VICTORIA UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES

Image Processing in Smalltalk

Fei Liu

(20 COMP 2)

June, 1992

TECHNICAL REPORT

VICTORIA UNIVERSITY OF TECHNOLOGY
BALLARAT ROAD (P O BOX 64) FOOTSCRAY
VICTORIA, AUSTRALIA 3011
TELEPHONE (03) 688-4249/4492
FACSIMILE (03) 687-7632

Campuses at
Footscray, Melton,
St Albans, Werribee

VICTORIA UNIVERSITY

Footscray Campus

IMAGE PROCESSING IN SMALLTALK

By FEI LIU

Department of Computer and Mathematical Sciences

Technical Report
June 1992

CONTENTS

IMAGE PROCESSING IN SMALLTALK

By **FEI LIU**

[Abstract] *This article describes the implementation of a binary-image algebra in* ***Smalltalk V286***. *Further, to facilitate use of the algebra, an image-processing environment has been created, together with a small-image data-base.*

Part 0. Introduction

This report describes the author's first step towards developing within Smalltalk /V286 an environment suitable for image-processing. The requirements at the outset were:-

1. The environment would include an expandable suite of readily invokable image-processing operations.
2. There would be a ready means of storing and accessing the images.
3. The environment would be graphic. We wished to work at the level of actual images rather than numerical arrays representing their bitmaps.

In mainstream image-processing the basic operation is *convolution* of an image with a *point-spread* function (or its equivalent in the frequency domain). However, Smalltalk's basic operation for working with binary bitmaps is the bit-block transfer (BitBlt) which appeared to the author as lending itself more readily to implementation of the operations of *mathematical morphology.*

In mathematical morphology we use a *structuring-element* to study an image by applying various binary operations to the image and the structuring-element. The first systematic treatment of mathematical morphology was the two-volume work by J. Serra [Serra 82]. However, a visit to Serra's work could well be prefaced by the reading of two relevant chapters of a recent book by R.M. Haralick and L. Shapiro [Haralick 92].

In the course of developing his Digital Optical Cellular Image Processor (DOCIP), K-S Huang [Huang 89] devised a Binary-Image Algebra (BIA) which allowed him to express morphological operations in terms of three fundamental operations.

The main part of the work reported here implements Huang's BIA in Smalltalk/V286. In order to advance the work more rapidly the author developed an small image data-base and a windowing system for accessing and processing the images retrieved from the data-base.

Part 1. Image processing in smalltalk

1. Smalltalk V286

SmallTalk V286 is a pure object-oriented programming language. Its advantages over other languages have been succinctly highlighted by Wilf R. LaLonde and John R. Pugh in their book *Inside Smalltalk*:-

> Programming in Smalltalk V286 is different from programming in traditional languages such as Pascal, Fortran , or Basic. A major difference is that the language is *object-oriented* rather than *procedure-oriented* and is based on concepts such as *objects* and *messages* rather than *records (structures)* and *functions*. Although these concepts are new to many programmers, they are often overshadowed by a more visible difference. Smalltalk V286 is much more than a programming language -- it is a complete program development environment. It integrates in a consistent manner such features as an editor, a debugger, print utilities, a window system , and a source code manager. Such features are traditionally associated with an operating system rather than a programming language. Smalltalk eliminates the sharp boundary between application and operating system by modelling everything as an *object*.

2. The fundamental operations of binary-image algebra (BIA)

Huang's binary-image algebra (BIA) expresses general operations on binary-images in terms of three fundamental operations:-

a). Complement of an image .
b). Union of two images .
c). Dilation of an original image with a reference image.

Our first step below will be to implement Huang's fundamental operations as methods in a class Image which we shall create as a subclass of the class Form. We shall then translate Huang's formulas for more general operations into Smalltalk methods. We shall retain Huang's numbering of formulas.

Complement

Huang's expression for the complement of an image X is given in his Formula 3.2 :-

$$\overline{X} = \{(x,y) \mid (x,y) \in W \wedge (x,y) \notin X\}$$

We may implement the operation in Smaltalk by means of the following instance method:-

```
  complement
"Answer the complement of an image.     F. L.   30  April   1992 "

  ^self reverse
```

We may test the method in the following way:-

```
      X:= ImageDataBase idbFromDisk:'image.dbs'.
      Z:= X at:'complement'.
      Z displayAt: 0@50.
      Z complement.
      Z displayAt: 50@50
```

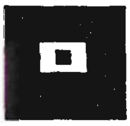


Fig. 1.1 Original

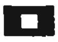


Fig. 1.2 Result

Union

Huang's expression for the union of an image X by an image R is given in his Formula 3.3 :-

$$X \cup R = \{(x,y) \mid (x,y) \in X \vee (x,y) \in R\}$$

We may implement the operation in Smaltalk by means of the following instance method:-

```
   union: imageR
 "Answer an image containing the image of the union of imageR and the receiver imageX.
F. L.   30  April   1992"

     self copy: (0@0 extent:(imageR extent)) from:imageR to:0@0 rule:7 .
```

We may test the method in the following way:-

```
      X:= ImageDataBase idbFromDisk:'image.dbs'.
```

```
    Y:= X at:'unionX'.
    Y displayAt: 0@20.
    Z:= X at:'unionR'.
    Z displayAt: 50@20.
    Y union: Z .
    Y displayAt: 100@20
```

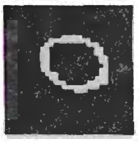


Fig. 2.1 Original

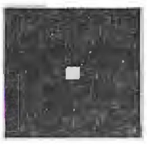


Fig. 2.2 Reference

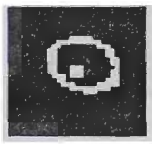


Fig. 2.3 Result

Dilation

Huang's expression for the union of an image X by an image R is given in his Formula 3.4 :-

$$X \oplus R = \begin{cases} \{(x_1 + x_2, y_1 + y_2) \in W \mid (x_1, y_1) \in X, (x_2, y_2) \in R\} & (X \neq \phi) \wedge (R \neq \phi) \\ \phi & otherwise \end{cases}$$

The methods *getPointsFrom* and *centre* are pre-defined in the class Image (see *the Appendix*). We may implement the operation in Smaltalk by means of the following instance method.:-

dilationBy:imageR

```
 "Dilation an image (imageX) by a reference image (imageR)    F. L.   30  April 1992"
   | a b c|
   b:= (Image width: (self width) height: (self height)) black.
   c:= (Image width: (self width) height: (self height)) black.
   a:= OrderedCollection new.
   a:= imageR getPointsFrom:imageR.
   1 to: a size do:[:i|
     b copy: (0@0 extent:(self extent)) from:self to:((a at:i)- (imageR centre)) rule:3.
     c union:b
       ] .
    self copy: (0@0 extent:(self extent)) from:c to:0@0 rule:3
```

We may test the method in the following way:-

```
    X:= ImageDataBase idbFromDisk:'image.dbs'.
    Y:=X at:'dilationX'.
    Y displayAt: 0@20.
    Z:=X at:'dilationR'.
    Z displayAt: 50@20.
    Y dilationBy: Z.
    Y displayAt: 100@20
```

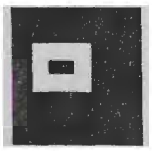


Fig. 3.1 Original

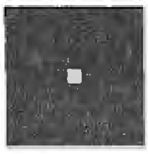


Fig. 3.2 Reference

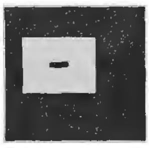


Fig. 3.3 Result

Reflect

Huang's expression for the reflection, through the origin, of an image X is given in his Formula 3.5 : -

$$\check{R} = \{(-x,-y) \mid (x,y) \in R\}.$$

We may implement the operation in Smaltalk by means of the following instance method.:-

```
reflect
  "Answer an image which containing the Reflected Reference Image .    F. L.   30  April
1992"
  | a b |
  a:= OrderedCollection new.
  b:= OrderedCollection new.
  a:= self getPointsFrom: self.
  1 to: a size do: [:i| b add:(self centre- (a at:i)+self centre ).
                    self at:(a at:i) put:0].
  1 to: b size do:[:i| self at:(b at:i) put: 1] .
```

We may test the method in the following way:-

```
     X:= ImageDataBase idbFromDisk:'image.dbs'.
     Z:=X at:'reflect'.
     Z displayAt: 50@20.
     Z reflect.
     Z displayAt: 100@20.
```

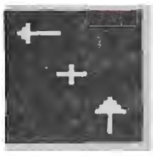


Fig. 4.1 Original

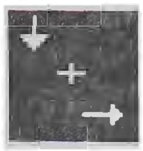


Fig. 4.2 Result

3. Translation of general BIA formulas into Smalltalk methods

Difference

Huang's expression for the difference of an image X by an image R is given in his Formula 4.1 :-

$$X/R = \{(x,y) \in X \mid (x,y) \notin R\} = X \cap \overline{R} = \overline{\overline{X} \cup R}$$

We may implement the operation in Smaltalk by means of the following instance method.:-

```
difference:imageR
 "Answer an image that containing the difference between   original image (imageX) and
reference image (imageR)       F. L.  30  April  1992  "

  ((self complement) union: imageR ) complement
```

