CPO-2/K-202:
A UNIVERSAL DIGITAL IMAGE ANALYSIS SYSTEM

by

Zenon KULPA, Janusz DERNAŁOWICZ,
Henryk T. NOWICKI*), Andrzej BIELIK

Polish Academy of Sciences
Institute of Biocybernetics and Biomedical Engineering
Department of Picture Recognition and Processing
00-818 WARSAW, Poland

*) Presently at:
Monument-Hospital "Center of Child's Health"
Department of Genetics
WARSAW-MIĘDZYLESIE, Poland

Abstract

Great universality and flexibility of an automatic picture analysis is most easily reached by the use of a computer-based picture processing system. A digital picture processing system designed, built up and utilized in the Institute of Biocybernetics and Biomedical Engineering is an example of such a system. It is very useful as a research tool to investigate methods of an analysis of pictures as well as it is used for many practical applications of this methods in science and technology.

The CPO-2/K-202 system consists of a minicomputer system and a special picture input/output device. The special pictorial peripheral, named CPO-2, for digital image conversion and for input/output of pictures to/from the computer system contains: a TV-camera and TV monitors, A/D and D/A converters, and an image buffer memory. The computer system comprises Polish K-202 minicomputer, a standard set of I/O peripherals and an appropriate software system for image processing. The software includes an assembler, a large library of image processing subroutines called PICASSO, and a PICASSO-SHOW family of interactive programming languages for picture processing.

Several application programs were written for the system, e.g. for analysis of pictures of leukemia cells, radar cloud covers, chromosome banding patterns as well as for measurement of limb joints angles for locomotion research.

1. Introduction

Great universality and flexibility of an automatic picture analysis is most easily reached by the use of computer-based picture processing systems. From the theoretical point of view, any picture processing algorithm can be realized by some program for a universal digital computer. A digital picture processing system designed, built up and utilized in our Institute in Department of Picture Recognition and Processing since 1974 represents an example of such a universal and flexible system [1-5]. It serves as a very useful research tool to investigate methods of picture analysis [23-29] as well as it can be used for many practical applications of these methods in science and technology [16-23, 33].

As the input information to the system any kinds of pictures can be given: natural scenes, photographs, drawings, negatives, diapositives, microscope pictures, and so on. As a picture processing tool serves a digital picture processing system based on a minicomputer with I/O devices augmented by a digital image converter. Results of picture processing can be of several types:

- qualitative results: selection, filtration, feature extraction,

recognition, comparison, data compression;
- quantitative results: measurements of length, area and size of picture objects, counts of their number, and other computable features (e.g. shape factors);
- processes on picture data banks: collection, searching and retrieval, etc.

All elements of the CPO-2/K-202 system can be grouped into two main groups:

1) the special peripheral, named CPO-2, for digital TV image conversion and for input/output of TV pictures to/from the computer system;
2) the computer system comprising the Polish K-202 minicomputer, a standard set of I/O peripherals and an appropriate software oriented for programming of picture processing algorithms.

2. System hardware

The block diagram of the system structure is shown in Fig. 1 (see also [1, 5]). Main operations and processes performed in the CPO-2 unit are:

1) Conversion of a physical image into an electrical video-signal by means of a standard TV-scanning process using a professional vidicon TV-camera.
2) Quantization of the video-signal into digital form and its coding to fit a computer-word format. The quantization process runs with the same speed as TV-scanning. It quantizes:
   - the value of gray intensity during every picture line into 16 levels (by the use of fast A/D converter), and
   - every picture line into 512 picture elements (by sampling the quantized video-signal at appropriate time intervals).
3) Storage of the digitized picture in a buffer core memory. The memory operates both as a refreshing memory for a TV-monitor and as a picture data store for the computer system.
4) Display of the picture, for visual inspection purposes, on two TV-monitors (black/white and colour). On the first monitor either the direct picture output from the TV-camera, or the signal after gray-levels quantization, or the digital picture from the buffer memory is shown in black and white, whereas on the second monitor the digital picture from the buffer memory is shown in artificial colours. For monitoring of inter-

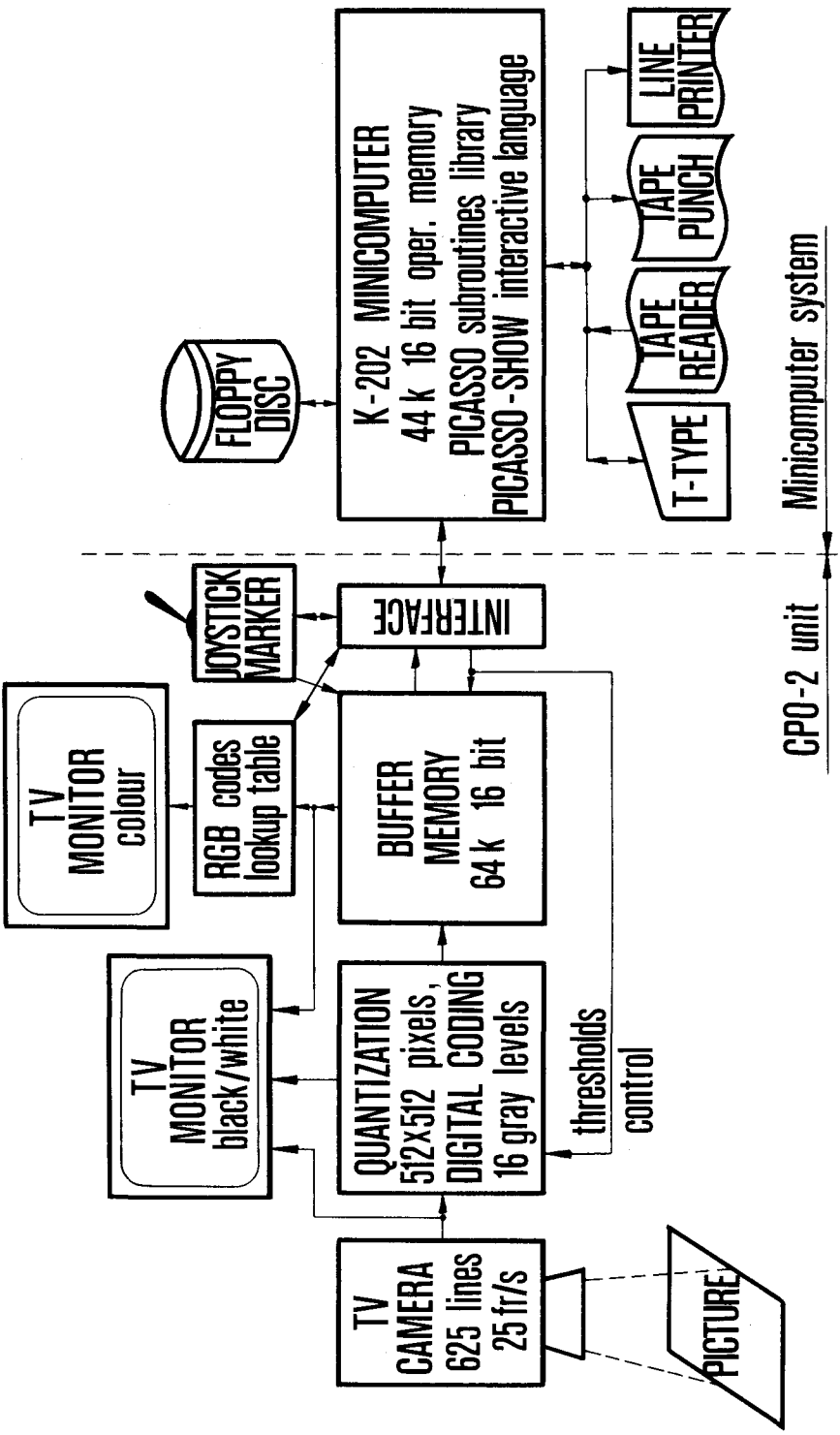


Fig. 1. The CPO-2/K-202 system structure

mediate stages of processing, any pictures can be send from the computer system to the buffer memory, so that they could be seen immediately on the screen.

