University of Glasgow
Department of Computing Science

Lilybank Gardens
Glasgow G12 8QQ

University of St Andrews
Department of Computational Science

North Haugh
St. Andrews KY16 8SX

Implementation Issues in Persistent Graphics

Persistent Programming
Research Report 23

Preface

This report has been accepted for publication in University Computing during Autumn 1986.

Implementation issues in Persistent Graphics

A.L. Brown and A. Dearle

Department of Computational Science

University of St Andrews

North Haugh

St Andrews KY16 9SX.

Introduction

Many of the present-day workstations feature high resolution raster displays and associated pointing devices. These systems provide a suitable medium for the construction of a high quality user interface. The graphics facilities of these systems are usually accessed at a low level. To be able to effectively use these systems a convenient high level interface must be built. These high level interfaces are usually only available as ad-hoc extensions to programming languages. This can present problems in combining graphical objects with those supported by the programming language. The problem is further aggravated when the long term storage of graphical objects is attempted. A solution to these problems would be the complete integration of graphics facilities into a persistent programming language. This paper describes the graphics facilities available in the persistent programming language PS-algol[1]. One implementation of the system on an ICL PERQ is also described.

What is persistence?

The persistence of a data object is the length of time the data object exists [2]. Previously, programming languages have treated this property of data consistently within the context of a program (either as values of local variables or as values on the heap) but have made radically different arrangements for data of longer term persistence (such as those in files or in a database). In PS-algol we view persistence as an orthogonal property of data and apply the Principle of Data Type Completeness[3], so that an object of any data type may have any persistence.

In a conventional system, a programmer has to visualise three mappings: from the real world to program representation, from the real world to database representation and from database representation to program representation. This greatly complicates the task and increases the potential for error due to inconsistent treatment of the mappings. We contend that allowing long term data storage without incurring such a multiplicity of representations and mappings is likely to have benefits in both software production and software maintenance. The immediate application of this

attribute of the language will be the construction of systems as a number of programs operating on a common body of data, particularly where data structures specific to the application are useful. One example of this is a suite of CAD programs.

The pictures provided as data types in the language are representations of annotated line drawings or bitmap images. Operators are provided which make it easy to construct such drawings or images. The communication with the user via modern terminals, which often have placement and graphics capabilities occupies a major portion of present programming. Provision of language constructs for this is intended to make the task less onerous, and the programs more portable. The problem of how to achieve similar effects on different devices has to be solved only once by the language implementor. We believe it is important to pursue further research into language constructs for user communication, as it is such a major part of programs and is poorly supported at present.

The introduction of functions as first class objects has a number of benefits[4]. The construction of functions by other functions, which are then yielded as a result provides a simple mechanism to hide information, restricting the operations on data and hence allowing the implementation of abstract data types[5]. Since we have arranged for long term persistent data, this means that hidden objects may be required to persist between runs of programs. This is a very powerful and as yet little explored feature of the language. It means for example that partial computations and the functions to progress them may be passed to another program. Data held in the persistent store may be viewed, protected and manipulated via functional abstractions. The storage of functions in the persistent store provides the equivalent of program libraries and allows for the separate compilation of program parts. The inclusion of procedures in persistent store provides automatic incremental type checked program linking.

The combination of persistence, graphics and first class functions in PS-algol result in a programming language which supports a wide range of programming activities.

Graphics

The PS-algol graphics facilities provide an integrated method of manipulating line drawings and

images. Line drawings have the data type picture and bitmaps the type image. An image is a rectangular grid of pixels of some 'colour'. Images may be manipulated by the raster operations provided by the language. These correspond to those generally available on systems with bitmap displays.

The picture drawing facilities of PS-algol are a particular implementation of the Outline system[6] which in turn took many ideas from GPL/1[7]. It allows the user to produce line drawings in an infinite two dimensional real space. The relationships between different parts of a picture are defined by mathematical transformations, therefore pictures are usually built up from a number of sub pictures. Pictures may be mapped on to an image thus providing flexibility in the way that line drawings may be manipulated.

Thus the system provides high level features for manipulating images in a finite two dimensional integer space and line drawings in an infinite two dimensional real space.

Pixels

The pixel is a base data type in PS-algol and is used to construct images. Two pixel literals are predefined in the language, these are **on** and **off**. These are said to have depth one, since they are only one pixel plane deep. Thus

let *a* = **on**

creates a pixel *a* with a depth of 1. Pixels may be concatenated in order to create pixels with depth greater than one, for example we may write,

let *b* = **on** & **off** & **off** & **on**

which creates a pixel *b* with a depth of 4. The expression on the right hand side of the above declaration is called a pixel sequence or simply a pixel.

Images

We use pixels in order to construct images. Images are initialised with a pixel expression and have the same depth as that expression. Images also have an X and Y dimension, for example,

let *c* = **image** 5 **by** 10 **of on**

creates an image *c* with 5 pixels in the X direction and 10 in the Y direction all initially on. The origin of all images is in the lower left corner, which has address 0,0. In this case the depth is 1. Full 3 dimensional images may also be created , for example,

let *d* = **image** 64 **by** 32 **of on & off & on & on**

creates an image which has depth 4.

In order to introduce the concept of images and the operations on them we will restrict ourselves for the present to images with a pixel depth of 1 which is the case on the PERQ. Everything that we say will be true for images of greater depth.

Images are first class data objects and may be assigned, passed as parameters or returned as results. e.g

let *b* := *a*

will assign the image *a* to the new one *b*. In order to map the operations usual on bitmapped screens, the assignment does not make a new copy of *a* but merely copies the pointer to it. Thus the image acts like a vector or pointer on assignment.