We may test the method in the following way:-

```
    X:= ImageDataBase idbFromDisk:'image.dbs'.
    Y:=X at:'differenceX'.
    Y displayAt: 0@20.
    Z:=X at:'differenceR'.
    Z displayAt: 50@20.
    Y difference: Z.
    Y displayAt: 100@20
```

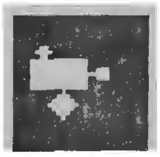


Fig. 5.1 Original

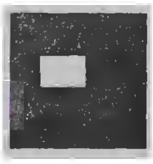


Fig. 5.2 Reference

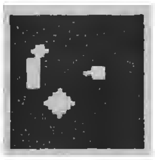


Fig. 5.3 Result

Intersection

Huang's expression for the intersection of an image X by an image R is given in his Formula 4.2 :-

$$X \cap R = \{(x, y) \mid (x, y) \in X \wedge (x, y) \in R\} = \overline{\overline{X} \cup \overline{R}}$$

We may implement the operation in Smaltalk by means of the following instance method.:-

```
intersection:imageR
 "Answer an image that contains the intersection of  original image (imageX) and
reference image (imageR)        F. L.   30  April   1992"

   ((self complement) union:(imageR complement)) complement
```

We may test the method in the following way:-

```
    X:= ImageDataBase idbFromDisk:'image.dbs'.
    Y:=X at:'intersectionX'.
    Y displayAt: 0@20.
    Z:=X at:'intersectionR'.
    Z displayAt: 50@20.
    Y intersection: Z.
    Y displayAt: 100@20
```

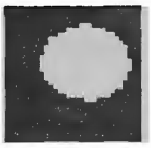


Fig. 6.1 Original

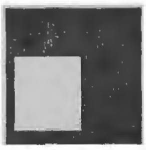


Fig. 6.2 Reference

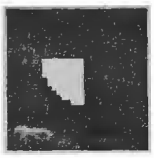


Fig. 6.3 Result

Erosion

Huang's expression for the erosion of an image X by an image R is given in his Formula 4.3 :-

$$X \ominus R = \overline{\overline{X} \oplus \check{R}}$$

We may implement the operation in Smaltalk by means of the following instance method.:-

```
erosionBy:imageR
 "Answer an image that containing the erosion of original   image (imageX) by reference
image (imageR)     F. L.  30  April  1992"

  ((self complement) dilationBy:(imageR reflect)) complement
```

We may test the method in the following way:-

```
    X:= ImageDataBase idbFromDisk:'image.dbs'.
    Y:=X at:'erosionX'.
    Y displayAt: 0@20.
    Z:=X at:'erosionR'.
    Z displayAt: 50@20.
    Y erosionBy: Z .
    Y displayAt: 100@20
```

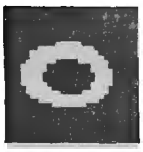


Fig. 7.1 Original

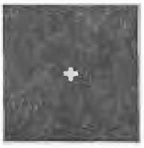


Fig. 7.2 Reference

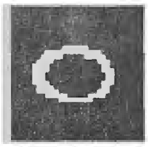


Fig. 7.3 Result

Symmetric Difference

Huang's expression for the symmetric difference of an image X by an image R is given in his Formula 4.4 :-

$$X \triangle R = (X/R) \cup (R/X) = \overline{\overline{X} \cup R} \cup \overline{\overline{R} \cup X}$$

We may implement the operation in Smaltalk by means of the following instance method.:-

```
symmetricDiff:imageR
 "Answer an image that containing the symmetric difference  between original image
(imageX) and reference image (imageR)      F. L.  30  April  1992"
   | a b |
   a:= (Image width: (self width) height: (self height)) black.
   a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
   a difference:imageR.
```

```
  b:= (Image width: (self width) height: (self height)) black.
  b copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
  imageR difference:b.
  a union:imageR.
  self copy: (0@0 extent:(self extent)) from:a to:0@0 rule:3
```

We may test the method in the following way:-

```
    X:= ImageDataBase idbFromDisk:'image.dbs'.
    Y:=X at:'symmetricDiffX'.
    Y displayAt: 0@20.
    Z:=X at:'symmetricDiffR'.
    Z displayAt: 50@20.
    Y symmetricDiff: Z .
    Y displayAt: 100@20
```

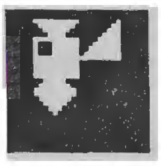


Fig.8.1 Original

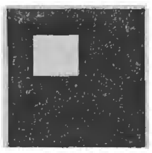


Fig. 8.2 Reference

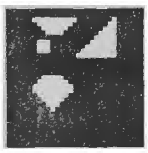


Fig. 8.3 Result

Opening

Huang's expression for the opening of an image X by an image R is given in his Formula 4.5 :-

$$X \circ R = (X \ominus R) \oplus R = \overline{\overline{X} \oplus \check{R}} \oplus R$$

We may implement the operation in Smaltalk by means of the following instance method.:-

```
openingBy:imageR
 "The opening   operation is an  erosion   followed  by a  dilation  with the same
reference imageR .  F. L.  30 April  1992"

   (self erosionBy:imageR) dilationBy:imageR
```

We may test the method in the following way:-

```
    X:= ImageDataBase idbFromDisk:'image.dbs'.
    Y:=X at:'openOrcloseX'.
    Y displayAt: 0@20.
    Z:=X at:'openOrcloseR'.
    Z displayAt: 50@20.
    Y openingBy: Z .
```

```
Y displayAt: 100@20
```

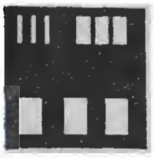


Fig. 9.1 Original

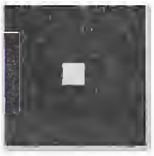


Fig. 9.2 Reference

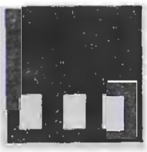


Fig. 9.3 Result

Closing

Huang's expression for the closing of an image X by an image R is given in his Formula 4.6 :-

$$\boldsymbol{X \bullet R = (X \oplus R) \ominus R = \overline{\overline{(X \oplus R)} \oplus \check{R}}}$$

We may implement the operation in Smaltalk by means of the following instance method.:-

```
closingBy:imageR
 "The closing operation is an dilation followed by an erosion with the same
reference imageR.   F. L.  30 April  1992  "

   (self dilationBy:imageR) erosionBy:imageR
```

We may test the method in the following way:-

```
     X:= ImageDataBase idbFromDisk:'image.dbs'.
     Y:=X at:'openOrcloseX'.
     Y displayAt: 0@20.
     Z:=X at:'openOrcloseR'.
     Z displayAt: 50@20.
     Y closingBy: Z .
     Y displayAt: 100@20
```

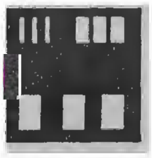


Fig. 10.1 Original

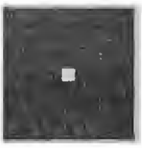


Fig. 10.2 Reference

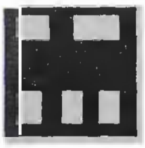


Fig. 10.3 Result

Hit or Miss Transform

Huang's expression for the hit or miss transform of an image X by an image pair (R1,R2) is given in his Formula 4.7 :-

$$X \circledast R = (X \ominus R_1) \cap (\overline{X} \ominus R_2) = \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}$$

We may implement the operation in Smaltalk by means of the following instance method.:-

hitMissTransBy:imageR1 and:imageR2
"The hit or miss transform of an image pair **R=(R1,R2)** is used to match the shape (or template) defined by the reference image pair R where R1 defines the foreground of the shape and R2 defines the background of the shape. F. L. 30 April 1992 "

```
| a b |
a:= (Image width: (self width) height: (self height)) black.
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
a erosionBy:imageR1.
b:= (Image width: (self width) height: (self height)) black.
b copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
(b complement) erosionBy:imageR2.
a intersection:b.
self copy: (0@0 extent:(self extent)) from:a to:0@0 rule:3
```

We may test the method in the following way:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'hitMissTrans'.
Y displayAt: 0@20.
Z:=X at:'triangle'.
Z displayAt: 50@20.
X:=X at:'cap'.
X displayAt: 100@20.
Y hitMissTransBy: Z and: X .
Y displayAt: 150@20
```

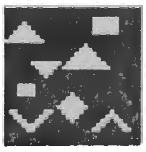


Fig. 11.1 Original

Fig. 11.2 and 2.3 Reference Pair

Fig. 11.4 Result

Thinning

Huang's expression for the thinning of an image X by an image pair (R1,R2) is given in his Formula 4.8 :-

$$X \circledcirc R = X/(X \circledast R) = \overline{\overline{X} \cup \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}}$$

We may implement the operation in Smaltalk by means of the following instance method.:-

thinningBy:imageR1 and:imageR2
"The thinning operation is anti extensive and decreases the size by removing the central points of the regions which match the reference image pair **R** = **(R1,R2)**. F. L. 30 April 1992"

```
 | a |
 a:= (Image width: (self width) height: (self height)) black.
 a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
 a hitMissTransBy:imageR1 and:imageR2.
 self difference: a
```

We may test the method in the following way:-

