Computer Programs in Biomedicine 11 (1980) 21–42
© Elsevier/North-Holland Biomedical Press

BMON2 – A DISTRIBUTED MONITOR SYSTEM FOR BIOLOGICAL IMAGE PROCESSING

Peter LEMKIN and Lewis LIPKIN
Image Processing Unit, Division of Cancer Biology and Diagnosis, National Cancer Institute, National Institutes of Health, Bethesda, MD 20205, USA

This paper presents an example of a distributed monitor system, BMON2, which was developed and is in daily use in a biological image processing environment. Some useful aspects of such a system are discussed, particularly those which make for easier biologist–user interaction and system extensibility. The principles of extension of the distributed monitor to a time-shared computer system is outlined.

Biological Buffer-memory Control-languages Data-acquisition Distributed
Distributed-processing Distributed-system Frame-buffer Image-memory Image-processing
Picture-processing Real-time-picture-processor Software Systems Transforms
BMON2 PDP8 RTPP

1. Introduction

The biologist, employing an interactive image processing system expects (1) an extensive repertoire of image processing and *I/O* procedures which are simply and easily invoked and (2) the ability to more or less freely combine such procedures into complex sequences of operations and interactively apply them to stored images. In pursuing his or her objectives of biologically significant and useful results, the biologist is intolerant of restrictive syntax and fatal errors which return him to ground zero. He is particularly resentful of constraints in the use of procedures which only become apparent when the procedure is part of a sequence. The biologist user tends to want to learn only so much about the system as is necessary for the solution of his problem. It is advantageous if his progression in the interactive development of sequences of effective procedures results in only minimal demands on him for additional system and syntactic information.

BMON2, a general purpose image processing system, evolved in large measure in the response to our experience of the biologist user's needs and behavior. Its repertoire of procedures, and some of its interactive characteristics parallel in part such interactive acquisition/processing systems as TICAS [28], SCANIT [2], and SCANSCANS [1]. Several significant differences, in system philosophy, in system design and in the principles of implementation when taken together make BMON2 quite unusual as well as particularly useful. Of these features, the distributed monitor concept, the concept of major states and substates, very large operators, the ease of addition of external procedures, the role of distributed syntax checking, and the user transparent multilevel image referencing will be presented here in more or less detail. The extension of BMON2 principles to a time sharing environment will essentially complete this paper. References to actually embodied image processing procedures will be minimal and only sufficient to convey the flavor of system operation and its design. Detailed descriptions of these operators may be found in the BMON2 reference manual [10]. A sample of a complex sequence of BMON2 commands will also be given.

1.1. Requirements for image processing of biological images

The general class of biological images covers a wide range of image characteristics. Even within the group of X-rays alone the disparities in say, image contrasts, between a skull series and a plain film of the abdomen is not only visually striking but in part determines the kinds of processing that may be easily attempted. It is truism, of course that the 'menu' of image processing procedures available in a given system in large measure reflects the kinds of images to be analyzed. In that regard, the repertoire of procedures in BMON2 reflects the (largely fortuitous) fact that the images with which we deal (cells and tissues, electron micrographs of nucleic acid strands, and 1- and 2-dimensional electrophoretic gels) are despite their varied origin, all characterized by relatively low contrast and by a moderate to marked degree of complexity. Classes of images such as these are particularly appropriate for strategies involving the stepwise experimental application of successive procedures to achieve a given analytic end.

The scope and range of the procedures that survive (in a Darwinian sense) in a system such as BMON2 also reflect the relative sophistication of the analytic objectives of the users. The same section of neural tissue can be regarded as the subject for a count of neurons in the sample, or on the other hand it may be viewed as one of a set for the establishment of subregions, i.e., the data are viewed in terms of potential correspondences and/or similarities of structure. In our material, establishment of correspondences between similar possibly temporally, spatially or ontogenetic related samples is a common theme among several of the major classes of images with which we deal. Some of the very large operators (see FLICKER and CMPGEL) developed for this system reflect this common interest which transcends a diversity of picture types.

All of those images mentioned are, to repeat, characteristically low contrast, complex images. In designing algorithms to process them, it is useful to be able to experiment with various image processing procedures under step by step control. Such interaction greatly facilitates algorithm development. This type of interaction, however, requires that the development system have an easily learned, simple and logical command structure. It should also reflect the higher level picture operations, data structures, and hardware of the implementation from the user's point of view.

For example, the READ or WRITE commands to move images between hardware image frame buffers and auxillary storage should behave in a generic manner. That is, there should be transparent appropriate default options which should force only minimal if any distinctions to be made by the user between disk and tape devices. As another example, parameters involving image buffer size or computing window (the region of interest within the image buffer) should be implicitly defined and should remain 'sticky', i.e., defined between operations where required.

It is obvious that facilities for procedural sequence construction and execution are even more essential in such a biological millieu. Two major such facilities exist at the primitive level: the assignment of picture operation statements to interactive command keys, and the execution of command sequences using interactive batch processing. For example, a procedure developed for dissecting and editing a previously segmented boundary can be applied, using interactive batch, to sequence and perform the bookkeeping on a large set of boundaries. Lists of images or other data to be processed can first be generated using one procedure and then processed, in turn, using sequences of other operations under interactive batch. As a case in point, in searching for polypeptide spot differences between pairs of 2-dimensional electrophoretic gel images, the sets of pairs of gels to be compared may be generated automatically and the output used to create a batch job which will drive the gel comparison operation [17,18].

The logging of intermediate results is useful in reconstructing successful sequences of operations or for recording measurements while interactively experimenting with image operations. Saving and later using non-image data structures produced by these operators (such as boundaries, lists, scalars etc.) also simplifies concatenating sequences of image operations dependent on these intermediate results.

Often we have found large subsystems are needed which involve combining several image operations in an efficient manner. Such subsets may have been developed either interactively or using sequences of image operators in batch. Combining operations often leads to dramatic increases in the usability of the system. Batch sequences were used in analyzing bone marrow smears [11,13] and in editing nucleic acid boundaries extracted from electron micrographs [14, 19,25]. The combining of a large subset of separate operators into a single multi-use operator is called a very large operator or VLO. In our system, these include operators to perform noise removal [14], boundary analysis and segment splitting [15], interactively drawn boundary extraction, boundary segmentation [11], boundary editing and marking, region isolation [16], and flicker-comparison and analysis of 2-dimensional gel images [17,18].

VLOs gain additional advantage by residing in a general purpose image processing system which they would not have if they were written to stand completely alone. These advantages include the ability to pre- and/or post-process images. VLOs developed in the context of the general purpose system can also take advantage of the generally large number of subroutines and code sequences existing for other operators, which may be used for the composition of the new operators.

One aspect of any general purpose system which is to be used by a number of researchers with different investigative interests is to make it as easy as possible to select a working subset of picture operators necessary for their particular problems. Learning enough of the system to 'get on the air' should also be simplified to encourage its use. Our experience with the BMON2 system has been that interested biologists could learn the minimum subset of picture operators necessary to perform useful work in their area in about one day or less. It should be noted that biologist-users should not be expected to learn the entire system in order to get useful work done, just as programmers using PL/1 do not need to know all aspects of the language in order to write programs in PL/1. A motivated but computer science naïve biologist may learn a viable subset of the system as easily and as rapidly as an experienced programmer.

The physical interface between the biologist's image data and an image analysis system should be relatively fast and simple. Various image forms such as 35 mm film, 4 X 5, 8 X 10 or 11 X 14 sheet film as well as optical microscope images should be easily scanned using the RTPP. A television frame image acquisition system is useful in those cases where image shading errors can be controlled and many frames of data are to be analyzed. Resultant processed images should be available through various image recording media (e.g., Polaroid for quick checking, 35 mm for quality images etc.) as well as being maintained in archival form on magnetic tape.

2. A distributed monitor system

One of the keys to a successful implementation achieving some of the goals just mentioned lies in being able to easily make changes to parts of the system without rebuilding the entire system each time a change is made. Constructing the system as a distributed program rather than a compiled system facilitates achievement of this goal. Such a distributed program would have the speed of a compiled system through the use of small compiled modules rather than through the use of a slower interpreter.

An important aspect of the DMS is the concept of a global state. The global state consists of a set of substates which may be modified at any particular time by one of several separate operators. By state and substate we mean that collection of global variables associated together. The DMS is implemented as a set of chained (overlayed) programs with a disk file copy of the state. The operators are implemented as programs meeting specified criteria such that they may be overlayed (in a manner to be discussed in detail later). Furthermore, these programs may be distributed arbitrarily over the file system of the given machine. One of these programs serves as the master. It communicates with the user command stream and dispatches control to the distributed operators. Figure 1 shows a block diagram illustrating the DMS concept. A distributed operator may itself be distributed. The primary constraints are that a distributed operator be started by and return to the master program, and that any state changes be instantiated in the disk backup copy of the state. In fig. 2, distributed operator Oi actually consists of the sequence of operators Oi1, Oi2, ..., Oi*m*. The last operator, Oi*m*, performs the possible state backup on disk and returns to the master program.

A batch operating system (such as the DEC OS8 [4]) is an example of a DMS. The programs to be run are the set {Oi} while the batch program corresponds to the master program. The state consists of a pointer to the current position in the batch input stream.

