INTRODUCTION

Picturelab is an interactive program system for experimentation in picture processing built using the SIS facility on a GE 635. Picturelab has been prepared as a replacement for a previous batch facility. The batch facility had the major drawback that most studies had to be run overnight. When graphic results were required microfilm processing time was added to the already long turn-around time. Since most present research in image processing continues to be done on a trial-and-error basis, the long turn-around times associated with the batch runs resulted in a combination of painfully slow progress and long running times.

In place of this, Picturelab permits interactive processing and immediate viewing of the processing results. This approach permits faster progress because of continuity of attention, quick elimination of fruitless paths, and capacity to test a much greater variety of algorithms in a given period of time. The material following is organized into a section introducing digital representation of pictures and sections discussing design considerations and outlining the basic portions of the Picturelab system. The appendix contains an example of an interactive session.

BRIEF INTRODUCTION TO DIGITAL REPRESENTATION OF PICTURE INFORMATION

Picture information is represented digitally by subdividing a picture into a rectangular array of points, and storing the level of grayness of each point as a binary integer. The gray level value is generally obtained in one of two ways, depending upon the storage medium. In the case of 35mm transparencies, the gray level is obtained by shining a spot of light through the transparency and measuring the amount of light transmitted. Since the spot of light must be moved about the picture, a conventional technique is to locate the picture in front of a Cathode Ray Tube and use beam positioning to control spot placement. The spot is made to move regularly along the rows and the amount of light that is transmitted is converted to a digital value and stored on magnetic tape. This type of scanner is referred to as a "flying spot scanner".

Picture information stored on opaque paper may be scanned by a facsimile device. In a facsimile scanner the material is wrapped around a drum and the drum is rotated under a spot of light. The reflectance is measured and converted into digital form and stored. During rotation the spot is made to move down along the length of paper.

To give an example of the volume of data produced, a 35mm slide is subdivided into approximately 1000 rows and 1000 columns. A five-bit gray-level value is stored for each of the 1 million row-column intersection points, thus producing 5 million bits of information. The regular two-dimensional array of intensity information is conventionally referred to as a raster or gray-scale or half-tone representation. Such a representation is utilized for present day television pictures.

DESIGN CONSIDERATIONS

The design criteria set forth for the Picturelab system were that the system should be (1) interactive, (2) command driven, (3) permit symbolic variable referencing, (4) permit the creation and execution of symbolic command files, (5) permit storing of partially processed pictures, and (6) be usable not only interactively but also as a batch program. In addition, the commands used for batch processing should be the same as the commands used for interaction whenever possible. Finally, of course, the arrays of picture information...
that could be processed were to be as large as possible. All of these criteria were actually achieved including the capacity to handle picture fragments up to 128 x 128 points.

The reason for making the system interactive was that, as previously mentioned, most picture processing research proceeds in a trial-and-error manner. A suitably implemented interactive system permits continuity of attention, quick elimination of fruitless paths, and faster progress down fruitful paths.

The decision to make the system command driven was based upon the desire to permit interaction to proceed quickly, to permit symbolic files of commands, and to permit commands punched on cards for batch input to be of exactly the same form as the commands used at the console.

The provision for symbolic variable referencing was founded on the desire to make the use of the commands as much as possible like programming in conventional high level languages (e.g., FORTRAN). One of the basic features of all programming languages is the use of symbolic names for variable information. This is accomplished in Picturelab by providing the user with a symbol table in which he may store variable names and associated values. He may then access the values for use in commands simply by supplying the variable name.

The provision for creating files of commands permits accumulating together, under a single name, a number of commands required to control a single activity. The single name may be used in place of the sequence of commands in other contexts. This is the familiar subroutine concept of higher level languages.

A major demand to be made upon the system was that of access to picture information. In order to permit efficient execution is was decided that information being processed should be held in core storage. It was not possible to permit holding in core an entire picture. Rather, portions of pictures called fragments are selected from within a picture for processing and placed within a system Data Area. The information that is accessed by processing programs is whatever picture fragment is stored in the Data Area at the time of access.

Another decision made about the storage of picture information was that, for a given picture fragment in the Data Area, the data should be stored with one point per word. Information from different pictures (raw and processed data, for example) is stored in different sequences of bits within the words. The two-dimensional array of bits obtained by taking the same sequence of bits within all the words used for a picture fragment is called a picture Plane. Using Planes permits optimizing the program that does the packing and unpacking of the data from a specific set of bits. This program can be made to execute faster than one unpacking data from varying sets of bits as would be required if many points from a single picture were stored within one word.