```
   X:= ImageDataBase idbFromDisk:'image.dbs'.
   Y:=X at:'thinOrthickX'.
   Y displayAt: 0@20.
   Z:=X at:'thinningRF'.
   Z displayAt: 50@20.
   W:=X at:'thinningRB'.
   W displayAt: 100@20.
   Y thinningBy: Z and: W .
   Y displayAt: 150@20
```

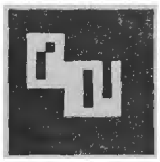
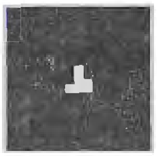


Fig. 12.1 Original



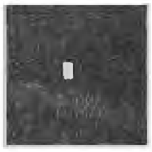


Fig. 12.2 and 2.3 Reference Pair

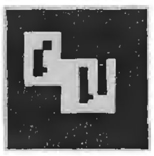


Fig. 12.4 Result

Thickening

Huang's expression for the thickening of an image X by an image pair (R1,R2) is given in his Formula 4.9 :-

$$X \odot R = X \cup (X \circledast R) = X \cup \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}$$

We may implement the operation in Smaltalk by means of the following instance method.:-

thickeningBy:imageR1 and:imageR2
"The thinning operation is extensive and increases the size by filling the image points where the regions match the reference image pair **R = (R1,R2)**. F. L. 30 April 1992"

```
 | a |
 a:= (Image width: (self width) height: (self height)) black.
 a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
 a hitMissTransBy:imageR1 and:imageR2.
 self union: a
```

We may test the method in the following way:-

```
    X:= ImageDataBase idbFromDisk:'image.dbs'.
    Y:=X at:'thinOrthickX'.
    Y displayAt: 0@20.
    Z:=X at:'thickeningRF'.
    Z displayAt: 50@20.
    W:=X at:'thickeningRB'.
    W displayAt: 100@20.
    Y thickeningBy: Z and: W .
    Y displayAt: 150@20
```

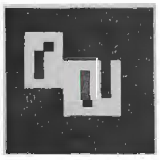


Fig. 13.1 Original

Fig. 13.2 and 13.3 Reference Pair

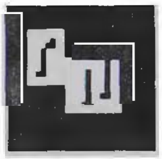


Fig. 13.4 Result

4. Examples: filters

One kind of morphological low pass filter, to remove high frequencies in the foreground of an image, can be achieved by opening:-

```
        X:= ImageDataBase idbFromDisk:'image.dbs'.
        Y:=X at:'openOrcloseX'.
        Y displayAt: 0@20.
        Z:=X at:'openOrcloseR'.
        Z displayAt: 50@20.
        Y openingBy: Z .
        Y displayAt: 100@20
```

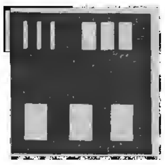


Fig. 14.1 Original

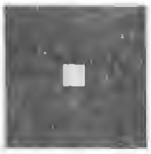


Fig. 14.2 Reference

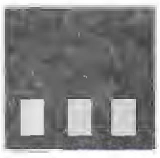


Fig. 14.3 Result

A second kind of morphological low pass filter, to remove high frequencies in the foreground of an image, can be achieved by closing:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'openOrcloseX'.
Y displayAt: 0@20.
Z:=X at:'openOrcloseR'.
Z displayAt: 50@20.
Y closingBy: Z .
Y displayAt: 100@20
```

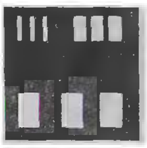


Fig. 15.1 Original

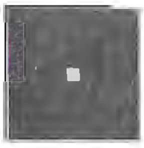


Fig. 15.2 Reference

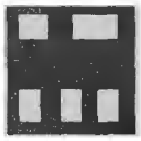


Fig. 15.3 Result

A morphological high pass filter which removes low frequencies in the foreground of an image X can be achieved by the difference of X and its opening:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'openOrcloseX'.
Y displayAt: 0@20.
Z:=X at:'openOrcloseR'.
Z displayAt: 50@20.
Y openingBy: Z .
X:= ImageDataBase idbFromDisk:'image.dbs'.
W:=X at:'openOrcloseX'.
W difference:Y.
W displayAt: 100@20
```

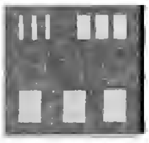


Fig. 16.1 Original

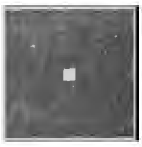


Fig. 16.2 Filter

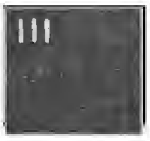


Fig. 16.3 Result

A morphological band pass filter which removes low frequencies and high frequencies in the foreground of an image X can be achieved by the difference of its opening with a smaller reference image R, and its opening with a larger reference image Q, where R belong to Q:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'openOrcloseX'.
Y displayAt: 0@20.
Z:=X at:'openOrcloseR'.
Z displayAt: 50@20.
Y openingBy: Z .
X:= ImageDataBase idbFromDisk:'image.dbs'.
Z:=X at:'openOrcloseX'.
Q:=X at:'openOrcloseQ'.
Q displayAt: 100@20.
Z openingBy: Q.
Y difference: Z.
Y displayAt:150@20
```

