POGO: Programmer-Oriented Graphics Operation*

by B. W. BOEHM, V. R. LAMB, R. L. MOBLEY, and J. E. RIEBER

The RAND Corporation
Santa Monica, California

INTRODUCTION

Wide-scale application of interactive computer graphics (ICG) is currently inhibited by two major difficulties:

1. Terminal time costs too much.
2. It takes too much effort and expertise to develop and modify ICG programs.

Several approaches, including RAND's Video Graphics System and other television-based or storage-tube based console designs, are currently being tried to overcome the first difficulty.

The POGO system described here represents an attempt to overcome the second difficulty, at least for a certain fairly general class of problems, which includes the interplay of computational programs with alphanumeric and curve input and display, but excludes highly dynamic interplay of computational programs with geometric manipulations.

POGO is fully operational; it allows a user to specify the nature of his ICG interfaces in a natural way at the graphics console itself, simplifying the programming process in two ways:

a. Unburdening the programmer of the tedious and artificial process of specifying ICG control "pages" (CRT displays) by transcribing coordinates from layout paper and stringing together calls to graphic support subroutines.

b. Permitting programmers to create ICG programs without spending a great deal of time learning the intricacies of the graphic subroutine package.

To an extent, these difficulties are removed in more specialized ICG packages—for such applications as simulations, circuit design and layout, trajectory analysis, curve fitting, and chemical analysis—which we would call user-oriented rather than programmer-oriented graphics operations. However, the second difficulty reappears as soon as the user wants some capability not expressible within the standard package (a fairly common occurrence), which can be achieved only by rewriting a piece of the package in a lower-level language.

This paper continues with some background information on the development and usage of POGO, followed by a description of POGO's general capabilities, illustrated by an example of its use.

Background and applications

Background

POGO is implemented on a 256 Kbyte IBM 360/40, furnished with an IBM 2250 graphic display console with light pen, keyboard and function keys, a RAND Tablet for freehand inputs, an SC-4060 hardcopy device, and some IBM 2311 disk drives. It is written in IGS (Integrated Graphics System), a set of FORTRAN-callable routines for elementary graphics manipulations similar to the IBM packages GPAK and GSP.

The "mother" of POGO (in the sense that necessity is the mother of invention) is a user-oriented ICG system for aerospace vehicle trajectory analysis called Graphic ROCKET. This system consists of a network...
of interconnected control "pages" (see Figures 1, 2) by which a user specifies the design and flight plan of a rocket vehicle, then specifies and views desired graphical displays of the resulting vehicle performance (Figure 3).

To create control pages for Graphic ROCKET, we had to visualize, usually with the help of a layout sheet, the positions and extent of lines and characters, and to determine their coordinates and character counts to enter as arguments for the appropriate IGS routines. Finally, we had to check the display on the 2250 to see if it came out the way we wanted. If not, we had to go through all the steps again. Modifying a control page to incorporate an extra user-desired option also required going through all the steps again.

Finding that other ICG application designers at RAND had similar problems (and having had them previously ourselves), we made a control page design program that was as general as possible without seriously compromising its simplicity. To complete the package, we modified a curve input program that had been developed for a graphic re-entry simulation, and the curve display program and filing routines from Graphic ROCKET, to make them available to the general ICG program developer in forms easy to use, and compatible with the design pages and with FORTRAN computational programs.

Applications

Since then, POGO has been used to create control pages and explanation pages for Graphic ROCKET and other models including on-line simulations and chemical models; it has further been useful for such processes as curve-fitting and digitizing map data. It has also been used to entirely specify the ICG interface for a model of fluid balance in the human body. Using the SC-4060 hardcopy option, which produces a paper copy and a 35 mm film copy of the 2250 scope contents whenever a function key is pushed, POGO has been a handy tool for composing slides for briefings and talks (e.g., Figure 4). In this vein, POGO has an attractive
potential for film animation, which we have not yet been able to pursue.

Using POGO: an example

This example involves a model of fluid balance in the human body. The rate of the body’s water excretion depends on the level of a hormone called ADH in the blood plasma, while the rate of ADH production depends on the amount of water in the plasma. The functional forms of these relationships are imprecisely known from physiological measurement. One would like to try a set of test functions (curves), specify some external inputs in curve form (e.g., “a drink of water”), and a set of initial conditions, integrate the set of coupled differential equations and compare the results with observed data.