A major constraint of a DMS is that all operator programs Oi, although they may chain to additional programs (e.g., in fig. 2, Oi1 → Oi2 → ... → Oi*m*), must return to the DMS master program when they are finished. During the passage of control between operators and suboperators, changes in state must be main-

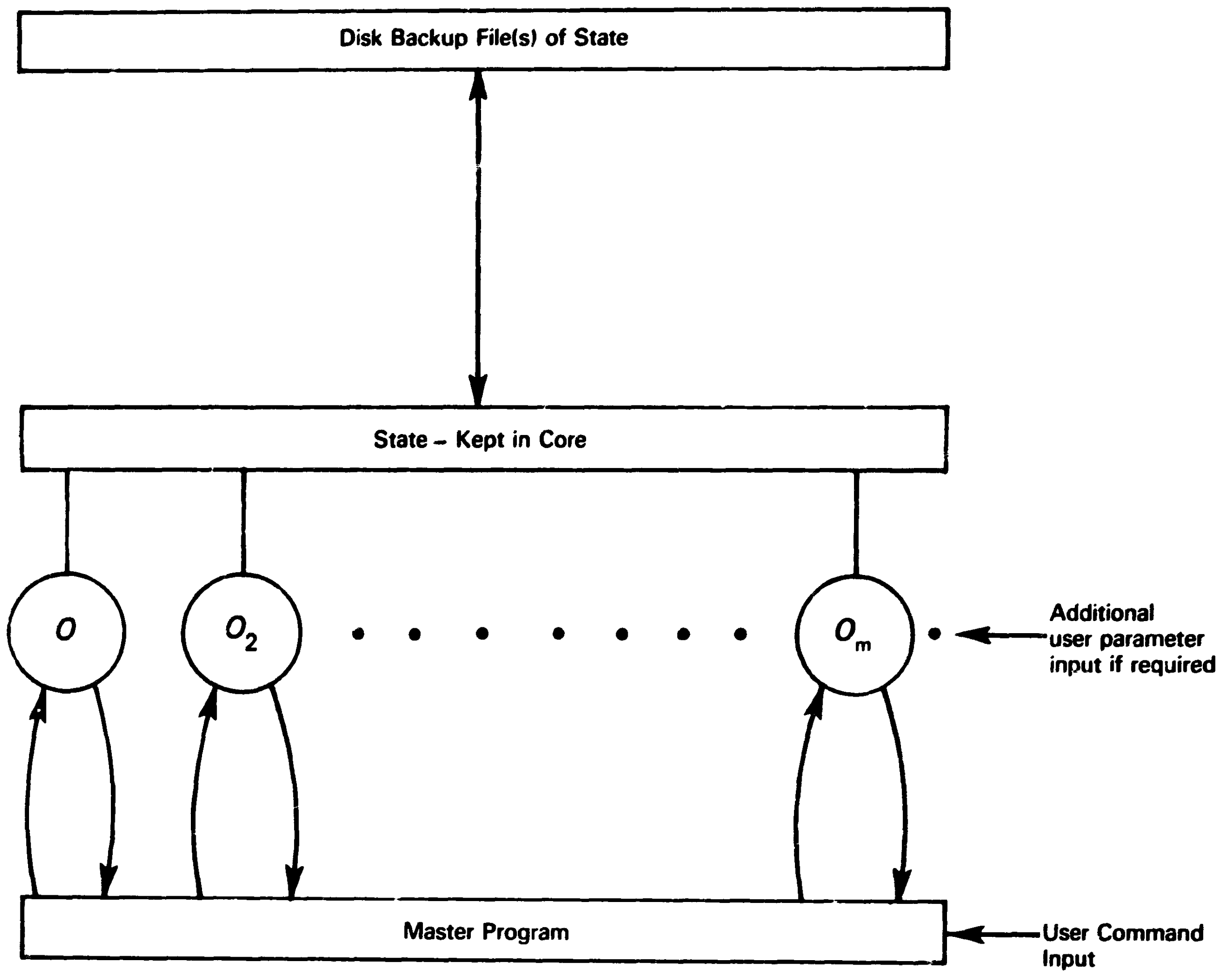
Disk Backup File(s) of State
State – Kept in Core
O
O2
Om
Additional user parameter input if required
Master Program
User Command Input

Fig. 1. Block diagram of a DMS. The master program receives messages from the user input stream. It then either executes the command itself or directs a distributed operator to execute it. The state in core is backed up when required on the disk. Additional user input may be required by the distributed operators. Control eventually returns to the master progam where it waits for another command to be sent.

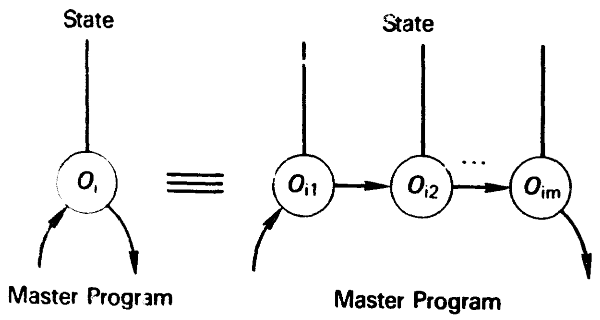

Fig 2. A distributed operator may itself be distributed. The primary constraints are that it be started by and return to the master program, and any state changes be instantiated in the disk backup copy of the state.

tained in the bookkeeping of the system. The master's convention is to (1) backup the state to a disk file before starting an external operator, and (2) restore the state from the disk file upon being restarted by an external operator upon its completion. It is necessary then, that an external operator changing the state must save the changed state on the disk when it is finished before restarting the master program.

2.1. Monitor state and distributed modules

The global state of a system is a set of instantiated parameters accessible by all parts of that system.

Many parts of the system not needing knowledge of the entire global state are restricted to access only a substate (a substate being a unique subset of these parameters). This access can be rigidly enforced or, less desirably, controlled by agreement when writing operator programs.

The global state should be capable of initialization either in part or in its entirety. This has the distinct advantage of repairing possible damage caused by a misguided external operator without destroying the entire state by reinitialization.

Such a global state concept is not new to computer science and has in fact been used in artificial intelligence research for some time [20]. The state has been commonly thought of as a blackboard where operators requiring state information may look if necessary and those generating information may place messages. Those operators not requiring the state however are not bothered by its existence.

There is a need to extend and delete the type of information contained in a state (i.e., its scope) over time as a system matures. Therefore, facilities must be included to allow this expansion and contraction to occur. Dynamic state variation therefore allows new operators with new substate requirements to be easily added. Major new substates can be added by using a subset tree of pointers and attributes in the global state. Such a tree can be started in the global state (so that it can always be found) and then traced as far as possible in the state found in core. Further pointers can then direct to the rest of the tree or the substates themselves as disk files. Removing a substate merely means finding its attribute and deleting the entry from the tree.

It is of interest that from both the view of the application programmer and even more significantly, the biological user, this places minimal requirements on what must be learned in order to add or use a new function. This tends obviously to make for greater biologist acceptability.

2.2. Operating system requirements imposed by a DMS

The CPU and core memory requirements for a DMS are modest while the disk space required depends on the number of external operators. As will be demonstrated in section 3, a very large DMS can be built on a machine as simple as a 12-bit word size PDP8 with 32 k 12-bit words of core memory and 1–6 M words of disk storage running under DEC's OS8 operating system [4]. Several features of a host monitor system are required for the successful implementation of a DMS. The most important is a disk file system that can access files whose names and devices are specified during run time. Another requirement is to be able to specify a program to be loaded and started (i.e., run) from the currently running program. This is sometimes called chaining or dynamic overlaying. In order to run a sequence of programs under batch, there must be a mechanism for intercepting batch stream data from a program. (In OS8, this mechanism is called the command decoder.) Thus the actual restrictions are minimal and should present no major difficulty in implementing a DMS on most computer systems.

2.3. Operators – built in and external

Operators in a DMS consist of two types: built in and external operators. Built in operators are physically part of the master program and externals are all others. Since built in operators will execute faster not having to be loaded and run as separate programs, care should be taken in selecting those operators to be made 'built ins'. Several informal 'rules' have evolved in our work which we use to decide whether a needed function is to be built in.

1. The operator should be small. Large operators will waste master program space.

2. The operator should be invoked fairly often. The additional overhead of making a rarely used operator external should not greatly contribute to average system overhead.

3. The execution time of the operator should be short. If the execution time is long, the additional time to evaluate it if it is an external operator would not contribute greatly to the system overhead.

4. There already exists a built in operator similar to the new operator for which most of the code can be shared resulting in minimal code overhead.

In a continually evolving system such as BMON2, a working set of built in operators is eventually produced by tuning the system through using it. Our working set evolved over several months with some additions and a few deletions to the working set of built in operators. After 4 years of using BMON2, we are still adding very large operators as well as new operators, but the list of 'built ins' has stabilized long since.

2.3.1. Secondary parsing of commands in distributed operators

Commands are generally received by the master program which parses it. Further parsing of a command may be required by particular operators, in which case secondary parsing is performed. This semantic checking phase verifies parameters expected against those actually specified for the distributed operator in the distributed operator itself. Furthermore, the operator may request additional user input which in turn must be parsed. By distributing the parsing of user input the complexity of the master program parser can be reduced but at the expense of some duplication of parser code in the distributed operator modules.

3. Current implementation – BMON2

3.1. BMON2 buffer memory hardware and software facilities

The BMON2 system will now be described in terms of the DMS concepts developed above. The particular hardware used by this system is discussed as well as the structure of the BMON2 monitor itself. The computer used is a DEC PDP8e with ~6 M 12-bit words of disk memory and 32 k 12-bit word core memory. It operates under DEC's OS8 single user (non-interrupt driven) operating system [4]. Image frame buffer memory is part of the real time picture processor (RTPP) [4–6]. The RTPP consists primarily of a PDP8e computer, a 11 frame/s TV display camera subsystem built around an IMANCO Quantimet 720 system, image frame buffer memories, and an interactive control desk. Another version of the RTPP has been constructed around a 27 frame/s CRT display with a 512 X 512 viewable area and without TV camera input. Figure 3 is a block diagram of the RTPP. Figure 4 shows the physical placement of the TV display and interactive control desk. Figure 5 illustrates the keys and switches of the control desk.

3.1.1. Structure of buffer memory frame buffer hardware

The RTPP buffer memory hardware consists of eight 256 X 256 arrays of 16-bit pixels (which may be used as two 8-bit pixels, designated as low and high byte, as well). The memories are addressed as BM0 through BM7. The 8-bit high image in a BM is addressed as BMiH and the low image as BMiL or BMi. Physically, each BM consists of 4 wirewrapped boards each, holding sixty-four 4 k-bits dynamic RAMs.

The BM controller interface allows multiple device requests to be made of the memories in the following manner: Vidicon or Plumbicon TV camera video to BM: BM to TV display; PDP8e memory to/from BM. Analog video from the TV camera is digitized at

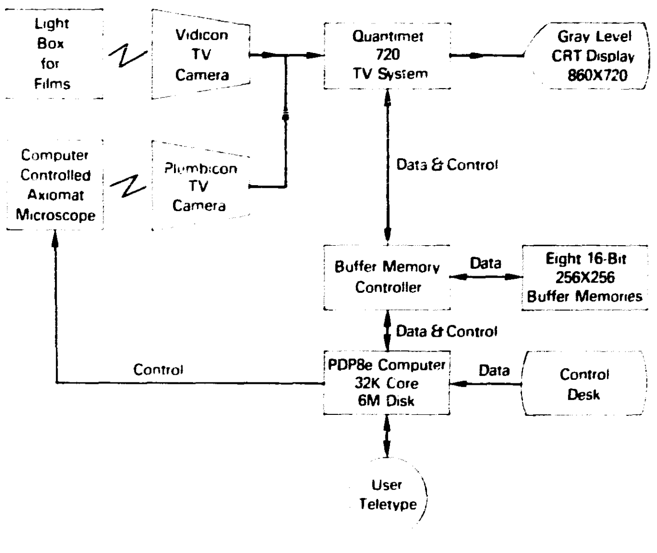

Fig. 3. Block diagram of the RTPP. The PDP8e computer directs the microscope state to positions determined either manually by the operator. Images may be acquired by the buffer memories for processing by the BMON2 system. Raw images as well as processed images may be displayed on the Quantimet 720 CRT display. TV camera input is from one of the two alternate TV cameras which are easily interchanged in <2 min. The user interacts with all of the above hardware via the PDP8e using the BMON2 image processing software system. RTPP = real time picture processor.