5) On-line communication with the computer system.

The pictures inputted by the CPO-2 device have resolution of 512 × 512 picture elements: a square picture area is divided into 512 lines (a part of the standard 625-line TV-frame is taken) and then every line is divided into 512 picture elements (by appropriate sampling). The number of distinguished gray levels of each picture element is 16. The quantization parameters, i.e. the position of the lowest level within the whole video-signal range (from black to white) and the distances between the levels can be changed either manually or from the computer, and set to one of 256 possible values. The computer can also read the actual values of these parameters.

The fast A/D converter, providing the quantization of a video -signal with the speed of TV scanning has been based on a set of differential comparators working parallelly. Their input thresholds are controlled by the above-mentioned quantization parameters. The 15-line output, corresponding to 16 gray levels (including 0-level), feeds an encoder.

In the encoding process the value of gray intensity of every point is expressed by a four-bit binary number. Then, the digitized video-signal is sampled and every block of 16 successive picture elements, lying along a scan line, is represented by a group of four 16-bit computer words. The bits placed at the same position in every word of the group represent the 4-bit code of gray intensity of the corresponding picture element. Schematically, the quantization process, encoding and sampling are shown on the block diagram in Fig. 2. This method of picture encoding ensures convenient operation with the picture in one of its 16-, 8-, 4-, or 2-gray levels versions, simply by taking into consideration only 4, 3, 2, or 1 word(s) from the group, corresponding to the most significant bits of the picture element code.

During the scanning of a new picture, the groups of words can be stored in the buffer memory one by one in such a manner that every word of a group is placed in a separate memory block. There are four such blocks in the memory, corresponding to the four words in each group. That is, in one memory cycle four 16-bit

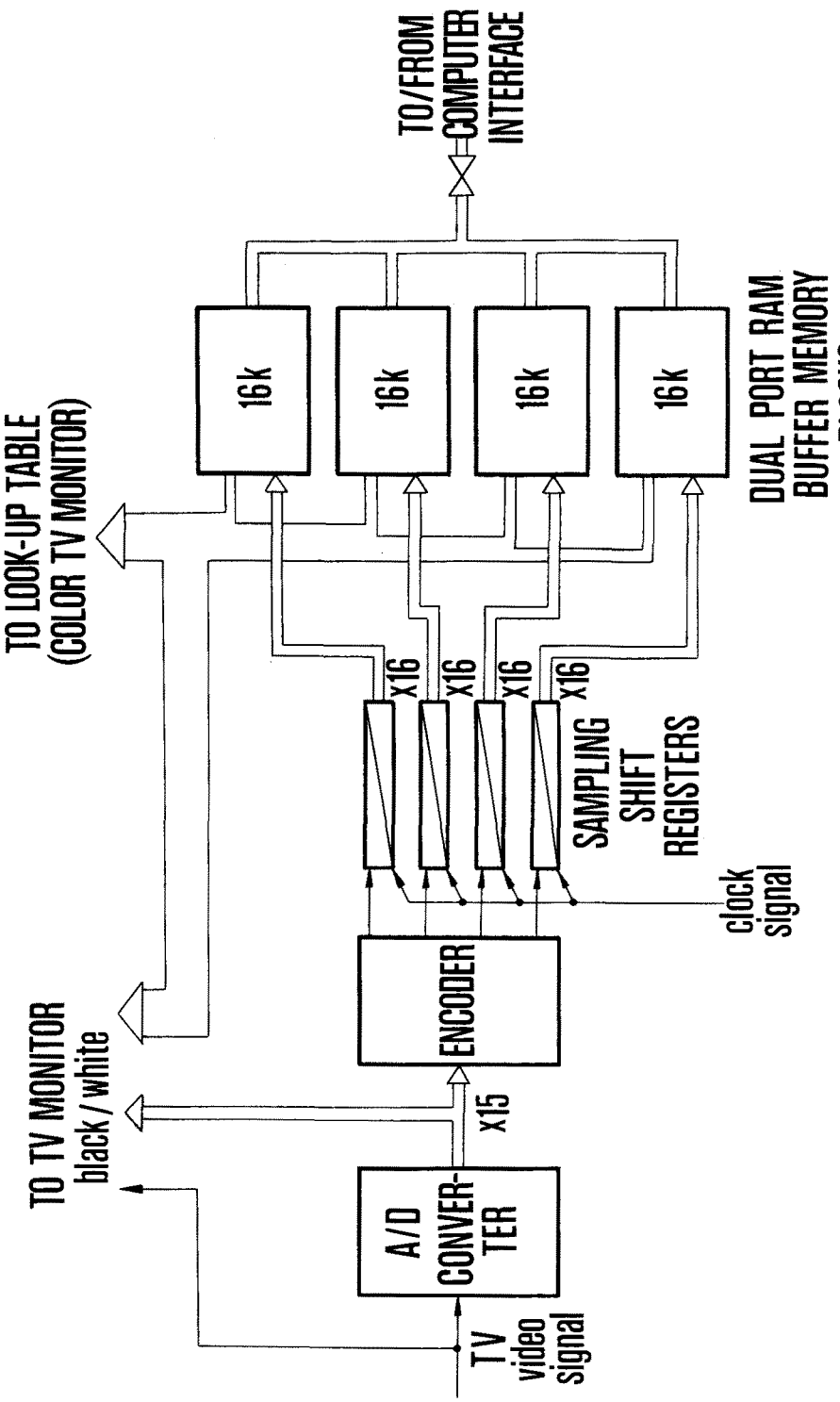


Fig. 2. Image quantization, encoding and storage in the system

words are stored or fetched simultaneously. Within a time interval of a single TV-picture frame (1/25s), the whole digitized picture is stored in the memory. Storage of a new picture can be made on the request of an operator (pushing an appropriate button) or on the signal from the computer, without necessity of the operator intervention.

The capacity of memory sections is of 16k words each. It corresponds to the number of picture elements (512 lines $\times$ 32 groups of 16 elements). In effect, the total storage capacity is equal to the amount of information contained in one 16 gray-levels picture (4 blocks $\times$ 16k words = 64k words = 1 048 576 bits).

Additionally, the CPO-2 unit is equipped with the joystick point marker, which allows either some intervention into the contents of picture information (correction, drawing of some picture elements, lines, etc.) or pointing to the computer program the position of objects chosen by the operator in a processed picture. A point marker (black or white) of the shape of a right angle corner pointing to the upper left is superimposed on a digital picture shown on the screen of the TV-monitor. The movement of the marker across the picture and its setting to some required position is usually done by hand, using a joystick manipulator. Depending on the operation mode, either every position of the marker and its trace can be memorized (in the buffer memory), changing therefore the picture contents, or the coordinates of its position can be send to the computer, as an answer to its request. The computer can also place the marker at any given position, sending the coordinates of its position to the marker.

The second TV-monitor of the system is a colour one. The quantized picture signal from the buffer memory can be seen on the monitor screen in "artificial colours". The correspondence of colours to different picture elements codes is determined by the look-up memory of RGB colour components, updated by the computer, so that to every individual gray-level code different combination of RGB signals may correspond. Every colour component (R, G or B) can be set individually into one of 16 levels. In effect, there is theoretically available 16^3 = 4096 different colours to represent every picture element code.