PS-algol supports eight raster operations these are, **ror, rand, xor, copy, nand, nor, not** and **xnor**

xor *b* **onto** *a*

performs a raster operation of *b* onto *a* using **xor**. Notice that *a* is altered 'in situ'. Both images have origin 0,0 and automatic clipping at the extremities of the destination image is performed.

It is often desirable to set up windows in images. The PS-algol limit operation allows this. For example,

let *c* = **limit** *a* **to** 1 **by** 5 **at** 3,2

sets *c* to be that part of *a* which starts at 3,2 and has size 1 by 5. *c* has an origin of 0,0 in itself and is therefore a window on *a*. Rastering sections of images onto sections of other images can be performed by expressions like,

xor limit *a* **to** 1 **by** 4 **at** 6,5 **onto**

limit *a* **to** 3 **by** 4 **at** 9,10

Automatic clipping on the edges of the limited region is performed. If the starting point of the limited region is omitted 0,0 is used and if the size of the region omitted then it is taken as the maximum possible. That is from the starting point to the edges of the host image. Limited regions of limited regions may also be defined.

The standard identifier *screen* is bound to an image representing the output screen. Performing a raster operation onto the image *screen* alters what may be seen by the user. e.g.

xor *a* **onto limit** *screen* **to** 4 **by** 5 **at** 4,7

will raster *a* onto the defined section of the screen. This will be visible to the user.

The standard identifier *cursor* is bound to an image representing the cursor. This allows the cursor to be manipulated in the same manner as any other image in the system.

In systems that support multiple planes the standard identifiers *screen* and *cursor* will have a depth greater than 1. All the operations that we have already seen on images (raster, limit and assignment) work more generally with depth. Thus the raster operations perform the raster function plane by plane in one to one correspondence between source and destination. Automatic depth clipping at the destination is performed and if the source is too small to fill all the destination's planes then these planes will remain unaltered. The limit and assignment operations also work with the depth of the image.

A window may be set up in the z-axis of an image using the depth selection operation. For example

let *b* = *a*(1|3)

yields *b* which is an alias for three contiguous planes of *a* starting at plane 1. *b* has the same X and Y dimensions as *a*.

Pixels may also be indexed. Thus

let *a* = **on & off**

gives a(0) as **on** and a(1) as **off**. This expression is an r-value only and may not be used on the left hand side of an assignment.

It should be noted that a(1) above is of type pixel which is not the same as

image 1 **by** 1 **of off**

which is of type image.

The standard function *Pixel* will yield a pixel from an image. For example, if a is defined as follows

let *a* = **image 9 by 9 of on & off & on**

Then *Pixel*(a,1,1) yields **on & off & on**.

There is a standard function that allows the programmer to gather information about the status of the mouse. The function is called *locator* and has form,

locator(→ **pntr**)

which returns a pointer to a structure of the following type,

structure *mouse*(**cint** *X.pos,Y.pos* ; ***bool** *the.buttons*)

The vector of booleans, *the.buttons*, indicate which of the buttons on the mouse are depressed. *X.pos* and *Y.pos* indicate the position of the mouse relative to the image *screen*.

Pixels

Many colour graphics systems allow the programmer to alter the colour map of the output device. That is, it is possible to alter the colour observed when a particular pixel is displayed. Usually this colour map is in the form of a lookup table maintained by the system. This table maps pixels onto integer encodings of the bits sent to the guns of the display device. The PS-algol provides two functions for manipulating the colour map of the device. The first is,

colour.map(**pixel** *p* ; **int** *i*)

This functions sets the integer produced by the colour map when pixel *p* is displayed to be *i*. The second function allows the user to interrogate the colour.map and is,

colour.of(**pixel** *p* → **int**)

This function returns the integer corresponding to the pixel p in the colour map.

Picture description in PS-algol

In PS-algol the picture descriptions are represented by the data type picture. The simplest

picture is a point. For example,

let *point* = [0.1,2.0]

represents the point with x-coordinate 0.1 and y-coordinate 2.0 in two-space. All the operations provided on pictures return a picture as their result, so arbitrarily complex pictures may be described and operated on.

Points in pictures are implicitly ordered. The binary operators on pictures operate between the last point of the first picture and the first point of the second picture. In the resulting picture the first point is the first point of the first picture and the last point is the last point of the second picture.

There are two binary operators on pictures, join '^' and combine '&'. The effect of the join operator is to produce a picture that is made up of its two operands with a line from the last point of the first operand to the first point of the second operand. Combine operates in a similar way without adding the joining line.

In addition to the binary operators pictures may also be transformed by shifting, rotating and scaling. For example:

shift *p* **by** *x.shift,y.shift*

will produce a new picture by adding *x.shift* to every x-coordinate and *y.shift* to every y-coordinate in the picture *p*.

rotate *p* **by** *no.of.degrees*

will produce a new picture by rotating the picture *p no.of.degrees* degrees clockwise about the origin.

scale *p* **by** *x.scaling,y.scaling*

will produce a new picture by multiplying the x and y-coordinates of every point in the picture *p* by *x.scaling* and *y.scaling* respectively.

Text can be included in pictures using the **text** statement. This takes a string of characters and a base line and constructs the picture of those characters along the base line.

let *p* = **text** "hello !" **from** 1,1 **to** 2,1

The characters will always be drawn from the first to last point of the base line. As a consequence text can be inverted by ending the base line on the left of its starting position.

Colour can also be specified in a picture but, unlike the other picture operations, the effect of this

will depend on the physical output device used.

Drawing a PS-algol picture

In order to view a picture it is necessary to map it onto an output device. The PS-algol standard function *draw* is provided to map pictures onto images. For example,

draw(an.image,a.pic,0,3,0,3)

will draw the section of the picture *a.pic* bounded by the box specified by the points (0.0,0.0) and (3.0,3.0) on the image *an.image*.

Note that the image may be the screen in which case the picture will be visible to the user.