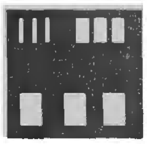


Fig 17.1 Original

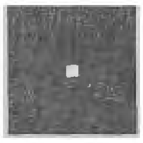


Fig. 17.2 Filter 1

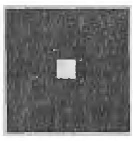


Fig. 17.3 Filter 2

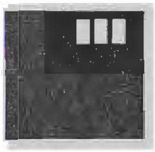


Fig. 17.4 Result

5. Examples: shape recognition (template matching)

In one form of shape recognition we may use the hit-or-miss transform to recognise locations of foreground points given by R1, and locations of background points given by R2.:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'hitMissTrans'.
Y displayAt: 0@20.
Z:=X at:'triangle'.
Z displayAt: 50@20.
X:=X at:'cap'.
X displayAt: 100@20.
Y hitMissTransBy: Z and: X .
Y displayAt: 150@20
```

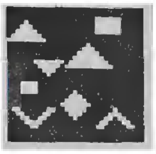


Fig. 18.1 Original

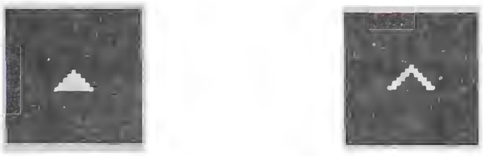


Fig. 18.2 and 18.3 Reference Pair

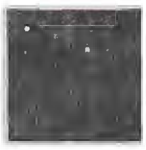


Fig. 18.4 Result

Part 2. Image Processing Environment

1. The small -image data-base

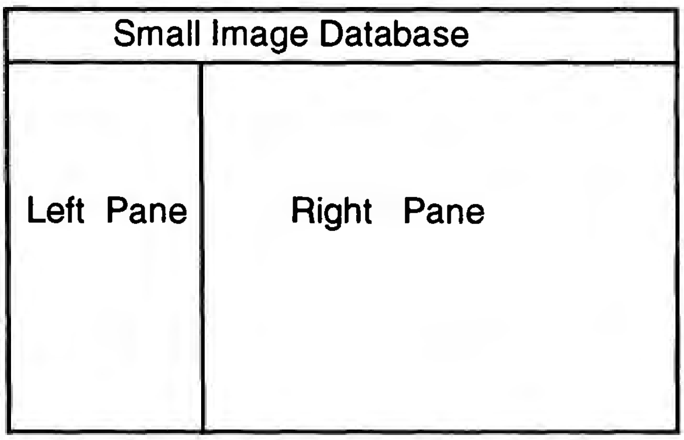


Fig. 19 The window of small image data base

Access to the small-image data-base is via a window with two panes. The left-pane shows a list of the names of images. The right-pane displays the selected image.

A menu for the left-pane offers the following options:-

- **add from disk** ----- Add new image from disk file .
- **bit editor** ----- Get into Bit Editor environment .
- **free drawing** ----- Get into Free Hand Drawing environment.
- **inspect** ----- To inspect the structure of the selected image.
- **remove** ----- To remove the selected image.
- **save IDB** ----- To save whole small image data base as a disk file.

The menu for the right-pane offers the following options:-

- **clear** ----- To clear the displayed image in the pane .
- **print** ----- To print out the selected image on a pin printer.
- **save as** ----- To save the selected image as a disk file.

In the first stage of the small-image data-base the image was stored as a file on disk containing a header followed by the (uncompressed) bitmap. The header had the following format:-

'**Image type**' space '**Image name**' space '**Width** (two bytes) **Height** (two bytes)'

2. The image-processing environment

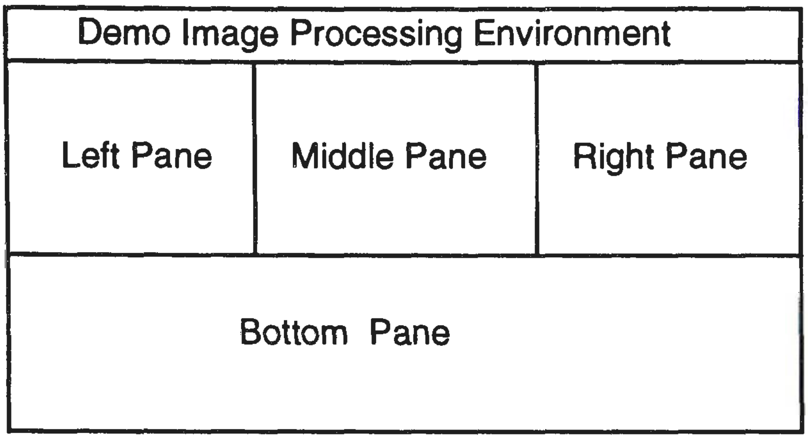


Fig. 20 The window of image processing environment

The window for the image processing environment has four panes. The *left upper* pane displays a selected original image; the *middle upper* pane displays the first selected reference image; and the *right upper* pane selects the second selected reference image. The *bottom pane* displays the image resulting from a selected operation with the images in the upper panes.

The menu for the left upper pane is:-

- **get it** ----- To get the image from selected image in small image data base as original image.

- **load IDB** ----- To load the small image data base from disk and open the data base environment.

- **from result** ----- To get the image from the result image. Then We can perform a sequence of image operation.

- **from disk** ----- To get the image from disk file.

The menu for the middle upper pane is:-

- **get it** ----- To get the image from selected image in small image data base as first reference image.

- **from result** ----- To get the image from the result image.

- **from disk** ----- To get the reference image from disk file.

The menu for the right upper pane is;-

- **get it** ----- To get the image from selected image in small image data base second reference image.

- **from result** ----- To get the image from the result image.

- **from disk** ----- To get the reference image from disk file.

The menu for the lower pane is;-

- **complement** ----- To reverse the original image.
- **union** ----- To union two images in upper left pane and upper middle pane.
- **dilation** ----- To dilate white part in original image by a reference image which displayed in upper middle pane.
- **erosion** ----- To erosion white parts in original image by a reference image which displayed in upper middle pane.
- **difference** ----- Get the difference between two images in upper left pane and upper middle pane.
- **reflect** ----- Get a reflection of the original image .
- **closing** ----- To close two images in upper left pane and upper middle pane.
- **opening** ----- To open two images in upper left pane and upper middle pane.
- **intersection** ----- Get the intersection between two images in upper left pane and upper middle pane.
- **symmetricDiff** ----- Get the symmetric difference between two images in upper left pane and upper middle pane.
- **hitMissTrans** ----- To get the hit and missing transform of original image in upper left pane and the reference image pair in upper middle pane and upper right pane.
- **thicken** ----- To get the thicken of original image in upper left pane and the reference image pair in upper middle pane and upper right pane.
- **thin** ----- To get the thinning of original image in upper left pane and the reference image pair in upper middle pane and upper right pane.
- **print** ----- To print out the result image on pin printer.
- **save as** ----- To save the result image as a disk file.

Part 3. Conclusion and further work

The aim of the present work was to produce an environment that would not only increase the author's understanding of morphological processes with images but would also facilitate exploratory research in the area. Further uses are now seen: as an expository tool, either directly via overhead projection of the monitor's display, or via hard-copy of images; and as a hands-on introduction to image-processing.

Future work will include:

- transferring the present work to the *Windows* version of Smalltalk V;
- extending the work from binary images to *grey-scale* images;
- introducing *data compression*;
- producing a *stand-alone version* accessible by people not familiar with Smalltalk.

Part 4. Acknowledgments

I would like to thank the many people who helped me in this work. I would especially like to thank ***Dr. Don Watson*** and ***Tom Peachey*** , the supervisors who encouraged me to go ahead with the original idea, guided my reading and assisted me in many ways. *Robert Hinterding* helped me with Smalltalk. *P. Rajendran* and *Damon Burgess* helped me accessing the facilities in our department.

Part 5. References:

- [Haralick 92] Haralick R.M., Shapiro L. G: Computer and Robot Vision. Addison-Wesley, 1992.

- [Huang 89] Huang K-S: A Digital Optical Cellular Image Processor. World Scientific, 1989.

- [Serra 82] Serra J: Image Analysis and Mathematical Morphology. Academic Press, 1982.

Appendix: The Classes and Methods

We have following classes and methods to support the small image data base and demo image processing environment:

ImageDataBase — Subclass of class Dictionary. F. L. 30 April 1992

```
Dictionary subclass: #ImageDataBase
 instanceVariableNames: ''
 classVariableNames: ''
 poolDictionaries: '' !
```

!ImageDataBase class methods !

idbFromDisk:fileName

```
   " Class method to retrieve a image dictionary on disk . --- Fred Apr 20, 1992."
 |inFile head aBitmap aBitmap1 aBitmap2 aBitmap3 aBitmap4
  anArray wTemp w h anImage aDictn aKey length|
   anImage:= Image new.
   aDictn:=ImageDataBase new.
   inFile := Disk file: fileName.
   inFile reset.
   [inFile atEnd] whileFalse:[
   head:= inFile nextWord.
   inFile next.
   aKey:= inFile nextWord.
   inFile next.
   (head='ColorForm') ifFalse:[ "black and white"
     h := inFile next asciiValue.
     wTemp := inFile next asciiValue.
     wTemp := (wTemp bitShift: 8) bitAnd:65280.
     w := wTemp bitOr: h.
     h := inFile next asciiValue.
     wTemp := inFile next asciiValue.
     wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
     h := wTemp bitOr: h.
     anImage := Image new width: w height: h .
     ((w\\16)=0 ) ifFalse:[
     w:=w+(16 - (w\\16)) deepCopy].
     aBitmap := Bitmap new: w*h/8.
     (1 to: w*h/8) do:
          [ :i | aBitmap at: We put: inFile next asciiValue].
     anImage bitmap: aBitmap .
                  ]
              ifTrue:[
     h := inFile next asciiValue.
     wTemp := inFile next asciiValue.
     wTemp := (wTemp bitShift: 8) bitAnd:65280.
     w := wTemp bitOr: h.
     h := inFile next asciiValue.
     wTemp := inFile next asciiValue.
     wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
     h := wTemp bitOr: h.
     anImage := (ColorForm new width: w height: h).
     ((w\\16)=0 ) ifFalse:[
     w:=w+(16 - (w\\16)) deepCopy].
     length:=w*h/8 deepCopy.
     aBitmap1:= Bitmap new:length.
     aBitmap2:= Bitmap new:length.
     aBitmap3:= Bitmap new:length.
     aBitmap4:= Bitmap new:length.
     (1 to: length) do:[ :i | aBitmap1 at: We put: (inFile next asciiValue)].
     (1 to: length) do:[ :i | aBitmap2 at: We put: (inFile next asciiValue)].
   (1 to: length) do:[ :i | aBitmap3 at: We put: (inFile next asciiValue)].
   (1 to: length) do:[ :i | aBitmap4 at: We put: (inFile next asciiValue)].
   anArray:= Array with:aBitmap1
               with:aBitmap2
               with:aBitmap3
               with:aBitmap4.
 anImage bitmap:anArray].
 aDictn at: aKey  put: anImage deepCopy ].
 inFile close .
 ^aDictn! !
```

!ImageDataBase methods !

idbStoreOnDisk:fileName

```
  " Instance method to store a image dictionary on disk as a file .      ---Fred  Apr 20, 1992."
  | a  outFile aWidth length|
  a:=OrderedCollection new.
  a:= self keys asOrderedCollection.
  outFile := Disk newFile: fileName.
  1 to: a size do:[:i|
  outFile nextPutAll: (self at:(a at:i)) class name.
  outFile nextPut: $ .
  outFile nextPutAll: (a at:i).
  outFile nextPut: $ .
  ((self at:(a at:i)) class name )='Image' ifTrue:[
    outFile nextTwoBytesPut: ((self at:(a at:i ))  width)  .
    outFile nextTwoBytesPut: ((self at:(a at:i ))  height) .
    (self at:(a at:i )) bitmap  do:
         [:ea | outFile nextPut: ea asCharacter]  ]
                              ifFalse:[
    outFile nextTwoBytesPut: ((self at:(a at:i)) width)  .
    outFile nextTwoBytesPut: ((self at:(a at:i)) height) .
    aWidth:=(self at:(a at:i)) width.
    ((aWidth\\16)=0 ) ifFalse:[
    aWidth:=aWidth+(16 - (aWidth\\16)) deepCopy].
    length:=aWidth*((self at:(a at:i)) height)/8.
    1 to: 4 do:[:aBitmap| (((self at:(a at:i )) bitmap) at: aBitmap) do:
            [:ea | outFile nextPut: ea asCharacter]]
                                        ]
            ].
   outFile close!
```

open

```
     "Open an ImageDataBase inspector window   on the receiver.    F. L.   30  April   1992"
   ImageDataBaseInspector new openOn: self! !
```

IdbFreeDraw ---- For free hand drawing in the environment.

