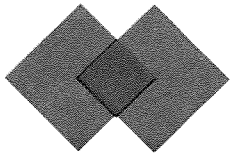

DAP 500

Image Processing Library

(man014.03)

AMT endeavours to ensure that the information in this document is correct, but does not accept responsibility for any error or omission.

Any procedure described in this document for operating AMT equipment should be read and understood by the operator before the equipment is used. To ensure that AMT equipment functions without risk to safety or health, such procedures should be strictly observed by the operator.

The development of AMT products and services is continuous and published information may not be up to date. Any particular issue of a product may contain part only of the facilities described in this document or may contain facilities not described here. It is important to check the current position with AMT.

Specifications and statements as to performance in this document are AMT estimates intended for general guidance. They may require adjustment in particular circumstances and are therefore not formal offers or undertakings.

Statements in this document are not part of a contract or program product licence save in so far as they are incorporated into a contract or licence by express reference. Issue of this document does not entitle the recipient to access to or use of the products described, and such access or use may be subject to separate contracts or licences.

Technical publication man014.03

First Edition Feb 1988
Second Edition November 1988
Third Edition February 1989

AMT filename:/cnm/image/ed3

AMT will be pleased to receive readers' views on the contents, organisation, etc of this publication. Please make contact at either of the addresses below:

Publications Manager
Active Memory Technology Ltd
65 Suttons Park Avenue
Reading
Berks, RG6 1AZ, UK

Tel: 0734 661111

Publications Manager
Active Memory Technology Inc
16802 Aston St Suite 103
Irvine
California, 92714, USA

Tel: (714) 261 8901

Preface

This manual describes the routines in release 2.0 of the Image Processing Library, which have been developed to facilitate image processing on AMT DAP Series machines. The routines are mainly written in FORTRAN-PLUS, and users are assumed to be familiar with this language.

The principles of FORTRAN-PLUS are outlined in:

DAP Series: Introduction to FORTRAN-PLUS (man001)

Detailed information on writing FORTRAN-PLUS programs is supplied in the AMT reference publication:

DAP Series: FORTRAN-PLUS Language (man002)

Developing and running programs on the DAP is described in the following publications:

DAP Series: Program Development under UNIX (man003)

DAP Series: Program Development under VAX/VMX (man004)

This is the third edition of the Image Processing Library.

Contents

Chapter 1

Introduction

1.1 Overview

The Image Processing Library has been designed to facilitate image processing on the AMT DAP. The DAP (standing for 'Distributed Array of Processors') is a massively parallel computer which attaches to a host as a peripheral processor.

The processors in the DAP are arranged in a square matrix; for example 32 x 32 in the case of the DAP 500 range and 64 x 64 in the DAP 600. The number of processors on one side of the square is called the *edge-size* of the DAP. ES is used as an abbreviation for edge-size in this manual.

This manual describes the routines included in the AMT Image Processing Library. The routines are divided into four chapters: Image Conversion Routines, Image Processing Primitives, Low Level Image Processing Routines and Image Analysis Routines, a summary of their contents is listed below. How to access the library is described in section 1.2. This is followed by a list of the routines in alphabetical order in chapter 2.

Chapter 3 Image Conversion Routines

There are three standard ways of storing images in DAP store: raster, sheet and crinkled mappings.

- By *raster images* we mean that the pixels of the image have been assigned a row-major ordering. Raster images are mapped into the DAP store by imposing long-vector (column-major) ordering on each DAP matrix and then column-major ordering on the set of matrices. Each pixel in the image is stored in the corresponding position in DAP store, the first pixel in the first position etc..
- A *sheet mapped image* is divided into tiles – each tile fills one DAP matrix. A set of tiles is stored in column-major ordering in DAP memory.
- A *crinkle mapped image* is conceptually divided into ES x ES tiles. The size of each tile is:

$$\frac{\text{height of the image}}{ES} \times \frac{\text{width of the image}}{ES}$$

Each tile is then 'crinkled' up and stored in column-major order under the corresponding processor element. Thus each DAP matrix holds one pixel value from each tile.

Image conversion routines convert between these three ways of mapping the image in DAP store. The image must be of size $2^M \times 2^N$ where $5 \leq N, M \leq 10$, and may be any (positive) number of bits deep, restricted only by storage considerations. Most of the routines in the following chapters assume that the image is held in DAP store using a crinkled mapping. Conversions between two's complement and unsigned formats are also included, since calculations in the library routines are performed using two's complement arithmetic.

CRINK_RASTER – converts an image from crinkled to raster ordering in DAP store.

CRINK_SHEET – converts an image from crinkled to sheet ordering in DAP store.

RASTER_CRINK – converts an image from raster to crinkled ordering in DAP store.

RASTER_SHEET – converts an image from raster to sheet ordering in DAP store.

SHEET_CRINK – converts an image from sheet to crinkled ordering in DAP store.

SHEET_RASTER – converts an image from sheet to raster ordering in DAP store.

TWO_UNSIG – converts an image from from two's complement to unsigned integer format maintaining the ordering of the grey-scale values.

UNSIG_TWO – converts an image from from unsigned to two's complement integer format maintaining the ordering of the grey-scale values.

Chapter 4 Image Processing Primitives

Image processing primitives are routines for adding, subtracting, and shifting entire images. Routines specified as 8-bits per pixel routines may also be used for 16-bits per pixel images by replacing the 8 in the routine name by 16 (and 16 if present by 32). For example:

MULT_8_TO_16 – multiplies two 8-bit two's complement images to form a 16-bit image.

MULT_16_TO_32 – multiplies two 16-bit two's complement images to form a 32-bit image.

The routines ADD_8, SCAL_ADD_8 and SUB_8 return their results modulo 256. The routines ADD_16, SCAL_ADD_16 and SUB_16 return their results modulo 65536.

Arithmetic operations

ADD_8 – adds together two 8-bit two's complement images. The result is returned modulo 256.

Chapter 4 Image Processing Primitives (continued)

SCAL_ADD_8 – adds an 8-bit two's complement scalar to an 8-bit two's complement image. The result is returned modulo 256.

SUB_8 – subtracts one 8-bit two's complement image from another. The result is returned modulo 256.

MULT_8_TO_16 – multiplies two 8-bit two's complement images to form a 16-bit image.

SCAL_MULT_8_TO_16 – multiplies an 8-bit two's complement image by an 8-bit two's complement scalar to form a 16-bit image.

SCAL_DIV_8 – divides an 8-bit two's complement image by an 8-bit two's complement scalar to form an 8-bit image.

Shifting operations

The shifting routines are for crinkled mapped images only. Shifts north are documented here but shifts to other directions can be made by replacing NORTH in the routine name by SOUTH, EAST or WEST as required.

SHIFT_IMAGE_NORTH_P – shifts an entire image north with planar boundary conditions. Zeroes are introduced at the boundaries.

SHIFT_IMAGE_NORTH_C – shifts an entire image north with cyclic (wrap-around) boundary conditions.

SHIFT_ROW_NORTH_P – shifts a row of matrices of an image north with planar boundary conditions. Zeroes are introduced at the boundaries.

SHIFT_ROW_NORTH_C – shifts a row of matrices of an image north with cyclic (wrap-around) boundary conditions.

SHIFT_COL_NORTH_P – shifts a column of matrices of an image north with planar boundary conditions. Zeroes are introduced at the boundaries.

SHIFT_COL_NORTH_C – shifts a column of matrices of an image north with cyclic (wrap-around) boundary conditions.

SHIFT_SHEET_NORTH_P – shifts a matrix of an image north with planar boundary conditions. Zeroes are introduced at the boundaries.

SHIFT_SHEET_NORTH_C – shifts a matrix of an image north with cyclic (wrap-around) boundary conditions.

Chapter 5 Low Level Image Processing Routines

Low-level image processing routines perform tasks such as edge detection routines, convolutions and Fast Fourier Transforms (FFTs). Routines specified as 8-bits per pixel may be used for 16-bits per pixel images by replacing the 8 in the name by 16, except for the COMP_FFT_2D routines which perform FFTs on images with 8, 16 and 24 bits per pixel.

ABS_THRESH_8 – thresholds an 8-bit two's complement image.

AVERAGE_8 – convolves a square averaging mask with an 8-bit two's complement image.

BOX_IN_BOX_8 – applies box-in-a-box prescreening to an 8-bit per pixel two's complement image. The result is compared with a user supplied threshold and a binary result returned.

C06_FFT_ESS – calculates the two dimensional discrete Fourier transform of ES x ES complex points.

C06_FFT_LV – performs a one dimensional Fast Fourier Transform of ES^2 complex points.