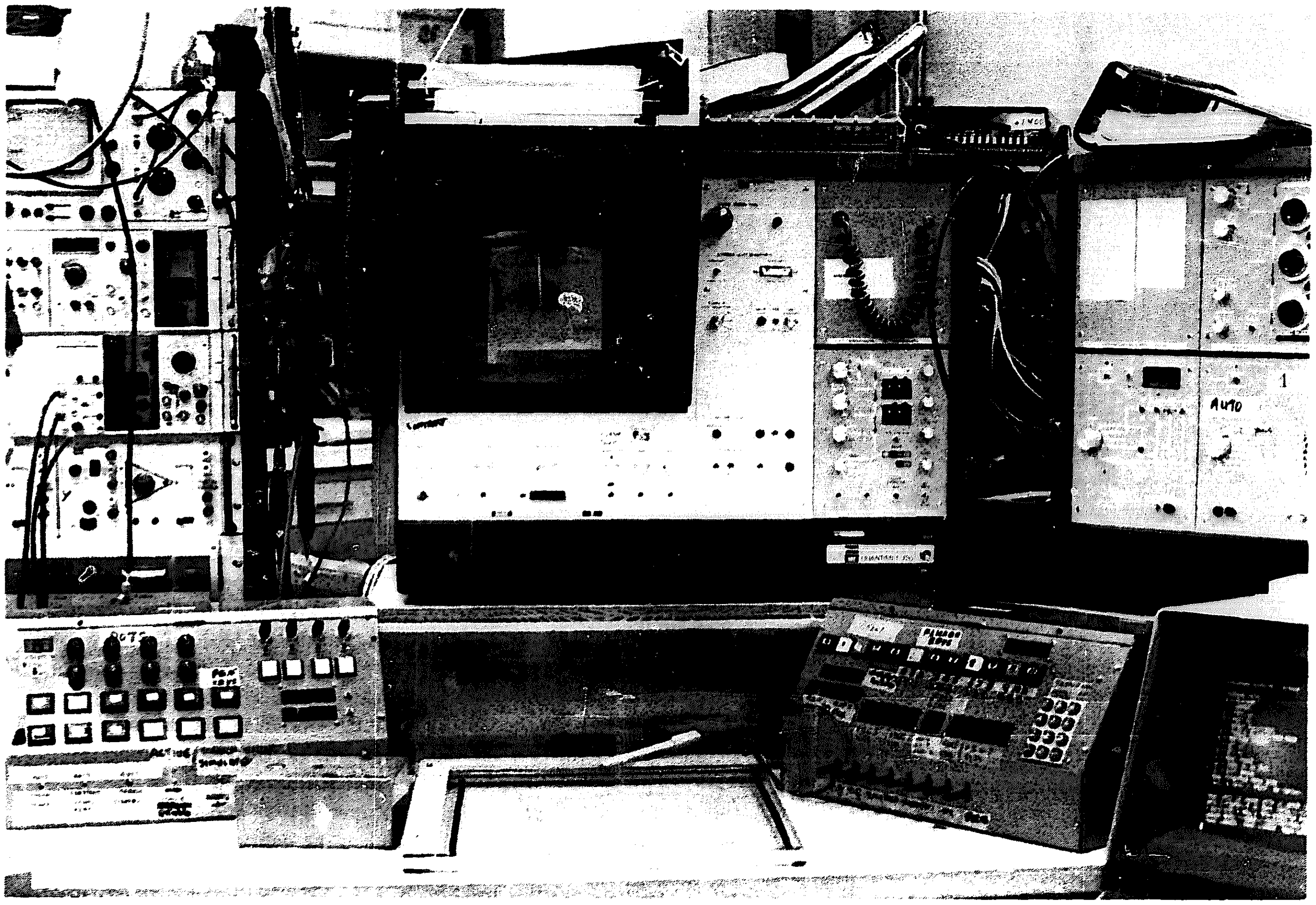

Fig. 4. Photograph of the TV display and interactive control desk.

8 Mhz rate to 8-bits. This digitized video is assembled into 4 pixel chunks which may then be written into the BMs. If BM data is to be displayed and/or normal camera video is to be displayed, then the 8-bit data multiplexed from the two sources are then converted back to analog and sent to the TV display controller. Because the BM cycles every 500 ns and the TV camera/display pixel rate is 125 ns, BM data is transfered in 4 pixel chunks and buffered accordingly. Thus random accessing of pixels takes a minimum of 500 ns instead of the 125 ns possible when transferring data in raster mode.

The pixel addressing system is the RTPP logical coordinate system (LCS). The LCS has (0, 0) as the upper left-hand corner (x, y) coordinates and (1023, 1023) as the lower right-hand corner. The visible screen size is effectively [0:860, 0:680]. Each BM is positioned independently in the LCS. Normally, the images are positioned adjacent to each other. Individual BMs or sets of BMs may be posted on the TV monitor independently, with either high or low bytes being shown. Addressing conflicts in BMs which overlap in LCS space are resolved with a hardwired priority network such that BM0 is displayed before BM1, BM1 before BM2 etc. The normally undisplayed byte of the BM having the gray scale image being displayed may be displayed as a binary overlay. This is useful for implementing line drawings. Furthermore, the PDP8e can synchronize with the start of each new display frame in order to decide on the correct time to change the BMs being displayed (the display state).

Pixel data may be transferred at 2.4 µs/pixel using direct memory access with the PDP8e. Data are transferred in either of 4 packed modes: low byte/pixel, high byte/pixel, 16-bit pixel, and 16-bit unpacked with sign extension for performing arithmetic. A 21-

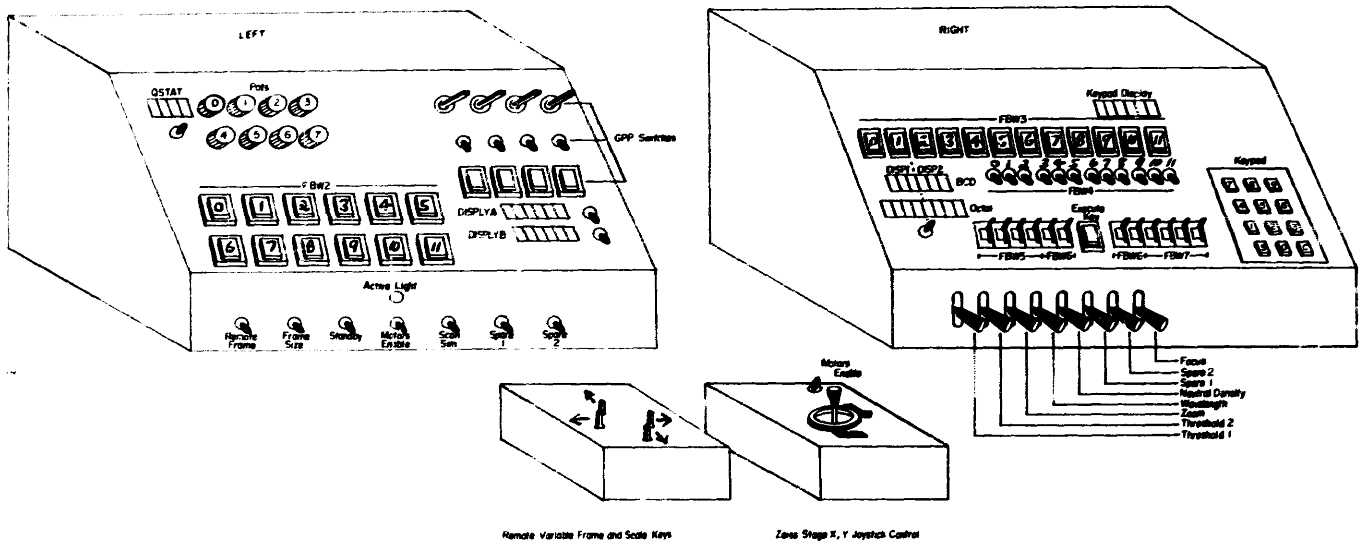

Fig. 5. Illustration of the RTPP interactive control desk keys and switches. The RTPP interactive control desk is situated next to the RTPP with the Quantimet-720 video display to the rear of the control desk and the Axiomat microscope off to the side. A Graphpen spark tablet is located immediately in front of the operator with the pushbuttons and lights mounted in two large boxes to the left and right. The remote Quantimet variable frame and scale keys are located in a small box with a movable cable as is a joystick for the Zeiss Axiomat (x, y) stepping stage. The latter has a long cable and may be used at the microscope for control of the stage while viewing through the eyepiece of the microscope. The control desk controls are listed as follows going from left to right and top to bottom (for the left box first): the QSTAT lights indicate the status of the Quantimet interface; the pots are connected to A/D channels in the PDP8e; FBW2 are lit 'command' keys for the PDP8e; the remote frame switch enables the remote frame and scale switches even when the PDP8e has not enabled them; the frame size switch freezes the frame and scale sizes so that a frame of fixed size may be moved around; the standby switch places the Quantimet display and system control in standby mode; the motors enable switch (also in joystick box) enables the stepping motors when the light above it is on. For the right control box, the controls are: keypad display of keypad input for the PDP8e; FBW3 'classification' keys for the PDP8e; DISP1/2 PDP8e display lights which are decoded as BCD in the top lights and as octal in the bottom lights; FBW4 PDP8e toggle switches; keypad to input 6 BCD digits to the PDP8e; FBW5/6/7 PDP8e octal digiswitches; Execute key used to execute (interpretively by the PDP8e) instructions given in the digiswitches; eight 5-position spring loaded toggle switches to control various stepping motors with fast and slow speeds in both forward and reverse directions.

bit address is needed to specify the transfer. The state variables specifying these parameters are also specified.