```
FreeDrawing subclass: #IdbFreeDraw
  instanceVariableNames:
    'anImage    fileName '
  classVariableNames: ''
  poolDictionaries: '' !
```

!IdbFreeDraw class methods ! !

!IdbFreeDraw methods !

loadFromFile

```
 " To get  image from a disk file."
   | aPrompter  head inFile w h  wTemp |
 fileName:= String new.
 aPrompter:=Prompter prompt:'file name?'
                 default:fileName .
 fileName:= aPrompter.
 (fileName=nil) ifTrue:[^nil].
 (fileName size=0) ifTrue:[^nil].
 inFile := Disk file: fileName.
 inFile reset.
 head:= inFile nextWord.
 inFile next.
 (head='color') ifFalse:[inFile reset].
 h := inFile next asciiValue.
 wTemp := inFile next asciiValue.
 wTemp := (wTemp bitShift: 8) bitAnd:65280.
 w := wTemp bitOr: h.
 h := inFile next asciiValue.
 wTemp := inFile next asciiValue.
 wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
 h := wTemp bitOr: h.
 ((w\\16)=0 ) ifFalse:[
 w:=w+(16 - (w\\16)) deepCopy].
 inFile close.
 (head='color') ifFalse:[ "black an white"
        anImage:=Image new width:w height:h.
        CursorManager execute change.
        anImage:=Image idbFromFile:fileName.
        CursorManager normal change.
        self clear.
        anImage displayAt:pane frame origin.
        ^anImage].
 anImage:=ColorForm new width:w height:h.
 CursorManager execute change.
 anImage:=ColorForm idbFromFile:fileName.
 CursorManager normal change.
 self clear.
 anImage displayAt:pane frame origin.
 ^anImage!
```

loadImage

```
  " Answer a menu , give the selectors of add an image to the IDB."
   | aMenu |
  aMenu:= (Menu
   labels: 'from Disk\from IDB' withCrs
   lines:  #(1)
   selectors: #(loadFromFile loadFromIdb ) )
   popUpAt:  Cursor position.
  aMenu isNil
   ifFalse: [self perform: aMenu]!
```

open

```
      "Open a free drawing window."
   | topPane |
   menu := Menu
labels:'clear \ change size \ change color \ change halftone \ change rule \ change font \ draw \ erase \ line \ rectangle \ circle \ ellipse \ curve \
fill \ copy \ paste \ bit edit \ save whole \ save selected \ load image' withCrs
      lines: #(1 5 7 13)
selectors: #(clear changeSize changeColor changeMask changeRule changeFont draw erase line rectangle circle ellipse curve fill
copyGraph pasteGraph bitEdit saveWhole saveSelected loadImage).
   topPane := TopPane new
      model: self;
      label: 'Form Editor';
      minimumSize: Display extent // 3;
      yourself.
```

```
topPane addSubpane:
   (pane :=
   FreeDrawPane new
      model: self;
      name: #initialize:;
      menu: #menu;
      change: #change).
topPane dispatcher open scheduleWindow!
```

saveSelected

```
" To save selected image on a disk file."
  | aPrompter aMenu |
fileName:= String new.
aPrompter:=Prompter prompt:'file name?'
               default:'image.drw' .
fileName:= aPrompter.
(fileName=nil) ifTrue:[^nil].
(fileName size=0) ifTrue:[^nil].
anImage:=Display compatibleForm fromUser deepCopy.
aMenu:= (Menu
  labels: 'as color\ as black & white' withCrs
  lines:  #(1)
  selectors:#(storeAsColor storeAsBlkAndWit))
  popUpAt:  Cursor position.
aMenu isNil
  ifFalse: [self perform: aMenu]!
```

saveWhole

```
" To save selected whole pane on a disk file."
  | aPrompter aMenu|
fileName:= String new.
anImage:= Display compatibleForm
             fromDisplay:(pane frame) deepCopy.
aPrompter:=Prompter prompt:'file name?'
               default:'image.drw' .
fileName:= aPrompter.
(fileName=nil) ifTrue:[^nil].
(fileName size=0) ifTrue:[^nil].
aMenu:= (Menu
  labels: 'as color\ as black & white' withCrs
  lines:  #(1)
  selectors:#(storeAsColor storeAsBlkAndWit))
  popUpAt:  Cursor position.
aMenu isNil
  ifFalse: [self perform: aMenu].!
```

storeAsBlkAndWit

```
CursorManager execute change.
  anImage storeAsBkAndWt:fileName.
CursorManager normal change.!
```

storeAsColor

```
CursorManager execute change.
  anImage storeColorOnFile:fileName.
CursorManager normal change.! !
```

IdbBtEdt ---- For Bit Editor in the environment.

```
BitEditor subclass: #IdbBtEdt
  instanceVariableNames: ''
  classVariableNames: ''
```

```
 poolDictionaries: '' !
```

```
!IdbBtEdt class methods ! !
```

```
!IdbBtEdt methods !
```

storeOnFile

```
 " To save selected image on a disk file."
   | aPrompter  fileName |
  aPrompter:=Prompter prompt:'file name?'
                default:'image.img'.
  fileName:= aPrompter.
  (fileName=nil) ifTrue:[^nil].
  (fileName size=0) ifTrue:[^nil].
  (imageForm class name)='Image' ifTrue:[
   CursorManager execute change.
   imageForm storeOnFile:fileName.
   CursorManager normal change.]
                    ifFalse:[
   CursorManager execute change.
   imageForm storeColorOnFile:fileName.
   CursorManager normal change.]!
```

windowMenu

```
     "Answer the window menu for the bit editor."
  ^Menu
     labels: 'cycle\close\save as' withCrs
     lines: #()
     selectors: #(cycle closeIt storeOnFile ).! !
```

IdbTopDispatcher ---- To reinitialize the window size.

```
TopDispatcher subclass: #IdbTopDispatcher
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !
```

```
!IdbTopDispatcher class methods ! !
```

```
!IdbTopDispatcher methods !
```

initWindowSize

```
     "Private - Answer default initial window extent."
  ^Display extent*1//2! !
```

ImageDataBaseInspector ----- The main class for the environment

```
DictionaryInspector subclass: #ImageDataBaseInspector
  instanceVariableNames:
   'anIdbTopPane    fileName '
```

```
 classVariableNames: ''
 poolDictionaries: '' !

!ImageDataBaseInspector class methods ! !

!ImageDataBaseInspector methods !
```

bitEdt

```
 | aPrompter anImage width height key size index|
 (instIndex=nil) ifTrue:[instIndex:=1] .
 aPrompter:=Prompter prompt:'Graph object name?'
        default: (instList at: instIndex) key.
  fileName:= aPrompter.
  (fileName = nil)  ifTrue:[^nil].
  fileName isEmpty ifTrue:[
  width:=(Prompter prompt:'Width = ?'
        default: '47') asInteger.
 (width = 0)  ifTrue:[^nil].
 height:=(Prompter prompt:'Height = ?'
        default: '47') asInteger.
 anImage:=Image new width:width height:height .
 IdbBtEdt new openOn: anImage ].
 anImage:=object at: (instList at: instIndex) key.
 IdbBtEdt new openOn: anImage .!
```

clear

```
     " To clear the instance pane"
    ^(Image new width:instPane frame
                  width height:instPane frame height; white)
         displayAt: instPane frame origin.!
```

freDrw

```
   IdbFreeDraw new!
```

fromDisk

```
  " To get image from disk file, insert into the  data base"
  |  anImage aPrompter key size index |
  aPrompter:=Prompter prompt:'file name?'
         default: fileName.
  fileName:= aPrompter.
  (fileName=nil ) ifTrue:[^nil].
  fileName isEmpty ifTrue:[^nil].
  CursorManager execute change.
  anImage:=Image idbFromFile:fileName.
  CursorManager normal change.
   key := Prompter
      prompt: 'new key expression'
      defaultExpression: String new.
   key isNil
      ifTrue: [^self].
   (object includesKey: key)
      ifTrue: [
          Menu message: 'key already in dictionary'.
          ^self].
   object at: key put: anImage.
   instList add:
      (Association key: key value: key printString).
   size := instList size.
   index := 1.
   [index > size
      or: [(instList at: index) key = key]]
            whileFalse: [index := index + 1].
   instIndex := index.
   self
      changed: #instVarList
```

```
        with: #restoreSelected:
        with: instIndex;
    changed: #instance:!
```

idbInsPaneMenu

```
  " Answer instance pane menu"
  | menu |
  menu := Menu
     . labels: 'clear\print\save as' withCrs
      lines: #(1)
      selectors: #(clear printImage storeOnFile).
 ^menu!
```

idbListMenu

```
     "Private - Answer  the dictionary   inspector list pane menu."
   ^Menu
      labels: 'add from disk\bit editor\free drawing\inspect\remove\save IDB' withCrs
      lines: #(1 3 5)
      selectors: #( fromDisk bitEdt freDrw inspectSelection remove storDic )!
```

instance: anImage

```
     Q:=instIndex.
     instIndex isNil
     ifTrue: [^(instPane clear)].
     instPane clear.
    ^(object at: (instList at: instIndex) key)
               displayAt: instPane frame origin
               clippingBox: ((instPane frame origin)
               extent:( instPane frame extent))!
```

loadDic