COMP_FFT_2D_8_TO_8 – calculates the two dimensional fast Fourier transform of an 8-bit two's complement complex image producing an 8-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

COMP_FFT_2D_8_TO_16 – calculates the two dimensional fast Fourier transform of an 8-bit two's complement complex image producing a 16-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

COMP_FFT_2D_8_TO_24 – calculates the two dimensional fast Fourier transform of an 8-bit two's complement complex image producing a 24-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

COMP_FFT_2D_16_TO_16 – calculates the two dimensional fast Fourier transform of a 16-bit two's complement complex image producing a 16-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

COMP_FFT_2D_16_TO_24 – calculates the two dimensional fast Fourier transform of a 16-bit two's complement complex image producing a 24-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

Chapter 5 Low Level Image Processing Routines (continued)

COMP_FFT_2D_24_TO_24 – calculates the two dimensional fast Fourier transform of a 24-bit two's complement complex image producing a 24-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

COMP_FFT_2D_REAL_3 – calculates the two dimensional fast Fourier transform of a 24-bit floating point complex image producing a 24-bit floating point complex image. The input data may be scaled; the output transform is scaled automatically.

CONVOLVE_8 – calculates the convolution of an 8-bit per pixel two's complement image with a square mask. The result may be scaled to avoid overflow.

DIFF_OF_GAUSS_8 – convolves one of a set of nine difference-of-Gaussian masks with an 8-bit two's complement image.

F01_G_MM – performs a general matrix multiply of two matrices A and B where A is a P x Q matrix and B is a Q x R matrix with P,Q and R in the range 1 to ES.

F01_M_INV – calculates, in place, the inverse of a given N x N matrix with N in the range 1 to ES.

F04_QR_GIVENS_SOLVE – solves the linear system $Ax = b$ for x where A is an n x n matrix with $2<n<ES+1$. The routine may be used to simultaneously solve for up to ES different right hand side vectors b.

FILL_IN_1 – performs a region growing process on a logical image starting from input seed points. The region growing is constrained by the input image – it can only grow where the image is .TRUE..

HISTOGRAM_C_8 – calculates the histogram for an 8-bit two's complement crinkle mapped image or tiles thereof.

HISTOGRAM_S_8 – calculates the histogram for an 8-bit two's complement sheet mapped image or tiles thereof.

KIRSCH_8 – applies a user-selected Kirsch compass gradient mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

LAPLACE_8 – applies a user-selected Laplacian mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

LINE_DET_8 – applies a user-selected line detection mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

NORMALIZE_8 – expands the range of grey-scale values used within an 8-bit two's complement image to the full range.

Chapter 5 Low Level Image Processing Routines (continued)

PERC_THRESH_8 – performs thresholding on an 8-bit two's complement image.

PREWITT_8 – applies a user-selected Prewitt compass gradient mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

PSEUDO_MEDIAN_8 – calculates the pseudo-median at each pixel in an 8-bit two's complement image for a rectangular neighbourhood centred on the image.

PURE_MEDIAN_8 – calculates the true median at each pixel in an 8-bit two's complement image for a rectangular neighbourhood centred on the image.

ROBERTS_8 – applies a user-selected Roberts edge detection mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

SOBEL_8 – sums the convolution of an 8-bit per pixel two's complement image with the 3 x 3 Sobel vertical and horizontal edge detection masks. The result may be scaled to avoid overflow.

ZERO_X_8 – finds the zero crossings in an 8-bit two's complement image.

Chapter 6 Image Analysis Routines

Image analysis routines allow the interesting features on an image to found and identified automatically. SEGMENT_16 and FEATURES_16 are available for analysing 16-bit images.

SEGMENT_8 – extracts the most 'interesting' blob or related set of blobs from an 8-bit two's complement image by successively applying difference-of-Gaussian convolution operators to the image.

LABEL_16 – labels the distinct 'blobs' in a binary image, returning a 16-bit two's complement image with all the pixels in each blob having one value, different for each blob.

FEATURES_8 – calculates invariant moments and other features for use in classification of 'blobs' of each blob in an 8-bit two's complement labelled image (see LABEL_16).

CLASSIF – calculates the class expectation index (how likely a 'blob' is to be in a given class) for each set of input blob features.

1.2 Access to the library

Routines in the image processing library are linked in at the consolidation stage of the compiling process. For details of compiling and linking see *DAP Series: Program Development Under UNIX*, or *DAP Series: Program Development Under VAX/VMS*.

1.2.1 Using the library under UNIX

Routines are linked in by specifying **iplib** with the **-l** (for library) flag in either **dapa** or **dapf**. For example:

```
dapf -o myfile.dd myfile.df -l iplib
```

dapf will compile the FORTRAN-PLUS program **myfile.df**, linking in any routines from the image processing library which are required and produce a DOF file **myfile.dd**. **dapf** and **dapa** will automatically find the version of the library to match the size of DAP that the program is being produced for.

1.2.2 Using the library under VAX/VMS

You can link routines from the image processing library into a program by using the /LIBRARY qualifier to the DLINK command. Two versions of the image processing library are supplied, IPLIB5 for DAP 500, and IPLIB6 for DAP 600; when linking routines from the image processing library into your program you need to specify the appropriate version of IPLIB.

For example, to compile and link a FORTRAN-PLUS program in the file **PICTURE.DFP** for a DAP 600 you can use the following commands:

```
$ DFORTRAN/DAPSIZE=64 PICTURE
$ DLINK/DAPSIZE=64 PICTURE,SYS$LIBRARY:IPLIB6/LIBRARY
```

To compile and link the FORTRAN-PLUS program in file **IMAGES.DFP** for a DAP 500 the commands would be:

```
$ DFORTRAN/DAPSIZE=32 IMAGES
$ DLINK/DAPSIZE=32 IMAGES,SYS$LIBRARY:IPLIB5/LIBRARY
```

You can define the logical name DAP*n*_LIBRARY by using the command:

```
$ DEFINE DAPn_LIBRARY SYS$LIBRARY:IPLIBn
```

where *n* is 5 (for DAP 500) or 6 (for DAP 600). This will cause DLINK to search IPLIB*n* automatically for unsatisfied external references. If you are going to use IPLIB*n* frequently, you could insert the DEFINE command above into your LOGIN.COM file. If there are several DAP users on the system, linked to a DAP 600 say, the system manager could include the command:

```
$ DEFINE/SYSTEM DAP6_LIBRARY SYS$LIBRARY:IPLIB6.DLB
```

into the site system start-up command file which would give all users automatic access to the library.

Similarly, the command:

```
$ DEFINE/SYSTEM DAP5_LIBRARY SYS$LIBRARY:IPLIB5.DLB
```

would do the same thing for a DAP 500 system.

On a system that has both DAP 500 and DAP 600, then both DAP5_LIBRARY and DAP6_LIBRARY can be defined, and users would pick up the version of IPLIB to match the /DAPSIZE that had been specified.

Chapter 2

Alphabetical listing of routines

Chapter 3

Image Conversion Routines

There are three standard ways of storing images in DAP store: raster, sheet and crinkled mappings.

- By *raster images* we mean that the pixels of the image have been assigned a row-major ordering. Raster images are mapped into the DAP store by imposing long-vector (column-major) ordering on each DAP matrix and then column-major ordering on the set of matrices. Each pixel in the image is stored in the corresponding position in DAP store, the first pixel in the first position etc..
- A *sheet mapped image* is divided into tiles – each ES x ES pixel tile is stored in one DAP matrix. A set of tiles is stored in column-major ordering in DAP memory.
- A *crinkle mapped image* is conceptually divided into ES^2 tiles of size (height $/ES$) x (width $/ES$). Each tile is then 'crinkled' up and stored in column-major order under the corresponding processing element. Thus each DAP matrix holds one pixel value from each tile.

Image conversion routines convert between these three ways of mapping the image in DAP store. The image must be of size 2^M x 2^N where $5 \leq N, M \leq 10$, and may be any (positive) number of bits deep, restricted only by storage considerations. Most of the routines in the following chapters assume that the image is held in DAP store using a crinkled mapping. Conversions between two's complement and unsigned formats are also included, since calculations in the library routines are performed using two's complement arithmetic.

3.1 CRINK_RASTER

1 Purpose

This routine converts an image from crinkled to raster ordering in DAP store.

2 Specification

CRINK_RASTER(IMAGE , BITS , NS_SIZE , WE_SIZE)

LOGICAL IMAGE(, , BITS , NS_SIZE , WE_SIZE)

INTEGER *4 BITS, NS_SIZE, WE_SIZE

3 Description

This routine converts images (in place) from crinkled to raster ordering in DAP store where:

crinkled mapping divides up an image into ES^2 tiles of size (height $/ES$) x (width $/ES$) each of which is then 'crinkled up' and stored under one processing element. Thus each DAP matrix in which the image is stored holds one pixel value from each tile. The number of bits per pixel is specified in BITS. Here DAP matrix refers to BITS consecutive bit planes

raster images are mapped into DAP store by imposing long-vector (column-major) ordering on each DAP matrix of pixel values and then column-major ordering on the set of matrices

4 References

None.

5 Arguments

IMAGE

IMAGE is the image to be remapped into DAP store.

BITS

On entry BITS is the number of bits per pixel in IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

LOG2, X05LOG2

8 Accuracy

Not applicable.

9 Further Comments

This routine uses Parallel Data Transforms.

10 Keywords

Parallel Data Transforms

3.2 CRINK_SHEET

1 Purpose

This routine converts an image from crinkled to sheet ordering in DAP store.