The 16 bit Polish K-202 minicomputer performs all image analysis programs on appropriate fragments (windows) of the input

picture, transmitted for this purpose into the operating memory from the CPO-2 image buffer. The minicomputer operating core memory (1.5μs cycle time) has two blocks of 16 bit words: the first (12k words) contains the operating system, and the second (32k words) contains user's programs and processed picture fragments. The instruction list includes about 90 basic instructions and programmed floating-point arithmetic. The standard set of peripherals consists of a teletype, paper tape reader/punch (ISO-7 code) and a line printer. A floppy-disc memory is being connected presently.

3. System software

The software for image processing in the CPO-2/K-202 system consists of the following parts:

1) Operating system SOK-1/CPO-2.
2) Assembler ASSK-3.
3) Library of basic picture processing subroutines PICASSO [1-6, 14].
4) A family of interactive languages PICASSO-SHOW [1-9, 14].
5) Application programs (usually written in one of the PICASSO-SHOW languages) [16-22].

The operating system presently in use is rather primitive - it is the standard SOK-1 system of the machine, augmented with a few subroutines to handle the CPO-2 device. It does not make use of the disc memory. The new operating system is under development. It is called COSMOS (COnceptually Simple Modular Operating System) and will be used with the floppy-disc memory being connected to the minicomputer (Fig. 1). Its structure will be based in part on the structure of the PICASSO-SHOW language interpreter and it will integrate into a single whole the functions of the operating system, the assembler and the PICASSO-SHOW interpreter.

The ASSK-3 assembler is also the standard assembler of the K-202 minicomputer. The library of picture processing subroutines PICASSO is written in assembly code, to achieve the highest possible efficiency of execution of these basic processing subroutines.

The PICASSO library and PICASSO-SHOW language will be described in some detail below (Sections 3.1 and 3.2). Some application

programs will be briefly described in Section 4.

A new high-level language for image processing (called PAL - Picture Analyzing Language) has been designed also [11-14] and will be implemented on the system. Meanwhile, some its ideas and parts of its compiler have been incorporated in the PICASSO-SHOW 3 language [8-10].

3.1. The PICASSO subroutines library

The PICASSO (PICture ASSembly-programmed Operations) package is a rather large set of subroutines for basic operations on pictures [1-6, 14]. It counts now about 170 operations. All operations assume the same structure of processed data items - numbers, pictures and number vectors. They are written so as to achieve maximal efficiency in execution time.

The program listings are standardized in order to become self-documented. Every subroutine is preceded by a standardized "comment header", summarizing all informations needed in order to use properly the subroutine in some program. The header describes parameters, results, non-local variables and subroutines called by the given one, machine register usage, error conditions and signals, and a form of the call. Within a subroutine body several standard conventions are also usually observed (e.g. in formation of variable and label names, program structuring) in order to enhance readability and facilitate modification and maintenance of the library. The fact that the library is included into the PICASSO-SHOW languages (see Section 3.2) is another reason for this standardization.

The most important convention adopted here is the form of basic data structures, namely pictures. The pictures are rectangular matrices of pixels, and to achieve greatest flexibility, they can have any dimensions and any number of bits of pixel values representation. In the memory, every picture is preceded by a header including the following parameters:

- XO, YO: coordinates of the lower left corner of the picture (in some absolute coordinate system),
- M, N: width and height of the picture (in pixels),
- S: the number of bits per pixel,
- L: the length of picture representation (in memory words).

Every picture operation uses this header to organize appropriately its processing of the picture.

Two different representations of pictures in memory are used, namely so-called "packed" and "stacked" representations. For the packed representation, all S bits representing the pixel value are stored in S consecutive bits of the same memory cell; one such cell contains usually several pixels. For the stacked representation, the picture is stored as S binary "planes", each containing a single bit of the representation of all pixels of the picture. Every memory cell in the plane contains the given bit of W consecutive pixels (along a row) of the picture, where W is the machine word length. The S bits of representation of some pixel are stored in S different memory cells, placed in the same positions of different planes. A binary picture (S=1) is a special case of a stacked picture, and consists of a single plane.

Utilizing the above packing of pixels into words and the fact that computers usually perform most operations with a single instruction over the whole word (bit-parallel), many PICASSO subroutines implement a semi-parallel processing method, gaining significantly in speed and efficiency over more serial processing requiring individual access to every single pixel.

Most of PICASSO picture operations are written in two versions, one for packed and one for stacked arguments. Some of them also have simplified versions operating on binary pictures. There are also operations processing only binary pictures (e.g. many propagation operations). The whole library is actually divided into 14 groups which are summarized (with examples of the most important operations in every group) in the table below. The greek letters α and β occuring in some names of operations stand for letters S or P (for α) or B, S or P (for β). These prefixes distinguish similar operations differing only by types of their arguments (i.e. Binary, Stacked, or Packed pictures, respectively).

1) CPO-2 device input/output

SCAN, DISP	input/output of a picture window,
SPOINT, DPOINT	input/output of single pixels,
COMPR, ENLARG	input/output of windows with linear scaling,
PUT, NEG	putting to a given value or negating windows in image buffer,

2) Changing picture form in memory

TOSTACK, TOPACK, PACK — changes between stacked and packed form of pictures,

αSCAL, αMOV — changes gray value scale of a picture,

3) Single-pixel operations

αREAD, αWRITE — read/write of single pixel,

SFIPO, SLIST — finding and listing of pixels with given values,

SLINE & NEXPLI, CIRC & NEXPCIR, RING & NEXPRI, ARC & NEXPAR — generate discrete lines, circles, rings and arcs (point by point) [24-26],

BAPROX — approximate binary contour with discrete straight line segments,

4) Input/output operations (to/from paper tape, to printer)

αDUMP, αLOAD — print/load: a number of different versions depending on the form of pictures on external medium,

HIST — print a histogram (also many versions),

5) Global features calculation

αWEIGHT, αCENTER — sum of gray values and center of gravity,

αHIST — gray level histogramming,

BAREA, BPERIM, VARPER — area and perimeter (corrected [24]) of blob,

BAXIS — main axis of inertia,

BWDOW — minimum window containing a blob,

NORMHI, MOMHIS — histogram normalization and moments,

FACSBL, FACSMA, FACSDA, FACSHA — different global shape factors [18-20, 27],

HIMA, HIPMA, WYMA — determination of local masks for texture filtering [28],

6) One-argument (pointwise) operations

αPUT — put all pixels to a given value,

αNEG — negate a picture,

αCOPY — copy a picture,

SDIVC — divide all pixel values by a given number,

STHRC — thresholding,

7) Two-argument (pointwise) operations

βOR, βAND, βDIF	logical,
βADD, βSUB, βDIV	arithmetic,
STHR	thresholding with pointwise different thresholds [18, 19],

8) Picture shifts

αSHIFT, SSH1D, SSH1X

9) Tests

αEQ	are two pictures equal?
αVAL, αBLACK, αWHITE	have all pixels the specified value?
BORD0, BORD1	does the white/black component touch the boundary of a picture?