This had been done by means of a batch-type FORTRAN program (Figure 5a) that accepted keypunch input curves and parameters and printed the numerical values of the resulting integration outputs, which were then manually plotted. As this is a tedious and time-consuming process, one would like to give the researcher the capability of tracing his curves directly into the machine and directly viewing the output curves. And, if the fit is not satisfactory, one would like to interrupt the integration, modify the numerical or curve inputs, and try again. This capability is what POGO allows a programmer to create in a simple, natural manner. The accompanying film will show the interactive aspects; this paper will just show some examples of the various pages.

Constructing graphical interfaces

Given any batch program, such as the fluid balance model, producing output functions from input functions and parameters, the following capabilities are needed to construct an interactive-graphics interface:

1. A means of interactively tracing, editing, labeling, and storing input curves.
2. A means of interactively recalling stored input curves and specifying which to use.
3. A means of identifying the types of input curves that may be traced and stored, and of relating them to the appropriate storage arrays in the batch program.
4. A means of interactively selecting, scaling, and displaying output curves.
5. A means of identifying the types of output curves that may be displayed, and of relating them to the appropriate storage arrays in the batch program.
6. A means of entering values of numerical parameters through the graphics console, and of relating them to the appropriate storage locations in the batch program.
7. A means of specifying decision options at the graphics console, including the flow of control between the various input and output pages.

POGO provides these capabilities in two phases of operation at the graphics console, the design phase and the execution phase, separated by a generally short programming phase. The nature of these phases and their interrelations are shown for the fluid balance model example, a fairly typical case, in Figure 5b.

Curve input

Capabilities 1 and 2 for creating, storing, and recalling input curves are provided during the execution
DESIGN PHASE - AT CONSOLE

Enter names of input arrays. (Fig. 8)
Enter names of output arrays. (Fig. 10)

PROGRAMMING PHASE

- Modify batch program. (Fig. 5a)
- Write, or use standard POGO, control page programs. (Fig. 12)
- Write, or use standard POGO, main control program.
- Compile, link edit, file on disk with control page data,
  using standard POGO job control sequence.

EXECUTION PHASE - AT CONSOLE

Read in from disk
Enter initial conditions (Fig. 11)
Trace curves (Fig. 6)
Select curve types (Fig. 7)
Enter control parameters
Display output graphs (Fig. 9)
Quit

Figure 5b—The POGO process: Fluid balance model example

Figure 6—POGO curve-tracing page

available, and the current list of curves of each type
appears when the appropriate “LIST” box is hit. One
can then specify which of these curves to use in the
calculation, to delete, or to display; the latter option
transfers control to the curve-tracing page with the
specified curve displayed. Up to ten curve types and
ten curves of each type can be stored; the “SCROLL”
options allow the user to reach the ones not currently
displayed.

Capability 3, for identifying types of input curves,
is provided during the design phase by the POGO page
shown in Figure 8. The user specifies, at the graphics
console, all the information POGO needs to label his
input curves and relate them to the appropriate arrays in his batch FORTRAN program, via the questionnaire form on the lower part of the page. When he pushes the “ENTER NEW DATA” box, the resulting information is summarized on the upper part of the page, and entered into the appropriate master tables on the POGO disk. “PUNCH DIMENSION STATEMENTS” produces a set of punched cards with the resulting FORTRAN array dimensions, if needed, for the user’s erstwhile batch program.

For example, suppose a user has entered the information shown in Figure 8 during the design phase. Then, if, in the execution phase, he hits the “USE” box on line 1 of the “LIST OF CURVES OF TYPE 2” area of the control page of Figure 7, POGO will go to region 2 of its curve input storage space on the disk, read off the values of the independent and dependent variables stored in item 1 of region 2, and store them in the arrays XA2 and F2 (from Figure 8), respectively.

Curve output

Capability 4, for selecting and displaying output curves, is provided during the execution phase by a standard POGO display page shown as Figure 9. This page allows one to graph values of two dependent variables (Y and Z), as functions of one independent variable (X). The windows along each axis show which of the output quantities is being used. Pushing the arrow next to a window brings in the next quantity on the list. Its name and unit appear in the window; its nominal upper and lower limits appear on the axis, and, on pushing the appropriate letter (X, Y, or Z), the corresponding output curve appears.