2 Specification

CRINK_SHEET(IMAGE,BITS,NS_SIZE,WE_SIZE)

LOGICAL IMAGE(,, BITS,NS_SIZE,WE_SIZE)

INTEGER *4 BITS, NS_SIZE, WE_SIZE

3 Description

This routine converts images (in place) from crinkled to sheet ordering in DAP store where:

crinkled mapping divides up an image into ES^2 tiles of size (height $/ES$) x (width $/ES$) each tile is then 'crinkled up' and stored under one processing element. Thus each DAP matrix in which the image is stored holds one pixel value from each tile. The number of bits per pixel is specified in BITS. Here DAP matrix refers to BITS consecutive bit planes

sheet mapping divides the image up into tiles of ES x ES pixels, each of which occupies one DAP matrix. The set of DAP matrices are stored in column-major order

4 References

None.

5 Arguments

IMAGE

IMAGE is the image to be remapped into DAP store.

BITS

On entry BITS is the number of bits per pixel in IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

LOG2, X05LOG2

8 Accuracy

Not applicable.

9 Further Comments

This routine uses the Parallel Data Transforms.

10 Keywords

Parallel Data Transforms

3.3 RASTER_CRINK

1 Purpose

This routine converts an image from raster to crinkled ordering in DAP store.

2 Specification

RASTER_CRINK (IMAGE , BITS , NS_SIZE , WE_SIZE)

LOGICAL IMAGE (, , BITS , NS_SIZE , WE_SIZE)

INTEGER *4 BITS, NS_SIZE, WE_SIZE

3 Description

This routine converts images (in place) from raster to crinkled ordering in DAP store where:

raster images are mapped into DAP store by imposing long-vector (column-major) ordering on each DAP matrix of pixel values and then column-major ordering on the set of matrices. Here DAP matrix refers to BITS consecutive bit planes

crinkled mapping divides up an image into ES^2 tiles of size (height $/ES$) x (width $/ES$) each tile is then 'crinkled up' and stored under one processing element. Thus each DAP matrix in which the image is stored holds one pixel value from each tile. The number of bits per pixel is specified in BITS

4 References

None.

5 Arguments

IMAGE

IMAGE is the image to be remapped into DAP store.

BITS

On entry BITS is the number of bits per pixel in IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

LOG2, X05LOG2

8 Accuracy

Not applicable.

9 Further Comments

This routine uses Parallel Data Transforms.

10 Keywords

Parallel Data Transforms

3.4 RASTER_SHEET

1 Purpose

This routine converts an image from raster to sheet ordering in DAP store.

2 Specification

RASTER_SHEET (IMAGE , BITS , NS_SIZE , WE_SIZE)

LOGICAL IMAGE (, , BITS , NS_SIZE , WE_SIZE)

INTEGER *4 BITS, NS_SIZE, WE_SIZE

3 Description

This routine converts images (in place) from raster to sheet ordering in DAP store where:

raster images are mapped into DAP store by imposing long-vector (column-major) ordering on each DAP matrix of pixel values and then column-major ordering on the set of matrices. Here DAP matrix refers to BITS consecutive bit planes

sheet mapping divides the image up into tiles of ES x ES pixels, each tile occupies one DAP matrix. The set of DAP matrices are stored in column-major order

4 References

None.

5 Arguments

IMAGE

IMAGE is the image to be remapped into DAP store.

BITS

On entry BITS is the number of bits per pixel in IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

LOG2, X05LOG2

8 Accuracy

Not applicable.

9 Further Comments

This routine uses Parallel Data Transforms.

10 Keywords

Parallel Data Transforms

3.5 SHEET_CRINK

1 Purpose

This routine converts an image from sheet to crinkled ordering in DAP store.

2 Specification

SHEET_CRINK (IMAGE , BITS , NS_SIZE , WE_SIZE)

LOGICAL IMAGE(, , BITS , NS_SIZE , WE_SIZE)

INTEGER *4 BITS, NS_SIZE, WE_SIZE

3 Description

This routine converts images (in place) from sheet to crinkled ordering in DAP store where:

sheet mapping divides the image up into tiles of ES x ES pixels, each of which occupies one DAP matrix. The set of DAP matrices are stored in column-major order. Here DAP matrix refers to BITS consecutive bit planes

crinkled mapping divides up an image into ES^2 tiles of size (height $/ES$) x (width $/ES$) each of which is then 'crinkled up' and stored under one processing element. Thus matrix in which the image is stored holds one pixel value from each tile. The number of bits per pixel is specified in BITS

4 References

None.

5 Arguments

IMAGE

IMAGE is the image to be remapped into DAP store.

BITS

On entry BITS is the number of bits per pixel in IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

LOG2, X05LOG2

8 Accuracy

Not applicable.

9 Further Comments

This routine uses the Parallel Data Transforms.

10 Keywords

Parallel Data Transforms

3.6 SHEET_RASTER

1 Purpose

This routine converts an image from sheet to raster ordering in DAP store.

2 Specification

SHEET_RASTER(IMAGE, BITS, NS_SIZE, WE_SIZE)

LOGICAL IMAGE(,, BITS, NS_SIZE, WE_SIZE)

INTEGER *4 BITS, NS_SIZE, WE_SIZE

3 Description

This routine converts images (in place) from sheet to raster ordering in DAP store where:

sheet mapping divides the image up into tiles of ES x ES pixels, each tile occupies one DAP matrix. The set of DAP matrices are stored in column-major order. Here DAP matrix refers to BITS consecutive bit planes

raster images are mapped into DAP store by imposing long-vector (column-major) ordering on each DAP matrix of pixel values and then column-major ordering on the set of matrices

4 References

None.

5 Arguments

IMAGE

IMAGE is the image to be remapped into DAP store.

BITS

On entry BITS is the number of bits per pixel in IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be one of 1, 2, 4, 8, 16, 32. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

LOG2, X05LOG2

8 Accuracy

Not applicable.

9 Further Comments

This routine uses the Parallel Data Transforms.

10 Keywords

Data Transformation

3.7 TWO_UNSIG

1 Purpose

This routine converts an image from from two's complement to unsigned integer format maintaining the ordering of the grey-scale values.

2 Specification

TWO_UNSIG(IMAGE,BITS,NS_SIZE,WE_SIZE)

LOGICAL IMAGE(,,BITS,NS_SIZE,WE_SIZE)

INTEGER *4 BITS, NS_SIZE, WE_SIZE

3 Description

This routine converts images from two's complement to unsigned format by flipping the most significant bit of the pixel value.

4 References

None.

5 Arguments

IMAGE

IMAGE is the image to be format converted.

BITS

On entry BITS is the number of bits per pixel in IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Processing

3.8 UNSIG_TWO

1 Purpose

This routine converts an image from from unsigned integer format to two's complement integer format maintaining the ordering of the grey-scale values.

2 Specification

UNSIG_TWO (IMAGE,BITS,NS_SIZE,WE_SIZE)

LOGICAL IMAGE(, , BITS,NS_SIZE,WE_SIZE)

INTEGER *4 BITS, NS_SIZE, WE_SIZE

3 Description

This routine converts images from unsigned to two's complement format by flipping the most significant bit of the pixel value.

4 References

None.

5 Arguments

IMAGE

IMAGE is the image to be format converted.

BITS

On entry BITS is the number of bits per pixel in IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Processing

Chapter 4

Image Processing Primitives

These routines provide image processing primitives such as image shifting and addition of two images. Routines specified as 8-bits per pixel routines may also be used for 16-bits per pixel images by replacing the 8 in the routine name by 16 (and 16 if present by 32). For example:

MULT_8_TO_16	– multiplies two 8-bit two's complement images to form a 16-bit image.
MULT_16_TO_32	– multiplies two 16-bit two's complement images to form a 32-bit image.

The routines ADD_8, SCAL_ADD_8 and SUB_8 return their results modulo 256. The routines ADD_16, SCAL_ADD_16 and SUB_16 return their results modulo 65536.

The shifting routines are for crinkled mapped images only. Shifts north are documented here but shifts to other directions can be made by replacing NORTH in the routine name by SOUTH, EAST or WEST as required.

Arithmetic Operations

4.1 ADD_8

1 Purpose

This routine adds together two 8-bit two's complement images. The result is returned modulo 256.

2 Specification

```
ADD_8(IMAGE1,IMAGE2,NS_SIZE,WE_SIZE,IMAGE_SUM)

INTEGER*1 IMAGE1 (,,NS_SIZE,WE_SIZE),
&         IMAGE2 (,,NS_SIZE,WE_SIZE),
&         IMAGE_SUM (,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE,WE_SIZE
```

3 Description

IMAGE1 is added modulo 256 to IMAGE2 pixel-by-pixel to form IMAGE_SUM

4 References

None.

5 Arguments

IMAGE1

On entry IMAGE1 is one of the 8-bit images to be added together. Each dimension of the image size must be $\geq ES$ and a multiple of ES. IMAGE1 may be overwritten by the sum if desired.

IMAGE2

On entry IMAGE2 is the other image to be added. The dimensions of IMAGE2 must be identical to those of IMAGE1. IMAGE2 may be overwritten by the sum if desired.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the images divided by ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width in pixels of the image divided by ES. Unchanged on exit.

