UvA-DARE (Digital Academic Repository)

SCILAIM: a multi-level interactive image processing environment

ten Kate, T.K.; van Balen, R.; Smeulders, A.W.M.; Groen, F.C.A.; den Boer, G.

DOI

[10.1016/0167-8655\(90\)90114-H](https://doi.org/10.1016/0167-8655(90)90114-H)

Publication date

1990

Published in

Pattern Recognition Letters

[Link to publication](#)

Citation for published version (APA):

ten Kate, T. K., van Balen, R., Smeulders, A. W. M., Groen, F. C. A., & den Boer, G. (1990). SCILAIM: a multi-level interactive image processing environment. *Pattern Recognition Letters*, 11(6), 429-441. [https://doi.org/10.1016/0167-8655\(90\)90114-H](https://doi.org/10.1016/0167-8655(90)90114-H)

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

SCILAIM: A multi-level interactive image processing environment

Ton K. TEN KATE*, Richard VAN BALEN**

Institute of Pathology, Free University Hospital, Amsterdam, The Netherlands

Arnold W.M. SMEULDERS**, Frans C.A. GROEN***, George A. DEN BOER

Faculty of Mathematics and Computer Science, University of Amsterdam, The Netherlands

Received 2 May 1989

Revised 5 December 1989

Abstract: A new transportable image processing environment is presented, which consists of a standard C interpreter (SCIL), a reconfigurable window manager, a command expander, a library handler, and an image processing library (AIM). SCIL permits rapid interactive image processing development in a higher level language (C), it offers machine compatible data structures. The library handler permits straightforward integration of external software packages and user-written functions. The interpreter is equipped with a command expander to provide parameter checking, default handling and command control without the burden of the C punctuation. The command expander operates on a command description file which is a text file containing per routine; type, default and range or legal values of its arguments. The window management system provides the possibility to enter commands through menus and dialogue boxes. The menus and dialogue boxes are reconfigurable and automatically generated from the command description file. AIM is an example of an external library, it consists of a collection of basic, fast image processing operations.

Key words: Image processing, interpreter, command expander, window management system.

1. Introduction

In the image processing community, interactive systems have earned a wide popularity. The reason is, that generally no off the shelf solutions for image processing problems are present. There is a large number of tools, these tools have to be combined, parameters to be optimized and sometimes problem dependent routines have to be developed and added. As the solution strategy may not always be immediately evident visual interaction during development assumes an important role. The need for these interactive systems has resulted in the development of an enormous variety of software systems for image processing. Preston (1981) gives a survey of dozens of image processing languages. He states: “*Research group after research group in many countries have been unable to resist the temptation to build new languages rather than to borrow from the old*”. He concludes that the diversity of the languages is not justified since the applications of image processing are not so disparate as to require unique and specialized data types, control structures and directives. Rather than creating a new highly specialized language it is better to utilize the full extent of an existing programming language.

*Now under contract grant praeventiefonds 28/736, sponsored in part by the SPIN programs 3DAI and 3DCV.

**Formerly with: Department of Medical Informatics, Erasmus University, Rotterdam.

***Also with: Faculty of Physics and Astronomy, Free University, Amsterdam.

In this paper our aim is to expound a method of making a high level programming language C especially effective as an interactive development system. First, we will review existing image processing systems. Thereafter, the structure of the newly proposed multi-level environment will be discussed. Next, the image processing toolbox will be presented. Finally, the effectivity in interactive image processing will be demonstrated.

2. Review of current systems

The functionality of existing systems for image processing vary from simple collections of routines to self-contained languages possibly dedicated to special hardware (Preston, 1983). A typical example of the first category is the SPIDER package consisting of about 400 subroutines, written in Fortran IV (Kaneko and Sakaue, 1983). This category of systems is characterized by using the command substitution of the operating system as the command language, letting the image processing tools operate as subtasks. Piper (1985) proposes the UNIX environment as the image processing vehicle, employing either the pipe mechanism to pass the output from one tool as input into the next one, or let each tool operate on a (disc) file. HIPS is another collection of UNIX programs where shell, filter and pipe mechanisms are used (Landy et al., 1984).