	Byte select	BM number	*Y* address	*X* address
	2-bits	3-bits	8-bits	8-bits
Fortran COMMON state variable	*I*BYTE	MEM	*IY*	*IX*

3.1.2. Structure of buffer memory software addressing

Although it is possible to code a picture operator to perform BM *I/O* directly, this is discouraged. Instead, two subroutine packages were developed to do this: BMIO and BMOMNI [8]. BMIO maximizes data rates but requires detailed knowledge of the system state variables in COMMON, while BMOMNI is for the naïve programmer, and requires little detailed system knowledge. BMIO consists of a set of Fortran procedures with parameters and data being passed through COMMON state variables. BMOMNI is called with an operation request number and all parameters and data are passed as subroutine arguments. BMOMNI actually contains redundant copies of BMIO subroutines as well as controlling >30 other RTPP *I/O* devices such as switches, display cursor, computing window display etc. The various *I/O* modes and subroutine entry names are listed in

Table 1
Picture memory I/O accessing procedures

a. Point or pixel byte mode: PACK2D/FETCH2D.
```
   COMMON args: [MEM, IBYTE, IY, IX, IZ].
   COMMON data: [IZ]
```
b. Line byte mode: T3BUF (buffer, *opr*).
```
   opr = 0/1 to read/write 3 lines packed
   opr = 2/3 to read/write 1 line unpacked.
   COMMON args: [MEM, IBYTE, IY]
   data: [buffer, opr]
```
c. 3 X 3 neighborhood byte mode using triple line buffering: GETI1.
```
   COMMON args: [MEM, IBYTE, IY, IX].
   COMMON data: [I10, I11, ..., I18] as:
     I13 I12 I11  (x - 1,y - 1) (x,y - 1)  (x + 1,y - 1)
     I14 I18 I10  (x - 1,y)     (x,y)      (x + 1,y)
     I15 I16 I17  (x - 1,y + 1) (x,y + 1)  (x + 1,y + 1)
   centered at I18 (x,y)
```
d. 3 X 3 neigborhood byte mode using fresh copy buffering: GETNGH.
```
   COMMON args: [MEM, IBYTE, IY, IX].
   COMMON data: [I10, I11, ..., I18] as:
     I13 I12 I11  (x - 1,y - 1) (x,y - 1)  (x + 1,y - 1)
     I14 I18 I10  (x - 1,y)     (x,y)      (x + 1,y)
     I15 I16 I17  (x - 1,y + 1) (x,y + 1)  (x + 1,y + 1)
   centered at I18 (x,y)
```
e. 512 pixel/line byte mode is simulated using 4 BMs and the T3BUF procedure in single line mode. The BMs are aligned as:
```
   BM0 BM1 or BM4 BM5 or BM0H BM1H or BM4H BM5H
   BM2 BM3    BM6 BM7    BM2H BM3H    BM6H BM7H
```

The following algorithm is used to perform the 512 to 256 mapping based on input line IY1:
```
Procedure IO512(IY1);
Begin '512 pixel line I/O'
DIMENSION IBUF[0 : 511];
IF IY1 > 255 THEN MEM: = 2 ELSE MEM: = 0;
IY: = IY1 Λ'377;
T3BUF(IBUF[0], opr);
MEM: = MEM + 1;
T3BUF(IBUF[256], opr);
End '512 pixel line I/O';
```

table 1 with the names of parameters passed through Fortran COMMON.

There is no reason why BMOMNI could not be profitably employed by a user. This has not occurred, which may reflect either the sufficiency of the repertoire or the unconcern of biologists with system problems. Perhaps both of these coupled with a flexible, biologically concerned programming staff is the reason.

3.1.3. Levels of accessing an image

An image may be processed at several levels of resolution. These include, from highest to lowest levels:
(a) The entire display;
(b) Within the entire display defining at least one BM;
(c) Within a BM defining a computing window;
(d) Within the computing window defining a cursor to point to a specific pixel.
Useful operator interaction requires that programs and users be able to articulate each of these image levels.

3.2. Software architecture

3.2.1. Major operator groups

The set of BMON2 operators may be more easily understood if they are grouped according to function. The major groups are listed in table 2 with the detailed operator lists given in appendix A. By grouping common operators together, the programmer can often take advantage of procedures common to several operators within a group.

Table 2
Operator groups

1. Image display and acquisition operators
2. Image input/output operators
3. Control desk
4. State initialization and state inquiry
5. Auxilliary programming
6. Synthetic image operators
7. Unary image point operators
8. Neighborhood unary image operators
9. Point binary image operators
10. Statistical display operators
11. Line drawing operators
12. Segmentation operators
13. Scalar measurement operators
14. Quantimet function operators
15. Texture measurement operators
16. 3 Dimensional reconstruction operators

3.2.2. Skeletons – structures for constructing new operators

A skeleton is the structure within which a new external operator may be constructed. It consists of all pre- and post-operator procedures necessary to smoothly interface a new operator with the BMON2 system. The structure of a skeleton is given here in outline form as algorithm A.

Algorithm A. Operator skeleton procedure

1. Print the name of the operator.
2. Do any additional syntax or semantic checking of the command line parse required for the particular operator. For example, BMs which must be present need to be checked for. If the specification is incorrect, then print a message and take the error return back to BMON2 (see section 3.3.5).

2.1. Restore the state in COMMON. (Not normally required.)

3. Evaluate the kernel of the operator.
4. Save COMMON in the state disk files only if there was any change to the state.
5. Chain back to BMON2 via the OS8 monitor.

BMON2 itself was constructed so that the kernel procedures required for the up to 64 built-in operators were included in the main program. These procedures were implemented by a set of 9 subinterpreter subroutines, auxillary subroutine packages BMAX1 through BMAX9. Each in turn can then be used to evaluate the selected function as required by passing the function number desired through a COMMON variable (IVAL). For example, the BMAX3 package contains the following functions listed in table 3. These auxillary functions may also be used in composition with other software presently in the existing external operators to create new operators.

3.2.3. Software state of BMON2

We have previously compared the software state of a DMS to a blackboard where messages may be left for various procedures which may require them. The BMON2 software state includes some of the following substate variables listed in table 4 which are instantiated in Fortran COMMON. The BMON2 state is saved on two disk files SVDDTG.DA and SVBMON.DA consisting of a partition of COMMON. When BMON2 is started, the state is restored from these files and when BMON2 chains to an external operator, the state is saved in these state files. External operators which change the state are required to save the new state in the state files before returning (via chaining) to BMON2. A parameterized subroutine BSCOMMON is called to swap COMMON to the disk for saving and restoring the state.

Table 3
Example of auxillary functions in procedure BMAX3.FT

IVAL	Function
1	BMj = HISTOGRAM(BMi)
2	Setup the computing window (KX1 : KX2, KY1 : KY2)
3	BMj = EDGE(BMi, threshold 1, threshold 2)
4	BMj = AVG8(BMi)
5	Q-register = EVAL(arg 1; arg 2; switches for add, subtract, etc.)
6	BMj = GRAYBAR
7	BMj = LAPLACE(BMi)
8	BMj = GRAD4(BMi)
9	BMj = SHOWHISTOGRAM
10	BMj = FILLPINHOLES(BMi)

3.2.4. BMON2 syntax

Commands are normally entered one per line and are prompted for by BMON2 printing a '*'. Some typical examples of command lines are given here to show the flavor of the language before its syntax is discussed in detail.

```
*INIT
*POST, BM0
*BM1 – COPY, BM0
*BM2 – SLICE, BM0, 150, 255
*BM2 – SLICE, BM0, P0
*BM3H – GRAYBAR/1
*BM2 – COLOR, 128
*BM5 – READ, MTA0:LL0027.PX/R
*SETGENSYM, LL, 0028
*MTA1:GENSYM.PX – WRITE, BM7
*BM3 – BM1, ADD, BM2
*BM4 – BM1, DIFF/U, BM2
*BM5 – GRAD4, BM2
```

The general form of a command always includes

Table 4
Some often used state variables

1. Parse state variables:
   - MCD[1 : 36] – unparsed command decoder input buffer
   - KOUTFILE[1 : 4] – output symbol name from parser
   - KINFILE[1 : 5, 1 : 4] – 5 input symbol names from parser
   - ICNUM[1 : 5] – 5 input symbols parsed as numbers from parser
   - ISW[1 : 36] – input binary switches, eg.,/S from parser
   - KDEVOUT[1] – output device name from parser
   - KDEVIN[1 : 5] – input device names from parser
   - IBM1, IHGH1 – 1st input BM number and byte from parser
   - IBM2, IHGH2 – 2nd BM number and byte from parser
   - JBM, JHGH – output BM number and byte from parser
   - IMA, IMB – 1st BM post bit pattern from parser
2. Buffer memory position variables:
   - LSAVE[x : y, BMO : BM7] – positions on in the display
   - IPSTA, IPSTB – BM A and B group post status words
   - IXPOSITION, IYPOSITION – current cursor
   - KX1 : KX2, KY1 : KY2 – computing window in LCS
3. Buffer memory addressing variables:
   - IX, IY, IZ, MEM, IBYTE – BM *I/O* address parameters
   - KX, KY, IX1, IX2, IY1, IY2 – free addressing variables
   - LASTY – last y address used in triple line buffering
   - LASTBM – last MEM address in triple line buffering
   - IBUF[1 : 4, 0 : 255] – assignable line in buffers
   - I10[1 : 9] eqv. I10. I11, I12, I13, I14, I15, I16, I17, I18 – 3 X 3 neighborhood
4. Extended state of other variables for stepping motor states:
   - MDPDATA[1 : 8, 1 : 12] – 12 stepping motor double precision substates i
     - [1 : 2, i]lower movement limit
     - [3 : 4, i]upper movement limit
     - [5 : 6, i]desired position limit
     - [7 : 8, i]current position limit
   - MSLOW[1 : 12] – slow stepping motor rates
   - MFAST[1 :12] – fast stepping motor rates
   - MACTIVE[bits 0 : 11] – 12 stepping motor active bits
5. Twenty-six Q-registers and other arithmetic variables:
   - ITMPSTK[1 : 26] – integer Q-registers low order
   - IQREG[1 : 26] – integer Q-registers high order
   - GENSYM[1 : 2] – file name generation variables
   - IA[1 : 2], IC[1 : 2] – free variables
   - FA, FB, FC – free variables
   - IDMAX, IDMIN – free variables
   - IVAL[1 : 2] – free variables

an operator. It may include one or more operands which may be buffer memory names and other operands depending on the particular operator.

The BMON2 command line syntax given in table 5 in the form of a backus normal form (BNF) grammar. Note that the '–' is the underline character and is equivalent to a back arrow on some teletypes. It is used to indicate assignment. The '<' character is equivalent to the '–' in the OS8 environment. Furthermore, if no output specification is required '–right side of specification' is equivalent to 'right side of specification'.

