A visual programming interface for an image processing environment

Koelma, D.C.; Smeulders, A.W.M.

DOI
10.1016/0167-8655(94)90125-2
Publication date
1994

Published in
Pattern Recognition Letters

November 1994

Pattern Recognition Letters

ELSEVIER

Pattern Recognition Letters 15 (1994) 1099–1109

A visual programming interface for an image processing environment ☆

Dennis Koelma *, Arnold Smeulders

Faculty of Mathematics and Computer Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, Netherlands

Received 20 May 1993; revised 25 May 1994

Abstract

The paper presents a visual programming interface for an image processing environment improving traditional interfaces by combining the user-friedliness of the menu & dialogue approach with programming capabilities. The interface gives a layman the opportunity to develop powerful image processing applications in a two-dimensional, data flow metaphor. This metaphor provides a better insight into the structure of an application than a traditional interface.

The visual programming interface was developed in the principle of dialogue independence (Hartson and Hix, 1989). The communication between the user interface components and the computational components is well defined without jeopardizing the expressive power of the user interface. This allows for the independent development of image processing functions and user interface aspects. The integration of new functions into the environment is trivial.

Keywords: Image processing environment; Visual programming; User interface; Dialogue independence

1. Introduction

The user interface of an image processing environment is a key aspect of the proper functioning of such an environment. A good user interface can significantly reduce the development effort of new image processing applications. The user interface also determines the usability of the environment to the various classes of users. Several types of user interfaces for image processing environments have been introduced in the past. Each of these user interfaces has its own specific advantages and disadvantages, to be discussed in the next section.

The search is for a user-friendly interface with programming capabilities. From the fact that image processing development very much is an interactive visual endeavor, it is obvious to consider a visual programming interface first. Visual programming (Shu, 1988; Chang, 1987; Myers, 1990; Raeder, 1985) has been advocated successfully to make programming more user-friendly. In this paper, it is investigated whether visual programming can be used to enhance the expressibility of the interface without the loss of user-friendliness.

The visual programming interface presented here differs from other interfaces in that those environments have a strong link between the visual programming layer and the image processing functions (Chang et al., 1985; Tanimoto, 1990; Blanford and Tanimoto, 1986; Rasure and Williams, 1991; Williams and Rasure, 1990; Lau-Kee et al., 1991) or the visualization functions (Upson et al., 1989; Dyer, 1990). In such a strongly linked system, functions are

☆ This research was supported by STW project AWI92-1691.

* Corresponding author. Email: koelma@fwi.uva.nl.

SSDI 0167-8655(94)00044-4

to be developed specifically in the visual programming context. Each function requires knowledge of the visual programming layer to function properly. This places severe restrictions on the ongoing development of image processing functions as well as on the re-usability of the software. The true separation of the user interface from the image processing functions would eliminate their interdependence. The visual programming interface can thus be used in combination with different image processing libraries, e.g. the SPIDER (Tamura et al., 1983) library or the KHOROS (Rasure and Williams, 1991) library, without the need for re-design of these libraries.

The paper discusses different types of user interfaces for image processing environments in Section 2. Section 3 describes the concept of the visual programming interface and the communication of the user interface with the image processing functions. Section 4 gives some details of the implementation of the visual programming interface in the image processing environment. Section 5 evaluates different aspects of a prototype version of the interface.

2. User interfaces for image processing environments

The oldest type of user interface is the programming language interface (Tamura et al., 1983; Dewaele et al., 1988). The interface effectively consists of a document that describes a library of subroutines or functions that can be used to construct new image processing applications. A programmer is able to use the full expressive power of the programming language in the development process. Such a setup may work for users with programming skills, but it does not serve the present day general need for image processing in a broad community. Another drawback of this type of interface is its limited interactive capability, hampering quick development in any case.

To simplify the programming language and to get immediate visual response, younger systems use a command based interface. These systems are usually based upon an interpreter of some kind. The interpreter can be an existing interpreter such as the Unix shell (Landy et al., 1984; Piper and Rutovitz, 1985). It can also be an interpreter for an existing programming language (ten Kate et al., 1990; CBP, 1991; Ibbotson, 1990) or some derivative thereof (Haralick and Minden, 1978; Gudmundsson, 1982; Groen et al., 1988; Bailey and Hodgson, 1988; TNO, 1991). In this type of system there is no obvious correspondence between a command and the data it operates on except for names of variables that indicate the (temporary) storage of the data. This makes programs hard to read and understand if the user is not a programmer. Another disadvantage of these systems is that they do not provide a good overview of the often very large number of available commands. They require the user to learn the commands, their parameters, and the calling sequence by heart. This severely limits the usability of the system for novel and casual users.