The size picture fragment that can be accommodated is directly related to the amount of storage available for the Data Area. It was decided that 18,000 words (enough to hold a 128 x 128 fragment) would be an acceptable compromise between the demand for larger fragments and the desire to conserve core storage.

The programs (called Routines) which implement the picture processing algorithms operate on whatever data happens to be in the Data Area at the time of execution. In general, the arguments to a Routine will indicate one Plane from which to obtain information and another Plane in which to store the results of the processing.

THE PICTURELAB SYSTEM

The fundamental structure of the system consists of two basic data tables and three basic program systems. See Figure 1.

The data tables are the system Symbol Table for holding variable names and their associated values and the system Data Area which holds the current picture fragment. When not in core these tables are maintained in a file system in addition to all of the processing Routines, and saved sequences of commands (called Processes). The three program subsystems and their responsibilities are listed below.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Processor</td>
<td>Create and maintain the Symbol Table, Initiate execution of Routines, Create and maintain Processes.</td>
</tr>
<tr>
<td>I/O System</td>
<td>Read picture information into Data Area from Tape or Disc,</td>
</tr>
</tbody>
</table>
Store picture information from Data Area on Tape or Disc.

Library of Routines Perform picture processing algorithms.

The Routines that actually perform the picture processing are written in FORTRAN and are compiled exterior to the Picturelab system. These Routines are then converted into executable form by the GE system loader and stored within the file system as executable core images. Commanding the execution of a Routine results in its being loaded along with the Data Area and control being passed to it. At the completion of execution, control is returned to the Command Processor for communication with the user to obtain his next command.

GRAPHIC CAPABILITIES

Since human comprehension of pictures depends on pictorial information rather than lists of data, it was felt mandatory that the user should be able, during a console session, to view the picture information stored within the Data Area. This is accomplished through use of the SIGHT2 console. The console consists of a Digital Equipment Corporation PDP-7 computer and a Cathode Ray Tube display. The display has 1024 x 1024 addressable points and a spot may be illuminated at one of eight levels of intensity. The intensity modulation permits direct viewing of half-tone (gray-scale) pictures on the scope face. The PDP-7 contains 8192 words of core storage. Picture information for display is held in core storage in the form of scope vector commands. The commands draw horizontal vectors of constant intensity. Changes in intensity are accomplished by starting a new vector with a new intensity.

The size picture that may be stored depends upon the frequency of changes of intensity. More than one picture may be stored, if space permits, and selection of the picture to be displayed is accomplished by means of function buttons. Sample capabilities are one half-tone picture or three or four binary (two level: black and white) pictures. Figure 2 is a photograph of a half-tone display.

In addition to the interactive graphic capability provision is included for the generation of microfilm from binary pictures using a Stromberg Datagraphix 4000 Microfilm Recorder. The information plotted on one frame may include data from a number of picture fragments, thus permitting an entire one million point binary picture to be plotted on a single frame. In effect this creates a 35mm transparency of the same size as the original.

INTERESTING IMPLEMENTATION APPROACHES

The combined constraints of needing to work within a relatively small amount of core storage, desiring to have execution proceed as quickly as possible and desiring to do most of the coding in Fortran led us to utilizing a number of sophisticated approaches which may be of interest to others attempting similar systems.

The approaches to be discussed here include (1) swapping of the SIS executive and a special command structure to improve the speed when this is done, (2) dynamic modification of subroutine calling sequences to improve execution time, (3) recursive overlaying of the entire Command Processor in order to implement a block structure capability for the Processes, and (4) inclusion of an arithmetic statement compiler and executor.

SIS, the system upon which Picturelab is built, normally executes in 32K words of core storage. This area is subdivided into about 20K for the SIS executive itself, and 12K for user programs, called Maps, that execute under control of the executive. In order to accommodate up to 18,000 words of picture data it was decided to swap the SIS executive, which provides a file system and a device-independent I/O system, out of core and utilize the freed space for the picture information.

This necessitated the writing of a minimal (3K) executive to control algorithm execution. The executive provides restricted I/O capability and can access only files for which it has absolute disk addresses. This executive (called the Runner) is called by the Command Processor.