IMAGE_SUM

On exit IMAGE_SUM is the sum of IMAGE1 and IMAGE2 modulo 256.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Processing

4.2 SCAL_ADD_8

1 Purpose

This routine adds an 8-bit two's complement scalar to an 8-bit two's complement image. The result is returned modulo 256.

2 Specification

```
SCAL_ADD_8 ( IMAGE,SCAL,NS_SIZE,WE_SIZE,IMAGE_SUM)

INTEGER*1 IMAGE (,,NS_SIZE,WE_SIZE),
&         SCAL,
&         IMAGE_SUM (,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE,WE_SIZE
```

3 Description

SCAL is added modulo 256 to each pixel of IMAGE to form IMAGE_SUM.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the 8-bit image to be added to a scalar. Each dimension of the image size must be $\geq ES$ and a multiple of ES. IMAGE may be overwritten by the sum if desired.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. Unchanged on exit.

IMAGE_SUM

On exit IMAGE_SUM is the sum of IMAGE and SCAL modulo 256.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Processing

4.3 SUB_8

1 Purpose

This routine subtracts two 8-bit two's complement images. The result is returned modulo 256.

2 Specification

```
SUB_8(IMAGE1,IMAGE2,NS_SIZE,WE_SIZE,IMAGE_DIFF)

INTEGER*1 IMAGE1 (,,NS_SIZE,WE_SIZE),
&         IMAGE2 (,,NS_SIZE,WE_SIZE),
&         IMAGE_DIFF(,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE,WE_SIZE
```

3 Description

IMAGE2 is subtracted modulo 256 from IMAGE1 pixel-by-pixel to form IMAGE_DIFF.

4 References

None.

5 Arguments

IMAGE1

On entry is one of the 8-bit images to be subtracted. Each dimension of the image size must be $\geq ES$ and a multiple of ES. IMAGE1 may be overwritten by the result if desired.

IMAGE2

On entry IMAGE2 is the other image to be subtracted. The dimensions of IMAGE2 must be identical to those of IMAGE1. IMAGE2 may be overwritten by the result if desired.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the images divided by ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the images divided by ES. Unchanged on exit.

IMAGE_DIFF

On exit IMAGE_DIFF is IMAGE1 minus IMAGE2 modulo 256.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Processing

4.4 MULT_8_TO_16

1 Purpose

This routine multiplies two 8-bit two's complement images to form a 16-bit image.

2 Specification

```
MULT_8_TO_16 (IMAGE1,IMAGE2,NS_SIZE,WE_SIZE,IMAGE_PROD)

INTEGER*1 IMAGE1 (,,NS_SIZE,WE_SIZE),
&         IMAGE2 (,,NS_SIZE,WE_SIZE)

INTEGER*2 IMAGE_PROD(,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE,WE_SIZE
```

3 Description

IMAGE1 is multiplied by IMAGE2 pixel-by-pixel to form IMAGE_PROD.

4 References

None.

5 Arguments

IMAGE1

On entry is one of the 8-bit images to be multiplied together. Each dimension of the image size must be $\geq ES$ and a multiple of ES. Unchanged on exit.

IMAGE2

On entry IMAGE2 is the other image to be multiplied. The dimensions of IMAGE2 must be identical to those of IMAGE1. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the images divided by ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the images divided by ES. Unchanged on exit.

IMAGE_PROD

On exit IMAGE_PROD is the product of IMAGE1 and IMAGE2.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Processing

4.5 SCAL_MULT_8_TO_16

1 Purpose

This routine multiplies an 8-bit two's complement image by an 8-bit two's complement scalar to form a 16-bit image.

2 Specification

```
SCAL_MULT_8_TO_16 (IMAGE,SCAL,NS_SIZE,WE_SIZE,IMAGE_PROD)

INTEGER*1 IMAGE (,,NS_SIZE,WE_SIZE),
&         SCAL

INTEGER*2 IMAGE_PROD (,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE,WE_SIZE
```

3 Description

IMAGE is multiplied by SCAL pixel-by-pixel to form IMAGE_PROD.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the 8-bit image to be multiplied by SCAL. Each dimension of the image size must be $\geq ES$ and a multiple of ES. Unchanged on exit.

SCAL

On entry SCAL is the scalar that multiplies IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. Unchanged on exit.

IMAGE_PROD

On exit IMAGE_PROD is the product of IMAGE and SCAL.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Processing

4.6 SCAL_DIV_8

1 Purpose

This routine divides an 8-bit two's complement image by an 8-bit two's complement scalar to form an 8-bit image.

2 Specification

```
SCAL_DIV_8(IMAGE,SCAL,NS_SIZE,WE_SIZE,IMAGE_QUOT)

INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE),
&         SCAL,
&         IMAGE_QUOT(,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE,WE_SIZE
```

3 Description

IMAGE is divided by SCAL pixel-by-pixel to form IMAGE_QUOT.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the 8-bit image to be divided by SCAL. Each dimension of the image size must be $\geq ES$ and a multiple of ES. Unchanged on exit.

SCAL

On entry SCAL is the scalar that divides IMAGE. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. Unchanged on exit.

IMAGE_QUOT

On exit IMAGE_QUOT is the quotient of IMAGE and SCAL.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Processing

Shifts

4.7 SHIFT_IMAGE_NORTH_P

1 Purpose

SHIFT_IMAGE_NORTH_P shifts an entire image north with planar boundary conditions. Zeroes are introduced at the boundaries.

2 Specification

```
      SUBROUTINE SHIFT_IMAGE_NORTH_P (IMAGE, BITS, NS_SIZE, WE_SIZE,
     &                                SHIFT_IMAGE, DIST)

      LOGICAL IMAGE (,,BITS,NS_SIZE,WE_SIZE),
     &        SHIFT_IMAGE (,,BITS,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, DIST, BITS
```

3 Description

All of IMAGE is shifted north by DIST pixels with planar boundary conditions. The precision of IMAGE is specified in BITS.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the image to be shifted. Each dimension of the image size must be in the range ES to $32ES$ and a multiple of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit, unless IMAGE and SHIFT_IMAGE occupy the same area of store.

BITS

On entry BITS is the precision in bits of the image. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of IMAGE in pixels divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of IMAGE in pixels divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SHIFT_IMAGE

On exit SHIFT_IMAGE contains the shifted version of IMAGE. SHIFT_IMAGE may occupy the same area of store as IMAGE. (They must occupy identical or disjoint but not overlapping areas of store.)

DIST

On entry DIST is the distance IMAGE is to be moved. DIST must be non-negative. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

260 stack planes are allocated. $BITS \times DISTmoduloNS\_SIZE$ planes will be needed. For east or west shifts $BITS \times DISTmoduloWE\_SIZE$ planes will be needed. For large shifts to be performed you will need to increase the program's stack allocation (see stack size in *DAP Series: Program Development under UNIX* or *DAP Series: Program Development under VAX/VMS*).

10 Keywords

Shifting, Image Processing

4.8 SHIFT_IMAGE_NORTH_C

1 Purpose

SHIFT_IMAGE_NORTH_C shifts an entire image north with cyclic (wrap-around) boundary conditions.

2 Specification

```
      SUBROUTINE SHIFT_IMAGE_NORTH_C(IMAGE, BITS, NS_SIZE, WE_SIZE,
     &                               SHIFT_IMAGE, DIST)

      LOGICAL IMAGE (,,BITS,NS_SIZE,WE_SIZE),
     &        SHIFT_IMAGE (,,BITS,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, DIST, BITS
```

3 Description

All of IMAGE is shifted north by DIST pixels with cyclic boundary conditions. The precision of IMAGE is specified in BITS.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the image to be shifted. Each dimension of the image size must be in the range ES to $32ES$ and a multiple of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit, unless IMAGE and SHIFT_IMAGE occupy the same area of store.

BITS

On entry BITS is the precision in bits of the image. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of IMAGE divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width in pixels of IMAGE divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SHIFT_IMAGE

On exit SHIFT_IMAGE contains the shifted version of IMAGE. SHIFT_IMAGE may occupy the same area of store as IMAGE. (They must occupy identical or disjoint but not overlapping areas of store.)

DIST

On entry DIST is the distance IMAGE is to be moved. DIST must be non-negative. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

260 stack planes are allocated. $BITS \times DISTmoduloNS\_SIZE$ planes will be needed. For east or west shifts $BITS \times DISTmoduloWE\_SIZE$ planes will be needed. For large shifts to be performed you will need to increase the program's stack allocation (see stack size in *DAP Series: Program Development under UNIX* or *DAP Series: Program Development under VAX/VMS*).

10 Keywords

Shifting, Image Processing

4.9 SHIFT_ROW_NORTH_P

1 Purpose

SHIFT_ROW_NORTH_P shifts a row of matrices of an image north with planar boundary conditions. Zeroes are introduced at the boundaries.