To provide a quick overview of the available commands some of the command based systems are equipped with a menu & dialogue interface (ten Kate et al., 1990; CBP, 1991; Groen et al., 1988). The menu gives an overview of the available commands and the dialogue box simplifies the parameter handling of the command. This results in a user-friendly interface that is also suited for first time use. The menu based interface, however, lacks the programming capabilities of the programming language interface and the command based interfaces. It provides a history mechanism to generate command files but no more. These command files have no control structures and do not allow for the hierarchical structuring of an image processing solution. The programs give no insight into the cooperation of several commands due to the linear ordering of the commands and the use of variable names to denote intermediate results.

The aim of the visual programming interface is to combine the user-friendliness of the menu & dialogue interface with some of the programming capabilities of the other interfaces. The interface should support the increasing number of laymen involved in the development of image processing applications.

3. Design of the visual programming interface

In the design of the visual programming interface two aspects have been taken into special consideration: (1) which metaphor to use in the visual programming interface; and (2) what is the definition of the communication link between the user interface and the computational components.

3.1. Selection of the visual programming metaphor

The choice of a proper metaphor for the visual programming interface is essential since programming is easier to learn when a physical or mechanical analogy is present for the computational model (Tanimoto and Glinert, 1986). This is especially true when the problem is already understood in terms of the analogy. In general, the metaphor should be able to express applications clearly to simplify the transcription of solutions into programs and vice versa. The metaphor should be simple to avoid confusing the user with unnecessary details. In order to select a proper metaphor for the visual programming interface, we first characterize the type of the image processing applications we consider here.

Image processing applications are characterized as streams of functions operating on data fields holding the digital intensity values. In a later stage of processing the data field may be restricted to an interesting part in the image and/or the data field may be transformed into a few numericals characterizing the image or an object therein. The functions used in an application are recruited from a library of image processing routines with standardized data structures. The functions are grouped in subtasks trying to solve different, independent parts of the problem. The behaviour of these subtasks is determined by adjusting parameters which can be optimized more or less separately. Often, the application designer evaluates the performance of different techniques aimed at solving a certain subtask.

The described type of image processing application has previously been developed using interfaces like the menu & dialogue interface and the command based interface. The applications are frequently encountered in industrial, medical or microscopical image processing, as well as in geometrical information systems and document image analysis.

In the visual programming interface the functions that are used to construct an application should be represented by icons to aid in the expression of their functionality. The construction of the application should not be based on an algorithmic metaphor and the explicit use of variables should be avoided as laymen are not familiar with these concepts and their use. It would complicate the use of the visual programming interface. As a general principle, programming constructs that are not essential in the development of the application should be omitted. In our class of applications these constructs include the definition of data structures, recursive function calling and pointer handling. They are not dealt with in the visual programming interface; they can be used inside the image processing functions of the library, however.

The aforementioned considerations make hierarchical data flow graphs the best choice for the metaphor of the visual programming interface. Data flow provides a view of computation which shows the data flowing along arcs from one node (filter function) to another, being transformed as it goes. Data flow graphs are a better alternative than flowcharts and Nassi–Shneiderman diagrams as these are based on an algorithmic metaphor. Compared to iconic programming and form manipulation languages, data flow graphs provide a clear expression of the flow of control in an application. State transition diagrams and petri nets are not really suited for development of image processing applications as they concentrate on the state a process is in and not on its results.

3.2. The communication link

Dialogue independence (Hartson and Hix, 1989) is an important issue in present day user interface design. It expresses the need for the separate development of the user interface and the computational components. This implies that image processing functions can be developed without taking the user interface into consideration in the design and implementation process.

Dialogue independence is based on the formal definition of the communication between the user interface and the computational components. The addition of new image processing functions should become almost trivial by limiting the communication to the bare essential and keeping the formal definition simple. Care should be taken that the thin communication link does not jeopardize the expressive power of the user interface.

We have based the definition of the communication link on three assumptions: the existence of a C-callable interface to the image processing functions, the adoption of the data flow metaphor in the computational structure of the library, and ability to cre-

ate, copy, and destroy images. They are valid in most image processing libraries.

A communication link consists of a list of image processing functions and the definition of data objects. The formal description of a function holds the name of the C-callable function and its parameters. The data objects form the parameter types of the functions and are defined by functions to create, copy, and destroy them. The definitions of a communication link constitute all information needed by the visual programming interface to execute visual programs. The library has no knowledge of the existence of the user interface.

4. Implementation

The visual programming interface has been built on top of the image processing environment SCIL_Image (CBP, 1991). SCIL_Image is an interactive environment with multiple interfaces based on a C-interpreter, to which any C-callable software library can be added. A function can be added to SCIL_Image by inserting an entry in a command description file. Upon completion the function is accessible at all levels of the interface. These user levels are a C-interpreter, a command expander, a menu & dialogue interface, and a visual programming interface (Koelma et al., 1992).