Implementation

Many host machines are unable to support the PS-algol picture and image operations consequently system performance may be greatly impaired. For example, raster operations on most mainframe computers are prohibitively expensive although alternative techniques have been found [8,9]. However graphics manipulation is not usually performed unless some suitable output device is available. With this in mind the language implementor may choose to make the evaluation of the graphics operations lazy rather than strict. That is instead of performing an operation immediately a data structure is built to represent the evaluation of the expression. When the object is eventually displayed the data structure may be evaluated in the context of the output device thereby ensuring the maximum efficiency on the hardware available. Another advantage of this lazy evaluation is that the data structure (unevaluated) may be used to represent the object in the persistent store.

The implementation of PS-algol on the ICL PERQ uses microcode assist for raster operations and lazy evaluation for pictures. The data structure used for PS-algol pictures is a directed acyclic graph (DAG). It is a graph since there may be several paths to the same sub-structure, directed since the ordering of the points uniquely defines a traversal and acyclic since recursive referencing is not possible for pictures. A fuller description of how this may be done is given below. The implementation issues for persistence are described elsewhere[10].

Implementation of the PS-algol pictures

The PS-algol picture facilities are purely applicative in nature. For this reason the implementation technique chosen is also applicative. When a picture expression is parsed the PS-algol compiler expands the picture expression to one that creates a set of predefined PS-algol structures. Hence every PS-algol picture is represented by a tree of PS-algol structures. The structures and the expanded expressions used to represent picture expressions are as follows:

a) **structure** *point.node*(**creal** *point.x,point.y*)

[*x,y*] compiles to *point.node*(*x,y*)

b) **structure** *operation.node*(**cpntr** *left.pic,right.pic* ; **cint** *operation*)

p1 & *p2* compiles to *operation.node*(*p1,p2,1*)

p1 ^ *p2* compiles to *operation.node*(*p1,p2,2*)

c) **structure** *transform.node*(**cint** *transform* ; **cpntr** *transform.pic* ;
creal *transform.x,transform.y*)

scale *p* **by** *x,y* compiles to *transform.node*(*1,p,x,y*)

shift *p* **by** *x,y* compiles to *transform.node*(*2,p,x,y*)

rotate *p* **by** *degrees* compiles to *transform.node*(*3,p,degrees,0*)

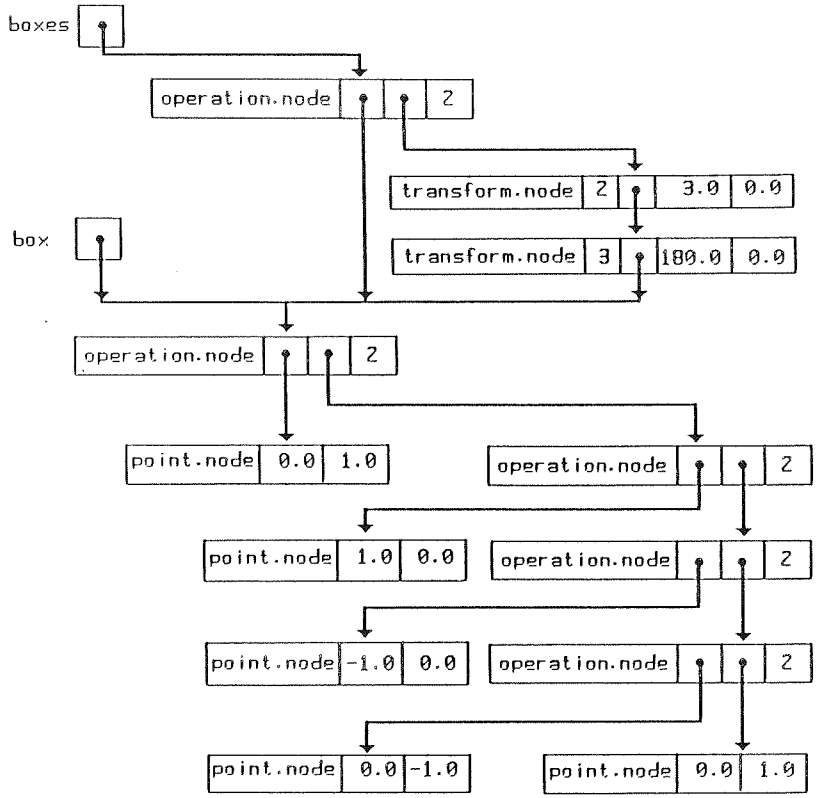
d) **structure** *text.node*(**cstring** *message* ; **creal** *xxl,yyl,xxr,yyr*)

text *a.string* **from** *x1,y1* **to** *x2,y2* compiles to *text.node*(*a.string,x1,y1,x2,y2*)

e) **structure** *colour.node*(**cpntr** *colour.pic* ; **cpixel** *shade*)

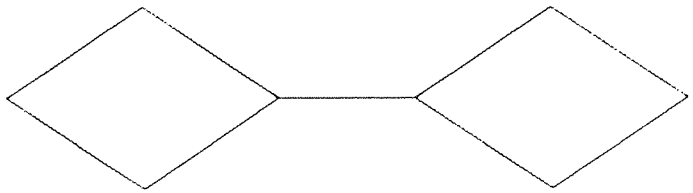
colour *p* **in** *a.pixel* compiles to *colour.node*(*p,a.pixel*)

let *box* = [0,1]^ [1,0]^ [-1,0]^ [0,-1]^ [0,1]
let *boxes* = *box* ^ **shift rotate** *box* **by** 180 **by** 3,0
is represented by:



draw(*screen*,*boxes*,-2,-4,-2,2)

will produce:



Internal Outline representation
Figure 1.