One may change scale by writing in new values with the stylus, find the numerical values associated with any point in the output region by pushing “NUMERICAL VALUES” and indicating the point with the stylus, or superimpose a grid on the output region. Generally, output points appear as they are being calculated; one may push boxes to “HOLD” the calculation and to “GO” again. And, one can store current curves on the disk and recall them for comparison with future runs by using the “STORE” and “COMPARE” options.

Capability 5, for relating the quantities produced by the erstwhile batch computation program to the routines controlling the Output Graph Page of Figure 9, is handled during the design phase in much the same manner as the corresponding input specification Figure 10. With this page, the user sits at the graphics console and enters, for each variable he wishes to display (up to twenty), its FORTRAN array name and dimension in his computational program, some description for the labels along the axes on the display page, and some nominal lower and upper limits. As with the corresponding input specification page, “ENTER NEW DATA” updates the appropriate master tables on the POGO disk, and “PUNCH DIMENSION STATEMENTS’ produces FORTRAN array dimensions on cards, if desired.
Control page design and interfacing

Capabilities 6 and 7, for entering parameter values and specifying decision options at the graphics console, are provided by control pages, which the user designs himself. He can do this at the graphics console by means of a set of POGO routines that allow him to create strings of text, fields for numerical values, option boxes, and geometric figures. He may then use the Tablet stylus to move these around the CRT screen until he is satisfied with the layout. He may enter codes that will be used to relate the numerical fields and option boxes to his FORTRAN program, and then press a button, which has POGO punch out a set of cards that will recreate the display at any later time.

Figure 11 shows one of the control pages that was created with the DESIGN program for the fluid balance model. The number associated with an input field indicates the location in the input storage array into which POGO will store values entered in that field; when one of the boxes is hit, the number accompanying that box will be returned to the user's control program for him to analyze what to do next.

To manage the control pages at execution time, POGO has a standard set of routines that can be incorporated in the programmer's FORTRAN control program in any way he desires. These routines include:

- RECALL
  - given a page number, creates the corresponding page on the screen.

- ACTION
  - waits until the user interrupts via keyboard, function key, light pen, or Tablet, then returns to the control program with numerical codes identifying the type of interrupt and its location.

- SAVAL
  - tests values of variables on the screen to see if they have been changed since the last such test. If so, they are converted to floating point and stored in the location corresponding to their ID number.

By stringing together CALL's to those routines in his FORTRAN control program, the programmer can allow the user to switch from one display page to another, enter new values, select multiple-choice options, or transfer to the curve input and display pages. He generally writes these programs after having composed the displays (see Figure 5), but can work the other way around also.

Figure 12 is an annotated listing of the subroutine
SUBROUTINE CTRLPG(IPG)

COMMON/PARAM/D(100)
--storage array for input parameters

COMMON/JPAGES/JCURVP,JGRAFP,JGRI DP,JUSERP(10)
--two standard POGO cards

COMMON/FLAGS/IPGNXT,NPOINT(20),INPNTS(10),IRUN
--two standard POGO cards

COMMON/MODES/Z(200)
--a communication area for all graphics routines

DIMENSION BCOVAL(3,15),KVAL(15)
--for storing BCD images and pointers

CALL GETIDG(Z,ID)
--provides local ID number for display

CALL RECALL(Z,JUSERP(IPG),0,BCOVAL,KVAL,OMY,OMY)
--places display of Fig. 11 on screen