The communication link between the user interface components and the image processing functions is defined by the description of the functions and the data objects. The description of a function consists of its name, place in the menu hierarchy, icon, and the description of its parameters. For example, Fig. 1 shows the entry that describes the function *erosion 3×3*. The description of a parameter consists of the data object type, the parameter type, the default value, the range or set of admitted values, and the prompt. The parameter type specifies whether it is an input, output, or control parameter. Input and output parameters are represented by pins on the left resp. right side of the icon of the function. Control parameters are input parameters that are not represented by pins to avoid screen clutter. The control parameters are adjusted through a dialogue box. The parameters of the functions consist of data objects. These data objects are defined by functions to create, copy, and destroy them. Four types of data object have been defined to date: images, integers, doubles and strings.

FUNC erosion3x3
MENU morphology
ICON erosion3x3.bitmap
ARGS

image	in	B	-	-	Input binary image
image	out	B	-	-	Output binary image
int	in	1	1	-	Number of iterations
choice	con	8	4|8	-	Connectivity
switch	con	Off	On	Off	Set edge pixels

Fig. 1. The description of the function *erosion3×3*.

The visual programming interface consists of a library handler and multiple worksheets. The library handler (Fig. 2) gives a hierarchical overview of the image processing functions in the library (or libraries) known to the interface. The functions are divided into groups with the same hierarchy as in the

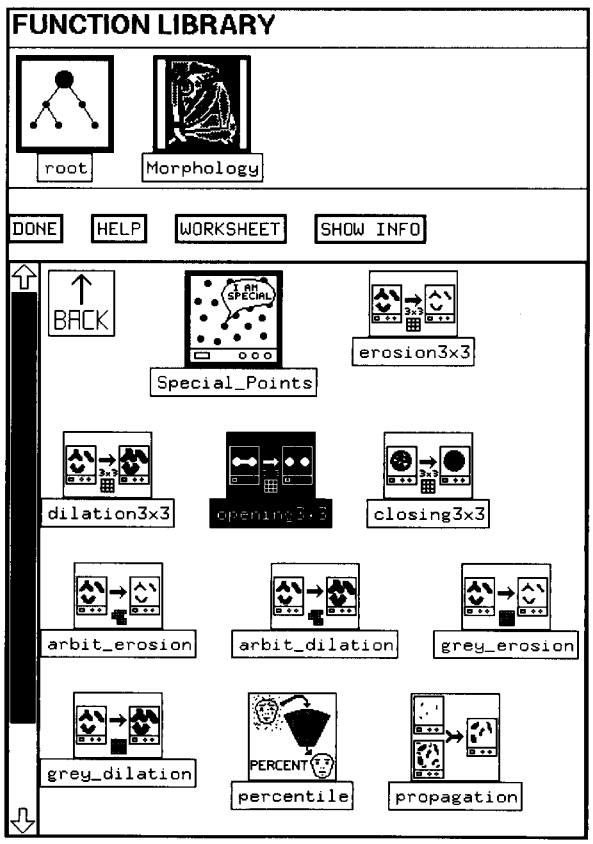

Fig. 2. An example of the face of the library handler.

menu system. Groups are distinguished from functions by their thick border. The user can move to a specific group by selecting its icon. A function can be selected from the library by clicking its icon (*opening3×3* in Fig. 2).

The worksheets (Fig. 3) are used to construct hierarchical data flow graphs from the functions in a library. A worksheet consists of a control panel and a workspace. The functions of the control panel can be executed by first selecting the operation icon in the control panel and then the data on the workspace, or vice versa. Only part of the workspace is visible as indicated by the scroll bars. The size of the workspace is unlimited.

Using the worksheet editor the user can copy functions from the library to a workspace, move the functions around the workspace, add comments to the functions, and remove functions from the workspace. The user can make connections between functions by selecting an input pin and an output pin with the same data type. An output pin can be connected to multiple input pins but an input pin can have only one connected output pin. A hierarchical structure in the data flow graph can be created by combining several functions into a singly visual function with its own user defined icon. The visual function has the same status as the functions from the library. However, it can be edited in a separate worksheet or split into its constituting functions again.

The constructed data flow graph is executed with the aid of the C-interpreter. As soon as a node in the graph has valid data on all input pins, data objects of the appropriate type are created at the output pins and the C-interpreter will execute the function that corresponds with the node. In case the user selects a demand driven order of execution, nodes that need to be executed are maintained in a list until nodes are reached that have valid data on all input pins and the functions in the list can be executed. Optionally, the data objects that are no longer needed can be destroyed to reduce memory allocation. Data flow