Implementation of PS-algol picture drawing

To draw a PS-algol picture the tree representation must be traversed and the picture's points calculated A convenient way of programming this tree traversal is to use a recursive procedure. PS-algol is a suitable tool for writing such recursive procedures. Hence it would be convenient to be able to use it to write the *draw* standard function. However to allow this a mechanism must be introduced to coerce the type of a picture expression from **pic** to **pntr**.

This is done using a special function called **pic.pntr** that takes a picture as a parameter and returns a pointer to the picture's description. All the PS-algol structures used to represent pictures only contain constant fields. Therefore the type coercion does not allow a PS-algol program to modify a picture's description.

The *draw* standard function is composed of three sections. An initial transformation calculation, a tree traversal procedure and a line drawing package.

The initial transformation calculation

The first step in drawing a picture is to calculate the transformation necessary to map the points in the picture onto the output device. This calculation is done by mapping the picture window supplied to *draw* onto the visible area of the output device. A recursive procedure can then be called to apply this transformation to the picture description.

The tree traversal procedure

The purpose of the tree traversal procedure is to calculate the points and lines that compose the picture. To do this it must be supplied with the transformation necessary to map the points it finds to the output device. It must also know what colour the picture should be drawn in. Therefore the tree traversal procedure is parameterised with a transformation, a colour value and a pointer to a picture description. The initial call to the procedure is supplied with a default colour value.

The action taken for each possible type of node in a picture description will now be described. There are 5 different types of node, a transformation node, an operation node, a colour node, a point

node and a text node.

a) A transformation node.

A transformation node in a picture description represents a sub picture and a transformation that must be applied to every point in the sub picture, therefore when a transformation node is encountered by the traversal procedure there will be two transformations that must be applied to the node's sub picture. The transformation represented by the node and the transformation supplied to the traversal procedure. However these two transformations occur after any found in the node's sub picture. Also the new transformation always occurs before the one passed as a parameter. Therefore they can be composed to form a new transformation to be applied to the sub picture.

All transformations are represented by matrices [11], therefore the composition of transformations can be performed using matrix multiplication. Once the composed transformation matrix has been calculated it can be used as a parameter when the traversal procedure is applied to the node's sub picture.

b) An operation node.

An operation node represents two sub pictures together with an operator for combining them. The two sub pictures are the left and right operands of a join or combine operation. When an operation node is encountered the left sub picture is traversed. A global flag is then set to indicate whether a join or combine operation is being performed. Finally the right sub picture is traversed.

This order of traversal ensures that the first point traversed in a sub picture is the first point of the sub picture. In the same way the last point traversed is the last point of the sub picture. Also the global flag indicating the operation being performed will not be reset until a point has been traversed. Therefore when a point node is found it can be treated as the first point in a sub picture. Then if the operation being performed is a join, a line can be drawn from it to the previous point to join their respective sub pictures.

c) A colour node.

A colour node represents a request to draw all of a sub picture in a particular colour.

This request is enforced on all of the sub picture regardless of any colour nodes it may contain. Therefore when a colour node is traversed a check is made on the colour parameter. If it is not the default colour value then the colour node's sub picture is traversed using the colour parameter. However if the colour parameter is the default colour, the colour node's colour value will be used to traverse its sub picture.

d) A point node.

At the leaves of a picture description the tree traversal procedure always encounters point nodes. When one is found the transformation parameter will be sufficient to map the point onto the screen. In addition the colour of the point will also be known. A request can then be made to draw the point in the appropriate colour. The global flag indicating the type of combine operation being performed is then checked. If the operation is a join operation then a line is drawn using the colour parameter from the point to the previous point.

e) A text node.

A text node represents a piece of text and a line along which it should be drawn. When a text node is found the tree traversal procedure constructs a picture description for the characters making up the text. This new picture description is then traversed in place of the text node.

The line drawing package

The *draw* standard function must be supplied with procedures to plot points and draw lines. It is possible that some of the points and line segments to be drawn will not be visible on the output device. i.e. they were outside the original picture window and so were mapped to outside the visible area of the output device. It is assumed that the drawing procedures will perform the necessary clipping operations to remove these points and line segments.

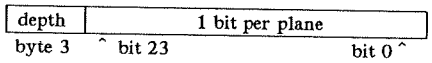
The device dependent parts of the *draw* standard function are limited to the drawing procedures and the assumed dimensions of the output device. Therefore any type of output device can be supported simply by supplying *draw* with the appropriate drawing procedures and the dimensions of

the output device. The PS-algol line drawing graphics currently support PS-algol rasters, Tektronix T4010 and T4107 terminals and PERQ/PNX windows[12].

Implementation of the PS-algol raster graphics

Pixels

Pixels are composed of a value on or off for each of the pixel's planes. Pixels are implemented as binary values encoded as bits in an integer. The least significant bit, represents plane 0, the next represents plane 1 and so on. In order that the depth of a pixel may be found the most significant byte of the integer is used to record the pixel's depth. The resulting format for a pixel is as follows, note this diagram assumes an integer made up of 4, 8 bit bytes.



The format of a pixel.
Figure 2.

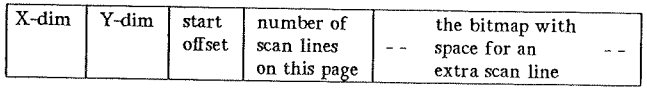
The abstract machine is augmented with two new instructions and one standard function to manipulate pixels. The first of the instructions, *subpixel.op*, takes an integer index and applies it to a pixel to yield a new pixel. As part of this operation a check is made that the index is legal. The second new instruction, *formpixel.op*, takes a set of pixels and composes them to form a new pixel. No special instruction was necessary for the literal pixel values on and off since the corresponding integers could be constructed by the compiler. The standard function *depth* is provided to allow a program to discover the depth of a pixel.