The second category of systems comprise the command-driven interactive systems, or command interpreters, characterized by prompt execution of image operations and (usually) the presence of default mechanisms. The functionality of command interpreters ranges from simple (those who offer the execution of (image) operations and the control of parameters) to sophisticated (those who in addition provide variable declaration, expression evaluation, flow control and/or procedure execution). An important feature of their practicality is the facility to construct a pseudo command from a sequence of basic commands. An example of a simple command interpreter is IRIS (1985). Haralick and Minden (1978) present the KANDIDATS interactive system, which supports a vast set of image processing functions. It includes prompting for parameter inputs and checking user input. There are no elements of flow control, variable manipulation and expression evaluation, however. A more sophisticated interpreter is TCL-IMAGE (Van Antwerpen, 1987), programmability to some degree of complexity is provided in an interpretive language adapted from Fortran. Gudmundsson (1982) describes PICAP, a high-level interactive image processing system. The supported interpretive language resembles a mixture of Algol and Fortran, although it lacks defaults and interactive argument prompting. The image processing is devoted to special purpose hardware, making the system non-transportable as a whole.

The next step up from the command-driven systems are the menu-based interactive systems. A menu interface allows for quick and efficient access to commands, at the expense of flexibility however. The IBAS system launched in the early 80's by KONTRON (1985) offers menu selectable commands with parameter specification through the keyboard. Flow control is supported on a rudimentary level when compared to more up-to-date command-interpreter systems. VISILOG of NOESIS (Moss and Howard, 1988) is a modern variant of a menu-based system with a window interface which offers user-friendly selection of commands. The package is mainly dedicated to the morphology operations, however adequate interactive programming facilities are lacking. AIM (Groen et al., 1988a) combines a window management system with a command interpreter. This attractive concept allows either selection of commands through menus and dialogue boxes, or by typing the command names. Defaults and prompting are supported, programmability is absent however.

The addressed categories of image processing systems all have their specific advantages and limitations. The first category offers the advantage of development in a standard programming language. However, they are less suited for interactive image processing and not user-friendly. The second category, the one of the command interpreters is geared towards interactive image processing. Although some command interpreters show similarities to higher programming languages, it turns out that at some point the interpretive language

cannot compete with the flexibility and complexity intrinsic to a standard higher programming language. Moreover, most, if not all existing interpreters in image processing maintain their own semantics and syntax, obstructing acceptability among novice users and standardizations in collaborations. And, from the perspective of the user, command-driven systems have the disadvantage that the names of commands, sequence and meaning of the parameters must be known by heart. A consistent user-interface and on-line information gives some relief but does not help in finding the command name of the intended next step. Finally, a menu-based system provides an user-friendly interface, sufficient to introduce the novice user the concepts of image analysis. However, for a more advanced level of expression menu-driven programming is too cumbersome to be used in practice. Also, it is difficult to expand the menus of these systems if possible at all.

3. The new environment

To overcome these limitations and to combine the virtues of the different categories we propose (and have implemented) a multi-level environment. See Figure 1. On the basis of the preceding argument the first layer of such a system must be an interpreter in a programming language standard. We have adopted the C-standard and named the interpreter SCIL (Van Balen and Ten Kate, 1988), (acronym for Standard C Interpreter Language). In the second layer, a command expander opens the way to effective interactive (image processing) design. The command expander also serves effectivity by inserting default values, prompting and checking the legality of the specified parameters, also hiding the C-level from the computer novice. In the third and top layer a window management system gives menus and dialogues to select commands, now hiding the command level from the naive user. We will now go over the functionality of the layers one by one.

Figure 1. The various levels and components of the multi-level programming environment SCIL. Correct input given at a higher level is transformed into legal input at a lower level.

```

/* function declarations at the top */
.
.
.
int percentile( );
.

/* entry table for the functions */
.
.
.
{"percentile", int_t, 000, 4, percentile, 0},
.
.
.

```

Figure 2. The concept of the library handler is to create an entry in the function table in SCIL of the function 'percentile'. At the top the function 'percentile' is declared in order to make the address known.

Library handler

To provide a flexible expansion of the development system, linking of external libraries takes place through a separate layer, the library handler. Compiled routines can then be integrated in the system by adding one line in the system function file, making use of the C-ability to define a pointer to a function. The functions have to be declared first so that the compiler can arrange for the addresses of the functions. It then suffices to fill the function table list with the function names. Thereafter, compilation and linking must take place to effectuate the functions.

To demonstrate the simplicity of the library handler a sample is given in Figure 2 to add a routine 'percentile' with 4 arguments. The actual entry takes place in an array of structures. Each entry consists of a number of fields involving the formal name, type, manual entry, number of arguments, function address and, when needed a pointer to a return description. When added to the system it is possible to invoke the function 'percentile' in the interpreter as a proper C function call:

```
percentile(a, b, 3, 90);
```

SCIL

In layer 1, the C-interpreter has been implemented. The use of a C-interpreter offers several advantages. First, one can utilize the control directives and complex data structures of C essential to a comfortable image processing environment. Secondly, since development takes place in the interpreter which maintains the standard C syntax, the source code eventually may be compiled and incorporated in the system. Thirdly, C opens the way to C++.