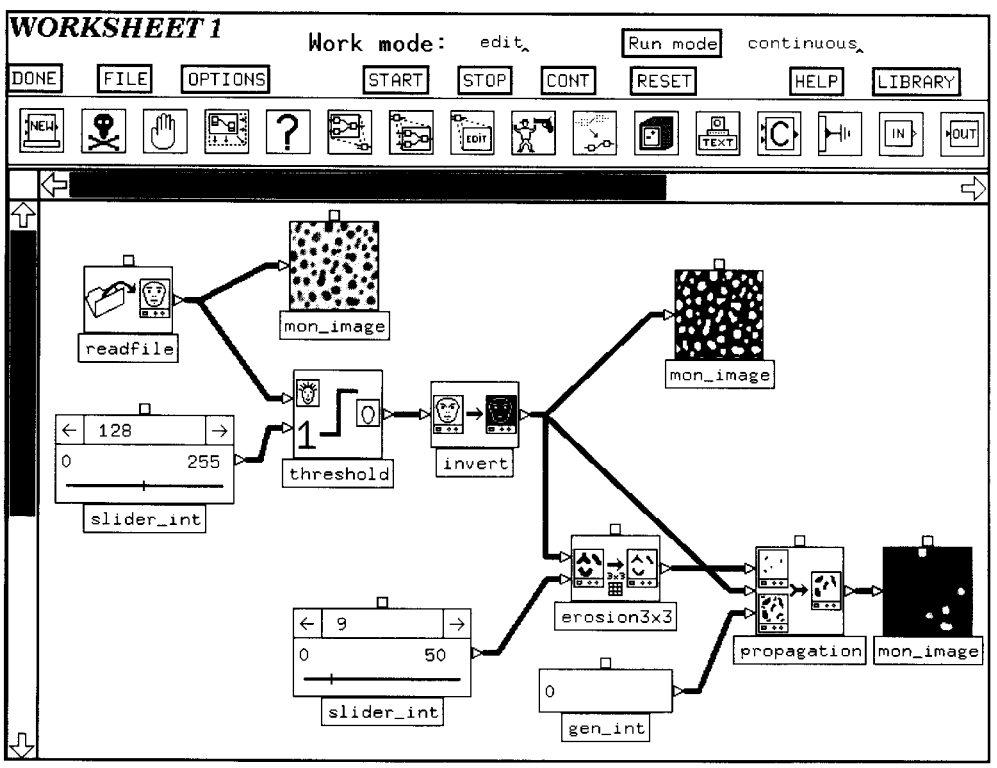

Fig. 3. Example of a worksheet.

graphs can be executed in single step or continuous mode.

Visual programs can be stored and retrieved as visual programs or they can be stored as C-programs. In the latter case they can be executed without the visual programming interface.

5. Results and evaluation

In this section results on the operation and usability of a prototype version of the visual programming interface are evaluated.

5.1. Dialogue independence

The integration of the Image (CBP, 1991) processing library with the visual programming interface shows that it is feasible to separate the user interface from the computational components completely. Parts of the Image library were designed and implemented over 10 years before the visual programming interface was designed. It was not modified in the integration process. The result also shows that the thin communication link does not limit the expressive power of the system in the sense that it has the desired expressibility. The expressibility is that of a controlless data flow graph.

We feel that the integration of any library of image processing functions is simple as long as the data flow metaphor is adopted. The addition of a function is trivial. The definition of data objects requires a little bit more insight in the use of those objects. On the other hand, the implementation should not be too difficult by taking the existing data objects as an example. Furthermore, the functions required for the definition of data objects are usually already present in the library. There are no restrictions to the complexity of new functions or data objects.

5.2. Overhead

The visual programming interface causes some additional computational overhead compared to a menu & dialogue interface or a command based interface. The overhead has two components. The first is caused by the algorithm that determines the next function to execute. This function is selected from a list of candidates that are connected with the already executed part of the program. The length of the list depends upon the size and structure of the program. Typically the list contains 5–10 items so the cost is small. Much longer lists would not pose a serious problem.

The second component of the overhead is the creation and deletion of data objects in the execution. Even without optimization of the computational burden by maintaining a pool of data objects, the overhead is small (less than 5%) compared to the computational effort of a typical image processing function.

5.3. Icons

Icons can be used to express the meaning of a function graphically. By Lodding's taxonomy (1983), there are three types of icons: respresentational (a picture), abstract (a symbol), and arbitrary (a sign). The icon can be used to express the meaning or the domain of a function.

The addition of icons to simply stating function names was found useful. Depending on the expressive power of the icon as compared to the function name of that particular routine, the interaction with the image processing functions was enhanced. Icons make the learning, using and remembering process easier because well designed pictures can convey more meaning than text. The integration of the Image library has revealed that the design of good icons is essential but also hard to do. In case neither the meaning or the domain of the function can be expressed clearly in the icon, the name can always be used as a port of refuge with the same level of expression as the menu & dialogue interface.

5.4. Data flow graphs

It is our experience that data flow graphs are a more natural way for a layman to express image processing applications than textual interfaces. The two-dimensional layout of the data flow graphs seems to be closer to the human way of thinking (Smith, 1975). This simplifies the translation of ideas to programs and enhances the understanding of a program. The data flow metaphor has a closer relation with the strongly data oriented nature of image processing applications.