```
 "To load all the images in data base onto disk"
    | aPrompter  fileName |
 aPrompter:=Prompter prompt:'file name?'
               default:'image.dic'.
  fileName:= aPrompter.
  (fileName=nil) ifTrue:[^nil].
  (fileName size=0)  ifTrue:[^nil].
  CursorManager execute change.
  object:=ImageDataBase idbFromDisk:fileName deepCopy.
  CursorManager normal change .
  anIdbTopPane dispatcher close  scheduleWindow.
  ^self openOn:object!
```

openOn: anObject

```
     "Open an inspector window on anObject.  Define  the pane sizes and behavior, and shedule the  window."
   object := anObject.
   instPane := GraphPane new
      menu: #idbInsPaneMenu;
      model: self;
      name: #instance:;
      framingRatio: (1/3 @ 0
         extent: 2/3 @ 1).
   anIdbTopPane := IdbTopPane new.
   anIdbTopPane
      label: ' Small Image Data Base  ';
      model: anIdbTopPane dispatcher;
      menu: #workSpaceMenu;
      minimumSize: 80@80;
      yourself.
   anIdbTopPane addSubpane:
      (ListPane new
         menu: #idbListMenu;
         model: self;
         name: #instVarList;
         change: #selectInstance:;
```

```
        returnIndex: true;
        framingRatio: (
            0@0 extent: 1/3 @ 1)).
  anIdbTopPane addSubpane: instPane.
  CursorManager normal change.
  self setInstList.
  anIdbTopPane dispatcher open scheduleWindow!
```

printImage

```
   "To print out the selected image on pin-printer"
 ^(object at: (instList at: instIndex) key)
     outputToPrinterUpright!
```

remove

```
      "Private - Remove the selected
       key from the dictionary."
  | assoc |
  instIndex isNil
     ifFalse: [
        assoc := instList at: instIndex.
        instList remove: assoc.
        object removeKey: assoc key.
        instIndex := nil.
        self
            changed: #instVarList with: #restore;
            changed: #instance:]!
```

selectInstance: anInteger

```
      "Private - Select the instance variable at   index position anInteger in the list."
   | lastIndex |
   lastIndex := instIndex.
   instIndex := anInteger.
   self changed: #instance:.
   instIndex = lastIndex
      ifTrue: [self inspectSelection]!
```

storDic

```
  "To store all the image data base on disk"
     | aPrompter |
  aPrompter:=Prompter prompt:'file name?'
                 default: fileName .
   fileName:= aPrompter.
   (fileName=nil) ifTrue:[^nil].
   (fileName size=0)  ifTrue:[^nil].
   CursorManager execute change.
   object idbStoreOnDisk:fileName.
   CursorManager normal change!
```

storeOnFile

```
  " To save selected image on a disk file."
     | aPrompter anImage |
    instIndex  isNil
       ifTrue: [^self].
   aPrompter:=Prompter prompt:'file name?'
                  default:fileName .
   fileName:= aPrompter.
   (fileName=nil) ifTrue:[^nil].
   (fileName size=0) ifTrue:[^nil].
   anImage:= (object at: (instList at: instIndex) key).
   (anImage class name)='Image' ifTrue:[
     CursorManager execute change.
     anImage storeOnFile:fileName.
     CursorManager normal change.]
                         ifFalse:[
     CursorManager execute change.
     anImage storeColorOnFile:fileName.
     CursorManager normal change.]! !
```

IdbTopPane ----- Subclass of TopPane

```
TopPane subclass: #IdbTopPane
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !

!IdbTopPane class methods ! !


!IdbTopPane methods !
```

defaultDispatcherClass

```
    "Answer the default dispatcher."  ^IdbTopDispatcher! !
```

Image ----- The class for image processing

```
Form subclass: #Image
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !

!Image class methods !
```

idbFromFile: fileName

```
   "To  retrieve  the stored image from disk-- both colour  image and black and white image.  FL  Apr 19, 1992."
   | inFile aBitmap aBitmap1 aBitmap2 aBitmap3 aBitmap4
    anArray w h anImage wTemp head length|
   inFile := Disk file: fileName.
   inFile reset.
   head:= inFile nextWord.
   (head='color') ifFalse:[ "black an white"
     inFile reset.
     h := inFile next asciiValue.
     wTemp := inFile next asciiValue.
     wTemp := (wTemp bitShift: 8) bitAnd:65280.
     w := wTemp bitOr: h.
     h := inFile next asciiValue.
     wTemp := inFile next asciiValue.
     wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
     h := wTemp bitOr: h.
     anImage := Image new width: w height: h .
     ((w\\16)=0 ) ifFalse:[
     w:=w+(16 - (w\\16)) deepCopy].
     aBitmap := Bitmap new: w*h/8.
     (1 to: w*h/8) do:
           [ :i | aBitmap at: We put: inFile next asciiValue].
     anImage bitmap: aBitmap .
     ^anImage
                  ].
   inFile next.
   h := inFile next asciiValue.
   wTemp := inFile next asciiValue.
   wTemp := (wTemp bitShift: 8) bitAnd:65280.
   w := wTemp bitOr: h.
   h := inFile next asciiValue.
   wTemp := inFile next asciiValue.
```

```
 wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
 h := wTemp bitOr: h.
 anImage := (ColorForm new width: w height: h).
 ((w\\16)=0 ) ifFalse:[
 w:=w+(16 - (w\\16)) deepCopy].
 length:=w*h/8 deepCopy.
 aBitmap1:= Bitmap new:length.
 aBitmap2:= Bitmap new:length.
 aBitmap3:= Bitmap new:length.
 aBitmap4:= Bitmap new:length.
 (1 to: length) do:[ :i | aBitmap1 at: We put: (inFile next asciiValue)].
 (1 to: length) do:[ :i | aBitmap2 at: We put: (inFile next asciiValue)].
 (1 to: length) do:[ :i | aBitmap3 at: We put: (inFile next asciiValue)].
 (1 to: length) do:[ :i | aBitmap4 at: We put: (inFile next asciiValue)].
 anArray:= Array with:aBitmap1
             with:aBitmap2
             with:aBitmap3
             with:aBitmap4.
 anImage bitmap:anArray.
 ^anImage! !
```

!Image methods !

center

```
     "Answer a Point,  the center of the receiver."
  ^((self width- 1)@(self height- 1)//2)!
```

closingBy:imageR

```
  "The opening operation is an dilation followed by an  erosion with the same reference imageR ."
   (self dilationBy:imageR) erosionBy:imageR!
```

complement

```
   "Answer the complement of an image."
  ^self reverse!
```

difference:imageR

```
  "Answer an image that containing the difference between  original image (imageX) and referrence image (imageR)"
   ((self complement) union: imageR ) complement!
```

dilationBy:imageR

```
   "Dilation an image (imageX) by a reference image (imageR)"
   | a b c|
   b:= (Image width: (self width) height: (self height)) black.
   c:= (Image width: (self width) height: (self height)) black.
   a:= OrderedCollection new.
   a:= imageR getPointsFrom:imageR.
   1 to: a size do:[:i|
     b copy: (0@0 extent:(self extent)) from:self to:((a at:i)- (imageR center)) rule:3.
     c union:b
       ].
    self copy: (0@0 extent:(self extent)) from:c to:0@0 rule:3!
```

erosionBy:imageR

```
  "Answer an image that containing the erosion of original  image (imageX) by referrence image (imageR)"
  ((self complement) dilationBy:(imageR reflect)) complement!
```

getPointsFrom:imageR

```
    "Method to get the position of each white pixel (forground) of the reference image , answer a ordered collection containing    white
points."
    | a |
    a:=OrderedCollection new.
    0 to: (imageR width- 1) do:[:i|
         0 to: (imageR height- 1) do:[:j|
            ((imageR at:(i@j))=1) ifTrue:[a add:i@j]]] .
```

```
^a!
```

hitMissTransBy:imageR1 and:imageR2

```
"The hit or miss transform of an image pair R=(R1,R2)  is used to match the shape (or template) defined by  the
reference image pair R where R1 defines the forground of the shape and R2 defines the background of the shape."
 | a b |
 a:= (Image width: (self width) height: (self height)) black.
 a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
 a erosionBy:imageR1.
 b:= (Image width: (self width) height: (self height)) black.
 b copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
 (b complement) erosionBy:imageR2.
 a intersection:b.
 self copy: (0@0 extent:(self extent)) from:a to:0@0 rule:3!
```

intersection:imageR

```
"Answer an image that containing the intersection of  original image (imageX) and referrence image (imageR)"
((self complement) union:(imageR complement)) complement!
```

openingBy:imageR

```
"The opening operation is an erosion followed by a  dilation with the same reference imageR ."
(self erosionBy:imageR) dilationBy:imageR!
```

reflect

```
"Answer an image which containing the Reflected Reference Image ."
| a b |
a:= OrderedCollection new.
b:= OrderedCollection new.
a:= self getPointsFrom: self.
1 to: a size do: [:i| b add:(self center- (a at:i)+self center ).
                   self at:(a at:i) put:0].
1 to: b size do:[:i| self at:(b at:i) put: 1] .!
```

storeOnFile: fileName