Images

The semantics of the data type **image** include the ability to alias sections of an image in one, two or three dimensions. To support this an image object is made up of 3 separate sections. A set of two dimensional bitmaps, a vector pointing to the bitmaps and an area descriptor.

Two dimensional bitmaps

Each plane of a PS-algol raster is represented by a PS-algol vector of integers. The vector consists of the bitmap for the plane plus some house keeping information such as its size. The format of one of these vectors is as follows:



A two dimensional bitmap.
Figure 3.

On some computers the raster operations provided impose alignment constraints on bitmaps. This in itself does not present a problem since space can always be allocated on the appropriate boundaries, however the PS-algol system incorporates a compacting garbage collector based on an algorithm by Lockwood Morris[13]. Hence PS-algol bitmaps may require realignment after every garbage collection. For this reason extra space in the heap object is allocated and the start of the bitmap within the object recorded. A further constraint imposed by the raster operations on some computers is that they do not operate over page boundaries. One way of overcoming this problem is to split the bitmap between scan lines into two rectangular sections, one on either side of the page boundary. To allow for realignment in this case the extra space needs to be a full scan line. In this way the bitmap can always be moved so that no scan line ever crosses a page boundary.

A vector of bitmaps

Each of the bitmap vectors implements a single plane of a PS-algol raster. To represent the ordering of planes in a raster, a PS-algol vector is used to point to each of the raster's planes. The first plane pointed to by the vector is plane 0, the second plane 1, and so on.

A two dimensional area description

The purpose of the final layer of a PS-algol raster is to specify the two dimensional area of interest. This is done by recording the X and Y dimensions of the area together with the X and Y offsets to the start of the area. The description of an area to which a raster applies is specified by a

pointer to a vector of bitmaps. However in some cases it may not be possible to represent a raster as a vector of bitmaps. For example on a UNIX system the screen may be represented by a special file type. To allow for this a file descriptor is held in addition to the vector of bitmaps. The format of an area description is as follows:

PS-algol heap header	pointer to vector of bitmaps	PS-algol file descriptor	X offset	Y offset	X dimension

An area descriptor for a PS-algol raster.
Figure 4.

Raster operations.

The operations permitted on rasters can be divided into three types, updates, area selections and dimension inspection. Rasters may only be updated by the destructive combination of two rasters. Such combinations overwrite a destination raster with the combination of a source raster and the destination raster under a specified combination rule. The following combination rules are supported. Each operation is described by the effect on two bits, D the destination and S the source. The bit values representing **on** and **off** correspond to **true** and **false** respectively. The symbol '~' is the PS-algol not operator.

a) copy - D := S

b) rand - D := S **and** D

c) ror - D := S **or** D

d) xor - D := (S **and** ~ D) **or** (~ S **and** D)

e) not - D := ~ S

f) nand - D := ~ (S **and** D)

g) nor - D := ~ (S **or** D)

h) xnor - D := (S **or** ~ D) **and** (~ S **or** D)

If the rasters have more than one plane the combination operation is repeated for each pair of planes. i.e. first between plane 0 of the source and plane 0 of the destination, then between plane 1 of the source and plane 1 of the destination, and so on. If the number of planes in the source and

destination rasters are different the combination operation stops when all the planes in one of the rasters have been used.

The selection operations are performed by copying the area descriptions and if necessary the vector of bitmaps.

When a PS-algol raster has **limit** applied to it a copy is made of the area descriptor to hold the new offset and dimension values. While the new values are being calculated a check is performed to make sure they describe a two dimensional area within the raster. The new area descriptor retains the pointer to the vector of bitmaps and the file descriptor because it is defined over the same raster.

When plane selection is performed on a raster, a copy of the vector of bitmaps is made that only contains entries for the selected planes. A copy of the area descriptor is then made to point to the copied vector. An attempt to select all the planes of a raster is treated as a null operation because the new vector of bitmaps and area descriptor would be identical to the originals.

The operations that inspect the dimensions of an image operate by indexing the objects that form a raster. They are all implemented via standard functions.

Implementation of the screen

The standard identifier *screen* is implemented by creating a dummy raster object. On the ICL PERQ this object is given a file descriptor for the special file representing the physical screen. The vector of bitmaps consists of a one entry vector with a null pointer. Whenever a raster operation is performed a check is made to see if the raster object contains a valid file descriptor. If it does the raster operation is performed using the file descriptor in place of the vector of bitmaps.

Implementation of the cursor

The standard identifier *cursor* is also implemented by creating a dummy raster object. However on the ICL PERQ the raster object has a valid vector of bitmaps but no file descriptor. The vector consists of 1 entry to the single bitmap composing the cursor. This bitmap always holds a copy of the cursor image displayed on the screen. By recording the pointer to this bitmap any updates to the

cursor image can be detected and propagated to the real cursor.

Implementation of the mouse

The locator standard function provides an interface to the mouse or tablet connected to a workstation. On the ICL PERQ a tablet is provided with either a 3 or 4 button puck. To implement this function a mouse structure is created on each call to locator. The cursor position is obtained from the underlying operating system and entered into the structure. The number of buttons provided on a puck or mouse can vary considerably. Therefore locator creates a vector of truth values with one entry for each available button.

On most workstations it is possible to specify the mode in which the tablet or mouse operates. For example on the ICL PERQ a read of the tablet can be delayed until some event occurs. Possible events are time outs, button presses, puck movement and the presence of keyboard input. The mode of operation selected for PS-algol was to delay the return of locator until the puck had moved at least 3 pixels or a button had been pressed or some keyboard input had become available. A more flexible choice of mode is currently under investigation.

Conclusions

The PS-algol system provides a persistent store facility with a small number of powerful graphics constructs. Thus the programmer is presented with an easily understood, high level interface to the facilities provided by today's workstations. With such facilities interactive applications are easier to write and maintain thus providing lower life cycle costs throughout the lifetime of the program.