Another favorable aspect of the use of data flow

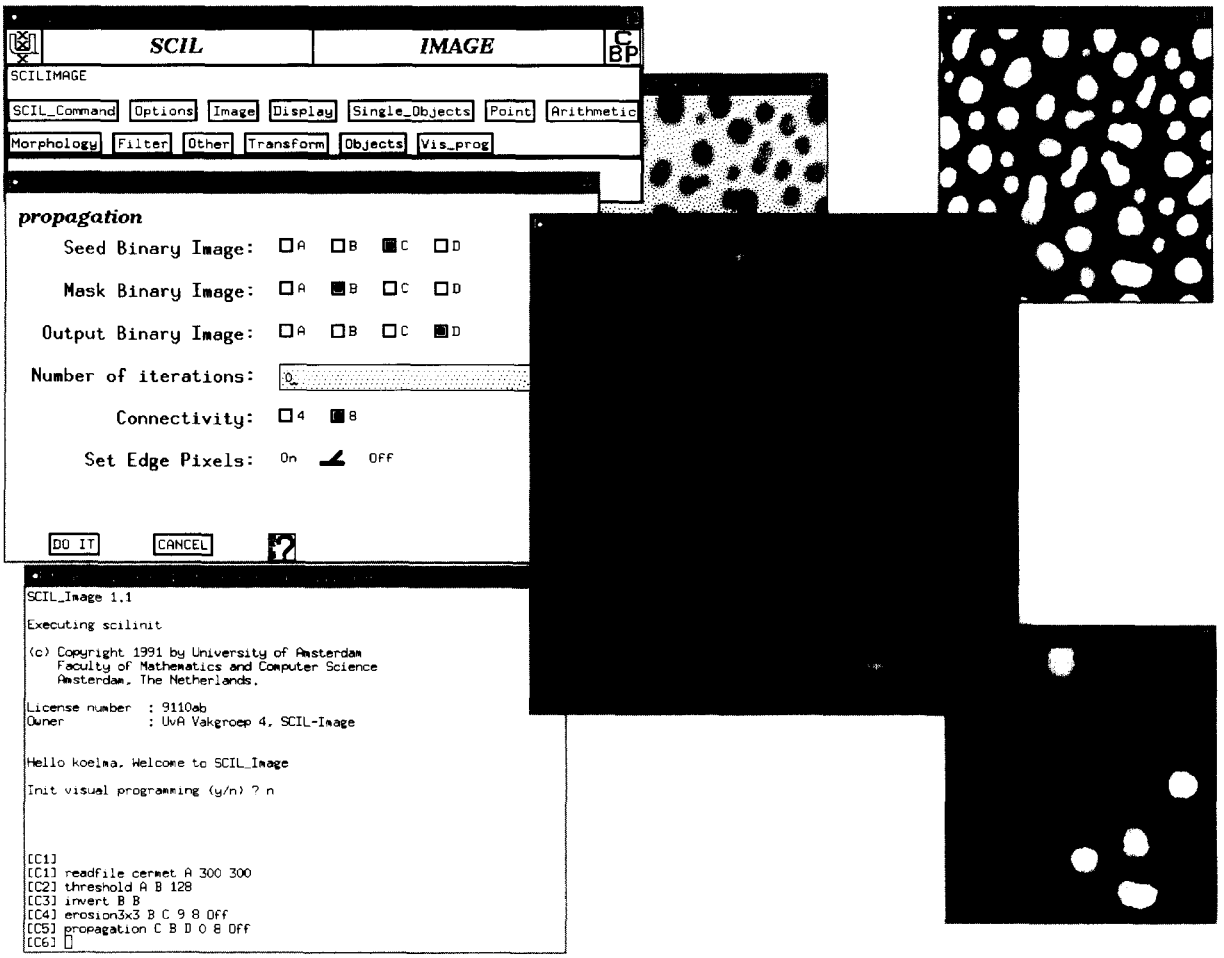

Fig. 4. Example of the screen of a command and menu & dialogue based interface.

graphs, is the absence of the explicit definition of variables. In a textual interface, explicit variable names are used to denote intermediate results in an application. The use of variables makes it more difficult to locate the result of a certain operation. For example, in the command and menu & dialogue based interface the correspondence between the operations in the lower left hand corner in Fig. 4 (also shown in Fig. 5) and the resulting images on the right side in Fig. 4 is not clear. The direct, visual relation between an operation and its result in a data flow graph makes it easier to understand the application. To illustrate the point, compare the data flow graph in Fig. 3 with a typical command interpreter instruction sequence in Fig. 5 expressing one and the same application. We

```
readfile cermet A 300 300
threshold A B 128
invert B B
erosion3x3 B C 9 8 Off
propagation C B D 0 8 Off
```

Fig. 5. The command instruction sequence in the lower left hand corner of Fig. 4.

maintain that for any user Fig. 3 is easier to interpret than the configuration in Fig. 4.