Buffer memories are specified as '**BM***nh*' where *n* is

Table 5
BMON2 BNF grammar

```
⟨cmd line⟩:: = ⟨cmd⟩ ⟨switches⟩
⟨cmd⟩:: = ⟨op⟩, ⟨args⟩
      :: = ⟨filespec⟩ _ ⟨op⟩, ⟨args⟩
      :: = ⟨filespec⟩ _ ⟨op⟩, ⟨BM⟩
      :: = ⟨BM⟩ _ ⟨op⟩, ⟨filespec⟩
      :: = ⟨op⟩, ⟨BM⟩, ⟨args⟩
      :: = ⟨BM⟩ _ ⟨op⟩, ⟨args⟩
      :: = ⟨op⟩, ⟨BM⟩, ⟨BM⟩, ⟨args⟩
      :: = ⟨BM⟩, ⟨op⟩, ⟨BM⟩, ⟨args⟩
      :: = ⟨BM⟩ _ ⟨op⟩, ⟨BM⟩, ⟨BM⟩, ⟨args⟩
      :: = ⟨BM⟩ _ ⟨BM⟩, ⟨op⟩, ⟨BM⟩, ⟨args⟩
⟨filespec⟩:: = ⟨device⟩ : ⟨file name⟩ . ⟨extension name⟩
⟨device⟩:: = SYS|DSKB|DSKC|DSKD|DSKE|DSKF|DSKG|
   DSKH|DTAO|DTA1|LPT
⟨file name⟩:: = GENSYM|(6 character alpha-numeric identi-
   fier beginning with letter)
⟨extension name⟩:: = (2 character alpha-numeric identifier)
⟨args⟩:: = ⟨args⟩, ⟨arg⟩|⟨arg⟩
⟨arg⟩:: = decimal number up to 4095
     :: = ⟨knob⟩
     :: = ⟨control desk switch⟩
     :: = ⟨keypad⟩
     :: = ⟨Q-register⟩
     :: = null
⟨knob⟩:: = P⟨octal digit⟩ (Knobs 0 : 7 with values [0 : 511])
⟨control desk switch⟩:: = FBW⟨decimal digit⟩ (control desk
   switches 0 : 9 with values [0 : 4095])
⟨keypad⟩:: = KDP (keypad values [0 : 999])
⟨Q-register⟩:: = QR⟨letter⟩
⟨letter⟩:: = A|B|C| ... |Y|Z
⟨op⟩:: = legal command
⟨BM⟩:: = BM⟨octal digit⟩⟨byte⟩
⟨octal digit⟩:: = 0|1|2|3|4|5|6|7
⟨decimal digit⟩:: = ⟨octal digit⟩ |8|9
⟨byte⟩:: = H|L|null
⟨switches⟩:: = /⟨letter⟩|/⟨octal digit⟩|
           :: = /S = ⟨2 digit octal number⟩
```

the buffer memory number and *h* denotes an optional byte selector (*h* = null or 'L' for low byte, *h* = 'H' for high byte). The notation ⟨BMi'⟩ (*h'* is the complement of *h*) denotes the other half of ⟨BMi⟩. So called software ⟨switches⟩ are denoted by '/' followed by an alpha-numeric character. Other parameters, denoted ⟨args⟩, may also be specified. Note that ⟨args⟩ includes decimal numbers up to 4095, Q-registers (see below), physical devices such as: the 8 knob pots (Pi), the control desk switches FBW1:9, and the keypad KPD (numbers 0:999). Twenty-six specially named registers called Q-registers are available for the operators (or the user) to pass integer parameters between operations. The syntax for the registers is 'QR⟨letter⟩'. The special symbol GENSYM may be used instead of a file name to indicate that BMON2 should generate a file name when referenced by incrementing a 4 digit postfix number appended to a two character prefix of the OS8 file name. A parser restriction prevents the specification of both decimal integers and other types of ⟨args⟩ in the same command line. This is actually not that difficult a restriction since integers may be stored in Q-registers.

3.3. Control structure

3.3.1. Top level BMON2 flow – command interpreter

The top level BMON2 control flow is shown in fig. 6. Upon starting, the program restores the COMMON state from the disk state files. It then enters a command input loop where operator intervention is tested. Commands entered as teletype strings are parsed (c.f., sections 3.2.2:3) and then interpreted (c.f., section 3.3.4).

3.3.2. Command parser

The BMON2 command parser is shown in fig. 7. The parser is embedded in the OS8 system by using its command decoder option. The command decoder specifies a command as an output name followed by a – followed by up to 5 input names. Software switches (as defined in section 3.2.4) may be used to modify and/or parameterize the command line.

3.3.3. Results of parsing a command line

A command line can contain the following information which is analyzed by the command decoder

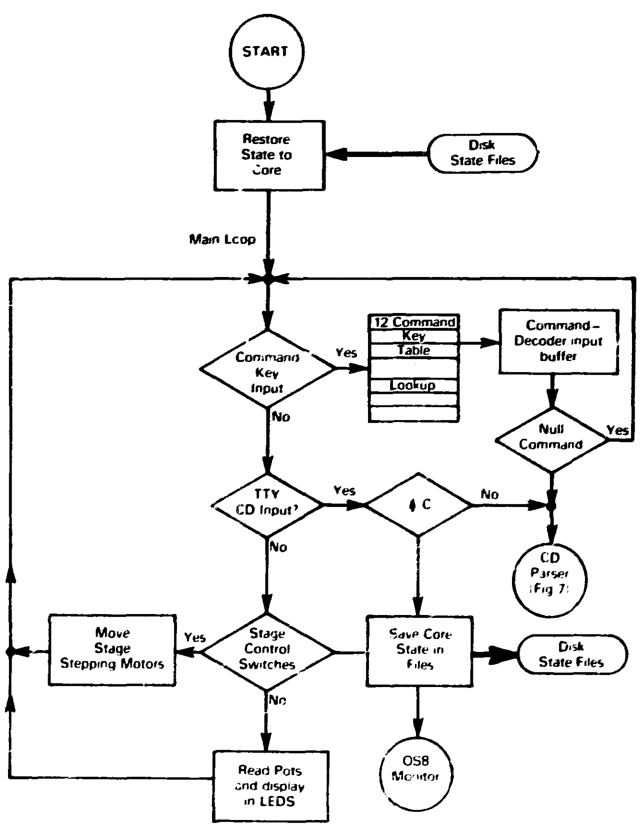

Fig. 6. Top level BMON2 control flow.

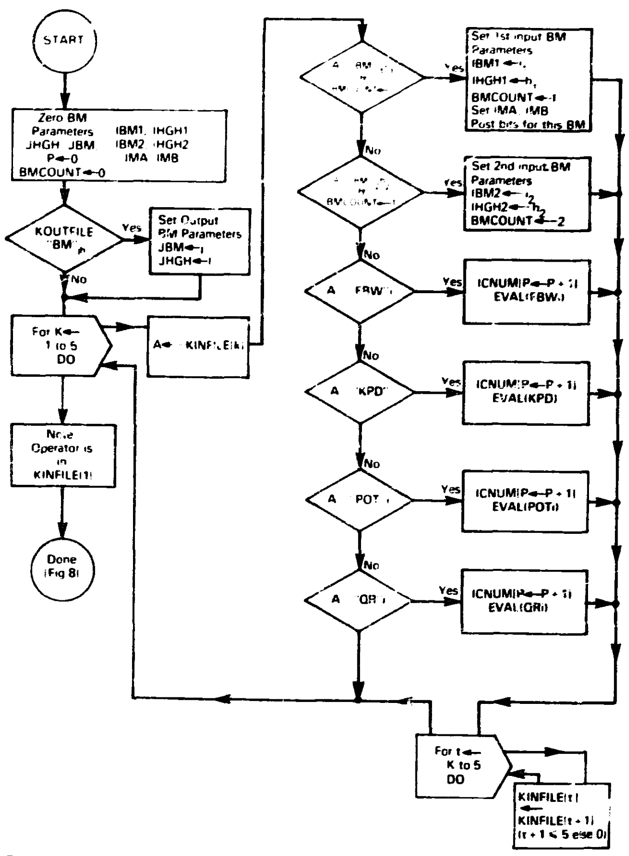
START
Zero BM Parameters IBM1, IHGH1 JHGH JBM IBM2 IHGH2 P←0 IMA IMB BMCOUNT←0
KOUTFILE "BM"?
Yes
No
Set Output BM Parameters JBM←i JHGH←i
For K← 1 to 5 DO
A← KINFILE(K)
None Operator is in KINFILE(1)
Done (Fig 8)
Set 1st input BM Parameters IBM1←i, IHGH1←h, BMCOUNT←1 Set IMA, IMB Post bits for this BM
Set 2nd input BM Parameters IBM2←i, IHGH2←h, BMCOUNT←2
A "FBW"?
ICNUM(P←P+1) EVAL(FBW)
A "KPD"?
ICNUM(P←P+1) EVAL(KPD)
A "POT"?
ICNUM(P←P+1) EVAL(POT)
A "QR"?
ICNUM(P←P+1) EVAL(QR)
For I← K to 5 DO
KINFILE(I) ← KINFILE(I+1) (I+1 ≤ 5 else 0)

Fig. 7. BMON2 command parser flow.

parser subroutine BCDSPEC.FT ('B'MON2 'C'ommand 'D'ecoder 'SPEC'ification).

1. One output argument (denoted by a '–' to its right).
2. Up to 5 input arguments separated by commas.
3. Up to 36 switches (denoted as /A, /B, ..., /Z, /0, /1, ..., /9).
4. Control desk command key assignment value specified as '=*nn*/S' where *nn* is the command key name 0–14 octal.

This information is parsed and appears as the following Fortran COMMON variables:

1. KDEVOUT OS8 device, KOUTFILE[1:4] as 4A2 format.
2. KDEVIN[1:5] OS8 devices, KINFILE[1:5, 1:4] as 5(4A2) format, or ICNUM[1:5] input arguments which have values [0:4095].
3. ISW[1:36] as 0 or 1.
4. MCD[39] as *nn* value.
5. MCD[1:36] – initial command string in 36A2 format.

A few examples will be given here to illustrate the characteristics of the command decoder parser.

Z – A, 3, B, 4

or

Z – 3, 4, A, B

are parsed into the global parse variable arrays:

Index	KOUTFILE	KINFILE	ICNUM
1	Z	A	3
2		B	4

Thus it is clear that the position of numeric arguments is not critical and that they may be mixed with non-numeric arguments. Syntax insensitivity is thus traded off against possible ambiguity which however is never allowed to produce a fatal error.

