphic solutions while requiring only a simple subset of the capabilities offered by the 2250. Indeed, the relative complexity of systems such as GPAK and GSP may hinder the effective graphic implementation of such problems.

2. Of overriding importance in practically any graphics application is the data structure used to represent the graphic entities (points, lines, text, etc.) and their interrelationships. However, in the same spirit as 1., it was felt that many useful problems could be solved without going the route of list structures, associative data structures, or whatever other golden calf might currently be in vogue. The 'data structure' used by the graphics subroutines consists of one or more two dimensional arrays which can be manipulated by both the system and the user.

The end product of these considerations is a system which is deficient in many of more complex aspects of graphics interaction (light pen tracking, hierarchical pictures, etc.), however one which is simple to learn and use, useful for a wide range of problems, and considerably less prone to system failure as compared with more complex systems.

## GENERAL DESCRIPTION

The following paragraphs describe some of the main concepts used in the system, particularly those involved in the interface between the user's program and the graphics subroutines. What is left unsaid will be clarified in the descriptions of the individual subroutines.

## The Data Array

Graphic data, whether generated by the user or the system on request from the user, is stored in one of several user defined arrays which are dimensioned 4xN, N being chosen by the user. To set up a standard example, the statement 'DIMENSION A(4,100)' would reserve space for 100 data items, the Ith item occupying A(1,I) through A(4,I). The first three of these locations, A(1,I)-A(3,I), are, with certain restrictions noted below, available to both the user and the graphics subroutines. The fourth, A(4,I), is reserved for system use and should never be refered to by the user (with the possible exception of copying one array into another).

The contents of the first three words of an item are a function of the type of item, which can be point, line, text, or plot. These will be described in turn and are shown graphically in figure 1.

## The Point Item

A(1,I) - The X coordinate, in user's units of the point. It should be noted that here and in all that follows, that all coordinate values are kept in single precision floating point (REAL*4) form.

A(2,I) - The Y coordinate of the point.

A(3,I) - A code representing the type of item contained in this 'column' of the array. For a point, this code is 1.0. The function of this code is explained in the section on adding and deleting data items.

## The Line Item

A line is specified by two consecutive points, the coordinates of which are given as above. The type code for each endpoint is 2.0.

## SUBROUTINE DESCRIPTIONS

In the following subroutine descriptions, all data types are named in accordance with FORTRAN naming conventions. If the subroutines are to be called from assembly language, the calling sequences should be appropriately modified (see IBM manual on Data Management Macros).

Name of Subroutine: GOPEN

Purpose: To initialize graphics system.

Calling Sequence: CALL GOPEN

Notes: GOPEN must be called before reference to any of the other graphics subroutines is made. Thereafter it should not be called again without an intervening call to GCLOSE (see description of GCLOSE).

Name of Subroutine: DEFINE

Purpose: To inform the graphics system of an array which is to be used for storing graphic data.

Calling Sequence: CALL DEFINE (ID,ARRAY,N,I,COX,COY,CEX,CEY)

Arguments:

ID - An identification number to be associated with this data array. ID must be an integer and in the range 1-4.

ARRAY - The name of an array dimensioned (4,N) which is to used for storing graphic data. (For further information, see the description of data arrays.)

N - see ARRAY

I - An integer variable equal to the number of data items currently in ARRAY. If ARRAY is empty when DEFINE is called, I should be set to zero. (NOTE: This must not be done by placing a zero in the calling list for DEFINE.)

COX,COY,CEX,CEY - The coordinates of the lower left and upper right hand corners of the area in which user data appears. These are used for determining which items in ARRAY should be displayed.

Name of Subroutine: DSPLAY

Purpose: To display the elements in a previously defined data array.

Calling Sequence: CALL DSPLAY (ID,AXTYP,OX,OY,EX,EY)

Arguments: ID - the ID (1-4) of the array to be displayed. AXTYP - The type of axes to be displayed:

| AXTYP | X | Y |
|---|---|---|
| 0 | no axes | |
| 1 | linear | linear |
| 2 | linear | log |
| 3 | log | linear |
| 4 | log | log |

OX,OY,EX,EY - The coordinates of the lower left and upper right hand corners of the screen area on which the array is to be displayed.