The graphics constructs described in this paper are independent of the persistence concept and so may be integrated with any high level language. We have described one implementation of these facilities and proposed other methods that may be used to support them.

Acknowledgements

This work is supported by SERC grants GRC 15907,GRC 4326.6 and a grant from ICL. We acknowledge the assistence of the other members of the PISA project at St Andrews University especially Ron Morrison and Peter Bailey who made many helpful comments during the preparation of this paper. We also acknowledge the close collaboration with Malcolm Atkinson's group at Glasgow University in this work and other aspects of language design. Since this paper was written PS-algol has been implementated on the Whitechapel MG1 by John Livingstone. We would like to thank him for his cooperation and helpful comments during this work.

References

1. PS-algol reference manual. Universities of Glasgow and St Andrews PPR-12 (1985).
2. Atkinson, M.P., Bailey, P.J., Cockshott, W.P., Chisholm, K.J. & Morrison, R. An approach to persistent programming. Computer Journal 26, 4 (1983), 360-365.
3. Strachey, C. Fundamental concepts in programming languages. Oxford University press (1967).
4. Atkinson, M.P. & Morrison, R. First Class Functions are Enough. Proc. 4th International Conference on Software Technology and Theoretical Computer Science. Bangalore, India(1984). In Lecture Notes in Computer Science, 181 (1984), 223-240. Springer-Verlag.
5. Liskov, B. & Zilles, S.N. Programming with abstract data types. ACM Sigplan Notices 9, 4 (1974), 50-59.
6. Morrison, R. Low cost computer graphics for micro computers. Software, Practice & Experience 12, 8 (1982), 767-776.
7. Smith, D.N. GPL/1 - a PL/1 extension for computer graphics. AFIPS (1971) 511-528.
8. Pike et al. Hardware/Software trade-offs for bitmap graphics on the Blit. Software, Practice & Experience 15, 2 (1985), 131-151.
9. Cole, A.J. Compaction techniques for raster scan graphics using space filling curves. Computer Journal (accepted for publication October 1985).
10. Cockshott, W.P., Atkinson, M.P., Bailey, P.J., Chisholm, K.J. & Morrison, R. The persistent object

management system. Software, Practice & Experience 14 (1984).

11. Newman, W. & Sproull, R. Principles of Interactive computer graphics. McGraw Hill (1981)

12. I.C.L., Introduction to PERQ. International Computers Ltd. RP10103. (1983).

13. Lockwood Morris, F. A Time and Space Efficient Garbage Compaction Algorithm. CACM 21,8 (1978), 662-665.

Bibliography

Copies of documents in this list may be obtained by writing to:

The Secretary,
Persistent Programming Research Group,
Department of Computing Science,
University of Glasgow,
Glasgow G12 8QQ
Scotland.

Books

Davie, A.J.T. & Morrison, R.
"Recursive Descent Compiling", Ellis-Horwood Press (1981).

Atkinson, M.P. (ed.)
"Databases", Pergammon Infotech State of the Art Report, Series 9, No.8, January 1982. (535 pages).

Cole, A.J. & Morrison, R.
"An introduction to programming with S-algol", Cambridge University Press, Cambridge, England, 1982.

Stocker, P.M., Atkinson, M.P. & Grey, P.M.D. (eds.)
"Databases - Role and Structure", Cambridge University Press, Cambridge, England, 1984.

Published Papers

Morrison, R.
"A method of implementing procedure entry and exit in block structured high level languages". Software, Practice and Experience 7, 5 (July 1977), 535-537.

Morrison, R. & Podolski, Z.
"The Graffiti graphics system", Proc. of the DECUS conference, Bath (April 1978), 5-10.

Atkinson, M.P.
"A note on the application of differential files to computer aided design", ACM SIGDA newsletter Summer 1978.

Atkinson, M.P.
"Programming Languages and Databases", Proceedings of the 4th International Conference on Very Large Data Bases, Berlin, (Ed. S.P. Yao), IEEE, Sept. 78, 408-419. (A revised version of this is available from the University of Edinburgh Department of Computer Science (EUCS) as CSR-26-78).

Atkinson, M.P.
"Progress in documentation: Database management systems in library automation and information retrieval", Journal of Documentation Vol.35, No.1, March 1979, 49-91. Available as EUCS departmental report CSR-43-79.

Gunn, H.I.E. & Morrison, R.
"On the implementation of constants", Information Processing Letters 9, 1 (July 1979), 1-4.

Atkinson, M.P.
"Data management for interactive graphics", Proceedings of the Infotech State of the Art Conference, October 1979. Available as EUCS departmental report CSR-51-80.

Atkinson, M.P. (ed.)
"Data design", Infotech State of the Art Report, Series 7, No.4, May 1980.

Morrison, R.
"Low cost computer graphics for micro computers", Software Practice and Experience, 12, 1981, 767-776.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"PS-algol: An Algol with a Persistent Heap", ACM SIGPLAN Notices Vol.17, No. 7, (July 1981) 24-31. Also as EUCS Departmental Report CSR-94-81.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Nepal - the New Edinburgh Persistent Algorithmic Language", in Database, Pergammon Infotech State of the Art Report, Series 9, No.8, 299-318 (January 1982) - also as EUCS Departmental Report CSR-90-81.

Morrison, R.
"S-algol: a simple algol", Computer Bulletin II/31 (March 1982).

Morrison, R.
"The string as a simple data type", Sigplan Notices, Vol.17,3, 46-52, 1982.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"Progress with Persistent Programming", presented at CREST course UEA, September 1982, revised in "Databases - Role and Structure", see PPRR-8-84.