2 Specification

```
      SUBROUTINE SHIFT_ROW_NORTH_P (IMAGE, BITS, NS_SIZE, WE_SIZE,
     &                              SHIFT_ROW, ROW, DIST)

      LOGICAL IMAGE (,,BITS,NS_SIZE,WE_SIZE),
     &        SHIFT_ROW (,,BITS,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, ROW, DIST, BITS
```

3 Description

SHIFT_ROW is the ROWth row of DAP matrices from the version of IMAGE shifted north by DIST pixels with planar boundary conditions. Thus if IMAGE1 were the (imaginary) shifted version of IMAGE, then SHIFT_ROW(,,J)=IMAGE1(,,ROW,J) for J=1,2,...WE_SIZE.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the image to be shifted. Each dimension of the image size must be in the range ES to $32ES$ and a multiple of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are vertical and horizon- tal co-ordinates of each crinkled tile. Unchanged on exit.

BITS

On entry BITS is the precision in bits of the image. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of IMAGE divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width in pixels of IMAGE divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SHIFT_ROW

On exit SHIFT_ROW contains the ROWth row of DAP matrices of the shifted version of IMAGE.

ROW

ROW is the row of the shifted image to be calculated.

DIST

On entry DIST is the distance IMAGE is to be moved. DIST must be non-negative. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

260 stack planes are allocated. $BITS \times DISTmoduloNS\_SIZE$ planes will be needed. For east or west shifts $BITS \times DISTmoduloWE\_SIZE$ planes will be needed. For large shifts to be performed you will need to increase the program's stack allocation (see stack size in *DAP Series: Program Development under UNIX or DAP Series: Program Development under VAX/VMS*).

10 Keywords

Shifting, Image Processing

4.10 SHIFT_ROW_NORTH_C

1 Purpose

SHIFT_ROW_NORTH_C shifts a row of matrices of an image north with cyclic (wrap-around) boundary conditions.

2 Specification

```
SUBROUTINE SHIFT_ROW_NORTH_C(IMAGE, BITS, NS_SIZE, WE_SIZE,
&                            SHIFT_ROW, ROW, DIST)

LOGICAL IMAGE(,,BITS,NS_SIZE,WE_SIZE), SHIFT_ROW(,,BITS,WE_SIZE)

INTEGER*4 NS_SIZE, WE_SIZE, ROW, DIST, BITS
```

3 Description

SHIFT_ROW is the ROWth row of DAP matrices from the version of IMAGE shifted north by DIST pixels with cyclic boundary conditions. Thus if IMAGE1 were the (imaginary) shifted version of IMAGE, then SHIFT_ROW(,,J)=IMAGE1(,,ROW,J) for J=1,2,...WE_SIZE.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the image to be shifted. Each dimension of the image size must be in the range ES to $32ES$ and a multiple of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

BITS

On entry BITS is the precision in bits of the image. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of IMAGE divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width in pixels of IMAGE divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SHIFT_ROW

On exit SHIFT_ROW contains the ROWth row of DAP matrices of the shifted version of IMAGE.

ROW

ROW is the row of the shifted image to be calculated.

DIST

On entry DIST is the distance IMAGE is to be moved. DIST must be non-negative. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

260 stack planes are allocated. $BITS \times DISTmoduloNS\_SIZE$ planes will be needed. For east or west shifts $BITS \times DISTmoduloWE\_SIZE$ planes will be needed. For large shifts to be performed you will need to increase the program's stack allocation (see stack size in *DAP Series: Program Development under UNIX* or *DAP Series: Program Development under VAX/VMS*).

10 Keywords

Shifting, Image Processing

4.11 SHIFT_COL_NORTH_P

1 Purpose

SHIFT_COL_NORTH_P shifts a column of matrices of an image north with planar boundary conditions. Zeroes are introduced at the boundaries.

2 Specification

```
SUBROUTINE SHIFT_COL_NORTH_P (IMAGE, BITS, NS_SIZE, WE_SIZE,
&                             SHIFT_COL, COL, DIST)

LOGICAL IMAGE(,,BITS,NS_SIZE,WE_SIZE), SHIFT_COL(,,BITS,NS_SIZE)

INTEGER*4 NS_SIZE, WE_SIZE, COL, DIST, BITS
```

3 Description

SHIFT_COL is the COLth column of DAP matrices from IMAGE shifted north by DIST pixels with planar boundary conditions. Thus if IMAGE1 were the (imaginary) shifted version of IMAGE, then SHIFT_COL(,,I)=IMAGE1(,,I,COL) for I=1,2,...NS_SIZE.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the image to be shifted. Each dimension of the image size must be in the range ES to $32ES$ and a multiple of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

BITS

On entry BITS is the precision in bits of the image. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of IMAGE divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width in pixels of IMAGE divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SHIFT_COL

On exit SHIFT_COL contains the COLth column of DAP matrices of the shifted version of IMAGE.

COL

COL is the column of the shifted image to be calculated.

DIST

On entry DIST is the distance IMAGE is to be moved. DIST must be non-negative. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

Not available.

8 Accuracy

Not applicable.

9 Further Comments

260 stack planes are allocated. $BITS \times DISTmoduloNS\_SIZE$ planes will be needed. For east or west shifts $BITS \times DISTmoduloWE\_SIZE$ planes will be needed. For large shifts to be performed you will need to increase the program's stack allocation (see stack size in *DAP Series: Program Development under UNIX* or *DAP Series: Program Development under VAX/VMS*).

10 Keywords

Shifting, Image Processing

4.12 SHIFT_COL_NORTH_C

1 Purpose

SHIFT_COL_NORTH_C shifts a column of matrices of an image north with cyclic (wrap-around) boundary conditions.

2 Specification

```
SUBROUTINE SHIFT_COL_NORTH_C (IMAGE, BITS, NS_SIZE, WE_SIZE,
&                             SHIFT_COL, COL, DIST)

LOGICAL IMAGE(,,BITS,NS_SIZE,WE_SIZE), SHIFT_COL(,,BITS,NS_SIZE)

INTEGER*4 NS_SIZE, WE_SIZE, COL, DIST, BITS
```

3 Description

SHIFT_COL is the COL^{th} column of DAP matrices from IMAGE shifted north by DIST pixels with cyclic boundary conditions. Thus if IMAGE1 were the (imaginary) shifted version of IMAGE, then SHIFT_COL(,,I)=IMAGE1(,,I,COL) for I=1,2,...NS_SIZE.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the image to be shifted. Each dimension of the image size must be $\geq ES$ and a multiple of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

BITS

On entry BITS is the precision in bits of the image.Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of IMAGE divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width in pixels of IMAGE divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SHIFT_COL

On exit SHIFT_COL contains the COL^{th} column of DAP matrices of the shifted version of IMAGE.

COL

COL is the column of the shifted image to be calculated.

DIST

On entry DIST is the distance IMAGE is to be moved. DIST must be non-negative. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

260 stack planes are allocated. $BITS \times DISTmoduloNS\_SIZE$ planes will be needed. For east or west shifts $BITS \times DISTmoduloWE\_SIZE$ planes will be needed. For large shifts to be performed you will need to increase the program's stack allocation (see stack size in *DAP Series: Program Development under UNIX* or *DAP Series: Program Development under VAX/VMS*).

10 Keywords

Shifting, Image Processing

4.13 SHIFT_SHEET_NORTH_P

1 Purpose

SHIFT_SHEET_NORTH_P shifts a matrix of an image north with planar boundary conditions. Zeroes are introduced at the boundaries.

2 Specification

```
      SUBROUTINE SHIFT_SHEET_NORTH_P (IMAGE, BITS, NS_SIZE, WE_SIZE,
     &                                SHIFT_SHEET, ROW, COL, DIST)

      LOGICAL IMAGE (,,BITS,NS_SIZE,WE_SIZE), SHIFT_SHEET(,,BITS)

      INTEGER*4 NS_SIZE, WE_SIZE, ROW, COL, DIST, BITS
```

3 Description

SHIFT_SHEET is the (ROWth,COLth) DAP matrix from IMAGE if it were shifted north by DIST pixels with planar boundary conditions. Thus if IMAGE1 were (the imaginary) shifted version of IMAGE, then SHIFT_SHEET(,)=IMAGE1(,,ROW,COL).

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the image to be shifted. Each dimension of the image size must be in the range ES to $32ES$ and a multiple of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

BITS

On entry BITS is the precision in bits of the image. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of IMAGE divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width in pixels of IMAGE divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SHIFT_SHEET

On exit SHIFT_SHEET contains the (ROWth,COLth) DAP matrix of the shifted version of IMAGE.

ROW

ROW is the row of the shifted image to be calculated.

DIST

On entry DIST is the distance IMAGE is to be moved. DIST must be non-negative. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Shifting, Image Processing

4.14 SHIFT_SHEET_NORTH_C

1 Purpose

SHIFT_SHEET_NORTH_C shifts a matrix of an image north with cyclic (wrap-around) boundary conditions.

2 Specification

```
SUBROUTINE SHIFT_SHEET_NORTH_C (IMAGE, BITS, NS_SIZE, WE_SIZE,
&                               SHIFT_SHEET, ROW, COL, DIST)

LOGICAL IMAGE(,,BITS,NS_SIZE,WE_SIZE), SHIFT_SHEET(,,BITS)

INTEGER*4 NS_SIZE, WE_SIZE, ROW, COL, DIST, BITS
```

3 Description

SHIFT_SHEET is the (ROWth,COLth) DAP matrix from IMAGE if it were shifted north by DIST pixels with cyclic boundary conditions. Thus if IMAGE1 were the (imaginary) shifted version of IMAGE, then SHIFT_SHEET(,)=IMAGE1(,,ROW,COL).