Further parsing is performed. The KOUTFILE symbol is checked to see whether it contains a symbol 'BM⟨digit⟩'. State variable JBM contains the digit (0 if none) and JHGH contains 1 if the symbol ended in an 'H' as in 'BM3H' indicating high byte. Similarly, the 5 KINFILE symbols are checked from left to right for 'BM⟨digit⟩' symbols. If one is found, it is put into (IBM1, IHGH1) as for (JBM, JHGH). The symbol is then removed from the KINFILE list and the list compressed. This process is repeated up to one more time in order to find the possible second input operand in (IBM2, IHGH2). The KINFILE list is then searched left to right for symbols which represent scalar values such as pots P0, P1, ..., P7; switches FBW1, FBW2, ..., FBW7; the keypad KPD and Q-registers QRA, QRB, ..., QRZ. If a symbol is found, it is removed as before from the KINFILE list and the corresponding device or register evaluated and the value stacked in the ICNUM list. Finally, the symbol left in the KINFILE[1, 1:4] array (leftmost symbol) will by definition be the operator. The parser thus reduces the command line to an operator prefix form (as in the LISP language) for easy evaluation. For example, in the following command where pots 0 and 1 have the values 123 and 234 respectively, the global parse variables arrays are defined as:

BM2H–SLICE, BM1, P0, P1

Index	KOUTFILE	KINFILE	ICNUM
1	⟨null⟩	SLICE	123
		⟨null⟩	234

JBM/JHGH	IBM1/IHGH1
2/1	1/0

Other state variable changes are noted such as checking whether the variable frame TV overlay is within the first input BM if the /U switch is specified. The variables (KX1:KX2, KY1:KY2) are set to the relative computing window values. Otherwise, they are defaulted to (0:255,0:255).

3.3.4. Interpreter – built in/external evaluation

Figure 8 shows the control flow of the BMON2 interpreter. External operators such as SEGBND,

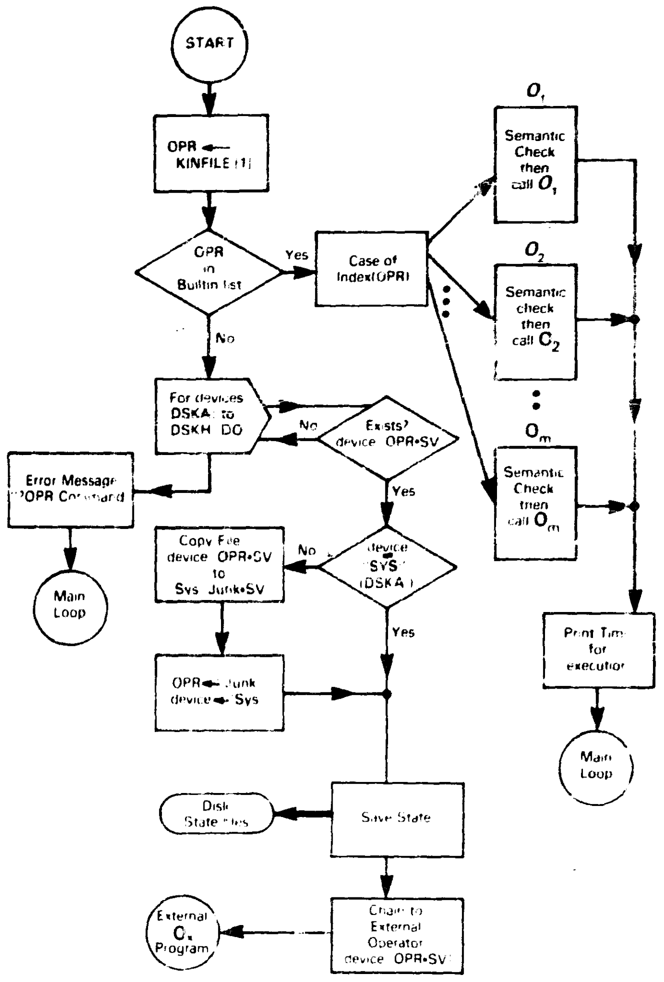

Fig. 8. BMON2 interpreter flow.

EXTRACT, ZOOM etc. are implemented as chained OS8 '.SV' core image files. When one of these is called, the state of BMON2 (including the parsed command line) is saved in system (SYS:) disk files (SVDDTG.DA, SVBMON.DA) before the chain is executed. The operator segment, on being started, has the option of restoring the state of BMON2 from these files or assuming that the state is left in COMMON. It then uses the parsed argument specifications in COMMON. After it performs the operation, it has the option of saving the new state of BMON2. It then chains back to BMON2. The chain operation is performed by BMON2 as follows in algorithm B.

Algorithm B. Operator evaluation

1. The operator is checked against a list of internal BMON2 operators. If it is an internal operator, it is executed within BMON2.
2. An unknown operator (potential CHAIN operator) X is looked up on the SYS: as 'SYS:X.SV'.
2.1. If it is found, then the state of BMON2 is saved, and the system chains to SYS:X.SV.
2.2. If X is not found, then BMON2 searches the rest of the disks in the following order: DSKB, DSKC, DSKD, DSKE, DSKF, DSKG, and DSKH. Any disks off line are not searched.
2.2.1. If it is found on any of these disks then X.SV is copied to SYS:JUNK.SV file and the system chains to SYS:JUNK.SV.
2.2.2. If X is not found, an error message '?X.SV' is printed and control returns to wait for the next command at the BMON2 level.

This mechanism thus permits an operator to be added or deleted from the BMON2 system by simply adding to (removing) from the PDP8e computer system disk packs with the operator program segments on them. (The system has a total of 4 dual surface disk drives.)

3.3.5. Distributed syntax checking

A major advantage of the DMS concept is that the peculiar syntax and semantic needs of a particular operator can be built into the operator itself. In particular, requirements as to BM specifications, disk usage, etc. may be checked at the operator level. Three internal subroutines are commonly used by many of the external operators; i.e., CKOUT. CKIN, and CKIN2. The first checks for an output BM specification by parsing the output symbol for 'BM' followed by a digit. CKIN does the same for the first input BM symbol. CKIN2 does a CKIN and checks for the existence of the next BM input symbol. For example, applying CKIN2 and CKOUT to the parse string

'BM2 _ BMO, ADD, BM1'

would be successful. Additional semantic checks may be made, as for example checking whether the input BM name is different from the output BM name, etc.

3.4. Adding new external operators to BMON2

Once again, our implementation of the DMS concept makes the addition of new external operators to the BMON2 system, easy and with few if any unforseen complications. This feature permits programs (such as BMON2 or the external operators themselves) to run other programs by requesting through the OS8 monitor. The external operators are programs, OS8 core image (OS8 .SV extension) files, created by compiling, loading and saving the programs. As noted in section 3.2.3 the critical component of interfacing these new operator programs to BMON2 is in saving and restoring the state information and the operator programs themselves.

Most new operators are created by taking the skeleton from an existing procedure of a similar type and changing the kernel of the procedure to perform the new operation (c.f., section 3.2.2). Each external operator has a prologue to restore the state and an epilogue to save it and chain back to BMON2. The external operator uses the COMMON declaration file, BMCMN.FT, during compilation so that any operations requiring COMMON will have it available. Actual BM *I/O* may be performed either using BMIO.FT or BMOMNI.FT (c.f., section 3.1.2).

3.5. The class of macro expansion batch operators

There has evolved a special calss of BMON2 operators quite different from the usual picture operator, the macro expansion batch operator. It is characterized by its use; namely to expand a set of parameters into a batch job for running BMON2. Currently 3 operators are in this group: APPLY, MAKLST and MAKCMP.

APPLY takes a template batch job and a set of parameters and does a simple parameter replacement and batch job submission. MAKLST is used to restore to the 3 scratch disks a list of 2-dimensional electrophoresis gel images from a set of magnetic tapes. It uses information stored in a data management file, GEL.DA, regarding picture names and magtape volumes where gel images are stored indexed by gel accession number. MAKCMP is also used in the 2-dimensional gel analysis to generate batch jobs for defining landmark spot sets (a landmark spot in a 2-dimensional gel is a well defined polypeptide spot identifiable in several gels). It is also used to pair spots between gels for a number of gels taken two at a time in a batch mode.

4. Annotated example of a BMON2 run

Appendix B shows the BMON2 operation sequence used in a batch job to acquire and preprocess an electron micrograph of nucleic acid molecules. The object of this sequence is to: acquire a field from the TV/micrograph input system; preprocess it preparatory to segmentation; segment it into a file consisting of a set of boundaries; post process this file into a set of edited boundary data files which are then saved on magnetic tape.

Steps 1–3 define the sample window to be scanned. Step 4 acquires the image at 2X magnification and then smoothes it by averaging and reducing it to 1X size. Step 5 complements the image so that the strands are black (since we are working with negative film input. Step 6 low pass filters the image to remove the effect of shading error in the image. Steps 7–9 find the threshold and slice the image such that the image consists primarily of strands and noisy blobs. Major gap repair is done where possible in step 10 using an interactive Graphpen to edit the image. Small non-linear blobs are removed in step 11. The image is segmented in step 12 and separate boundary data files created with marked endpoints in step 13. Finally the boundary data files are saved on magnetic tape in step 14.

5. Discussion

5.1. How DMS implementation helps image processing user

In solving a problem, one often breaks larger unsolved problems into smaller solvable problems. A DMS permits an investigator to create and test, independent of the requirements and state expectations of the rest of the system operators, new operators in an attempt to solve these subproblems. Because of this independence, creating new operators is relatively easy. By accumulating experience with sequences of such operators, more compact and efficient 'repackaged' operators embodying the same sequence of operators can then be created, which may and frequently do evolve into what we have termed very large operators.

5.1.1. Minimum 'window' of system knowledge required for extension

The software interface between the DMS and a particular operator is known as the window (somewhat similar to the concept of gateway used when discussing spheres of protection in time sharing systems). It may be thought of as two funnels connected through the narrow ends to each other. The window actually consists of a subset of the state variables. A minimum window size is desirable in order to maximize the ease of understanding this interface. By partitioning the state into substates, only those substates actually used need be understood. This requires an understanding of the classes of substates which exist and only then detailed knowledge of those of interest. At the user level again, this strategem minimizes the demand on the user to learn new system features in order to use the new operator.

5.1.2. Concept of levels of attention – substates

Because each operator requires only a subset of the total state and because these substates are often able to be partitioned, substates can be treated both conceptually and physically as separate entities. This leads to relatively simple and efficient data structure implementations. A further advantage is the small number of state variables a programmer has to contend with in a substate. This makes learning and using a particular substate relatively easy.

5.2. Extensibility of DMS concept to time-shared systems.

There is nothing inherent in the DMS design concept which prevents its being used in a time shared environment. In fact we are in the process of building such a structure. The DMS concept should be easily transferred to a time shared system such as for example the DECSYSTEM-10 or -20. The DMS master program, (i.e. 'BMON20'), should be a shared re-entrant program. Each user would create, upon first using BMON20, his own copy of the major state file in his own disk file region. Thus multiple users could use the system simultaneously. As operators were invoked by users which require additional substate files, these files would be created and that user's major state file modified to reflect this change.