CALL ACTION(Z,ICH,IVAL,ID)
--waits for user action

IF(ICH.EQ.2 .OR. IVAL.EQ.500) GO TO 2
--check inputs on box strike or end key

IF(IVAL.GE.1001 .AND. IVAL.LE.1010) GO TO 3
--box strike to change page

GO TO 1
--other actions ignored

CALL SAVAL(Z,D,BCDVAL,ID,KVAL,NVAL)
--convert new numerical entries

GO TO 1
--other actions ignored

CALL CTRLPG(IPGNXT)
--return to control program

IF(ICH.EQ.0 .OR. CH.GT.1000)
--check inputs on box strike or end key

IF(IVAL.GE.1001 .AND. IVAL.LE.1010) GO TO 3
--box strike to change page

GO TO 1
--other actions ignored

CALL SAVAL(Z,D,BCDVAL,ID,KVAL,NVAL)
--convert new numerical entries

GO TO 1
--other actions ignored

CALL DISPLG(Z,0,0,0)
--clears display screen

RETURN
--return to control program

END

Figure 12—POGO control page subroutine

CTRLPG, which uses these POGO routines to manage the initial conditions page of Figure 11 at execution time. First, the page is put onto the screen with RECALL, then ACTION waits for a user action. Suppose he enters the number “43.6” in the field next to “INITIAL WATER LEVEL”, and the number “3” in the field next to “NUMBER OF SIGNIFICANT DIGITS” (either via the keyboard or the Tablet stylus), and then pushes the “CHECK INPUTS” box (with the light pen or Tablet stylus). Control will pass from the subroutine ACTION with IVAL = 500; the next statement results in a “GO TO 2” that passes control to SAVAL, which will return with the value 43.6 placed in FORTRAN location D(2) and the value 3.0 placed in location D(14).

The subsequent “GO TO 1” returns control to ACTION, which waits for further action; suppose the user pushes the “TO INPUT CURVES PAGE” box. Control now passes from ACTION with IVAL = 1002; the subsequent tests produce a “GO TO 3,” which computes the number of the page to be placed on the screen next (IVAL = 1000 = 2), clears the display screen, and returns control to the main program. Of course, during the programming phase of Figure 5, one must also write a main program that calls the appropriate subroutine when its number is returned.

CALL CTRLPG(IPGNXT)
in his main routine and include CTRLPG in his load module.

1. Even if the user wants extra features, such as special option or decision boxes, on his control pages, the control routine he writes will be generally simple and straightforward. Furthermore, it will be transparent to such control page modifications as adding or changing commentary, adding new “values” fields, and moving entities around the page.

3. The output display control page of Figure 9 and the input and output array specification pages of Figures 8 and 10 were laid out with the POGO DESIGN program. The corresponding control programs involved CALL’s to ACTION, SAVAL, etc., but also required some IGS-level programming for scrolling, curve display management, and numerical values. These figures give some idea of the range of POGO capabilities for composing and managing control pages. This “bootstrapping” capability also allowed us to shorten the development times of the total POGO package, probably by a couple of months; it also makes these pages very easy to modify.

Control page design
All of the figures in this paper, except Figures 3, 5, 6, 12, and 13, were created with the POGO DESIGN program; this gives some idea of the range of its capabilities.

The facilities available to compose and interface displays are indicated in Figure 13, which show the POGO function keyboard layout. To use any POGO facility, the user simply presses the corresponding function key. Short descriptions of some of these facilities follow:

Small Characters

The user indicates with the Tablet stylus where...
on the "page" he would like his character string to begin, then enters a string of characters from the keyboard.

**Touch-Up**

Placed the console in the character recognition mode. The user may modify any of the characters on the screen by writing over them freehand with the Tablet stylus.

**Move**

The user points to the character string or geometric entity he wants to move with the Tablet stylus, and drags it around the screen with the stylus until it is where he wants it. Lifting the stylus completes the action.

**Plain Boxes**

The user points with the Tablet stylus to define the lower left corner of the box. Pointing again defines the upper right corner; the user may drag the position of this corner around until he indicates (by lifting the stylus) that he is satisfied with the box.

**Values**

The user points with the stylus to define a place to store the value of a variable; the position is denoted by underscores.

**Fancy Boxes**

These are similar to plain boxes, except they have a dot at the center to serve as a target for the light pen.

**Insert Codes**

By each box and each "values" position in the current display, the user is presented with a line of underscores to furnish a numerical code which will identify this box or value this FORTRAN control program.

**Joined Lines**

The user can draw arbitrary geometric figures consisting of joined line segments with the stylus.

**Recall Files**

They allow the user to recall a previously created display for review or modification.

**Output Display**

POGO asks the user to provide a name for the current display. When this is done, POGO punches out a set of cards with the information necessary to recreate the display and identify its components to the FORTRAN control routines.