4 References

None.

5 Arguments

IMAGE

On entry IMAGE is the image to be shifted. Each dimension of the image size must be in the range ES to $32ES$ and a multiple of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

BITS

On entry BITS is the precision in bits of the image. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of IMAGE divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width in pixels of IMAGE divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SHIFT_SHEET

On exit SHIFT_SHEET contains the (ROWth,COLth) DAP matrix of the shifted version of IMAGE.

ROW

ROW is the row of the shifted image to be calculated.

DIST

On entry DIST is the distance IMAGE is to be moved. DIST must be non-negative. Unchanged on exit.

6 Errors

No explicit error checking is done.

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Shifting, Image Processing

Chapter 5

Low Level Image Processing Routines

These routines perform low level image processing tasks such as convolutions and FFT's. Routines specified as 8-bits per pixel may be used for 16-bits per pixel images by replacing the 8 in the name by 16. For example:

AVERAGE_8 – convolves a square averaging mask with an 8-bit two's complement image.

AVERAGE_16 – convolves a square averaging mask with a 16-bit two's complement image.

The COMP_FFT_2D routines which perform FFTs on images with 8, 16 and 24 bits per pixel, are listed in full.

5.1 ABS_THRESH_8

1 Purpose

ABS_THRESH_8 thresholds an 8-bit two's complement image.

2 Specification

```
      SUBROUTINE ABS_THRESH_8 (IMAGE, NS_SIZE, WE_SIZE, THRESH,
     &                         HIGH_TIDE, FLIP, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, HIGH_TIDE, IFAIL

      LOGICAL THRESH(,,NS_SIZE, WE_SIZE), FLIP
```

3 Description

THRESH is a logical bit-map of the image which has .TRUE. values where the IMAGE grey-scale values exceed HIGH_TIDE and .FALSE. otherwise. (This is reversed if FLIP is .FALSE. instead of .TRUE.)

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be thresholded. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1-32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1-32. Unchanged on exit.

THRESH

On exit THRESH contains a logical bit-map of from the thresholding of IMAGE. THRESH stores the bit-map using a crinkled mapping (see IMAGE above).

HIGH_TIDE

HIGH_TIDE is the threshold with which IMAGE is compared.

FLIP

If FLIP is .TRUE., THRESH is .TRUE. when IMAGE is greater than HIGH_TIDE. If FLIP is .FALSE., THRESH is .FALSE. when IMAGE is greater than HIGH_TIDE.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below.)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=13	HIGH_TIDE out of range −128 to 127

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Thresholding, Image Processing

5.2 AVERAGE_8

1 Purpose

AVERAGE_8 convolves a square averaging mask with an 8-bit two's complement image.

2 Specification

```
      SUBROUTINE AVERAGE_8 (IMAGE,NS_SIZE, WE_SIZE, AVG,
     &                      MASK_SIZE, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE), AVG(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, MASK_SIZE, IFAIL
```

3 Description

AVG is the convolution of IMAGE and a centred square averaging mask. IMAGE is assumed to be extended by a field of zeros for the convolution.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved with the averaging mask. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile.

IMAGE is overwritten.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1-32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1-32. Unchanged on exit.

AVG

On exit AVG contains the convolution of IMAGE with an averaging mask. AVG stores the convolution using a crinkled mapping (see IMAGE above).

MASK_SIZE

On entry MASK_SIZE must be one of 3, 5, 7,...,31. This is length of the side of the averaging mask. Unchanged on exit.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below.)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	MASK_SIZE invalid

7 Auxiliary Routines

Image Processing Library routines SHIFT_COL_WEST_P, SHIFT_COL_EAST_P, SHIFT_ROW_WEST_P and SHIFT_ROW_EAST_P are called.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Convolution, Local Averaging, Image Processing

5.3 BOX_IN_BOX_8

1 Purpose

BOX_IN_BOX_8 applies box-in-a-box prescreening to an 8-bit per pixel two's complement image. The result is compared with a user supplied threshold and a binary result returned.

2 Specification

```
      SUBROUTINE BOX_IN_BOX_8 (IMAGE, NS_SIZE, WE_SIZE, BOX,
     &                         BOX_SIZE, THRESH, W1, W2, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE),
     &          W1(,,NS_SIZE,WE_SIZE),
     &          W2(,,NS_SIZE,WE_SIZE)

      LOGICAL BOX(,,NS_SIZE, WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, BOX_SIZE, THRESH, IFAIL
```

3 Description

For each pixel in IMAGE the grey-scale values in a square box centred on the pixel are averaged. An average of the grey-scale values is also calculated for a box of approximately one-half the area of the first box. The average over the large box is subtracted from the average over the small box and the result is compared with THRESH. If the result is greater than THRESH, BOX will be .TRUE otherwise BOX will be .FALSE..

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be pre-screened. Each dimension of IMAGE must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile.

IMAGE is overwritten.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

BOX

On exit BOX contains the result the thresholding operation stored using a crinkled mapping.

BOX_SIZE

On entry BOX_SIZE is the size of the square outer box in pixels. It must be one of 5, 7, 9,...,31. Unchanged on exit.

THRESH

On entry THRESH is the threshold with which the difference of the two averages is compared. Unchanged on exit.

W1

Workspace.

W2

Workspace.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	BOX_SIZE out of range

7 Auxiliary Routines

The Image Processing Library routines AVERAGE_8, SHIFT_COL_WEST_P, SHIFT_COL_EAST_P, SHIFT_ROW_WEST_P and SHIFT_ROW_EAST_P are called.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Prescreening, Image Processing

5.4 C06_FFT_ESS

1 Purpose

C06_FFT_ESS calculates the two dimensional discrete Fourier transform of ES x ES complex points.

2 Specification

SUBROUTINE C06_FFT_ESS(X , Y , INVERS , FIRST)

REAL X(,) , Y(,)

LOGICAL INVERS , FIRST

3 Description

The two dimensional transform is calculated by performing independent sets of row and column ES-point transforms.

The data is then in bit reversed order independently in rows and columns, and a final shuffle is performed to reorder the data.

For a description of the general theory of FFTs see [1].

4 References

[1] BRIGHAM E.O.
The Fast Fourier Transform.
Prentice-Hall, 1974

5 Arguments

X – REAL MATRIX

On entry X contains the real part of the data to be transformed. On exit X contains the real part of the transformed data.

Y – REAL MATRIX

On entry Y contains the imaginary part of the data to be transformed. On exit Y contains the imaginary part of the transformed data.

INVERS – LOGICAL

If INVERS is set to .FALSE. the transform:

$$X_{jk} + iY_{jk} = \sum_m \sum_n (A_{mn} + iB_{mn}) exp 2\pi i \frac{(j-1)(m-1)}{ES} + \frac{(k-1)(n-1)}{ES}$$

is calculated, where $j = 1, 2, \ldots, ES$; $k = 1, 2, \ldots, ES$ and the summations are also over $m = 1, 2, \ldots, ES$ and $n = 1, 2, \ldots, ES$.

If INVERS is set to .TRUE. the transform:

$$A_{mn} + iB_{mn} = \sum_j \sum_k (X_{jk} + Y_{jk}) exp{-2\pi i} \frac{(m-1)(j-1)}{ES} + \frac{(n-1)(k-1)}{ES}$$

is calculated, where $m = 1, 2, \ldots, ES$; $n = 1, 2, \ldots, ES$ and the summations are also over $j = 1, 2, \ldots, ES$ and $k = 1, 2, \ldots, ES$.

FIRST – LOGICAL

If FIRST is set to .TRUE. the exponential coefficients for the transform are calculated. Consequently FIRST must be set to .TRUE. the first time this routine is called within a program, but may be set to .FALSE. for all subsequent calls.

6 Errors

None.

7 Auxiliary Routines

This routine calls the DAP library routines Z_C06_F2DCOEFF, Z_C06_ROWFFT, Z_C06_COLFFT and Z_C06_F2DBREV.

8 Accuracy

Accuracy will be data dependent. Some indication of the accuracy may be obtained by performing a subsequent inverse transform and comparing the results with the original data.

9 Further Comments

This routine uses a common block with the name CC06FFTESSQ. Consequently the user program must not use a common block with this name.

10 Keywords

Fast Fourier Transform

11 Example

Not available.

5.5 C06_FFT_LV

1 Purpose

C06_FFT_LV performs a one dimensional finite Fourier transform of ES^2 complex points.

2 Specification

SUBROUTINE C06_FFT_LV (X , Y , INVERS , FIRST)

REAL X(,) , Y(,)

LOGICAL INVERS , FIRST

3 Description

The data is considered as ES^2 complex points in long vector order, and the transform is calculated by performing linked row and column transforms. The first step is to calculate ES-point transforms along each row of complex data. The results of the row transforms are multiplied by a second set of exponential factors and then ES-point transforms are calculated along each column in a similar way to the row transforms but using different exponential factors. The exponential factors are set up in such a way as to ensure that the row and column transforms are linked correctly to give the required one dimensional transform. The final step reorders the data which is in bit reversed order.