Notes: The existing elements in the data array are scaled from user to screen coordinates and displayed. Other displayed arrays are unaffected. 'ID' becomes the "current" or default id; that is, the id which is assumed in subsequent calls to data routines (LINE, POINT, etc.) if an id is not explicitly included in the calling list.

Name of Subroutine: LINE

Purpose: To add a line to a data array and the current display.

Calling Sequence: CALL LINE (XU,Y1,X2,Y2,ID)

Arguments: X1,Y1,X2,Y2 - The coordinates in user's units of the endpoints of the line to be plotted.
ID - (optional) The id of the data array to which the point is to be added. If ID is omitted, the point is added to the current data array.

Notes: The user's index for the array is incremented. A line takes up two spaces in the user's data array. If the coordinates place one of the endpoints outside the display area for the array, the line will be clipped.

Name of Subroutine: FLPBCD

Purpose: To convert a real variable to alphanumeric form.

Calling Sequence: CALL FLPBCD (VAR,W,D,AREA)

Arguments:
VAR - The name of the REAL*4 variable to be translated.
W,D - The total field width and number of decimals places for the number. The number is printed in FORTRAN FORMAT EW.D.
AREA - An array into which the EBCDIC representation of the number is placed. AREA should be at least W characters long.

Notes:
If W-6<D or D≤0, the first word of AREA (AREA(1)) is set to zero. The contents of AREA can be subsequently displayed by a call to WRITE.

(This routine has been taken in entirety from GPAK.)

Name of Subroutine: FKEY

Purpose: To provide interactive program control through the function keyboard.

Calling sequence: CALL FKEY (N,MASK)

Arguments: N - The number of the function key depressed (integer 0-31).
MASK - (optional) The name of a mask which determines which keys are to be enabled (see below). If this paramter is omitted, all keys are enabled.

Notes: The specified keys are lit, and the computer is put into a wait state which is left when one of the enabled keys is depressed. Usually this subroutine will be used with a subsequent computed GO-TO statement for program branching.
A mask which determines which keys are enabled can be specified using Z format in a DATA statement. Each bit (0-31) corresponds to a key; if the bit is '1' the key will be enabled. For example, if MASK1 is specified by 'DATA MASK1 /ZFF00C031/', the instruction CALL FKEY (N,MASK1) will light keys 0-7, 16,17,26,27, and 31.

Name of Subroutine: LPEN

Purpose: To enable light pen detection of displayed graphic data.

Calling Sequence: CALL LPEN (X,Y,ITEM,ID)

Arguments:
X,Y,- The coordinates of the point detected.
ITEM - The index of the item detected.
ID - (optional) The id of the data array to which the detected item belongs. If ID is omitted from the parameter list, it is assumed that a detect on an item belonging to the current data array is desired; control returns to the user only on such a detection--others are effectively ignored.

Notes:
Detection of a line will result in X,Y being set to the actual coordinates of the point detected, however ITEM will always be set to the index of data item representing the second endpoint of the line.
The X,Y coordinates will be found particularly useful in detecting a point which is part of a plot (see description of PLOT).
N.B.: In order to insure correct light pen correlation (i.e., determining which item in a data array was detected), the display orders for each data array are generated as a block. However, when data items are added to data arrays via calls to LINE, POINT, etc., the corresponding graphic orders are appended to the end of the existing program of graphic orders. For this reason, REGEN should be called before a call to LPEN if data has recently been added in order to insure accurate light pen correlation.

Name of Subroutine: FIND

Purpose: To provide coordinate input from the light pen.

Calling Sequence: CALL FIND(X,Y)

Arguments: X,Y - The coordinates in user's units of the light pen position.

Notes: The screen is temporarily flooded with points and a light pen interrupt is sought. When the position of the pen is detected, X and Y are set to the coordinates of the pen in user's units. Note that scaling to user's coordinates is always with respect to the scale factors of the current data array. No change is made to the existing display.