Morrison, R.
"Towards simpler programming languages: S-algol", IUCC Bulletin 4, 3 (October 1982), 130-133.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Problems with persistent programming languages", presented at the Workshop on programming languages and database systems, University of Pennsylvania. October 1982. Circulated (revised) in the Workshop proceedings 1983, see PPRR-2-83.

Atkinson, M.P.
"Data management", in Encyclopedia of Computer Science and Engineering 2nd Edition, Ralston & Meek (editors) January 1983. van Nostrand Reinhold.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"Algorithms for a Persistent Heap", Software Practice and Experience, Vol.13, No.3, 259-272 (March 1983). Also as EUCS Departmental Report CSR-109-82.

Atkinson, M.P., Chisholm, K.J. & Cockshott, W.P.
"CMS - A chunk management system", Software Practice and Experience, Vol.13, No.3 (March 1983), 273-285. Also as EUCS Departmental Report CSR-110-82.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"Current progress with persistent programming", presented at the DEC workshop on Programming Languages and Databases, Boston, April 1983.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"An approach to persistent programming", The Computer Journal, 1983, Vol.26, No.4, 360-365 - see PPRR-2-83.

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, W.P. & Morrison, R.
"PS-algol a language for persistent programming", 10th Australian Computer Conference, Melbourne, Sept. 1983, 70-79 - see PPRR-2-83.

Morrison, R., Weatherill, M., Podolski, Z. & Bailey, P.J.
"High level language support for 3-dimension graphics", Eurographics Conference Zagreb, North Holand, 7-17, Sept. 1983. (ed. P.J.W. ten Hagen).

Cockshott, W.P., Atkinson, M.P., Chisholm, K.J., Bailey, P.J. & Morrison, R.
"POMS : a persistent object management system", Software Practice and Exerience, Vol.14, No.1, 49-71, January 1984.

Kulkarni, K.G. & Atkinson, M.P.
"Experimenting with the Functional Data Model", in Databases - Role and Structure, Cambridge University Press, Cambridge, England, 1984.

Atkinson, M.P. & Morrison, R.
"Persistent First Class Procedures are Enough", Foundations of Software Technology and Theoretical Computer Science (ed. M. Joseph & R. Shyamasundar) Lecture Notes in Computer Science 181, Springer Verlag, Berlin (1984).

Atkinson, M.P., Bocca, J.B., Elsey, T.J., Fiddian, N.J., Flower, M., Gray, P.M.D. Gray, W.A., Hepp, P.E., Johnson, R.G., Milne, W., Norrie, M.C., Omololu, A.O., Oxborrow, E.A., Shave, M.J.R., Smith, A.M., Stocker, P.M. & Walker, J.
"The Proteus distributed database system", proceedings of the third British National Conference on Databases, (ed. J. Longstaff), BCS Workshop Series, Cambridge University Press, Cambridge, England, (July 1984).

Atkinson, M.P. & Morrison, R.
"Procedures as persistent data objects", ACM TOPLAS 7, 4, 539-559, (Oct. 1985) - see PPRR-9-84.

Morrison, R.,Bailey, P.J., Dearle, A., Brown, P. & Atkinson, M.P.
"The persistent store as an enabling technology for integrated support environments", 8th International Conference on Software Engineering, Imperial College, London (August 1985), 166-172 - see PPRR-15-85.

Atkinson, M.P. & Morrison, R.
"Types, bindings and parameters in a persistent environment", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 1-24 - see PPRR-16-85.

Davie, A.J.T.
"Conditional declarations and pattern matching", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 278-283 - see PPRR-16-85.

Krablin, G.L.
"Building flexible multilevel transactions in a distributed persistent environment, proceedings of Data Types and Persistence Workshop, Appin, August 1985, 86-117 - see PPRR-16-85.

Buneman, O.P.
"Data types for data base programming", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 291-303 - see PPRR-16-85.

Cockshott, W.P.
"Addressing mechanisms and persistent programming", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 363-383 - see PPRR-16-85.

Norrie, M.C.
"PS-algol: A user perspective", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 399-410 - see PPRR-16-85.

Owoso, G.O.
"On the need for a Flexible Type System in Persistent Programming Languages", proceedings of Data Types and Persistence Workshop, Appin, August 1985, 423-438 - see PPRR-16-85.

Morrison, R., Brown, A.L., Bailey, P.J., Davie, A.J.T. & Dearle, A.
"A persistent graphics facility for the ICL PERQ", Software Practice and Experience, Vol.14, No.3, (1986) - see PPRR-10-84.

Atkinson, M.P. and Morrison R.
"Integrated Persistent Programming Systems", proceedings of the 19th Annual Hawaii International Conference on System Sciences, January 7-10, 1986 (ed. B. D. Shriver), vol IIA, Software, 842-854, Western Periodicals Co., 1300 Rayman St., North Hollywood, Calif. 91605, USA - see PPRR-19-85.

Atkinson, M.P., Morrison, R. and Pratten, G.D.
"A Persistent Information Space Architecture", proceedings of the 9th Australian Computing Science Conference, January, 1986 - see PPRR-21-85.

Kulkarni, K.G. & Atkinson, M.P.
"EFDM : Extended Functional Data Model", The Computer Journal, Vol.29, No.1, (1986) 38-45.

Internal Reports

Morrison, R.
"S-Algol language reference manual", University of St Andrews CS-79-1, 1979.

Bailey, P.J., Maritz, P. & Morrison, R.
"The S-algol abstract machine", University of St Andrews CS-80-2, 1980.

Atkinson, M.P., Hepp, P.E., Ivanov, H., McDuff, A., Proctor, R. & Wilson, A.G.
"EDQUSE reference manual", Department of Computer Science, University of Edinburgh, September 1981.

Hepp, P.E. and Norrie, M.C.
"RAQUEL: User Manual", Department of Computer Science Report CSR-188-85, University of Edinburgh.