For a description of the general theory of FFTs see [1].

4 References

[1] BRIGHAM E.O.
The Fast Fourier Transform.
Prentice-Hall, 1974

5 Arguments

X – REAL MATRIX

On entry X contains the real part of the data to be transformed. On exit X contains the transformed real part of the data.

Y – REAL MATRIX

On entry Y contains the imaginary part of the data to be transformed. On exit Y contains the transformed imaginary part of the data.

INVERS - LOGICAL

If INVERS is set to .FALSE. the transform

$$X_j + iY_j = \sum_{k=1}^{ES^2} (A_k + iB_k) exp(2\pi \frac{i(j-1)(k-1)}{ES^2})$$

is calculated, where j = 1, 2, ... , ES^2 and the summation is over k = 1, 2, ... ,ES^2.

If INVERS is set to .TRUE. the transform

$$A_k + iB_k = \sum_{j=1}^{ES^2} (X_j + iY_j) exp(-2\pi \frac{i(j-1)(k-1)}{ES^2})$$

is calculated,where k = 1, 2, ... , ES^2 and the summation is over j = 1, 2, ... , ES^2.

The argument is unchanged on exit.

FIRST – LOGICAL

If FIRST is set to .TRUE. the exponential coefficients for the transform are calculated. Consequently FIRST must be set to .TRUE. the first time this routine is called within a program,but may be set to .FALSE. for all subsequent calls.

The argument is unchanged on exit.

6 Error Indicators

None.

7 Auxiliary Routines

The routine calls the DAP library routines Z_C06FFT1DCOEFF, Z_C06ROWFFT, Z_C06COLFFT, Z_C06FFT1DBREV.

8 Accuracy

Accuracy will be data dependent. You can get some idea of the accuracy by carrying out the transform, then carrying out the inverse transform and comparing the results with the original data.

9 Further Comments

The routine uses a common block with name CC06FFTLV. Consequently the your program must not use a common block with this name.

10 Keywords

Fast Fourier Transform

11 Example

Not available.

5.6 COMP_FFT_2D_8_TO_8

1 Purpose

COMP_FFT_2D_8_TO_8 calculates the two dimensional fast Fourier transform of an 8-bit two's complement complex image producing an 8-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

2 Specification

```
      SUBROUTINE COMP_FFT_2D_8_TO_8 (DATA_IN, DATA_OUT, WORK,
     &                               NS_SIZE, WE_SIZE,
     &                               SCALE_FACTOR, INVERS, IFAIL)

      INTEGER*1 DATA_IN (,,NS_SIZE,WE_SIZE,2)
     &          DATA_OUT (,,NS_SIZE,WE_SIZE,2)

      REAL*3 WORK (,,NS_SIZE,WE_SIZE,2)

      INTEGER*4 NS_SIZE, WE_SIZE, SCALE_FACTOR, IFAIL

      LOGICAL INVERS
```

3 Description

The two dimensional transform is calculated by performing independent row and column transforms on the image. The forward Fourier transform is defined to be:

$$\tilde{X}(k_1,k_2) = \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1} exp\left\{\frac{-2\pi i}{N_1 N_2}(k_1 j_1 + k_2 j_2)\right\} X(j_1,j_2)$$

where:

$N_1 = NS\_SIZE \times ES$
$N_2 = WE\_SIZE \times ES$

If INVERS is .TRUE. then the complex conjugate of the exponential is used. Normalisation is not performed for either the forward or the inverse transform.

4 References

None.

5 Arguments

DATA_IN

On entry DATA_IN contains the image to be transformed. The dimensions of the image must be specified in NS_SIZE and WE_SIZE. The image is assumed to be stored in DATA_IN using a crinkled mapping where the third and fourth co-ordinates of DATA_IN are the vertical and horizontal co-ordinates of each crinkled tile. The real part of the image is stored in the first half of DATA_IN, and the imaginary part in the second half. Unchanged on exit, unless DATA_IN has been EQUIVALENCEd to DATA_OUT or WORK.

DATA_OUT

On exit DATA_OUT contains the two dimensional Fourier transform of DATA_IN.

WORK

Used as work space.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image, divided by ES. NS_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor by which the input data has been scaled. (positive if multiplied by; negative if divided by). On exit SCALE_FACTOR contains the factor by which the result has been scaled.

INVERS

If INVERS is set to .TRUE. the inverse discrete Fourier transform is performed. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

This routine calls the Image Processing Library subroutines COMP_FFT_2D_REAL_3, X05LOG2 and X05PI, and the parallel data transform library is also used.

8 Accuracy

Not available.

9 Further Comments

The transform is calculated in place; DATA_IN, DATA_OUT and WORK may all occupy disjoint areas of DAP store or any two may be EQUIVALENCEd or all three may be EQUIVALENCEd to start at the same address. For example:

```
EQUIVALENCE (DATA_IN,DATA_OUT)
EQUIVALENCE (DATA_IN,DATA_OUT,WORK)
```

would both work. But

```
EQUIVALENCE (DATA_IN,WORK),(DATA_OUT,WORK(,,NS_SIZE,3))
```

would not work.

10 Keywords

Fast Fourier Transform, Image Processing

5.7 COMP_FFT_2D_8_TO_16

1 Purpose

COMP_FFT_2D_8_TO_16 calculates the two dimensional fast Fourier transform of an 8-bit two's complement complex image producing a 16-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

2 Specification

```
SUBROUTINE COMP_FFT_2D_8_TO_16 (DATA_IN, DATA_OUT, WORK,
&                               NS_SIZE, WE_SIZE,
&                               SCALE_FACTOR, INVERS, IFAIL)

INTEGER*1 DATA_IN(,,NS_SIZE,WE_SIZE,2)

INTEGER*2 DATA_OUT(,,NS_SIZE,WE_SIZE,2)

REAL*3 WORK(,,NS_SIZE,WE_SIZE,2)

INTEGER*4 NS_SIZE, WE_SIZE, SCALE_FACTOR, IFAIL

LOGICAL INVERS
```

3 Description

The two dimensional transform is calculated by performing independent row and column transforms on the image. The forward Fourier transform is defined to be:

$$\tilde{X}(k_1,k_2) = \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1} exp\left\{\frac{-2\pi i}{N_1N_2}(k_1j_1 + k_2j_2)\right\} X(j_1,j_2)$$

where:

$N_1 = NS\_SIZE \times ES$
$N_2 = WE\_SIZE \times ES$

If INVERS is .TRUE. then the complex conjugate of the exponential is used. Normalisation is not performed for either the forward or the inverse transform.

4 References

None.

5 Arguments

DATA_IN

On entry DATA_IN contains the image to be transformed. The dimensions of the image must be specified in NS_SIZE and WE_SIZE. The image is assumed to be stored in DATA_IN using a crinkled mapping where the third and fourth co-ordinates of DATA_IN are the vertical and horizontal co-ordinates of each crinkled tile. The real part of the image is stored in the first half of DATA_IN, and the imaginary part in the second half. Unchanged on exit, unless DATA_IN has been EQUIVALENCEd to DATA_OUT or WORK.

DATA_OUT

On exit DATA_OUT contains the two dimensional Fourier transform of DATA_IN.

WORK

Used as work space.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor by which the input data has been scaled. (positive if multiplied by; negative if divided by). On exit SCALE_FACTOR contains the factor by which the result has been scaled.

INVERS

If INVERS is set to .TRUE. the inverse discrete Fourier transform is performed. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

This routine calls the Image Processing Library subroutines COMP_FFT_2D_REAL_3, X05LOG2 and X05PI, and the parallel data transform library is also used.

8 Accuracy

Not available.

9 Further Comments

The transform is calculated in place; DATA_IN, DATA_OUT and WORK may all occupy disjoint areas of DAP store or any two may be EQUIVALENCEd or all three may be EQUIVALENCEd to start at the same address. For example:

```
EQUIVALENCE (DATA_IN,DATA_OUT)
EQUIVALENCE (DATA_IN,DATA_OUT,WORK)
```

would both work. But

```
EQUIVALENCE (DATA_IN,WORK),(DATA_OUT,WORK(,,NS_SIZE,3))
```

would not work.