**An example using the DESIGN program:**

**Furniture arrangement**

Figure 14 shows the layout of an apartment suite and the outlines of various pieces of furniture. Pushing the MOVE key in the DESIGN program allows one to drag the images of the items of furniture about the screen until he has a satisfactory layout. Or, by using the other DESIGN options, he may create more furniture or modify the outline of the suite. This application makes a nice demonstration of the potential use of interactive graphics in problems of spatial distribution. The entire application was composed in a twenty-minute console session with the DESIGN program.
FURNITURE ARRANGEMENT

POINT TO THE FURNITURE WITH THE STYLUS
AND MOVE IT AROUND THE HOUSE AS YOU LIKE
OR... BUILD YOUR OWN FURNITURE

Figure 14—A POGO DESIGN application:
Furniture arrangement

COMMENTS AND CONCLUSIONS

Versatility

The POGO pages are modular and need not all be used for an application. Thus, a POGO program may consist completely of control pages, as in decision tree applications, or perhaps just curve input and output, as in computing convolution integrals. Further, the modules have clean, well-defined interfaces and can be (and have been) used to design and update the control page parts of a special-purpose ICG applications system, or input and file traced curves for interactive or non-interactive programs.

Although the DESIGN program and the curve-tracing program require the RAND Tablet, the rest of the POGO pages (including pages composed with the DESIGN program) will run on a configuration having only light pen and keyboard input.

Areas for improvement

Some POGO improvements can be accomplished fairly straightforwardly, and these we plan to incorporate, including:

1. Wider range of output pages: charts, histograms, more numerical information.
2. Simple operations on output curves, particularly integrals and derivatives.
3. Reformattting displays at execution time (e.g., for summarizing a sequence of decisions).
4. Scaling and overlay of displays at design or execution time (e.g., for map and network studies).

Others are more difficult, such as incorporating a hierarchical structure in DESIGN program constructs, due to the nature of IGS, our source language. On others, such as a more advanced file and retrieval system, we are waiting for more information on usage patterns.

Finally, one would like to combine the design and execution phases, indicate the flow of control and processing activities at the console along with the page design, and then directly execute the resulting program. RAND is doing some research toward building such a capability, but it is more difficult than POGO by at least an order of magnitude.

Development and usage experience

Total development time for POGO to date has been about one man-year. Machine usage on the 360/40 was about 100 hours for development. As mentioned above, our ability to bootstrap some of the curve input and output pages with the DESIGN program reduced our development time by a couple of months, and makes these pages far easier to modify.

We are just beginning to instrument POGO to measure user interaction and response times. One interesting usage observation is that people tend to get tired of the continual, precise interaction involved in control page design and sign off after one to two hours.

On responsiveness in interactive graphics systems

On our Graphic ROCKET application, we estimate that the POGO DESIGN page has cut our control page development times by factors of four to ten below those required for the manual layout-paper approach. Further, the work is far more palatable, and our error rate is cut to virtually nil.

The most important consequence of the above factors is that they have lowered considerably our responsiveness threshold on providing users with additional capabilities not in the basic Graphic ROCKET package. On most interactive graphics systems we have seen, this extension threshold is quite high and constitutes a major usage bottleneck.

If there is any general reason for this, we feel it is due to a tendency to design complete ICG systems by deductive inference from an abstract model of typical user performance at a console, producing “closed” systems, which are quite responsive in the small but quite unresponsive in the large. Our experience with Graphic ROCKET and POGO users indicates that
general characterizations of user activity are still quite risky, and that more overall responsiveness is gained by the prototype approach: deliberately designing an austere but extensible prototype, then refining it by inductive inference from observed usage patterns.

REFERENCES

1 M R DAVIS T O ELLIS
   *The RAND tablet: A man-machine graphical communication device*
   The RAND Corporation RM-4122-ARPA August 1964

2 G D BROWN C H BUSH
   *The integrated graphics system for the IBM 2250*
   The RAND Corporation RM-5531-ARPA October 1968

3 B W BOEHM J E RIEBER
   *Graphical aids to aerospace vehicle mission analysis*
   The RAND Corporation P-3660 October 1967

4 A S PRIVER B W BOEHM
   *Curve fitting and editing via interactive graphics*
   The RAND Corporation P-3742 December 1967
   (Also in *Interactive systems for experimental applied mathematics* M. Klerer and J. Reinfelds Academic Press 1968 343–45)

5 R TURN R L MOBLEY J P HAVERTY M WARSHAW
   *An application of interactive computer graphics to on-line ballistic missile defense simulation*
   The RAND Corporation RM-5590-ARPA August 1968

6 G F GRONER
   *Real-time recognition of handprinted text*
   The RAND Corporation RM-5016-ARPA October 1966