All SCIL syntax is fully and exactly compatible with standard C (Kernighan and Ritchie, 1978). SCIL supports all C-statements with the exception of the static initialization of arrays or structures, local static variables, the sequence operator, bit fields, the goto statement and pointers to functions. They are not (yet) incorporated in SCIL as the use of these features was initially felt to be less relevant for image processing purposes. Currently, we are implementing these last elements to reach full and complete compatibility with the C-standard.

Command expander

On the aspect of user-friendliness, a higher programming language cannot compete with existing command interpreters. The user has to pay for the flexibility of a higher-level language by a rather cumbersome function call mechanism. One has to provide the full name of the function, followed by the correct number of arguments of the proper type and value, and, with the precise C punctuation. To overcome the burden, in the second layer the command expander was developed. The basic task of the command expander is to enter commands without the C punctuation and transpose it to the associated C function call. Simultaneously, arguments are checked for consistency and when arguments are missing the list is completed with default values. Interactive argument prompting is invoked when requested by typing a question mark, when an illegal value is encountered, or when automatic conversion to the proper type fails.

The command expander operates on a command description file in which the information on all commands and their argument specifications is assembled. The argument specifications per routine include type, default and range or legal values. This approach has the advantage that image processing functionality and the user interface are truly separated, and, it greatly facilitates the addition of commands since it circumvents the need to embed an interface for each individual function. To enter a command specification it suffices to edit the command description file, in the following syntax:

```
<command name><menutag1> ... <menutagk>[comment]
<TAB><type 1st argument><default1><min1><max1><prompt1>
:
<TAB><type nth argument><defaultn><minn><maxn><promptn>
```

Continuing the example from the previous section, the command description for 'percentile' looks like:

percentile	NLINEAR_FILT			
image	A	-	-	Input Image
image	B	-	-	Output Image
odd	3	3	23	Filter Size
int	50	0	100	Percentile

Apart from the 'ordinary' types known from C such as int, float, char etc. some object oriented argument types are supported, e.g. odd, even, switch, choice and confirm. The 'image' type is a special argument type, in fact it is an unknown argument type but actually an user-supplied function.

The syntax fed into the command expander is as follows:

```
<command>::=<ComName>{<arg>{,}<arg>}<arg>::="?"|<number>|<variable>
```

Figure 3 shows some of the features of the command expander with the 'percentile' command.

Menu-and-dialogue generator

The menu-and-dialogue generator allows for a quick and efficient selection of commands with minimum input effort. The window interface is created at run-time from specifications in the command description file. Through interaction with the control panel the user exercises control over the menus. Menus are activated by mouse automatically showing the options in pop up menus, see Figure 4. The dialogue boxes are automatically maintained when a command is selected, giving immediately an inventory of all parameters together with their defaults and ranges, if relevant. The parameter types of a command and its ranges are used to specify the graphical attribute in the dialogue box. This information determines whether

Figure 3. Some features of the command expander with the added 'percentile' command, in the commands window bottom left.

it is a number, slider, cycle or a sequence of buttons etc. The menu-and-dialogue generator performs an educated guess to produce the most convenient dialogue. In this way additional specifications in the command description file are not needed, serving simplicity in use, consistency throughout the user-interface and allowing dynamic adaptation. For example, if a sequence of images is created the corresponding at-

Figure 4. The SCILAIM control panel and the pop-up menu at the top and top right.

Figure 5. A dialogue box showing the parameter information, defaults and ranges, with various action buttons.

tribute gradually changes from buttons, to inverted choices to a cycle, depending upon the actual number of images. Figure 5 shows an example of a dialogue box. Logical command grouping under each menu or sub-menu of course improves the effectivity in image processing. Besides, since the interface is run-time generated from a specification the grouping and the menus are easily tailored to ones preferences or customized for a particular application.

4. AIM

The image processing package AIM (Groen et al., 1988a), consists of an extensive set of image processing operations. AIM has been interfaced to SCIL as a set of subroutines loaded by the library handler. In AIM attention has been given to those implementations which are tailored to the flexibility of sequential processors such as the general (micro) processors (Groen et al., 1988b). The obvious advantage of these software implementations over special purpose hardware is that they can easily be ported to other architectures e.g. to benefit from faster execution or to become more widespread in use. At present the set of image operations consist of approximately 200 commands, they comprise:

File operations. Images of various sizes may be loaded and stored in different formats.