Searching for the external operator program files could then be done in a more extensive environment. For example, BMON20 might first search the user's disk file structure for the special operator. If this search fails, it might then search a special DMS system-wide structure for the operator. A useful extension to the command syntax might be to employ an external operator from a specific user's disk area. This facility would let several users try an operator under development before it is finally put into the DMS file area for general use.

As an added benefit of a DMS, unnecessarily large programs modules consisting of a set of operators can be broken down into smaller modules and still be linked together in a reasonable way through the DMS. The effect of developing a working set of these smaller modules reduces physical core demands although at the added expense of more paging and disk storage. The paging expense however can be held to reasonable limits by careful design of the modules.

5.2.1. Substate implementation and its implications

Since it is impossible to completely define the major state file during the initial system design, a facility must be built into the system to allow the extension of the state during system development. This is done by setting aside a region of the state space itself which points to the extensions of the state resident on disk files. These extensions could consist of lists of 2-tuples: (substate file name, substate attribute). Part of the attribute list would be the name of the author of the new substate, in order to uniquely define it. Such an attribute could then be used to uniquely label the substate, enabling an operator requiring the substate to find it. Thus a user not requiring many substates would have a minimum number of auxillary substate files and minimum overhead. Furthermore, each user would have a unique set of these substate files tailored to his particular use of the system. Overhead in accessing substates could be reduced by maintaining a demand paging region in the state for the current substate.

Flexibility in substate structures allows for 'lean' but understandable major states. Passing the major state between DMS programs could be done either through disk files or through one of several interprocess communication mechanisms available for passing messages through core memory (e.g., in TOPS-10 systems TMPCOR files, IPCF, pseudo teletypes). In either case, the state must eventually be backed up permanently on disk files. An appropriate time to do this is when there is a 'significant' state change affecting long term operation, such as when a new substate is created or deleted.

5.2.2. Requirements for allocation of frame buffer resources

As was discussed in section 3.1, certain types of picture memory *I/O* operations are quite often used. A similar situation occurs in attempting to perform picture processing in a time shared environment. However, further complications arise because of the allocation of precious frame buffer hardware resources. One solution is to restrict image processing on the system only to the user currently 'owning' the frame buffer resource. Clearly, this is an inadequate solution. A better solution would be to require all frame buffer interaction to proceed through a 'gateway' procedure which would (1) in the case of owning the frame buffer, use it directly, or (2) in the case of not owning the frame buffer, simulate it as a set of disk files. We believe that the latter solution would enable many more users to take advantage of the system without extravagant use of system disk resources. This 'gateway' procedure could also be responsible for communicating with the hardware frame buffer's own CPU for performing offline image operations.

This gateway program would appear to be similar to BMOMNI in BMON2 but with the various necessary extensions required for a time shared environment. Among these would be the ability to ASSIGN and DEASSIGN (as in the DECSYSTEM-10 monitor command) a buffer memory handler and RTPP controller. Once assigned, a device could not be accessed by other users even though no program may be currently using it (as is the case which occurs during chaining between DMS modules). Return of a device to the available pool may be made automatic or as an explicit courtesy to other users.

In summary, the types of frame buffer *I/O* for both the byte and word size pixels typically involved are: pixel; neighborhood; line; image; boundary list. The neighborhood *I/O* should allow the user to define a neighborhood (typically $n \times n$ square) and have the system set up n-line buffers (and maintain them) to do the actual *I/O*. The latter should be transparent to the user. As an extension, a direction list neighborhood definition could be specified, as well as giving offsets relative to a center with the center specified as absolute coordinates. This would permit arbitrary neighborhood definition. The packing of images in core should be convenient for rapid access for pixel/line operations. This may necessitate maintaining two sets of line buffers in core, one packed and the other unpacked.

If several non-refresh type displays are available to time sharing, it should be possible to use these to interrogate the picture memory. Although not as desirable for most applications as a refresh gray scale display, they would permit wider use of the system. This interaction can be facilitated by adding a new operator, SHOW, with the following syntax.

```
display type – SHOW, BM, (display window)
```

The display window may be defaulted but is certainly a function of the type of display used and its gray scale simulation mechanism.

Another useful change would be to allow the mixing of picture files with BMs in the command specification. For example, the following sequence can be replaced with a much simpler one if display of the data is not required.

```
BM0 – READ, A.PIX
BM1 – READ, B.PIX
BM2 – BM0, ADD, BM1
C.PIX – WRITE, BM2
```

may be replaced by:

```
C.PIX – A.PIX, ADD, B.PIX
```

5.2.3. Extensions to the syntax

There are no inherent limitations to extending the syntax. However, the side effects of any extension should be carefully weighed. For example, the addition of other data structures (such as lists) to the state would either have the effect of enlarging the state (bad for routine chaining) or of requiring their accession through a substate (less efficient during usage but probably preferable over the long term). Algol style control block-structure type syntax would be desirable but must be implemented in such a way so as to minimize the problems just discussed, possibly using a control stack. Since the operators are not known until evaluation time, any control stack data structure might contain the actual string names as an efficient coding mechanism.

Another useful extension would be to allocate a small number (possibly on the order of 30–50) user defined scalar variables to replace the Q-registers which are difficult to remember. The new syntax might to be preface the symbol with a '%' as in '%AREA'. A fixed region in the major state area could be allocated for the set of 2-tuples (%variable, scalar value) constituting the user variables.

6. Conclusion

We have detailed an instance of a distributed picture processing monitor system. The advantages of a DMS for interactive use by non-programmer users are indicated. The further advantages of a distributed monitor system in both adding and maintaining large numbers of large new operators are enumerated. The extension of the distributed monitor system concept to a time sharing environment in the context of a large main frame is outlined.

Acknowledgements

We wish to thank Bruce Shapiro, Earl Smith and Morton Schultz for useful discussions on the DMS concepts and their extension to the time share environment as well as useful criticism of the sections on BMON2.

Appendix A – Lists of BMON2 operators by group

The set of BMON2 operators are most easily understood if they are functionally grouped. In the list, those followed by a '*' are external operators while the others are built into BMON2 proper. The function of a given individual operator is often apparent from its name. Detailed description of its operation is found in the user's manual [10]. Those operators whose algorithms were obtained from specific papers are referenced as such.

1. Image display and acquisition operators:
   - GET – acquire TV images into buffer memories
   - SMPGET* – acquire 512 X 512 image sampled/averaged to 256 X 256 BMs
   - FSTGET* – acquire 16 BM sequential images at current F and S
   - POST – post BMs on the TV display
   - UNPOST – unpost BMs from the TV display
   - POSXYBM – position a BM by (x, y) position on the display
   - POSFSBM – position a BM by window position on the display
   - SETFSXY – set the window position by (x, y) values
   - SETFSBM – position the window over the specified BM
   - SETFSREL – position the window relative from one BM to another
   - FINDFS* – find the minimum enclosing window in a BM
   - STDBM – set the standard BM positions on the display
   - ALL384 – set all BM positions to the center of the LCS
   - SHOWMOVIE* – show a specified sequence of BMs repeatedly
   - GELMVI* – show two 512 X 512 images with camera control

2. Image input/output operators:
   - WRITE* – write BM(s) to OS8 devices
   - READ* – read BM(s) from OS8 devices
   - WINDMP* – print the decimal values of the BM window
   - MAG10* [7] – magtape file utility
   - PIXMTA* – BM(s) data acquisition to magtape
   - REVIEW* – magtape image files to BM(s)
   - BNDPRINT* [15] – boundary analysis and display
   - CAMERA* – automatic camera control

3. Control desk:
   - CMDKEYS – print command key assignments
   - SAVCMD – save current command key assignment in a file
   - RSTCMD – restore command key assignment from a file

4. State initialization and state inquiry:
   INIT – initialize the BMON2 state or substates
   EXIT – exit BMON2 back to OS8 saving the state
   PARAMETERS – print state or substate parameters
   SETGENSYM – define the file name generator
   LOADQR – load a scalar into a Q-register
   EVAL – do scalar arithmetic
   HELP* – search a document file for command information
   OPENFILE – start spooling BMON2 output
   CLOSEFILE – stop spooling BMON2 output
   MOVSTATE* – move the microscope stepping motors by value

5. Auxilliary programming:
   BATCH – submit an OS8 batch job from BMON2
   NOBATCH – turn off OS8 batch if it is on
   APPLY* – create a batch job from a macro file and parameters
   SETIOT – execute the PDP8e *I/O* instruction with parameter

6. Synthetic image operators:
   COLOR – assign a buffer memory a specified gray value
   ZERO – zero the buffer memory
   GRAYBAR – fill a BM with a graybar (16 step log or 256)
   TEXT – draw a teletype specified text message in a BM
   GRID – draw a grid of size $N \times N$ color G in a BM
   WHITENOISE – color a BM with white noise

7. Unary image point operators:
   COPY – copy a BM into
   COMPLEMENT – complement a BM
   CONTRAST – contrast stretch a BM
   DEFCONTRAST* – draw a contrast function with the Graphpen
   FNCONTRAST* – apply the contrast function to a BM
   SCALE – scale a BM using a linear transformation
   SLICE – threshold slice a BM
   SHIFT – translate a BM in (x, y)
   ROTATE* – rotate a BM a specified angle

8. Neighborhood unary image operators:
   ZOOM* – magnify a window in BM(s) by repeating pixels
   AVG8 – 8-neighbor average a BM window
   AVGN* – $N \times N$ average a BM window
   MIDPOINT* – 3 X 3 midpoint filter a BM window
   MEDIAN* – 3 X 3 median filter a BM window
   LAPLACIAN – 3 X 3 Laplacian filter a BM window
   GRAD4 [27] – 4-neighbor gradient of a BM window
   EDGE [27] – 3 X 3 edge filter a BM window
   GRADN* – $N \times N$ neighborhood gradient a BM window
   MTV* [23] – 3 X 3 MTV filter a BM window
   VARIANCE* – 3 X 3 variance filter a BM window
   GRAD8* – 8-neighbor gradient of a BM window
   FILTER* – direction list filter a BM window
   FILLPINHOLES – fill pin holes in a BM window
   CIRCLE – copy a circular BM window
   RECTANGLE – copy a rectangular BM window
   PROPAGATE* – propagate a BM window
   PROP2* – propagate a BM window
   RUNFILTER* – run length filter a BM window
   NGHSE* – gray scale shrink/expand a BM window
   NOTCH* [19] – notch filter a BM window
   NCH512* [19] – notch filter a 512 X 512 BM window
   FILGAP* – gap fill a BM window