The data flow graph also simplifies the construction of a hierarchical structure in the program. This can be done by selecting the functions that form a coherent part in the application (Fig. 3) and combining them into a single function (Fig. 6).

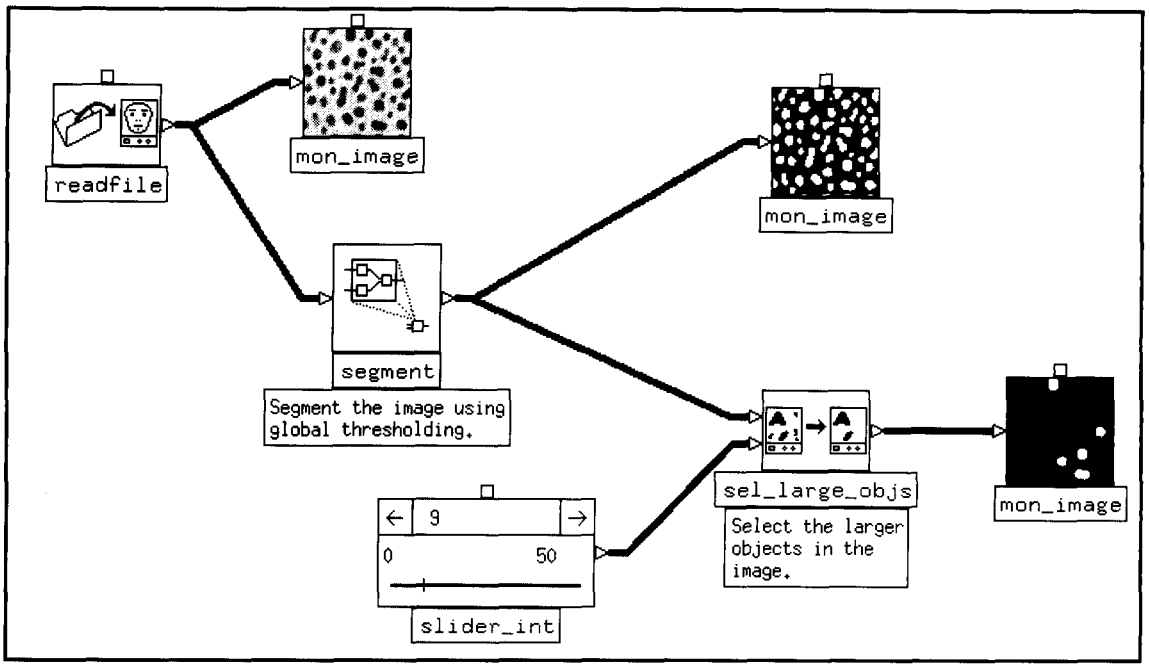

Fig. 6. Example of a hierarchical structure.

A considerable disadvantage of data flow graphs is that they require a large space on the screen. This problem can be solved only partially through the use of scrolling and/or abstraction mechanisms such as a hierarchical structure in the program.

5.5. Parallelism

Parallelism can be exploited in several ways using large-grain data flow (Babb, 1984). The obvious way is to use the data flow dependence to determine the functions that can be executed in parallel at a certain point in the execution of the program. The functions can be distributed over several processors in a shared memory configuration. The scheduling overhead of a parallel implementation is small because of the computational complexity of a typical image processing function. The overhead can be avoided by connecting the icon of a function to a processor, designating the function to be executed on that processor. The shared memory configuration is not essential in case the communication link is extended with functionality to copy objects across a network. However, such an approach would introduce more overhead and requires extensions to the current implementation.

A less common way to use parallelism is to compare different approaches to the solution of a subtask on line. The task can be assigned to several, parallel processes using alternative data structures and/or numerical solutions. Upon completion of the task, differences in the computational effort or numerical results in the outcome can be evaluated. Such a procedure could be a standard element of image processing functions making solutions more efficient and robust. This approach also requires synchronization in the execution of the data flow graph.

5.6. Execution and parameter adjustment

In our implementation, the execution of the data flow graphs is guided by an interpreter. The interpreter can provide a high level of interaction with the user since the program can be re-executed immediately upon changes made to it. In many cases, the program does not require re-execution from the start because the data flow dependence determines in which part the modification will have effect. All data is kept in memory so updating is done quickly.

The interpreter also facilitates direct animation of the execution by highlighting the icon of the function under execution. The visualization of the execution through animation helps in understanding, debugging and optimizing the program.

Parameter adjustment of the tasks that solve the various, independent parts of the problem is an important aspect in image processing application development. In data flow graphs the effect of parameter adjustment can be traced easily through the data flow dependence. The effect can be evaluated almost immediately since adjustment of the parameters is done mostly downstream the program being developed and it is not necessary to re-execute the program from the start. For example, changing the sizes of the selected objects in the graph in Fig. 3 by pulling the slider connected to the function *erosion3×3* will cause only the functions *erosion3×3*, *propagation*, and *mon_int* to be re-executed. The monitor will then show the new objects. This clearly provides a higher level of interaction than the other types of interfaces can achieve.