Arithmetic grey value operations. These include add, multiply, divide, minimum and maximum either on images with images or images with constants.

Bitwise grey value operations. These are the and, or, exor and not operators acting upon and between grey value images.

Bitplane operations. This group contains the and, or, exor and not operators together with the setting, resetting and copying of bitplanes.

Grey value filters. Both linear and non-linear filters are supported. Among the linear filters are laplace, gradient, roberts, gaussian and uniform filter. User defined convolutions by separable kernel and by a rotational symmetrical kernel are available, both restricted to a 3×3 neighborhood size. A fast implementation

of a 3×3 median filter is present and a general percentile filter up to a 23×23 matrix. Other non-linear filters are the kuwahara, minimum, maximum, local contrast stretching and edge preserving smoothing filters.

Point operations. This group comprises histogram equalization, contrast stretching, table lookup, grey value mapping and interactive or automatic thresholding.

Morphological operations. The morphological group supports operations which focus on cellular logic processing. Erosions, dilations, skeletons and propagation are present. Contour, majority vote, removal of

Sample of Image.h

```
#define PIXEL unsigned char

typedef struct Image {
    PIXEL      *lcore;           /* upper-left corner in core */
    int         llenx;           /* x axis dimension */
    int         lleny;           /* y axis dimension */
    unsigned long lsize;         /* image size */
    int         ltype;           /* image type */
    int         lgrey;           /* number of grey values */
    char        *Vport;          /* image handle */
    int         Vbitplane;       /* bitplane of Vport */
    char        *Swap;           /* swap area of viewport */
    char        Image[40];       /* image name */
} IMAGE;

#define ImageData( lp )      ( lp ) -> lcore
#define ImageWidth( lp )    ( lp ) -> llenx
#define ImageHeight( lp )   ( lp ) -> lleny
#define ImageSize( lp )     ( lp ) -> lsize
#define ImageType( lp )     ( lp ) -> ltype
#define ImageNgrey( lp )    ( lp ) -> lgrey
#define ImageVport( lp )    ( lp ) -> Vport
#define ImageBitplane( lp ) ( lp ) -> Vbitplane
#define ImageSwap( lp )     ( lp ) -> Swap
#define ImageName( lp )     ( lp ) -> Image
```

Figure 6. The initialization procedure of image processing in SCILAIM, including a sample of the definition file 'image.h'.

pepper and salt noise are other binary operations. Special linefigure pixels can be detected, single pixels, end pixels link pixels and vertices.

Measure. This group is used to analyze specific aspects of an image, such as image data, calculation of an histogram, labeling of binary images, the measurement of area, perimeter and contour ratio of binary objects in an image. The euclidian distance transform is also available.

Display. The display commands control the source and nature of the display, if automatic display is required and the display table. Functions are available to create and destroy images or set the display size.

Utilities. This group provides commands to copy parts of images, assign a value to an image, computation of various grey ramps, scale, copy and sample images.

Fourier. The fast fourier transform and its inverse allow frequency analysis, frequency domain filtering and complex multiplication. In addition display of magnitude, real and imaginary part of a complex image and the conversion of complex images to grey value images and vice versa are supported.

Graph. These commands transform binary linefigure images to vector and graph representations, translating to straight and circular line segments.

In AIM there are four standard 8 bit images available represented by the letters A through D. Images are by default 256 by 256 pixels but may be of arbitrary size.

5. SCILAIM, a session

When inserting AIM as a function library in SCIL the powerful combination arises as image processing is pre-eminently apt for interaction with prompt execution. At startup a function 'initaim' defines the window structure pointers A to D, allocates memory for the image structure and buffers, and couples the window display to the image structure. Figure 6 shows the relevant steps in the start up procedure. Additional images may be created with user defined names and sizes. Created images are treated as objects, they are known to the menu-and-dialogue generator and thus appear in dialogue boxes, see Figure 7. This object

Figure 7. Sample session of SCILAIM showing the result of multiplicative shading correction by estimating the background image through min-max filtering.

oriented mechanism allows for a robust memory management. The image structure and some macros to access the image structure are defined in the include file 'image.h'. After initialization the user can intermix C instructions and (expandable) commands. Consider the next sample where shading correction is performed, approximating the background image through minimum and maximum filtering. If multiplicative shading is assumed, the original image has to be divided by the estimated background image, multiplied by a constant. This constant is usually taken as the modulus value determined from the reference image. For example the user can type:

<i>user</i>	<i>meaning</i>
readf cell	read cell image in <i>A</i>
lmax <i>a b</i> 31	maximum filtering <i>A</i> to <i>B</i>
lmin <i>b b</i> 31	minimum filtering <i>B</i>
unif <i>b b</i> 31	uniform filtering <i>B</i>
float data[256];	declare histogram array, 'data'
histoh(<i>b</i> , data);	get histogram data
int <i>i</i> , modus=0;	declare integer variable ' <i>i</i> ', 'modus'
for(<i>i</i> =0; <i>i</i> <256; <i>i</i> ++)	determine modus from histogram data in loop
if(data[<i>i</i>]>data[modus])	
modus= <i>i</i> ;	
div <i>a b c</i> modus	divide original with estimated background image multiplied by modus

See Figure 7. The pointers A to D are actually variables in the interpreter pointing at the image structure internally used by the AIM routines. This enables the user to access the pixel values directly in the interpreter. One might inspect or alter the actual pixel values just as one would in a traditional C program. The next few lines show a sample to map the grey levels to relate linearly to optical density according to:

$$d(x, y) = -\log_{10} t(x, y) \quad (1)$$

<i>user</i>	<i>description</i>
#include "image.h"	obtain IMAGE structure description and definitions
PIXEL table[256];	declare lookup table
int <i>i</i> ;	integer ' <i>i</i> '
for(<i>i</i> =0; <i>i</i> <256; <i>i</i> ++)	fill table in loop
table[<i>i</i>] = -100*log 10((double)(<i>i</i> +1)/256.);	
PIXEL * <i>p</i> ;	declare pixel pointer ' <i>p</i> '
IMAGE * <i>im</i> =(IMAGE *) <i>a</i> ;	' <i>im</i> ' points at image structure of <i>A</i>
<i>p</i> = ImageData(<i>im</i>);	sets ' <i>p</i> ' at start of image data
<i>i</i> = ImageWidth(<i>im</i>)*ImageHeight(<i>im</i>);	set ' <i>i</i> ' to image size
while(<i>i</i> --)	do the mapping in loop
* <i>p</i> ++ = table[* <i>p</i>];	
gdisplay <i>a</i> ;	update display

Apart from typing C instructions in the direct command mode it is also possible to create C functions and call them from the interpreter. For example to construct a grey ramp first a file ramp.c has to be created:

```

File ramp.c:
#include "image.h"
ramp(im)
IMAGE *im;
{
    PIXEL line[256], *p;
    int i;
    p = ImageData(im);
    for(i = 255; i >= 0; --i) line[i] = i;
    for(i = 0; i < 256; i++){
        memcpy(p, line, 256);
        p += ImageWidth(im);
    }
    gdisplay(im);
}

```

In the interpreter session it now suffices to type:

```
ramp(a);
```

to automatically load and let the interpreter execute the ramp. It is feasible to describe useful tools as commands in the command description file to take full advantage of the command expander and menu-and-dialogue generator.

Diagnostics

The interpreter will complain when the syntax is not correct. The error messages are intended to be self explanatory. Forgetting a semicolon for example will yield,

```

[C1] int i
int i → semicolon ';' expected
[C2]

```

When errors occur in loaded functions the line number where the error was encountered in addition is reported. Even when syntactically correct, program faults may occur through illegal memory references, illegal instructions or floating point exceptions. If either one of these situations are provoked the interpreter will stop executing, report the corresponding message and where it was detected, and fall back to a known state. Beside program faults outside world signals can be generated to interrupt execution of loops, programs etc.

6. Conclusions and discussion

In this paper we have presented an interactive multi-level image processing environment, which provides user support at various levels of acquaintance with a computer. The primary requirements in our setting are to introduce the novice user to the fundamentals of image processing (with the dialogue generator), to accommodate the experienced scientist with a high-level design tool (at the command expander level) and furnish the application programmer an interactive standard programming language (the C interpreter). The user community consists of operators, students, programmers and research scientists in medicine, biology, physics, robotics and computer science in a dozen groups in Europe. Applications include image analysis

of tumor tissue, image cytometry, positional reconstruction of neuronal data, 3D-image analysis and reconstruction.