10) Local operations (3×3 neighbourhood usually)

BCLEAN	"salt & pepper" noise removal,
BCONT	contour extraction,
BLINEND, BLICOS	line ends and intersections extraction,
SAV	local averaging,
BCURV	local line curvature determination,

11) Propagation operations (on binary pictures)

BPRO4, BPRO8, BPRO48, BPRO84	basic propagation operations,
BCON, BCCOM, BCOMPS & NEXCOM	connected components extraction,
BFILL, BFILLG	hole filling,
BTOUCH	extraction of components touching a boundary,
BLOB, BLOBC1, BLOBR	blob extraction [18, 19],
BTHINL	thinning (ACL algorithm [30]),
BARC, BLIC & NEXLI	extraction of discrete arcs,

12) Object extraction operations

HUECK	simplified Hueckel operator [31, 32],
HITHR, LOCTHR, THR	dynamic thresholding [18, 19],
SFILTT	extraction of objects of a given texture [28] (see operations in 5th group),

13) Object-generation operations

BFRAM, SZER, SMAX	setting values on the boundary,
SCIRC	circular disk generation [25, 26],
BLINSEG	straight lines drawing [24],

14) Correction operations

CORSHW, CORSHB	additive shading correction.

3.2. The PICASSO-SHOW languages

The PICASSO-SHOW languages comprise a family of interactive, command-oriented picture processing languages for a minicomputer-based picture processing systems. Earlier languages of the family, called PICASSO-SHOW [1, 6, 7], PICASSO-SHOW 1.5 and PICASSO-SHOW 1.6 [2-5, 14] have been working for about 4 years as a basic programming tools for the CPO-2/K-202 picture processing system. Basing on experience gained with these versions, design principles of the new version, called PICASSO-SHOW 3 [8, 9, 14] have been developed (the PICASSO-SHOW 2 version has been proposed also, although not implemented).

The PICASSO-SHOW 3 language is oriented towards picture processing operations from the PICASSO library [1-6, 14]. Individual operations can be executed in the way of interaction between a human operator and the interpreter, or they can be grouped into programs, stored in the operating memory and run there. The former mode of work is called interactive one. The latter one comprises two distinct modes, oriented either toward convenient program development and debugging (the so-called interpretative mode) or towards fast running of debugged programs (the program mode).

The set of operations of the PICASSO-SHOW 3 language is not fixed. Any user-defined subroutine can be appended to the language as its normal operation and any subset of the PICASSO library can be selected as the set of PICASSO-SHOW 3 operations.

3.2.1. The PICASSO-SHOW 3 language

Basic executable units of the language are called statements. In the interactive mode, the statement is executed immediately after it has been written. Under the interpretative mode, a sequence of statements (optionally preceded by label declarations) constitutes a program. A labeled statement can be executed under

interactive mode as well - in this case the label declaration has no result. Generally, all statements are legal under all modes, though sometimes there are minor differences in their results.

Statements of the language are of three types: declarations, instructions and assignments. A declaration generates some object the instructions are to deal with, supplying the interpreter with parameters of the object (e.g. size). An instruction evokes, via the system vocabulary of operations, some operation from the system library, passes to it its parameters and starts its execution. An assignment fixes the numerical value of a symbolic number of an object.

Objects processed by the PICASSO-SHOW 3 instructions are of five general types: pictures, numerical vectors, numerical variables, atoms and vectors of atoms.

The structure of pictures in memory is the same as that accepted by the PICASSO subroutines library (see Section 3.1). Besides "stacked" and "packed" types of pictures, a "binary" type is introduced which corresponds to pictures of two possible gray levels (black or white only). Pictures of these three types have different names. As a parameter of PICASSO operation, a binary picture is a special case of a stacked one; the distinction is made because some PICASSO subroutines operate on binary pictures only, what should be made visible in program texts.

Numerical vectors are one-dimensional sequences of memory cells. Their elements can be interpreted also as numerical variables of any numerical type.

Numerical variables are of three types: integer, long integer and real. There is ten predefined standard numerical variables for every type - they need not be declared.

Atoms are sets of fields (dynamic records). Fields are ordered pairs consisting of a field selector (see below) and a field value. A field has a type attached to it, which determines the type of its value. An atom as a whole has also a type assigned to it. Atom types are significant only when the atom is used as a field selector (see below), othervise the atom type has no significance at all. The atom type allows building hierarchical data structures of any complexity, for structural picture description and processing. The atom concept has been borrowed from the PAL language [9, 11-14].

Objects used as instruction arguments are referred to by names. Only some fixed set of object names can be used and their form is also standard. Generally, there exist ten different names for objects of every type, e.g. ten different static numerical variables of every numerical type, ten numerical vectors, ten stacked pictures, etc. For numerical variables and atoms this restriction does not limit a programmer because appropriate vector elements can be used as well. For other objects the restriction on the number of the object names does not affect the number of the objects themselves. An object can exist without a name as a value of some atom field (where it is accessible by a name of the field), or as an auxiliary parameter of an instruction (where it is created by a "generator", see below). By a field name the object may be referenced to in atom-dealing instructions, but it must be assigned to an object name before it is used in other instructions.

The fixed form of object names simplifies argument reading subroutines, which is important especially when working under interactive or interpretative modes (no need for any identifier tables). It also simplifies manual translation of PICASSO-SHOW 3 programs into assembly code and makes the programs more "semantically legible", because standard object names indicate immediately their types.

An object name consists of a letter (specifying the type of the object), a digit (specifying the number of the object) and, eventually, a vector element index (if needed). The digit may be replaced by a symbolic number (a single letter). The use of this device simplifies linking of different program fragments, because all used object numbers may be symbolic, so that changing the number requires only changing appropriate assignment (see below) instead of changing many names.

A declaration generates an object, allocates a space for it in operating memory and assigns to it a name. The declaration consists of a list of names of generated objects and a list of parameters of the objects. Objects generated in the same declaration must be of the same type (except for stacked and packed pictures which may be declared together). The parameters may be written explicitly or by a reference to some existing object of the same type. E.g. the stacked picture S3 may be declared in two

ways:

*S3: (100, 200, 128, 64, 4), or

*S3: S5,

In the first case the size of the generated picture is given explicitly by the parameter list (in parentheses), while in the second one the parameters of the existing S5 picture are used. Another examples, in this case of the atom declaration:

*AI1: (FI01:5, FS21:S3),

*AR2: A1,

The parameters of an atom define the initial set of its fields. Field selectors consist of the letter F, the type indicator, and a pair of digits.

An object without a name can be also generated. The form of such an "object generator" is similar to that of a declaration, but it does not contain the object name, and it can be used only as argument of an instruction. The aim of using the generator is to provide an instruction with an auxiliary object, some intermediate results of the instruction can be kept in. The contents of such auxiliary objects are not important before and after the execution of the operation, so there is no need of assigning any names to them. Examples of possible forms of generators of auxiliary packed pictures are:

*P(10, 50, 100, 100, 2) <a picture with given parameters>

*P3 <the same parameters as in the P3 picture>

Instructions perform operations on objects. The operation is defined by a subroutine attached to the instruction in the instruction module of the system library (see below). Instruction name is a typical alphanumerical character sequence. As arguments of an instruction, objects, arithmetical expressions and texts can be used. In order to avoid unnecessary declarations of temporary arguments or to shorten the notation of some arguments, two additional conventions were introduced: generators (described above) and "windows".

When working under interactive mode, one often encounters the need of executing a sequence of operations consisting of scanning an image from input device, performing some image operation (e.g. from PICASSO package) and displaying the result immediately on the screen. The "window" allows to condense the notation of this

sequence of actions into one statement. A window is a picture name placed as an instruction argument together with a command for transmitting the picture to/from the image buffer memory. In the following example, one instruction with windows replaces four normal instructions. The upward-pointing arrow symbol denotes "display" command while the left-pointing arrow denotes "scan" command. The window may also contain some parameters, describing the place in the buffer (coordinates on the screen) to/from where the transmission would take place. For instance:

```
    AND, S1←, S2←, S3↑,
```

is equivalent to the sequence of instructions (simplified):

```
    SCAN, S1,
    SCAN, S2,
    AND, S1, S2, S3,      < the result is on the picture S3 >
    DISP, S3,
```

After that instruction the result of the AND operation is immediately seen on the TV-monitor screen.