5.7. Control and data structures

The use of control and data structures is currently not supported in our implementation of data flow graphs. However, the standard control structures and facilities for the definition of data structures of the C-language can be used in the C-icon. The C-icon is a function that is used like any other function in the data flow graph. The number of parameters of the function and their type can be adjusted through pop-up menus. The parameters are available in the C-function body through symbolic names. For example, the function *threshold* in Fig. 3 could be replaced with a C-iron with two input parameters (an image and an integer), one output parameter (an image), and the following body:

```
threshold(INPUT1, INPUT2, OUTPUT1);
```

The body may contain any valid C-function body and can be changed at all times without the need for recompilation.

Control structures are not supported in the visual language as we feel it is impossible to combine user-friendliness and flexibility in these structures. The problem cannot be solved by limiting the flexibility of a general control structure to create a specialized control structure that is still useful in image processing applications. For example, an iterative procedure is very hard to visualize as is shown in the attempt in Fig. 7. In the KHOROS environment (Rasure and Williams, 1990) the control structures are based upon the use of variables and thus are not visual or even visible. Prograph (Cox et al., 1989) uses a visual expression for control structures but has a very complex model for the execution of these structures that cannot be derived easily from its visual expression.

All attempts to specify a visual control structure suffer from essentially the same problem: the loss of the one directional, natural flow of the data. The problem is caused by the need to specify multiple connections between the specification of the function, of the condition, and of the iteration. These multiple connections are needed to represent the initial and iterated values of the data and the iterator.

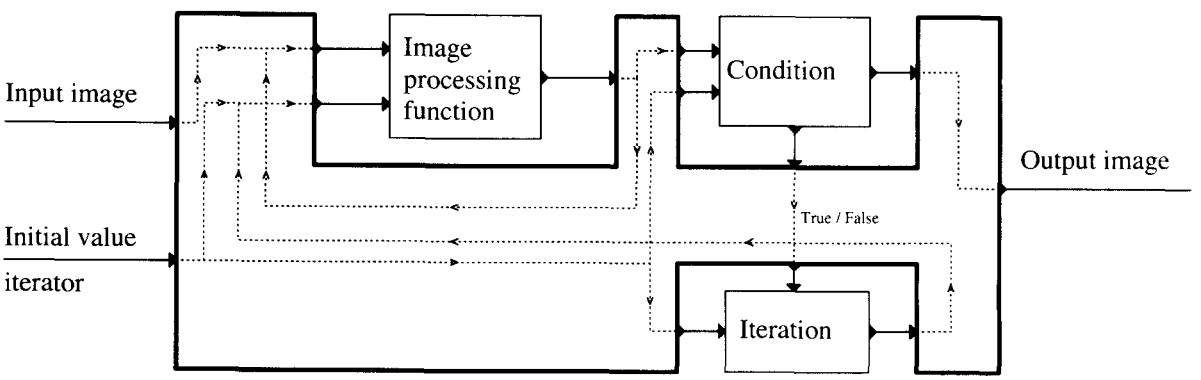

Fig. 7. An attempt to specify an iterative procedure in a data flow graph.

This makes the graphical expression of control artificial. Until a clear graphical expression has been found we rely on the C-icon.

The definition of data structures also has been omitted from the visual programming language. Handling data structures is not of primary importance in the high level image processing applications considered here. The immediate drawback of this limitation is the inability to reason within hierarchical objects. For example, one can extract an individual frame from a video sequence, perform some operation on it and insert it back in the video sequence again. But one cannot manipulate frames as members of the same video sequence (other than using the frames as explicit parameters).

6. Conclusions

We have presented the implementation of the visual programming interface of an image processing environment that is independent of the underlying library of image processing functions. It is a true interface in the sense that it is not an integral part of the image processing library. The assumptions made by the interface about the library are common among those libraries. The integration of the interface with an image processing library does not call for re-design or even re-compilation of the library. The dialogue independence makes it possible to develop image processing functions without having to take the user interface aspects into consideration. It should, for example, not be too cumbersome to combine the presented interface with the SPIDER package (Tamura et al., 1983) or the image processing library of the KHOROS environment (Rasure and Williams, 1991).

The prime objective of the visual programming interface for image processing is to interactively design solutions without the need to learn a textual, procedural programming language. It is our experience that even advanced users benefit from the visual programming interface due to the interactive way image processing applications grow. In the conversion of existing C-programs to visual programs several bugs, inconsistencies, and program flow inefficiencies were revealed.

Among the various alternatives in visual programming, data flow graphs are a natural way to express image processing solutions in many domains because they are strongly data oriented. It is our experience that data flow graphs provide more insight in the structure of a program than the one-dimensional, textual alternative that makes use of explicitly defined variables. Their main deficiency is their inability to use other than trivial control structures and to define data structures. On the other hand, the data flow graphs provide a higher level of interaction than the commonly used command based interface and menu & dialogue interface without causing much computational overhead. They also show great potential for parallelism.