Compared to existing systems, for the purpose of image processing, this new environment can provide several major advantages. The library handler provides straightforward integration of external libraries and user-developed routines. The interpreter SCIL ensures algorithm development with the full complexity and flexibility of the C programming language. The user-interface at the command and the menu-and-dialogue level has the advantage of being generated from specifications assembled in the command description file. Therefore, the environment easily allows for tailoring the user-interface to a particular application or end-user group. Eventually, the designed applications may be compiled and used as a stand-alone task or added to the system.

The environment also has growth potential. Object oriented programming languages such as C++ are a to C particularly profitable extension (Piper and Rutovitz, 1988). It provides the facility to construct 'classes' of variables and to overload arithmetic operators specific for the members of that class. If, for example A and B are instances of the class 'image' it is standard C++ to implement the '+' operator, as in $A+B$ as the pixelwise sum of two images and to simultaneously interpret $A+2$ as an integer addition to image A . In addition, C++ provides better protection and managing objects of a class than conventional C does. Therefore, upgrading SCIL to SCIL++ will increase the functionality with respect to image processing further.

The present SCIL environment has been implemented under UNIX on SUN (1988) Microsystems SUN-386i, SUN-3 and SUN-4 workstations. The menu-and-dialogue generator uses SUN's multiwindow management system. The window-dependent components will be re-written to run also under X11 to secure the transportability to a large variety of workstations. Porting SCILAIM to the HP9000 workstation proved to be very simple (where currently the menu-and-dialogue level was omitted, awaiting the X11 implementation (Scheifler and Gettys, 1986)).

It is hoped that this paper will stimulate a renewed interest in image processing languages and will help establish standards in transportable image processing software. The availability of such a platform could lead to an amplified exchange of programs, algorithms and concepts in the community.

References

- Groen, F.C.A., R.J. Ekkers and R. de Vries (1988a). Image processing with personal computers. *Signal Processing* 15, 279-291.
- Groen, F.C.A., P.P. Jonker and R.P.W. Duin (1988b). Hardware versus software implementations of fast image processing algorithms. *NATO Advanced Research Workshop on Real-time Object and Environment Measurement and Classification*, Maratea, Italy, 31 August-3 September 1987.
- Gudmundsson, B. (1982). An interactive high-level language system for picture processing. *Computer Graphics and Image Processing* 18, 392-403.
- Haralick, R.M. and G. Minden (1978). Kandidats: An interactive image processing system. *Computer Graphics and Image Processing* 8, 1-15.
- IBAS (1985). *IBAS 2000 Reference Manual*. Kontron Image Analysis Systems. 4.3.
- IRIS (1985). *IRIS IPS/68k Reference Manual*. Kontron Image Analysis Systems 1.1.
- Kaneko M. and K. Sakaue (1983). Design and Implementation of SPIDER—A transportable image processing software package. *Computer Vision, Graphics and Image Processing* 23, 273-294.
- Kernighan, B.W. and D.M. Ritchie (1978). *The C Programming Language*. Prentice-Hall, Englewoods Cliffs, NJ.
- Landy, M.S., Y. Cohen and G. Sperlin (1984). HIPS—A Unix-based image processing system. *Computer Vision, Graphics and Image Processing* 25, 331-347.
- Moss, M.C. and C.V. Howard (1988). Look before you leap. *Laboratory Practice* 37 (1), 15-23.
- Piper, J. and D. Rutovitz (1985). Data structures for image processing in a C language and Unix environment. *Pattern Recognition Letters* 3, 119-129.
- Piper, J. and D. Rutovitz (1988). An investigation of object-oriented programming as the basis for an image processing and analysis system. *Proc. 9th ICPR*, Rome 14-17 November 1988, 1015-1019.

- Preston, K. Jr., (1981). Image processing software. A survey. In: L.N. Kanal and A. Rosenfeld, Eds., *Progress in Pattern Recognition*. North-Holland, Amsterdam.
- Preston K. (1983). Progress in image processing languages. In: *Computing Structures for Image Processing*. Academic Press, New York, 195-211.
- Scheifler, R.W. and J. Gettys (1986). The X window system. *ACM Trans. Graphics* 5 (2), 79-109.
- SUN (1988). *Sunview 1 Programmer's Guide*. Sun Microsystems, Inc.
- Van Antwerpen, G. and R.J. Van Munster (1987). *TIPS Users Manual*. Institute of Applied Physics - TNO, Delft, The Netherlands.
- Van Balen, R. and T.K. Ten Kate (1988). *SCIL Version 3.0 user's manual*. Free University Amsterdam, 1988.