Figure 2—Enlargement of a portion of a raster picture
Processor when the execution of a Routine is requested. It swaps the SIS executive, loads the Routine to do the actual processing, and after its completion restores the SIS executive and returns control to the Command Processor.

In order to increase the speed at which the execution of routines could proceed a special mode (called Deferred Execution) was provided which permits accumulation of a series of Routine references and, on command, executes the entire list, swapping the SIS executive out at the beginning and restoring it only after completion of execution of the entire list of Routines.

The second approach, dynamic modification of subroutine calling sequences, speeds up the unpacking of data. This arises during the execution of algorithms on picture fragments; the desired picture information must be unpacked from a picture Plane within the word operated upon, and then restored to another Plane. It was desired to make this unpacking as fast as possible, but also it was desired to permit the Routines to be written in Fortran. In Fortran most bitpacking operations, such as unpacking data from within a word, are slow. One alternative frequently taken is to use subroutines to perform these functions; however, frequently the subroutine linkage can take more time for such routines than the actual processing instructions. The approach taken in Picturelab is to provide bit manipulating subroutines, which, on first execution, overwrite the calling sequence with the instructions to be executed and then execute them. Subsequent passages through the subroutine reference result in immediate in-line execution rather than a subroutine call. Naturally, this works only when the variability in subroutine argument values are suitably restricted.

Another approach of potential interest is that of the method used to permit local symbols within Processes. In order to avoid symbol clashes between symbols used in different Processes (recall that these are files of sequences of commands to the Command Processor), it was desirable to permit symbols local to a Process and the capability to hand down symbols not redefined within a Process.

There is an inherently recursive structure to the procedure of unwinding these Processes, however, Fortran is not oriented toward recursive processing. The solution was to take advantage of the stacking capability of SIS. A Map, executing under control of SIS may ask for another Map to be executed, during which time the entire core image of calling Map is placed on top of a Map stack. Upon completion of execution of the called Map its core image is destroyed and the calling Map restored. This restoration process is called peeling back. A Map may call itself, resulting in the placement of the existing core image on the Map stack and the loading of a fresh core image. This may be repeated to any (reasonable) depth.

During the execution of a Process, each time the Command Processor encounters a reference to another Process, it simply writes out the Symbol Table and a small amount of other bookkeeping data, stacks itself and calls in a new image which loads the Symbol Table and data; adds new local symbols to the end of the Symbol Table; and executes the called process. Upon completion of the execution of a Process, the Command Processor removes the local symbols, saves the Symbol Table and bookkeeping data and peels back to the previous version of itself. This approach permits an inherently recursive structure to be handled by a relatively simple Fortran Program.

A final implementation item of potential interest is an immediate execution arithmetic statement processing Routine. All Routines, the programs which operate on picture information in the Data Area, must be compiled outside the SIS system and loaded into the system as absolute core images by means of an off-line run. To permit flexibility during a console session in spite of this requirement, a routine was written which will dynamically compile single assignment statements. These statements permit both logical and arithmetic operators. For example $A \leftarrow (B \geq C)$, $D + (B < = C)$, $(-D)$ places the contents of $D$ into $A$ if $B$ is greater than $C$ otherwise $-D$ is placed in $A$.

Of particular note is the fact that if the variable on the left of the assignment arrow refers to a picture Plane, the result is stored in all the points within the picture; if a Plane is mentioned to the right of the arrow the execution utilizes the corresponding picture data for all the points in the picture in the execution of the statement. This capability coupled with the capacity to loop within a Process provides the system user with a very powerful on-line programming capability for processing picture data. This routine provides a happy compromise between the flexibility of an interpreter and the speed of executable code.

SUMMARY

To summarize, Picturelab is an interactive system intended for use for research in processing digitized pictures. The system contains a Symbol Table for symbolically handling variable information, a Data Area for in-core storage of pictures to be processed, and provision for files containing lists of symbolic commands.

System action is directed by commands entered into a Command Processor. These commands manipulate the Symbol Table, call into execution picture processing Routines, or manipulate files of commands. In addition,
the user may gain access to data on secondary storage through an I/O system which permits reading picture information into and out of the Data Area.

Visual interaction with the picture information in the Data Area is accomplished by means of the SIGHT console and Stromberg Datagraphix Microfilm output.