```
 "DW 23 Mar., 1992 "
 | outFile  |
 outFile := Disk newFile: fileName.
 "Delete existing contents"
 outFile nextTwoBytesPut: (self width)  .
 outFile nextTwoBytesPut: (self height) .
 self bitmap do:
    [:ea | outFile nextPut: ea asCharacter].
 outFile close.!
```

symmetricDiff:imageR

```
"Answer an image that containing the symmetric difference  between original image (imageX) and referrence image (imageR)"
 | a b |
 a:= (Image width: (self width) height: (self height)) black.
 a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
 a difference:imageR.
 b:= (Image width: (self width) height: (self height)) black.
 b copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
 imageR difference:b.
 a union:imageR.
 self copy: (0@0 extent:(self extent)) from:a to:0@0 rule:3!
```

thickeningBy:imageR1 and:imageR2

```
 "The thinning operation is extensive and increases the   size by filling the image points where the regions    match the reference image
pair R = (R1,R2)."
 | a |
 a:= (Image width: (self width) height: (self height)) black.
 a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
 a hitMissTransBy:imageR1 and:imageR2.
 self union: a!
```

thinningBy:imageR1 and:imageR2

```
   "The thinning operation is antiextensive and decreases  the size by removeing the central points of the regions  which match the reference
image pair R = (R1,R2)."
   | a |
   a:= (Image width: (self width) height: (self height)) black.
   a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
   a hitMissTransBy:imageR1 and:imageR2.
   self difference: a!
```

union: imageR

```
      "Answer an image containing the image of the union of imageR and the receiver imageX.   F. L.   30  April   1992"
     ^self copy: (0@0 extent:(imageR extent)) from:imageR to:0@0 rule:7 .! !
```

ImageProcessing ---- The main class to perform demo image processing environment.

```
Object subclass: #ImageProcessing
  instanceVariableNames:
    'object leftPane midlePane rightPane bottomPane fileName instList instIndex indexSet resultImage
resultImageA resultImageB resultImageC '
  classVariableNames: ''
  poolDictionaries: '' !
```

!ImageProcessing class methods ! !

!ImageProcessing methods !

bottomPaneMenu

```
 ^Menu
   labels: 'complement\union\dilation\reflect\erosion\difference\closing\opening
\intersection\symmetricDiff\hitMissTrans\thicken\thin\print\save as' withCrs
   lines: #(1 5 9 13)
   selectors: #(complement union dilation reflect erosion difference closing  opening
intersection symmetricDiff  hitMissTrans thicken thin printResult storeOnFile)!
```

closing

```
    | a b |
   bottomPane clear.
   a:=resultImageA deepCopy.
   b:=resultImageB deepCopy.
   CursorManager execute change.
   a closingBy: b.
   CursorManager normal change.
   resultImage:=a deepCopy.
  ^resultImage displayAt: bottomPane frame origin
     clippingBox: ((bottomPane frame origin)
     extent:( bottomPane frame extent))!
```

complement

```
    | a |
   bottomPane clear.
   a:=resultImageA deepCopy.
   CursorManager execute change.
   a complement.
   CursorManager normal change.
   resultImage:=a deepCopy.
  ^resultImage displayAt: bottomPane frame origin
     clippingBox: ((bottomPane frame origin)
  extent:( bottomPane frame extent))!
```

difference

```
 | a b |
 bottomPane clear.
 a:=resultImageA deepCopy.
 b:=resultImageB deepCopy.
 CursorManager execute change.
 a difference: b.
 CursorManager normal change.
 resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
   clippingBox: ((bottomPane frame origin)
   extent:( bottomPane frame extent))!
```

dilation

```
 | a b |
 bottomPane clear.
 a:=resultImageA deepCopy.
 b:=resultImageB deepCopy.
 CursorManager execute change.
 a dilationBy: b.
 CursorManager normal change.
 resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
   clippingBox: ((bottomPane frame origin)
   extent:( bottomPane frame extent))!
```

erosion

```
 | a b |
 bottomPane clear.
 a:=resultImageA deepCopy.
 b:=resultImageB deepCopy.
 CursorManager execute change.
 a erosionBy: b.
 CursorManager normal change.
 resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
   clippingBox: ((bottomPane frame origin)
   extent:( bottomPane frame extent))!
```

fromDisk

```
 " To get image from disk file, insert into the
   data base"
 | aPrompter key size index |
 aPrompter:=Prompter prompt:'file name?'
        default: fileName.
 fileName:= aPrompter.
 (fileName=nil ) ifTrue:[^nil].
 fileName isEmpty ifTrue:[^nil].
 CursorManager execute change.
 resultImage:=Image idbFromFile:fileName.
 CursorManager normal change.
  leftPane hasCursor ifTrue:[
 resultImageA:=resultImage deepCopy.
^resultImageA
   displayAt: leftPane frame origin
   clippingBox: ((leftPane frame origin)
   extent:( leftPane frame extent))].
  midlePane hasCursor ifTrue:[
 resultImageB:=resultImage deepCopy.
^resultImageB
   displayAt: midlePane frame origin
   clippingBox: ((midlePane frame origin)
   extent:( midlePane frame extent))].
  rightPane hasCursor ifTrue:[
 resultImageB:=resultImage deepCopy.
^resultImageC
```

```
    displayAt: rightPane frame origin
    clippingBox: ((rightPane frame origin)
    extent:( rightPane frame extent))]!
```

fromResult

```
   "To get the image from the result image for
    another operation.   Fred Liu  May 11, 1992 "
   leftPane hasCursor ifTrue:[
 resultImageA:=resultImage deepCopy.
 ^resultImageA
    displayAt: leftPane frame origin
    clippingBox: ((leftPane frame origin)
    extent:( leftPane frame extent))].
   midlePane hasCursor ifTrue:[
 resultImageB:=resultImage deepCopy.
 ^resultImageB
    displayAt: midlePane frame origin
    clippingBox: ((midlePane frame origin)
    extent:( midlePane frame extent))].
   rightPane hasCursor ifTrue:[
 resultImageC:=resultImage deepCopy.
 ^resultImageC
    displayAt: rightPane frame origin
    clippingBox: ((rightPane frame origin)
    extent:( rightPane frame extent))]!
```

getIt

```
  "To get the image from the Image Data Base."
    instIndex:=Q.
   leftPane hasCursor ifTrue:[
     indexSet at:1 put: instIndex.
    ^ (self changed: #instanceLft:)].
   midlePane hasCursor ifTrue:[
     indexSet at:2 put: instIndex.
    ^ (self changed: #instanceMid:)].
   rightPane hasCursor ifTrue:[
     indexSet at:3 put: instIndex.
    ^( self changed: #instanceRit:)]!
```

hitMissTrans

```
   | a b c |
   bottomPane clear.
   a:=resultImageA deepCopy.
   b:=resultImageB deepCopy.
   c:=resultImageC deepCopy.
   CursorManager execute change.
   a hitMissTransBy: b and: c.
   CursorManager normal change.
   resultImage:=a deepCopy.
   ^resultImage displayAt: bottomPane frame origin
     clippingBox: ((bottomPane frame origin)
     extent:( bottomPane frame extent))!
```

instanceBtm: anImage

```
       instIndex isNil
       ifTrue: [^( bottomPane clear)].
       bottomPane clear.
     ^resultImage
                 displayAt: bottomPane frame origin
                 clippingBox: ((bottomPane frame origin)
                 extent:( bottomPane frame extent))!
```

instanceLft: anImage

```
     instIndex isNil
     ifTrue: [^leftPane clear].
     leftPane clear.
     resultImageA:=(object at: (instList at: instIndex) key) deepCopy.
     ^resultImageA
```

```
            displayAt: leftPane frame origin
            clippingBox: ((leftPane frame origin)
            extent:( leftPane frame extent))!
```

instanceMid: anImage

```
  instIndex isNil
  ifTrue: [^midlePane clear].
  midlePane clear.
  resultImageB:=(object at: (instList at: instIndex) key) deepCopy.
  ^resultImageB
            displayAt: midlePane frame origin
            clippingBox: ((midlePane frame origin)
            extent:( midlePane frame extent))!
```

instanceRit: anImage

```
   instIndex isNil
   ifTrue: [^(rightPane clear)].
   rightPane clear.
  resultImageC:=(object at: (instList at: instIndex) key) deepCopy.
  ^resultImageC
            displayAt: rightPane frame origin
            clippingBox: ((rightPane frame origin)
            extent:( rightPane frame extent))!
```

intersection

```
  | a b |
 bottomPane clear.
 a:=resultImageA deepCopy.
 b:=resultImageB deepCopy.
 CursorManager execute change.
 a intersection: b.
 CursorManager normal change.
 resultImage:=a deepCopy.
 ^resultImage displayAt: bottomPane frame origin
   clippingBox: ((bottomPane frame origin)
   extent:( bottomPane frame extent))!
```

leftPaneMenu

```
  ^Menu
    labels: 'get it\load IDB\from result\from disk' withCrs
    lines: #(0)
    selectors: #( getIt openIdb fromResult fromDisk)!
```

openIdb

```
 (ImageDataBaseInspector allInstances size)=0
        ifFalse:[^nil].
 object open!
```

opening