Arithmetical expressions may be used anywhere as numerical arguments. Four arithmetical operators are allowed as well as parentheses (with arbitrary nesting). Two-argument operators deal with pairs of operands of the same type (integer, long integer, or real). To convert an operand to appropriate type, conversion operators are used.

An object name consists of a letter and a digit. The letter defines the type of the object and the digit - its number. However, it is often convenient to use symbolic names of objects. In a symbolic name, the digit is replaced by a letter. The letter obtains its value by means of an assignment, having the form:

```
    letter = digit,
```

Using actual values of letters (as defined by assignments) the interpreter (in the course of loading a program) changes all symbolic names into explicit ones.

Labels are of two kinds: global and local. Label denotations consist of the symbol "$" and a name. The names of global labels have the same form as instruction names. Local labels names consist of two digits. Global labels are accessible everywhere in a program while the scope of local labels is restricted to the program segment between pairs of consecutive global label declarations.

Comments, having the form of strings of characters enclosed in angle brackets "< " and " >", can be placed anywhere, even within instruction names.

3.2.2. Instructions repertoire

The set of instructions available in the language consists of the so-called "system instructions" and any set of other instructions (usually a subset of the PICASSO library, see Section 3.1) chosen by a programmer in the phase of assembling "instruction modules" into the system vocabulary (Section 3.2.3).

The system instructions are permanently resident in the system. They can be classified into the following groups:

- jumps and testing instructions,
- loop organization instructions,
- subroutines organization instructions,
- editing instructions,
- execution control instructions,
- list processing instructions,
- other.

Each jump instruction has a label as an argument. For conditional jumps there is another argument (sometimes implicit) which decides whether the jump is to be performed or not. There are six jump instructions:

GOTO (unconditional); GOKEY (if some key is on);
GOL, GOE, GONE, GOG (if the value of the IO variable is less, equal, not equal or greater than 0, respectively).

Closely connected with jumps is a set of testing operations. This set contains arithmetical comparison instruction COMP and some PICASSO operations of the same character. They set the variable IO to -1, 0 or +1, depending on the fulfilment of some conditions. The IO variable is accessible for a programmer as any other variable, and can be set to any value with the SET instruction as well.

Loops in a program are organized by using pairs of BEGLOOPi -- ENDLOOPi instructions, where the letter "i" denotes a digit. The digit is a number of the loop. The full form of the BEGLOOPi instructions is:

BEGLOOPi, an1, an2, an3,

where an1, an2, are numerical arguments setting boundaries of the

loop counter of standard name Ki, and the an3 numerical argument is the step of the counter. Thus the BEGLOOPi instruction is roughly equivalent to the ALGOL 60 construction:

for Ki := an1 step an3 until an2 do begin

and the ENDLOOPi instruction is equivalent to the end instruction closing the loop body. An important difference is that the body is always executed at least once. Loops can be nested, but then they must have different numbers.

Subroutines are implemented by means of two operations: the operation CALL that puts on a stack a return address (of the statement following the CALL) and jumps to some label (starting label of the subroutine), and the operation RETURN that pops up the stack and jumps to the statement the popped stack element was an address of. Thus recursive calling of subroutines is possible. Nevertheless, there is no special mechanism for passing arguments to and results from the subroutine - they have to be transferred within global variables and objects. It should be explained that the subroutine on the language level has nothing in common with the instruction subroutine realizing some language instruction. The latter is written in assembly language as a part of some instruction module (see the next Section). For example, the PICASSO--SHOW 3 program below computes recursively the factorial of a number given in the variable L1, puts the result into L2, then prints it out and returns to the interactive mode:

```
    SET, L2, L1,
    CALL, ∅1,
    PRL, L2,
    DO, 3, < EXIT TO INTERACTION WITH TELETYPE >
∅1: < A FACTORIAL SUBROUTINE >
    SET, L1, L1-1,
    COMP, L1, 1,   GOL, ∅2,
    SET, L2, L2 * L1,
    CALL, ∅1, < RECURSIVE CALL OF FACTORIAL >
∅2: RETURN,
```

To execute this program, one should place a number into L1 (say, the number is 5) by writing on the teletype:

```
    SET, L1, 5,
```

and activate the program:

```
    DO,,
```

After a while the system responds with the factorial of the number 5:

120

and waits for the next command to be written on the teletype.

The LOAD instruction reads the text of a program from an input device, places it in the system memory and numbers its lines. The PRINT instruction outputs the required fragment or the whole program to an output device. The INS and REPL instructions insert or replace fragments of a program respectively.

The DO instruction switches the mode of work between interactive and interpretative modes. The COMPILE instruction translates a program to the intermediate code allowing its fast interpretation, and the RUN instruction runs this code, i.e. sets the program mode.

The STOP instruction halts program execution if a special key is on, otherwise it has no result. To restart a program after the STOP or other interruption (e.g. an error), the GO instruction can be used.

The ON instruction changes the reaction of the interpreter after an error has been detected in a program. The standard reaction is the printout of an error message and halting the program, i.e. returning to the interactive mode. Once the ON instruction has been executed, the interpreter does not halt the program after an error message (of the error specified by the parameter of the ON instruction), but resumes its execution from the point marked by a label given by another argument of the ON instruction.

List processing instructions allow dynamic extension and compression of vectors (either numerical vectors or vectors of atoms) and access to atom fields. The dimension of a vector is changed by the ALTER instruction. Access to a field of an atom is given by instructions OF (reading) and ASSOC (assigning); with the latter instruction a new field can be also added to the atom, whereas the FREE instruction removes a field from it. The ISF instruction tests the existence of a given field in an atom.

There are some other system instructions, e.g. input/output ones (dealing with numbers, characters and texts), CPO-2 device control, etc. Some of them are closely dependent on the hardware of the system, others are more general and rather typical for many programming languages.

3.2.3. Structure of the interpreter

The main concept of the interpreter is the idea of operation vocabulary. The vocabulary consists of entries describing all instructions legal in the system. Instructions are organized into "instruction modules". A module of a single instruction (or a set of closely related instructions) consists of a subroutine (or subroutines) performing the operation (or several related operations), some entries of the operation vocabulary, and possibly some entries of the linker vocabulary (if the subroutine calls another subroutines). The modules are constructed in such a way that instruction can be added to the system library with the use of a special linker as well as with the standard assembler.

An entry of the operation vocabulary consists of a sequence of characters (the six initial characters of the instruction name), an address of an entry point in the subrotine body, an address of the next vocabulary entry, a sequence of descriptions of arguments (operation parameters) and the end marker. Because argument descriptions simply name subroutines to be activated for the arguments reading and setting, the entry provides a "procedural" description of types of arguments and their sequence.

Every argument reading subroutine reads an argument of a defined type, checks it for its correctness, changes to the form of a parameter of the main subroutine which performs the operation, and passes it to that subroutine. Some subroutines which appear in the operation vocabulary entries do not read any arguments but perform some auxiliary actions. For example, the HELP subroutine, used in order to facilitate a dialogue with an uninitiated user, prints on the monitor any prescribed text, giving the user additional informations, e.g. about the type and meaning of subsequent arguments to be written. There is also a set of subroutines controlling checking of argument parameters, e.g. which pictorial arguments should have the same size.