ACKNOWLEDGMENTS
The Picturelab system is a result of the effort of a number of people. Not included in the list of authors are Mr. K. J. Busch who contributed substantially to the design and Miss K. M. Keller who implemented most of the Command Processor.

REFERENCES
1 K J BUSCH G W R LUDERER
The slave interactive system: A one-user interactive executive grafted on a remote-batch computing system
2 W S BARTLETT K J BUSCH M L FLYNN R L SALMON
SIGHT—A Satellite Interactive Graphic Terminal
Proceedings of 23rd National ACM Conference Brandon Systems Press Inc

APPENDIX
Sample picturelab session

This Appendix presents a sample of the use of Picturelab commands. The operations described below are shown in Table 1 at the end of the Appendix. Lines preceded by a star were printed by the system.

A. Three planes are defined to be global variables. The numbers give the starting bit and number of bits within a 36-bit word. The masks defined are then listed.

B. Control is passed to the I/O system, the Data Area cleared, an input file defined, and the label on the file is printed. A picture fragment of size 128 x 128 starting at row 641, column 636 is requested. The picture is read and control returned to the Command processor.

C. A routine applying a constant threshold to the picture in plane INPLAN is executed with the results going into plane OUTPLA. Then the original picture is displayed followed by the threshold results.

D. A Routine applying an adaptive threshold is executed and the results displayed.

E. Control is passed to the I/O system in order to save the picture on a file, SAVE THRESH, labeled "Results of Thresholding". The values saved are from the plane OUTPLA created by the adaptive thresholding routine.

F. Then a Process is built containing commands to perform an adaptive threshold and then display the results. This is filed under the name ADTHRE.

G. This Process is then executed with a tracing capability enabled to permit monitoring the flow of control.
### TABLE I—Sample Console Session

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>DEFINE PLANES</td>
</tr>
<tr>
<td>2.</td>
<td>ENTER I/O SYSTEM, READ PICTURE FRAME</td>
</tr>
<tr>
<td>3.</td>
<td>APPLY CONSTANT THRESHOLD, DISPLAY</td>
</tr>
<tr>
<td>4.</td>
<td>ADAPTIVE THRESHOLD</td>
</tr>
<tr>
<td>5.</td>
<td>SAVE OUTPUT OF ADAPTIVE THRESHOLD</td>
</tr>
<tr>
<td>6.</td>
<td>BUILD A PROCESS TO DO ADAPTIVE THRESHOLD</td>
</tr>
<tr>
<td>7.</td>
<td>EXECUTE PROCESS</td>
</tr>
<tr>
<td>8.</td>
<td>TERMINATE</td>
</tr>
</tbody>
</table>

---

*PICTURELAB. COMMAND PROCESSOR. TYPE COMMAND, MAP, ROUTINE OR PROCESS NAME*

- **GLOBAL INPLAN PLANE** 31 6
- **GLOBAL OUTPLA PLANE** 15 1
- **GLOBAL PLDISP PLANE** 32 3
- **PIVAL INPLAN OUTPLA PLDISP**

* INPLAN PLANE 31 6 000000000077
* OUTPLA PLANE 15 1 000010000000
* PLDISP PLANE 32 3 000000000034

*I0SYS*

* I/O SYSTEM
  CLEARD
  INPUT 21 1

* ENGINEERING DRAWING #1 FROM MH361
  WINDOW 128 128
  SUBPIC 641 636
  RDPCT INPLAN
  PEEL

*COMMAND PROCESSOR.*

- **THRESH INPLAN OUTPLA** 15
- **DISPLA PLDISP**
- **DISPLA OUTPLA**
- **ADAPT INPLAN OUTPLA**
- **DISPLA OUTPLA**

*I0SYS*

* I/O SYSTEM
  OLABEL “RESULTS OF THRESHOLDING”
  OOUTPUT SAVE THRESH
  WRPC1 OUTPLA
  PEEL

*COMMAND PROCESSOR.*

- **BUILD**
- **PARS PLIN PLOUT**
- **10 ADAPT PLIN PLOUT**
- **20 DISPLA PLOUT**
- **FILE ADTHRE**
- **TRACE ON**
- **ADTHRE INPLAN OUTPLA**

* ENTERING ADAPT
* ENTERING DISPLA
* COMMAND PROCESSOR

PEEL
BYE

H. TERMINATE