```
  | a b |
  bottomPane clear.
  a:=resultImageA deepCopy.
  b:=resultImageB deepCopy.
  CursorManager execute change.
  a openingBy: b.
  CursorManager normal change.
  resultImage:=a deepCopy.
  ^resultImage displayAt: bottomPane frame origin
    clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!
```

openOn:anObject

```
  "To open an image processing environment"
  | anIdbTopPane |
  object:=anObject.
```

```
indexSet:=Array new:4.
anIdbTopPane := IdbTopPane new.
anIdbTopPane
   label: 'Demo Image Processing';
   model: anIdbTopPane dispatcher;
   menu: #workSpaceMenu;
   minimumSize: 80@80;
   yourself.
anIdbTopPane addSubpane:
  (leftPane:=GraphPane new
   menu: #leftPaneMenu;
   model: self;
   name: #instanceLft:;
   framingRatio: (0 @ 0
      extent: 3/7 @ (1/2))).
anIdbTopPane addSubpane:
  (midlePane:=GraphPane new
   menu: #rightPaneMenu;
   model: self;
   name: #instanceMid:;
   framingRatio: (3/7 @ 0
      extent: 2/7 @ (1/2))).
 anIdbTopPane addSubpane:
  (rightPane:=GraphPane new
   menu: #rightPaneMenu;
   model: self;
   name: #instanceRit:;
   framingRatio: (5/7 @ 0
      extent: 2/7 @ (1/2))).
anIdbTopPane addSubpane:
  (bottomPane:=GraphPane new
   menu: #bottomPaneMenu;
   model: self;
   name: #instanceBtm:;
   framingRatio: (0 @ (1/2)
      extent: 1 @ (1/2))).
CursorManager normal change.
self setInstList.
anIdbTopPane dispatcher open scheduleWindow!
```

printResult

```
  "To print out the result image on pin-printer"
   ^resultImage outputToPrinterUpright!
```

reflect

```
  | a |
  bottomPane clear.
  a:=resultImageA deepCopy.
  CursorManager execute change.
  a reflect.
  CursorManager normal change.
  resultImage:=a deepCopy.
 ^resultImage displayAt: bottomPane frame origin
```

```
    clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!
```

rightPaneMenu

```
  ^Menu
    labels: 'get it\from disk\from result' withCrs
    lines: #(0)
    selectors: #( getIt fromDisk fromResult)!
```

setInstList

```
     "Private - Compute instList, an
      OrderedCollection of key strings
      for the list pane."
  | aSet |
  aSet := Set new: object size.
  object keysDo: [:aKey |
     aSet add:
        (Association key: aKey value: aKey printString)].
  instIndex := nil.
  (instList := SortedCollection new)
     sortBlock: [:a :b| a value < b value];
     addAll: aSet!
```

storeOnFile

```
 " To save selected image on a disk file."
   | aPrompter |
  instIndex  isNil
     ifTrue: [^self].
 aPrompter:=Prompter prompt:'file name?'
                default:fileName .
 fileName:= aPrompter.
 (fileName=nil) ifTrue:[^nil].
 (fileName size=0) ifTrue:[^nil].
 (resultImage class name)='Image' ifTrue:[
   CursorManager execute change.
   resultImage storeOnFile:fileName.
   CursorManager normal change.]
                        ifFalse:[
   CursorManager execute change.
   resultImage storeColorOnFile:fileName.
   CursorManager normal change.]!
```

symmetricDiff

```
   | a b |
  bottomPane clear.
  a:=resultImageA deepCopy.
  b:=resultImageB deepCopy.
  CursorManager execute change.
  a symmetricDiff: b.
  CursorManager normal change.
  resultImage:=a deepCopy.
  ^resultImage displayAt: bottomPane frame origin
    clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!
```

thicken

```
   | a b c |
  bottomPane clear.
  a:=resultImageA deepCopy.
  b:=resultImageB deepCopy.
  c:=resultImageC deepCopy.
  CursorManager execute change.
  a thickeningBy: b and: c.
  CursorManager normal change.
  resultImage:=a deepCopy.
  ^resultImage displayAt: bottomPane frame origin
    clippingBox: ((bottomPane frame origin)
  extent:( bottomPane frame extent))!
```

thin

```
  |abc|
 bottomPane clear.
 a:=resultImageA deepCopy.
 b:=resultImageB deepCopy.
 c:=resultImageC deepCopy.
 CursorManager execute change.
 a thinningBy: b and: c.
 CursorManager normal change.
 resultImage:=a deepCopy.
 ^resultImage displayAt: bottomPane frame origin
   clippingBox: ((bottomPane frame origin)
   extent:( bottomPane frame extent))!
```

union

```
  |ab|
 bottomPane clear.
  a:=resultImageA deepCopy.
 b:=resultImageB deepCopy.
 CursorManager execute change.
 a union: b.
 CursorManager normal change.
 resultImage:=a deepCopy.
 ^resultImage displayAt: bottomPane frame origin
   clippingBox: ((bottomPane frame origin)
   extent:( bottomPane frame extent))! !
```

There are some methods which We wrote is in the system classes:

clear

```
   " Instance method of class GraphPane --- To clear the instance pane"
 ^(Image new width:self frame
             width height:self frame height; white)
       displayAt: self frame origin.
```

storeAsBkAndWt: fileName

```
  "Instance method of ColorForm . To store an Colorform as a black and white form. "
  | anImage length aBitmap  aWidth|
  aWidth:=self width.
  ((aWidth\\16)=0 ) ifFalse:[
  aWidth:=aWidth+(16 - (aWidth\\16)) deepCopy].
  length:=aWidth*(self height)/8.
  aBitmap:= Bitmap new:length.
  1 to: length do:[:i|aBitmap at:i put:((( ((bits at: 1 ) at: We )
                                bitAnd: ((bits at: 2) at: i))
                                bitAnd: ((bits at: 3) at: i))
                                bitAnd: ((bits at: 4) at: i))].
  anImage:= Image new width:self width height:self height.
  anImage bitmap: aBitmap.
 anImage storeOnFile:fileName
```

storeColorOnFile: fileName

```
  "Instance method of ColorForm .FL  Apr 14, 1992., 1992 "
  | outFile   length aWidth|
  outFile := Disk newFile: fileName.
  outFile nextPutAll: 'color'.
  outFile nextPut:$ .
  outFile nextTwoBytesPut: (self width)  .
  outFile nextTwoBytesPut: (self height) .
  aWidth:=self width.
  ((aWidth\\16)=0 ) ifFalse:[
  aWidth:=aWidth+(16 - (aWidth\\16)) deepCopy].
  length:=aWidth*(self height)/8.
   1 to: 4 do:[:aBitmap| (bits at: aBitmap) do:
        [:ea | outFile nextPut: ea asCharacter].] .
  outFile close.
```

changeToBkAndWt

```
 "Instance method of ColorForm .Answer an image which is a change from a color form."
   | anImage length aBitmap anArray |
  length:=(self width)*(self height) deepCopy.
  length inspect.
  ((length\\16)=0 ) ifFalse:[
  length:=length+(16 - (length\\16)) deepCopy].
  anArray:= self getBits:length.
  aBitmap:= Bitmap new:length.
  1 to: length do:[:i|aBitmap at:i put:((( ((anArray at: 1 ) at: We )
                                bitOr: ((anArray at: 2) at: i))
                                bitOr: ((anArray at: 3) at: i))
                                bitOr: ((anArray at: 4) at: i))].
   anImage:= Image new width:self width height:self height.
   anImage bitmap: aBitmap.
   ^anImage
```

colorFromFile: fileName

```
   "Class method of class ColorForm . FL 23 Mar. , 1992"
   | inFile aBitmap1 aBitmap2 aBitmap3 aBitmap4 anArray w h anImage wTemp head length|
   inFile := Disk file: fileName.
   inFile reset.
   head:= inFile nextWord.
   (head='color') ifFalse:[^nil].
   inFile next.
   h := inFile next asciiValue.
   wTemp := inFile next asciiValue.
   wTemp := (wTemp bitShift: 8) bitAnd:65280.
   w := wTemp bitOr: h.
   h := inFile next asciiValue.
   wTemp := inFile next asciiValue.
   wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
   h := wTemp bitOr: h.
   anImage := (ColorForm new width: w height: h).
   ((w\\16)=0 ) ifFalse:[
   w:=w+(16 - (w\\16)) deepCopy].
   length:=w*h/8 deepCopy.
   aBitmap1:= Bitmap new:length.
   aBitmap2:= Bitmap new:length.
   aBitmap3:= Bitmap new:length.
   aBitmap4:= Bitmap new:length.
   (1 to: length) do:[ :i | aBitmap1 at: We put: (inFile next asciiValue)].
   (1 to: length) do:[ :i | aBitmap2 at: We put: (inFile next asciiValue)].
   (1 to: length) do:[ :i | aBitmap3 at: We put: (inFile next asciiValue)].
   (1 to: length) do:[ :i | aBitmap4 at: We put: (inFile next asciiValue)].
   anArray:= Array with:aBitmap1
              with:aBitmap2
              with:aBitmap3
              with:aBitmap4.
   anImage bitmap:anArray.
   ^anImage
```