All PICASSO-SHOW 3 declarable objects as well as some tables of the interpreter (e.g. the table of global label names) are administered by the SETSYS dynamic storage allocation system [10]. The SETSYS is an autonomous system of storage allocation procedures, and its use in the PICASSO-SHOW 3 interpreter is one of its possible applications.

Basically, SETSYS consists of two levels: semantic (or user)

level and memory (or implementation) level. The user level essentially coincides with the list processing capabilities of PICASSO-SHOW 3. I.e., it allows:

- creation and deletion of objects,
- attachment and detachment of elements to/from objects,
- getting and putting values from/to elements of objects.

There are four types of objects: simple, vector, atom, picture. Simple objects correspond roughly to PICASSO-SHOW variables except that they may contain references to other objects and are dynamic (may be deleted from the computation). Vectors behave like double-ended queues and, additionally, indexed access to their elements is possible. Atoms are sets of named values which can be freely accessed, added to and deleted from the atom by means of their names. Picture is a problem-oriented data type. Elements of composite objects (vectors, atoms) are of simple type.

Morphology of the above objects and operations on them is realized in terms of memory level of SETSYS. The memory level operates on the so-called "sets". Sets are blocks of consecutive memory cells placed in a predetermined pool of memory cells, called a heap. A set consists of the useful part (used for storing elements of objects it represents) and the spare part (used for eventual future extensions).

Possible operations on sets are: creation, deletion, extension and contraction. Deletion simply releases block of cells occupied by the set, which thereafter becomes the so-called hole. Contraction of the set reduces the number of its elements, adding the cells occupied by them to the spare part of the set. Creation and extension in their turn both consist of allocating new free storage (in the case of extension, the possibility of using the spare part of the set to be extended is tried first). If the spare part was not sufficient to complete the required extension, a sufficiently large hole is searched for and the sets and holes between the extended set and the found hole are shifted in order to use the hole to enlarge the set. Similarly, for creation of a new set, the multistage strategy of acquiring necessary amount of free cells is adopted. In each subsequent stage the complexity of the algorithm increases, until the success is achieved:

1) try to seize a hole,
2) try to allocate free storage from the heap,

3) repeat (2) after hole merging,
4) repeat (2) after spare parts retrieval and merging,
5) perform garbage collection (i.e. recovery of sets which are not referred to by any other set accessible directly or indirectly from the actual program).

Conceptually, SETSYS is an elaborated version of the so-called MINIPAL/SET system [15]. It was initially intended for use in the PAL language compiler [10-14].

4. Applications

The CPO-2/K-202 image processing system has been used for several practical applications. Application programs (mostly written in PICASSO-SHOW language) have been used for processing of various kinds of pictures, mainly biomedical. More important realized programs include:

a) calculation of blood vessels width ratio in eye-fundus photographs [2, 17],
b) calculation of areas and shape descriptors of the optic disc and cup in eye-fundus images [2, 16, 17],
c) ERG curves digitization [23],
d) analysis of copper ore samples,
e) blood groups precipitation data recognition,
f) quantitative measurement of shape changes of moving leukemia cells [18-20],
g) measurements of radar pictures of cloud covers,
h) calculation of limb joints angles for animal locomotion research [21-23],
i) determination of banding profiles of chromosomes [33],
j) muscle tissue analysis.

One of the most elaborated programs is that for cells shape changes measurement ((f) above). The program (strictly speaking, several its versions, called CSC-1, ..., CSC-4) has been used in investigations of leukemia cells motility and adhesiveness properties (in connection with cancer research [20]). The time-lapse films of a cell culture have been analyzed off-line, frame by frame, on the CPO-2/K-202 system. In every frame usually several cells were analyzed. Various quantitative features (about 20 different quantities) have been measured for every cell image. Pre-

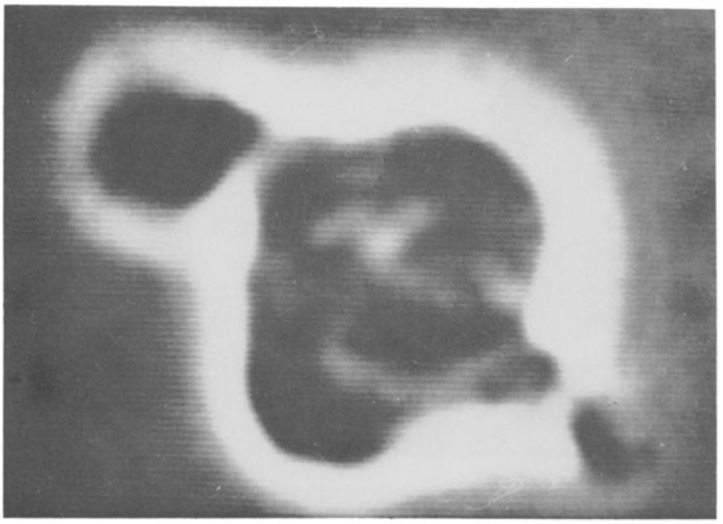
liminary analysis of the biological significance of obtained parameters has been attempted in [20].

The first stage of analysis is aimed for extraction of cell outline from the background (Fig. 3a-e). The image, quantized into 16 gray levels by the CPO-2 device, is then binarized by dynamic thresholding method [18, 19]. By this method, the image is thresholded with different thresholds in different parts of the image. These local thresholds are determined from analysis of gray-level histograms calculated for small windows of the image. If the histogram is markedly bimodal, the threshold is set to the gray level value corresponding to the minimum between the modes. Otherwise, the threshold is undetermined for this window, and it receives its default value by some iterative interpolation process involving thresholds of nearest "good" windows. The binary picture thus obtained (Fig. 3c) is then filtered out to remove the background components touching the boundary and filling holes within the cell component (Fig. 3d). In this stage, the image can be edited by the operator (using the joy-stick point-marker of the system, Fig. 1), e.g. in order to cut off eventual "bridges" joining the cell component to the background (due to minute cell contour imperfections). The main component representing the cell is then extracted and subjected to some boundary-smoothing operation and its contour is extracted finally (Fig. 3e).

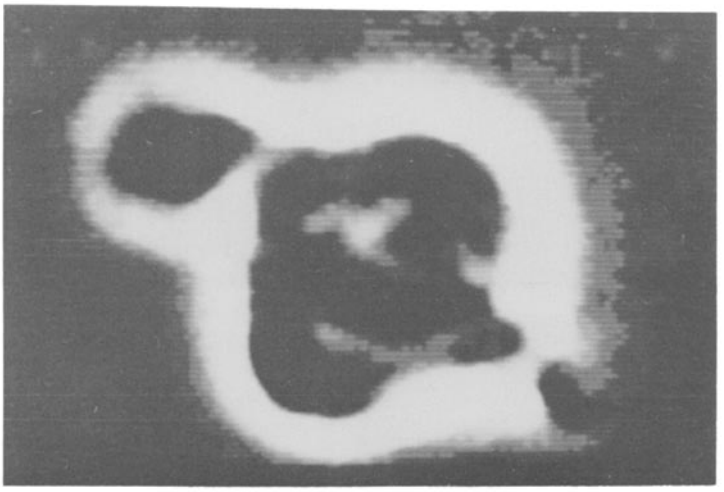
The second stage consists of measuring various quantitative features of the extracted cell. Among others, the program calculates:

- coordinates of the center of gravity,
- area and perimeter,
- various global shape factors (see [18-20, 27] for details),
- direction of main axis of inertia,
- length (along the axis) and width (perpendicular to the axis),
- cross-sections for several positions along and perpendicular to the axis.