References

Babb, R.G. (1984). Parallel processing with large-grain data flow techniques. *IEEE Computer* 17 (7), 55–61.

Bailey, D.G. and R.M. Hodgson (1988). VIPS – a digital image processing algorithm development environment. *Image and Vision Computing* 6 (3), 176–184.

Blanford, R.P. and S.L. Tanimoto (1986). The PyramidCalc system for research in pyramid machine algorithms. *Proc.* 1986 *Workshop on Visual Languages*, 138–142.

CBP, University of Amsterdam for the Centre of Image Processing and Pattern Recognition (1991). *SCIL_Image manual.*

Chang, S.-K. (1987). Visual languages, a tutorial and survey. *IEEE Software*, 29–39.

Chang, S.-K., E. Jungert, S. Levialdi, G. Tortora and T. Ichikawa (1985). An image processing language with icon assisted navigation. *IEEE Trans. Software Engineering*, 811–819.

Cox, P.T., F.R. Giles and T. Pietrzykowski (1989). Prograph: a step towards liberating programming from textual conditioning. *Proc.* 1989 *Workshop on Visual Languages*, 150–156.

Dewaele, P., D. van den Oudenhoven, J. Vandeneede, R. Bartels, P. Wambacq and A. Oosterlinck (1988). LILY: a software package for image processing. In: E.S. Glesema and L.N. Kanal, Eds., *Pattern Recognition and Artificial Intelligence.* North-Holland, Amsterdam, 17–33.

Dyer, D.S. (1990). A dataflow toolkit for visualization. *IEEE Computer Graphics Appl.*, 60–79.

Groen, F.C.A., R.J. Ekkers and R. de Vries (1988). Image processing with personal computers. *Signal Process.* 15 (3), 279–291.

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.

Hartson, H.R. and D. Hix (1989). Human-computer interface

development: concepts for systems and its management. *ACM Computing Surveys* 21 (1), 5–92.

Ibbotson, J. (1990). *IBM AIX image assistant/6000 user guide.*

Koelma, D., R. van Balen and A. Smeulders (1992). SCIL-VP: a multi-purpose visual programming environment. *Proc.* 1992 *ACM/SIGAPP Symposium on Applied Computing*, 1188–1198.

Landy, M.S., Y. Cohen, and G. Sperling (1984). HIPS: a Unix-based image processing system. *Computer Vision, Graphics, and Image Processing* 25, 331–347.

Lau-Kee, D., A. Billyard, R. Faichney, Y. Kozato, P. Otto, M. Smith and I. Wilkinson (1991). VPL: an active, declarative visual programming system. *Proc.* 1991 *Workshop on Visual Languages*, 40–46.

Lodding, K.N. (1983). Iconic interfacing. *IEEE Computer Graphics Appl.* 3, 11–20.

Myers, B.A. (1990). Taxonomies of visual programming and program visualization. *J. Visual Languages and Computing*, 97–123.

Piper, J. and D. Rurovitz (1985). Data structures for image processing in a C language and Unix environment. *Pattern Recognition Lett.* 3, 119–129.

Raeder, G. (1985). A survey of current graphical programming techniques. *IEEE Computer*, 11–25.

Rasure, J.R. and C.S. Williams (1991). An integrated data flow visual language and software development environment. *J. Visual Languages and Computing*, 217–246.

Shu, N.C. (1988). *Visual Programming.* Van Nostrand Reinhold, New York.

Smith, D.C. (1975). Pygmalion: a creative programming environment. Report no. stan-cs-75-499, Stanford.

Tamura, H., S. Sakane, F. Tomita, N. Yokoya, M. Kaneko, and K. Sakaue (1983). Design and implementation of SPIDER – a transportable image processing software package. *Computer Vision, Graphics, and Image Processing* 23, 273–294.

Tanimoto, S.L. (1990). VIVA: a visual language for image processing. *J. Visual Languages and Computing*, 127–139.

Tanimoto, S.L. and E.P. Glinert (1986). Designing iconic programming systems: representation and learnability. *Proc.* 1986 *Workshop on Visual Languages*, 54–60.

ten Kate, T.K., R. van Balen, A.W.M. Smeulders, F.C.A. Groen and G.A. den Boer (1990). SCILAIM: a multi-level interactive image processing environment. *Pattern Recognition Lett.* 11, 429–441.

TNO, Institute of applied physics (1991). *TCL-Image user's manual.*

Upson, C. et al. (1989). The application visualization system: a computational environment for scientific visualization. *IEEE Computer Graphics Appl.*, 30–42.

Williams, C.S. and J.R. Rasure (1990). A visual language for image processing. *Proc.* 1990 *Workshop on Visual Languages*, 86–91.