Name of Subroutine: DELETE

Purpose: To delete an item from a data array.

Calling Sequence: CALL DELETE (ITEM,ID)

Arguments: ITEM - The index corresponding to the item to be deleted.ID (optional) - The id of the data array from which the item is to be deleted. If omitted, the deletion is made from the current grid.

Notes: ITEM can consist of an integer, integer variable, or the name of the index associated with the data array. (The latter case will result in the last item in the array being deleted.) If a line is to be deleted, ITEM can correspond to either endpoint; both endpoints will be deleted.Items after the deletion are moved down in the array so as to fill up the vacancy left by the deletion. The user's index is updated.

Name of Subroutine: BLANK

Purpose: To render invisible all plotted entities associated with a data array(s).

Calling Sequence: CALL BLANK (ID1,ID2,ID3,...)

Arguments: ID1,ID2,ID3,... - A list of ids corresponding to the data arrays to be blanked. The list must contain at least one id.

Notes: All items in the given data array(s) are deleted from the current display. The items for each array can be restored by a call to UNBLNK or DSPLAY. Blanking an array which is already blanked has no effect.

Name of Subroutine: UNBLNK

Purpose: To redisplay the items in a previously blanked array.

Calling Sequence: CALL UNBLNK (ID1,ID2,ID3,...)

Arguments: ID1,ID2,ID3,... - A list of ids corresponding to the data arrays to be unblanked. The list must contain at least one id.

Notes: All previously displayed items in the blanked array(s) are redisplayed. Unblanking a display which is already displayed has no effect.

Name of Subroutine: ERASE

Purpose: To delete all items in a data array.

Calling Sequence: CALL ERASE (ID)

Arguments: ID - The id of the data array to be erased.

Notes: The user's index for the array is set to zero, thus emptying the array of data items as far as the graphics system is concerned. All data items belonging to array are deleted from the current display. The user can "restore" the array by resetting the index of the array to its former value and calling REGEN.

Name of Subroutine: CALCMP

Purpose: To provide a hard copy of the current display.

Calling Sequence: CALL CALCMP (IHGHT)

Arguments: IHGHT - An integer representing the height in inches of the CalComp graph to be generated.

Notes: The screen will be blanked, and the current display will be translated into a format suitable for CalComp input and written to tape. (Note that a DD card for a CalComp tape must be included in the job if this routine is to be called.) The output of the CalComp Plotter will be exactly the same as the current display with the exception of the scale factor determined by the height of the screen (10 inches) as compared with IHGHT. When control returns to the user, the current display will be restored. The routine may be called more than once to provide hard copy for several displays.

## ABNORMAL CONDITIONS

Programs involving graphics applications are not inherantly more prone to errors or logical flaws; however the results of such are often more spectacular than those incurred by other programs. The most common sequence of events is the following: a user is sitting at the scope busily interacting with his program. On pushing a function key or indicating part of a display with the light pen, he notes that the expected response does not occur. Thereupon he decides to attempt the action again or perhaps several more times. Presently the system dies, frequently noting its passing with certain tintinnabulous effects. The cause of this event (leaving unasked, for the moment, certain questions best addressed to the designers of the system) was the user's second and subsequent attempts to obtain a response from his program. His first attempt, in all likelihood, resulted in a program check or other abnormal condition causing termination (or dumping) of his program. As it transpires, interrupts received from the scope during this termination process are the cause of massive trauma for OS, thus the events described above.

In order to prevent this from occurring, the following axioms should be closely observed. Whenever a scope program does not react in the expected manner, particularly when it doesn't react at all, immediately attempt to ascertain the problem. If your program is dumping (the operator will be able to tell you this), do not attempt any more actions at the scope (it may already be too late).

The graphics subroutines include a procedure for catching a small subset of abnormal conditions before they cause termination of the job. If one of these conditions should occur, the current display will be replaced with a message notifying the user of the conditions and giving instructions for obtaining a dump. The job will then be terminated. (Note: if there is any suspicion that one of the graphics subroutines has malfunctioned, a dump should be taken.)