Then, the cell is decomposed into the cell body and extensions. The body is extracted by iterative circular propagation with the center shifted after every iteration to the center of extracted "candidate body" [18, 19, 25, 26]. Usually from 2 to 4 iterations suffice to obtain the final result, as in Fig. 3f. Parts of the

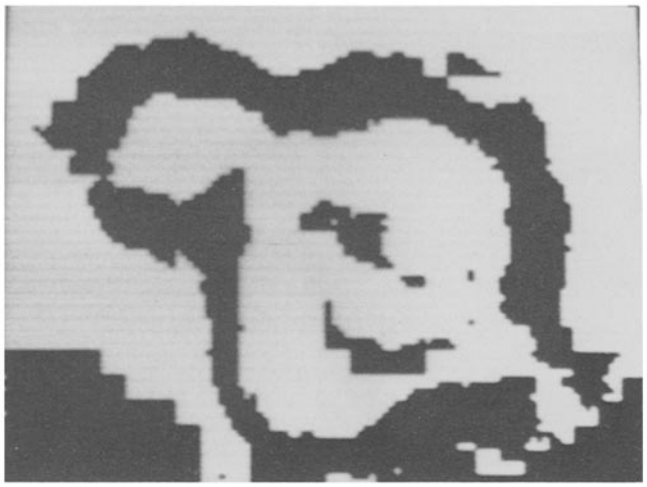


(a)
cell image from
the camera



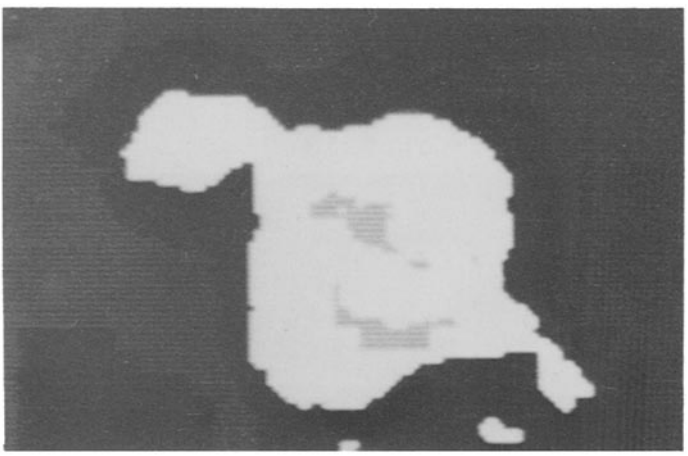
(b)
cell image after
quantization
(16 gray levels)

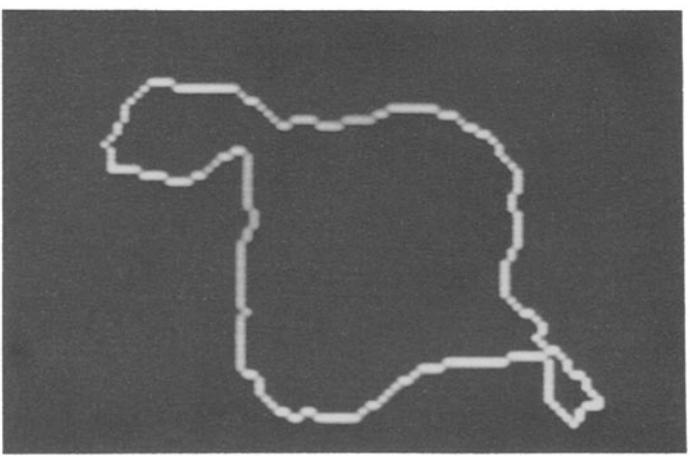
Fig. 3. Leukemia cells analysis example



(c)
cell image binarized
by dynamic thresholding

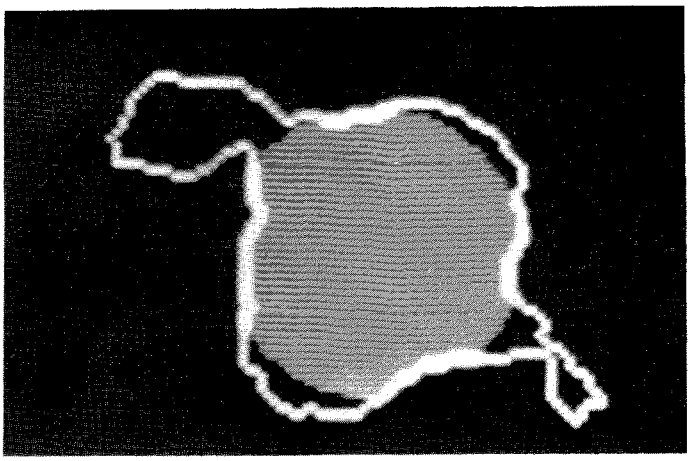
(d)
cell image after gap-
-filling & border-
-touching component
removal



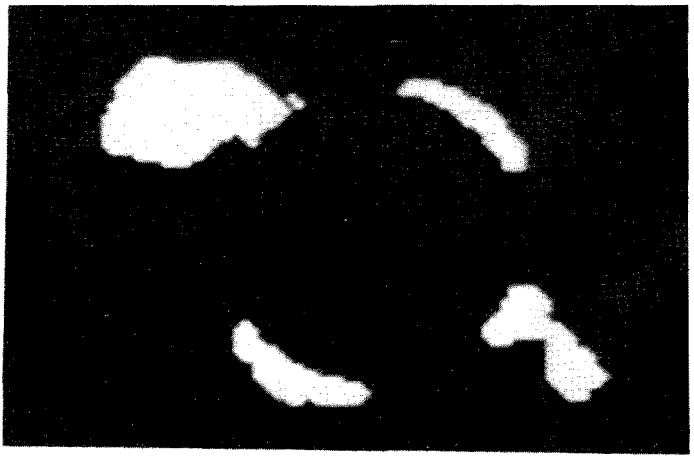


(e)
final cell image
(contour)

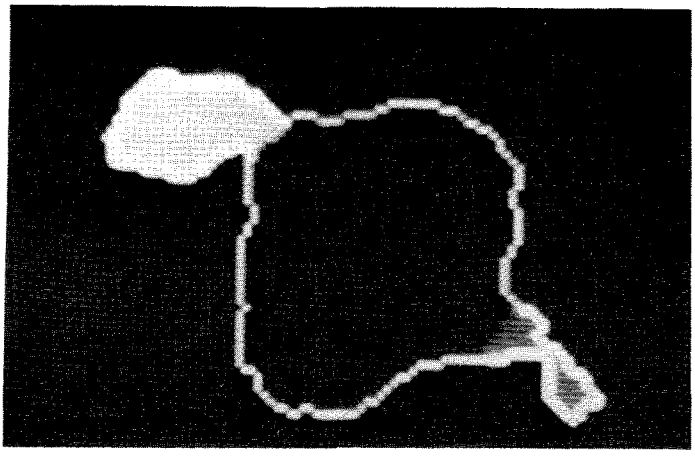
Fig. 3. (continued)



(f)
cell body
extraction



(g)
five candidate
extensions



(h)
two true extensions
superimposed
on the contour

Fig. 3. (continued)

cell protruding from the body (Fig. 3g) are then examined as eventual extensions. True extensions are distinguished by the dimensions (should be large enough) and the percentage of that part of their perimeter which touches the body to their whole perimeter (Fig. 3h). Several so-called structural features are then calculated, among others:

- radius of the body (the number of circular propagation steps),
- number of true extensions,
- ratio of areas of the largest extension and the body,
- position of the largest extension (relation of centers of gravity of the body and the extension),
- direction of the extension main axis (also with relation to the cell axis).