9. Point binary image operators:
   ADD – pixel by pixel add two BM windows
   SUB – pixel by pixel subtract two BM windows
   MUL – pixel by pixel multiply two BM windows
   DIV – pixel by pixel divide two BM windows
   AND – pixel by pixel bit-AND two BM windows
   OR – pixel by pixel inclusive bit-OR two BM windows
   MAX – pixel by pixel maximum of two BM windows
   MIN – pixel by pixel minimum of two BM windows
   DIFF – pixel by pixel absolute difference of two BM windows
   ISOLATE [16]* – region isolation of connected component image and gray scale image
   SHADE* [24] – shade correct a BM window

10. Statistical display operators:
 HIST – compute the gray scale distribution of a BM window
 SHOWHISTOGRAM – display gray scale distribution in a BM
 SMOOTHISTOGRAM* [26] – smooth a gray scale distribution
 PLOT2D* – plot two BM windows in a third BM

11. Line drawing operators:
 GRAPHPEN – edit a BM with a Graphpen
 EXTRACT* – extract BM measurements with drawn boundaries
 DRW512* – edit a 512 X 512 BM image with Graphpen
 BDEDIT* [25] – edit a boundary data file overlaying a BM image
 BTT* [12] – compute the boundary trace transform in a BM

12. Segmentation operators:
 SEGBND* [11,22] – segment a BM into a set of boundaries and a connected component (CC) BM image
 SEG2PS* [22] – segment a BM window into a CC BM image
 SEG512* [18,22] – segment a 512 X 512 BM window into a CC image
 RMVBLOB* [14] – remove compact blobs less than size N
 DYNBND* – dynamic boundary follower

13. Scalar measurement operators:
 AREA – compute area of a BM window
 DENSITY – compute the density of a BM window
 PERIMETER – compute the perimeter of objects above threshold in a BM window
 SUMDIFF – compute the sum of the differences of two BMs
 COMASS* – compute the center of mass of the BM window data
 PTILE* [22] – compute the N'th percentile of current histogram
 TOTDENSITY* [18] – compute the total density and background of 512 X 512 image

14. Quantimet function operators:
 QDATA – acquire data from the QMT function computer hardware
 LOADTHRESHOLDS – load the detector module threshold values

15. Texture measurement operators:
 RLTEXTURE* [5] – compute run length texture features of BM window
 JGSTXTURE* [21] – compute joint gray scale texture features of a BM window
 JGSPLOT* [21] – compute and display joint gray scale texture features of BM window

16. 3-Dimensional reconstruction operators:
 RECONSTRUCT* – reconstruct projection from serial sections

17. Image comparison operators:
 FLICKER* [17] – flicker analysis of 2-dimensional gel images
 SCANSPOT* [18] – canonical spot data management system
 MAKLST* [18] – image accession number data management system
 CMPGEL* [18] – gel spot iist comparison system
 MAKCMP* – generate batch jobs for CMPGEL use

Appendix B – Sample BMON2 run

The following batch job will acquire and preprocess a number of nucleic acid molecules from electron micrographs of film. The result is a set of boundary data files suitable as input to a molecule analysis program [19,25] running on a DEC10.

```
$JOB MOLECULES.BI – ACQUIRE, PREPROCESS
   NUCLEIC ACID MOLECULES
.R BMON2
/1. Initialize system and unpost any images currently
   posted
*INIT/A
/
/2. Set frame size to 512 X 512
*SETFSXY,1,1,512,512
/
$MSG – POSITION THE SAMPLE FRAME OF
   IMAGE TO BE ACQUIRED
/3. Position BMs at frame currently set interactively
```

```
*INIT/B
/
/4. Acquire a 512 X 512 image at the current window
/and average it to a 256 X 256
*BM0–SMPGET/A
/
/5. Complement the image into itself
*BM0–COMPLEMENT, BM0
/
/6. Low pass filter the image to remove most of the
   shading
/error using a 32 X 32 window in a notch filter and
/compute the lowest background value automatically
*BM3–NOTCH,32/A
/
/7. Compute the gray scale distribution of the image
   and
/display the distribution in BM1
*BM1–HISTOGRAM,BM3
/
/8. Compute the 75'th percentile of density values of
   the
/histogram which will then be used to generate a
   noisy thresholded
/image of the molecules
*PTILE,75
/Note: The gray value corresponding to the 75'th
   percentile
/is stored in QRC
/
/9. Slice the image in BM3 back into BM0 at thresh-
   old QRC
/through 255 (black)
*BM0–SLICE, BM3, QRC
/
/10. Do gross image editing. Fill major gaps if
   required
*BM0–GRAPHPEN
/
/11. Remove blobs <20 pixels in radius in 2 passes
/Leave larger blobs intact
*BM2–RMVBLOBS,BM0,10
*BM0–COPY,BM2
*BM2–RMVBLOBS,BM0,10
/
/12. Segment the image into separate molecules
   boundaries
/First set up the boundary data file name generator to
/generate the boundary data file (BDF) name
   BD0001
/Size objects by perimeter in the range of 150:2000
   pixels
*SETGENSYM,BD,0
*BM1–SEGBND,BM2,150,2000/T/L/N/B/1
/
/13. Edit a single BDF list of boundaries (removing
   hairs
/and marking endpoints) into a set of BDFs
   BS0001.DA,
/BS0002.DA, etc.
/Each time a boundary for the BD0001.DA BDF is
   edited,
/it is overlayed in black in BM0 on top of a fresh
   copy of
/the gray scale original image in BM2
*BM0–BDEDIT,BD0001.DA,BS,0001,BM2
/Note: The endproduct is a set of marked molecule
   boundaries on
/the disk
/
/14. Dump the final set of boundaries on magtape
   and then
/delete them from the disk
$MSG – MOUNT MAGTAPE: REWIND, PUT
   ON-LINE, RING IN, ON UNIT 0
*MTA0:–MAG10,BS????.DA/D, BMON2
$END MOLECULES.BI
```

References

[1] E. Bengtsson, J. Holmquist, B. Olsen and B. Stenkvist, Comput. Progr. Biomed. 6 (1976) 39.

[2] J.F. Brenner, S.B. Dew, J.B. Horton, T. King, P.W. Neurath and W.D. Selles, J. Histochem. Cytochem. 24 (1976) 100.

[3] G. Carman, P. Lemkin, L. Lipkin, B. Shapiro, M. Schultz and P. Kaiser, J. Histochem. Cytochem. 22 (1974) 732.

[4] Digital Equipment Corp., OS8 Handbook (Maynard, MA, 1974).

[5] M. Galloway, Comput. Graph. Image Process. 4 (1975) 172.

[6] P. Lemkin, G. Carman, L. Lipkin, B. Shapiro, M. Schultz and P. Kaiser, J. Histochem. Cytochem. 22 (1974) 725.

[7] P. Lemkin, MAG10 – A PDP8e file based magtape utility. NCI/IP Tech. Rep. 20 (Nat. Tech. Info. Serv. PB261534/AS (or DECUS 8-879), Dec. 1976).

[8] P. Lemkin, BMOMNI - Fortran interface program for the RTPP buffer memory, quantimet and control desk, NCI/IP Tech. Rep. 23 (Nat. Tech. Info. Serv. PB261538/AS, Dec. 1976).
[9] P. Lemkin, G. Carman, L. Lipkin, B. Shapiro and M. Schultz, Real time picture processor - Description and specification. NCI/IP Tech. Rep. 7a (Nat. Tech. Info. Serv. PB269600/AS, June 1977).
[10] P. Lemkin, Buffer memory monitor system for interactive image processing. NCI/IP Tech. Rep. 21b (Nat. Tech. Info. Serv. PB269642/AS, June 1977).
[11] P. Lemkin, Bone marrow smear image analysis (PhD Dis. Univ. Maryland, College Park, MD, 1978).
[12] P. Lemkin, Comput. Graph. Image Process. 9 (1979) 150.
[13] P. Lemkin and L. Lipkin, Anal. Quant. Cytol. J. 1 (1979) 67.
[14] P. Lemkin, B. Shapiro, L. Lipkin, J. Maizel and J. Sklansky, Preprocessing of electron micrographs of nucleic acid molecules for automatic analysis of computer. II. Noise removal and gap filling, Comput. Biomed. Res. (1980) in press.
[15] P. Lemkin and L. Lipkin, Splitting touching nuclei in clusters of bone marrow smear cell images, Anal. Quant. Cytol. J. (1980) in press.
[16] P. Lemkin, Comput. Graph. Image Process. 10 (1979) 281.
[17] P. Lemkin, C. Merril, L. Lipkin, M. Van Keuren, W. Oertel, B. Shapiro, M. Wade, M. Schultz and E. Smith, Software aids for the analysis of 2-D gel electrophoresis images, Comput. Biomed. Res. (1980) in press.
[18] P. Lemkin, L. Lipkin, C. Merril and S. Shifrin, Protein abnormalities in macrophages bearing asbestos, Env. Health Persp. (1980) submitted.
[19] L. Lipkin, P. Lemkin, B. Shapiro and J. Sklansky, Comput. Biomed. Res. 12 (1979) 279.
[20] N. Nilsson, Problem-solving methods in artificial intelligence (McGraw Hill, NY, 1971).
[21] N.J. Pressman, Optical texture analysis for automatic cytology and histology: A Markovian approach (PhD Diss., Lawrence Livermore Lab., UCLA, Report UCRL-52155, Oct. 1976).
[22] A. Rosenfeld and A. Kak, Digital picture processing (Academic Press, NY, 1976).
[23] B. Schachter, L. Davis and A. Rosenfeld, Some experiments in image segmentation by clustering of local feature values. Pattern Recog. (1980) submitted.
[24] M. Schultz, L. Lipkin, M. Wade, P. Lemkin and G. Carman, J. Histochem. Cytochem. 22 (1974) 751.
[25] B. Shapiro, Biological shape description (PhD Diss. Univ. Maryland, College Park, MD, 1978).
[26] K.T. Smith, D.C. Solmon and S.L. Wagner, Practical and mathematical aspects of the problem of reconstructing objects from radiographs (Address to Far West Section Meet. Am. Math. Soc., Monterey, CA, April 1975).
[27] H. Wechsler and J. Sklansky, Automatic detection of rib contours in chest radiographs. 4th Int. Joint Conf. Artif. Intel. (1975) 688.
[28] G. Wied, G.F. Bahr and P.H. Bartels, Automatic analysis of cell images by TICAS, in: Automated cell identification and cell sorting, Wied, G. et al. ed. (Academic Press, NY, 1970).