Norrie, M.C.
"The Edinburgh Node of the Proteus Distributed Database System", Department of Computer Science Report CSR-191-85, University of Edinburgh.

In Preparation

Kulkarni, K.G. & Atkinson, M.P.
"EFDM : A DBMS based on the functional data model", to be submitted.

Atkinson, M.P. & Buneman, O.P.
"Database programming languages design", submitted to ACM Computing Surveys - see PPRR-17-85.

Morrison, R., Dearle, A., Bailey, P., Brown, A. & Atkinson, M.P.
"An integrated graphics programming system", presented at EUROGRAPHICS UK, Glasgow University, March 1986, to be published in Computer Graphics Forum - see PPRR-14-86.

Buneman, O.P. & Atkinson, M.P.
"Inheritance and Persistence in Database Programming Languages" - to be presented at ACM SIGMOD Conference 1986, Washington, USA, May 1986 - see PPRR-22-86.

Brown, A.L. and Dearle, A.
"Implementation Issues in Persistent Graphics" - to be published in University Computing, Autumn 1986 - see PPRR-23-86.

Theses

The following Ph.D. theses have been produced by member of the group and are available from

The Secretary,
Persistent Programming Group,
University of Glasgow,
Department of Computing Science,
Glasgow G12 8QQ,
Scotland.

W.P. Cockshott
Orthogonal Persistent, University of Edinburgh, February 1983.

K.G. Kulkarni
Evaluation of Functional Data Models for Database Design and Use, University of Edinburgh, 1983.

P.E. Hepp
A DBS Architecture Supporting Coexisting Query Languages and Data Models, University of Edinburgh, 1983.

G.D.M. Ross
Virtual Files: A Framework for Experimental Design, University of Edinburgh, 1983.

G.O. Owoso
Data Description and Manipulation in Persistent Programming Languages, University of Edinburgh, 1984.

Persistent Programming Research Reports

This series was started in May 1983. The following list gives those produced and those planned plus their status at 1st June 1986.

Copies of documents in this list may be obtained by writing to:

The Secretary,
The Persistent Programming Research Group,
Department of Computing Science,
University of Glasgow,
Glasgow G12 8QQ.

PPRR-1-83 The Persistent Object Management System - Atkinson,M.P., Chisholm, K.J. and Cockshott, W.P. £1.00

PPRR-2-83 PS-algol Papers: a collection of related papers on PS-algol - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R. £2.00

PPRR-4-83 The PS-algol reference manual - Atkinson, M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R. £2.00

PPRR-5-83 Experimenting with the Functional Data Model - Atkinson, M.P. and Kulkarni, K.G. £1.00

PPRR-6-83 A DBS Architecture supporting coexisting user interfaces: Description and Examples - Hepp, P.E. £1.00

PPRR-7-83 EFDM - User Manual - K.G.Kulkarni £1.00

PPRR-8-84 Progress with Persistent Programming - Atkinson,M.P., Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R. £2.00

PPRR-9-84 Procedures as Persistent Data Objects - Atkinson, M.P.,Bailey, P., Cockshott, W.P., Chisholm, K.J. and Morrison, R. £1.00

PPRR-10-84 A Persistent Graphics Facility for the ICL PERQ - Morrison, R., Brown, A.L., Bailey, P.J., Davie, A.J.T. and Dearle, A. £1.00

PPRR-11-85 PS-algol Abstract Machine Manual £1.00

PPRR-12-85 PS-algol Reference Manual - second edition £2.00

PPRR-13-85 CPOMS - A Revised Version of The Persistent Object Management System in C - Brown, A.L. and Cockshott, W.P. £2.00

PPRR-14-86 An Integrated Graphics Programming Environment - second edition - Morrison, R., Brown, A.L., Dearle, A. and Atkinson, M.P. £1.00

PPRR-15-85 The Persistent Store as an Enabling Technology for Integrated Project Support Environment - Morrison, R., Dearle, A, Bailey, P.J., Brown, A.L. and Atkinson, M.P. £1.00

PPRR-16-85 Proceedings of the Persistence and Data Types Workshop, Appin, August 1985 - ed. Atkinson, M.P., Buneman, O.P. and Morrison, R. £15.00

PPRR-17-85 Database Programming Language Design - Atkinson, M.P. and Buneman, O.P. £3.00

PPRR-18-85 The Persistent Store Machine - Cockshott, W.P. £2.00

PPRR-19-85 Integrated Persistent Programming Systems - Atkinson, M.P. and Morrison, R. £1.00

PPRR-20-85 Building a Microcomputer with Associative Virtual Memory - Cockshott, W.P. £1.00

PPRR-21-85 A Persistent Information Space Architecture - Atkinson, M.P., Morrison, R. and Pratten, G.D. £1.00

PPRR-22-86 Inheritance and Persistence in Database Programming Languages - Buneman, O.P. and Atkinson, M.P. £1.00

PPRR-23-86 Implementation Issues in Persistent Graphics - Brown, A.L. and Dearle, A. £1.00

PPRR-28-86 A Domain Theoretic Approach to Higher-Order Relations - Buneman, O.P. £1.00

PPRR-30-86 Data Types for Data Base Programming - Buneman, O.P. £1.00

Persistent Programming Research Reports
In Preparation

Some Applications Programmed in a Persistent Language -
Cooper, R.L. (ed).

PS-algol Applications Programming -
Cooper, R.L., Dearle, A., MacFarlane, D.K. and Philbrow, P.

A Compilation Technique for a Block Retention Language -
Cockshott, W.P. and Davie, A.J.T.

Thoughts on Concurrency -
Wai, F.

An Exception Handling Model in a Persistent Programming Language -
Philbrow, P.

Concurrency in Persistent Programming Languages -
Krablin, G.K.

Extracting Garbage and Statistics from a Persistent Store -
Campin, J.