The whole analysis of the cell by the CSC program takes several minutes of the system run-time, depending on the quality of the image (which affects the cell-extraction process) and complexity of the cell structure (which affects the decomposition process). The programs written for the system were (and still are) used to process and analyze many thousands of pictures.

Acknowledgments

The research reported here was supported by the Research Programme No. 10.4.

References

1. Z. Kulpa, J. Dernałowicz, H.T. Nowicki et al., System cyfrowej analizy obrazów CPO-2 (CPO-2 digital pictures analysis system, in Polish), Institute of Biocybernetics and Biomedical Engineering Reports, Vol. 1, Warsaw 1977.

2. Z. Kulpa, J. Dernałowicz, M. Raczkowska, M. Piotrowicz, Digital picture processing system CPO-2 and its biomedical applications, In: Selected Papers of the 1st Natl. Conf. on Biocybernetics and Biomedical Engineering, Polish Scientific Publ. (PWN), Warsaw 1978.

3. Z. Kulpa, J. Dernałowicz, Digital picture processing system CPO-2 and its biomedical applications, Proc. BIONIKA'77 Conf., vol. 3, Bratislava, Sept. 1977, 288-293.

4. Z. Kulpa, M. Sobolewski, Obrabotka i raspoznavanye izobrazhenyi s pomoshchyu universalnoy systiemy CPO-2/K-202 (Image processing and recognition using universal system CPO-2/K-202, in Russian), Proc. BIONIKA'78 Conf., vol.1, Leningrad, Oct. 1978, 182-192.

5. Z. Kulpa, J. Dernałowicz, Digital image analysis system CPO-2/K-202, general hardware and software description, Proc. IV Polish-Italian Bioengineering Symp. on "Pattern Recognition of Biomedical Objects", Porto Ischia/Arco Felice, Oct. 1978.

6. Z. Kulpa, H.T. Nowicki, Simple interactive picture processing system PICASSO-SHOW, Proc. 3rd Inter. Joint Conf. on Pattern Recognition, Coronado, Calif., Nov. 1976, 218-223.

7. Z. Kulpa, H.T. Nowicki, Simple interactive picture processing system PICASSO-SHOW, Proc. Inter. Seminar on "Experiences of Interactive Systems Use", Szklarska Poręba, Oct. 1977, Wrocław Tech. Univ. Press, Wrocław 1977, 101-115.

8. H.T. Nowicki, Interactive picture processing language PICASSO-SHOW 3 and its interpreter, as in [5].

9. Z. Kulpa, Propozycja podjęzyka przetwarzania list do systemu PICASSO-SHOW (A proposal of a list-processing sublanguage for the PICASSO-SHOW system, in Polish), Institute of Biocybernetics and Biomedical Engineering Internal Reports, Warsaw 1978.

10. A. Bielik, Z. Kulpa, System dynamicznej rezerwacji pamięci i przetwarzania listowego SETSYS/K-202 (Dynamic storage allocation and list processing system SETSYS/K-202, in Polish), ibid.

11. Z. Kulpa, An outline description of the picture analyzing language PAL, Proc. 9th Yugoslav International Symp. on Information Processing (INFORMATICA'74), Bled, Oct. 1974.

12. Z. Kulpa, Język analizy obrazów graficznych PAL (A graphic pictures analyzing language PAL, in Polish), Institute of Biocybernetics and Biomedical Engineering Internal Reports, Warsaw 1977.

13. Z. Kulpa, Konstrukcja języka programowania algorytmów cyfrowego przetwarzania złożonych obrazów wizualnych (Design of a programming language for digital processing algorithms of complex visual images, in Polish), Ph. D. Thesis, Institute of Computer Science, Warsaw 1979.

14. Z. Kulpa, PICASSO, PICASSO-SHOW and PAL - a development of a high-level software system for image processing, Proc. Workshop on High-Level Languages for Image Processing, Windsor, June 1979; Academic Press, 1981 (in press).

15. Z. Kulpa, System dynamicznego przydziału pamięci i przetwarzania listowego MINIPAL/SET 1204 (A dynamic storage allocation and list processing system MINIPAL/SET 1204, in Polish), Institute of Biocybernetics and Biomedical Engineering Internal Reports, Warsaw 1973.

16. K. Czechowicz-Janicka, K. Majewska, L. Prządka, M. Raczkowska, Surface and shape of the optic disc in healthy subjects in various age groups - application of computer picture processing, Ophtalmologica, 674, 1977, 1-4.

17. M. Rychwalska, M. Piotrowicz, Analysis of the eye fundus using digital image processing system CPO-2, Proc. BIONIKA'77 Conf., vol. 2, Bratislava, Sept. 1977, 192-195.

18. Z. Kulpa, A. Bielik, M. Piotrowicz, M. Rychwalska, Measurement of the shape characteristics of moving cells using computer image processing system CPO-2, Proc. Conf. BIOSIGMA'78, Paris, April 1978, 286-292.

19. A. Bielik, Z. Kulpa, M. Piotrowicz, M. Rychwalska, Use of computer image processing in quantitative cell morphology, as in [5].

20. K. Lewandowska, J. Doroszewski, G. Haemmerli, P. Sträuli, An attempt to analyze locomotion of leukemia cells by computer image processing, Computers in Biology and Medicine, vol. 9, 1979, 331-344.

21. Z. Kulpa, A. Gutowska, Measurement of limb movement coordination in cats using universal computer image processing system CPO-2, In: A. Morecki, K. Fidelius, eds., Biomechanics VII, Proc. VIIth Inter. Congress of Biomechanics, Warsaw, Sept. 1979, Polish Scientific Publ. (PWN), Warsaw 1980, 459-465.

22. Z. Kulpa, A. Gutowska, Limb movement coordination in cats measured by universal computer image processing system CPO-2, Proc. EUSIPCO-80 Conf., Lausanne, Sept. 1980 (Short Communication and Poster Digest), 85.

23. Z. Kulpa, Errors in object positioning with "centre of gravity" method, The Industrial Robot, vol. 5, Nr.2, 1978, 94-99.

24. Z. Kulpa, Area and perimeter measurement of blobs in discrete binary pictures, Computer Graphics and Image Processing, vol. 6, Nr.5, 1977, 434-451.

25. Z. Kulpa, On the properties of discrete circles, rings and disks, Computer Graphics and Image Processing, vol. 10, 1979, 348-365.

26. M. Doros, Algorithms for generation of discrete circles, rings and disks, Computer Graphics and Image Processing, vol. 10, 1979, 366-371.

27. Z. Kulpa, M. Piotrowicz, Shape factors of figures in discrete pictures, In: Selected Papers of the 3rd Natl. Conf. on Biocybernetics and Biomedical Engineering, Polish Scientific Publ. (PWN), Warsaw 1980.

28. M. Młodkowski, Texture discrimination using local masks, as in [5].

29. M. Młodkowski, S. Vitulano, Some experiments with two-dimensional C-transform applied to texture analysis, as in [5].

30. C. Arcelli, L. Cordella, S. Levialdi, Parallel thinning of binary pictures, Electron. Letters, vol. 11, Nr. 7, 1975.

31. M. H. Hueckel, An operator which locates edges in digitized pictures, J. ACM, vol. 18, 1971, 113-125.

32. L. Mérő, Z. Vassy, A simplified and fast version of the Hueckel operator, Proc. 4th Inter. Joint Conf. on Artificial Intelligence, Tbilisi, 1975, 650-655.

33. M. Piotrowicz, Z. Kulpa, Determination of profiles of banded chromosomes using computer image processing system CPO-2, Proc. EUSIPCO-80 Conf., Lausanne, Sept. 1980 (Short Communication and Poster Digest), 83-84.