10 Keywords

Fast Fourier Transform, Image Processing

5.8 COMP_FFT_2D_8_TO_24

1 Purpose

COMP_FFT_2D_8_TO_24 calculates the two dimensional fast Fourier transform of an 8-bit two's complement complex image producing a 24-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

2 Specification

```
      SUBROUTINE COMP_FFT_2D_8_TO_24 (DATA_IN, DATA_OUT, WORK,
     &                                NS_SIZE, WE_SIZE,
     &                                SCALE_FACTOR, INVERS, IFAIL)

      INTEGER*1 DATA_IN (,,NS_SIZE,WE_SIZE,2)

      INTEGER*3 DATA_OUT (,,NS_SIZE,WE_SIZE,2)

      REAL*3 WORK (,,NS_SIZE,WE_SIZE,2)

      INTEGER*4 NS_SIZE, WE_SIZE, SCALE_FACTOR, IFAIL

      LOGICAL INVERS
```

3 Description

The two dimensional transform is calculated by performing independent row and column transforms on the image.The forward Fourier transform is defined to be:

$$\tilde{X}(k_1,k_2) = \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1} exp\left\{\frac{-2\pi i}{N_1 N_2}(k_1 j_1 + k_2 j_2)\right\} X(j_1,j_2)$$

where:

$N_1 = NS\_SIZE \times ES$
$N_2 = WE\_SIZE \times ES$

If INVERS is .TRUE. then the complex conjugate of the exponential is used. Normalisation is not performed for either the forward or the inverse transform.

4 References

None.

5 Arguments

DATA_IN

On entry DATA_IN contains the image to be transformed. The dimensions of the image must be specified in NS_SIZE and WE_SIZE. The image is assumed to be stored in DATA_IN using a crinkled mapping where the third and fourth co-ordinates of DATA_IN are the vertical and horizontal co-ordinates of each crinkled tile. The real part of the image is stored in the first half of DATA_IN, and the imaginary part in the second half. Unchanged on exit, unless DATA_IN has been EQUIVALENCEd to DATA_OUT or WORK.

DATA_OUT

On exit DATA_OUT contains the two dimensional Fourier transform of DATA_IN.

WORK

Used as work space.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor by which the input data has been scaled. (positive if multiplied by; negative if divided by). On exit SCALE_FACTOR contains the factor by which the result has been scaled.

INVERS

If INVERS is set to .TRUE. the inverse discrete Fourier transform is performed. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

This routine calls the Image Processing Library subroutines COMP_FFT_2D_REAL_3, X05LOG2 and X05PI, and the parallel data transform library is also used.

8 Accuracy

Not available.

9 Further Comments

The transform is calculated in place; DATA_IN, DATA_OUT and WORK may all occupy disjoint areas of DAP store or any two may be EQUIVALENCEd or all three may be EQUIVALENCEd to start at the same address. For example:

```
EQUIVALENCE (DATA_IN,DATA_OUT)
EQUIVALENCE (DATA_IN,DATA_OUT,WORK)
```

would both work. But

```
EQUIVALENCE (DATA_IN,WORK),(DATA_OUT,WORK(,,NS_SIZE,3))
```

would not work.

10 Keywords

Fast Fourier Transform, Image Processing

5.9 COMP_FFT_2D_16_TO_16

1 Purpose

COMP_FFT_2D_16_TO_16 calculates the two dimensional fast Fourier transform of a 16-bit two's complement complex image producing a 16-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

2 Specification

```
SUBROUTINE COMP_FFT_2D_16_TO_16 (DATA_IN, DATA_OUT, WORK,
&                                NS_SIZE, WE_SIZE,
&                                SCALE_FACTOR, INVERS, IFAIL)

INTEGER*2 DATA_IN (,,NS_SIZE,WE_SIZE,2)
&         DATA_OUT (,,NS_SIZE,WE_SIZE,2)

REAL*3 WORK (,,NS_SIZE,WE_SIZE,2)

INTEGER*4 NS_SIZE, WE_SIZE, SCALE_FACTOR, IFAIL

LOGICAL INVERS
```

3 Description

The two dimensional transform is calculated by performing independent row and column transforms on the image.The forward Fourier transform is defined to be:

$$\tilde{X}(k_1,k_2) = \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1} exp\left\{\frac{-2\pi i}{N_1N_2}(k_1j_1 + k_2j_2)\right\} X(j_1,j_2)$$

where:

$N_1 = NS\_SIZE \times ES$
$N_2 = WE\_SIZE \times ES$

If INVERS is .TRUE. then the complex conjugate of the exponential is used. Normalisation is not performed for either the forward or the inverse transform.

4 References

None.

5 Arguments

DATA_IN

On entry DATA_IN contains the image to be transformed. The dimensions of the image must be specified in NS_SIZE and WE_SIZE. The image is assumed to be stored in DATA_IN using a crinkled mapping where the third and fourth co-ordinates of DATA_IN are the vertical and horizontal co-ordinates of each crinkled tile. The real part of the image is stored in the first half of DATA_IN, and the imaginary part in the second half. Unchanged on exit, unless DATA_IN has been EQUIVALENCEd to DATA_OUT or WORK.

DATA_OUT

On exit DATA_OUT contains the two dimensional Fourier transform of DATA_IN.

WORK

Used as work space.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor by which the input data has been scaled. (positive if multiplied by; negative if divided by). On exit SCALE_FACTOR contains the factor by which the result has been scaled.

INVERS

If INVERS is set to .TRUE. the inverse discrete Fourier transform is performed. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

This routine calls the Image Processing Library subroutines COMP_FFT_2D_REAL_3, X05LOG2 and X05PI, and the parallel data transform library is also used.

8 Accuracy

Not available.

9 Further Comments

The transform is calculated in place; DATA_IN, DATA_OUT and WORK may all occupy disjoint areas of DAP store or any two may be EQUIVALENCEd or all three may be EQUIVALENCEd to start at the same address. For example:

```
EQUIVALENCE (DATA_IN,DATA_OUT)
EQUIVALENCE (DATA_IN,DATA_OUT,WORK)
```

would both work. But

```
EQUIVALENCE (DATA_IN,WORK),(DATA_OUT,WORK(,,NS_SIZE,3))
```

would not work.

10 Keywords

Fast Fourier Transform, Image Processing

5.10 COMP_FFT_2D_16_TO_24

1 Purpose

COMP_FFT_2D_16_TO_24 calculates the two dimensional fast Fourier transform of a 16-bit two's complement complex image producing a 24-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

2 Specification

```
SUBROUTINE COMP_FFT_2D_16_TO_24(DATA_IN, DATA_OUT, WORK,
&                               NS_SIZE, WE_SIZE,
&                               SCALE_FACTOR, INVERS, IFAIL)

INTEGER*2 DATA_IN(,,NS_SIZE,WE_SIZE,2)

INTEGER*3 DATA_OUT(,,NS_SIZE,WE_SIZE,2)

REAL*3 WORK(,,NS_SIZE,WE_SIZE,2)

INTEGER*4 NS_SIZE, WE_SIZE, SCALE_FACTOR, IFAIL

LOGICAL INVERS
```

3 Description

The two dimensional transform is calculated by performing independent row and column transforms on the image. The forward Fourier transform is defined to be:

$$\tilde{X}(k_1,k_2) = \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1} exp\left\{\frac{-2\pi i}{N_1N_2}(k_1j_1 + k_2j_2)\right\} X(j_1,j_2)$$

where:

$N_1 = NS\_SIZE \times ES$
$N_2 = WE\_SIZE \times ES$

If INVERS is .TRUE. then the complex conjugate of the exponential is used. Normalisation is not performed for either the forward or the inverse transform.

4 References

None.

5 Arguments

DATA_IN

On entry DATA_IN contains the image to be transformed. The dimensions of the image must be specified in NS_SIZE and WE_SIZE. The image is assumed to be stored in DATA_IN using a crinkled mapping where the third and fourth co-ordinates of DATA_IN are the vertical and horizontal co-ordinates of each crinkled tile. The real part of the image is stored in the first half of DATA_IN, and the imaginary part in the second half. Unchanged on exit, unless DATA_IN has been EQUIVALENCEd to DATA_OUT or WORK.

DATA_OUT

On exit DATA_OUT contains the two dimensional Fourier transform of DATA_IN.

WORK

Used as work space.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor by which the input data has been scaled. (positive if multiplied by; negative if divided by). On exit SCALE_FACTOR contains the factor by which the result has been scaled.

INVERS

If INVERS is set to .TRUE. the inverse discrete Fourier transform is performed. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

This routine calls the Image Processing Library subroutines COMP_FFT_2D_REAL_3, X05LOG2 and X05PI, and the parallel data transform library is also used.

8 Accuracy

Not available.

9 Further Comments

The transform is calculated in place; DATA_IN, DATA_OUT and WORK may all occupy disjoint areas of DAP store or any two may be EQUIVALENCEd or all three may be EQUIVALENCEd to start at the same address. For example:

```
EQUIVALENCE (DATA_IN,DATA_OUT)
EQUIVALENCE (DATA_IN,DATA_OUT,WORK)
```

would both work. But

```
EQUIVALENCE (DATA_IN,WORK),(DATA_OUT,WORK(,,NS_SIZE,3))
```

would not work.

10 Keywords

Fast Fourier Transform, Image Processing

5.11 COMP_FFT_2D_24_TO_24

1 Purpose

COMP_FFT_2D_24_TO_24 calculates the two dimensional fast Fourier transform of a 24-bit two's complement complex image producing a 24-bit two's complement complex image. The input data may be scaled; the output transform is scaled automatically.

2 Specification

```
      SUBROUTINE COMP_FFT_2D_24_TO_24(DATA_IN, DATA_OUT, WORK,
     &                                NS_SIZE, WE_SIZE,
     &                                SCALE_FACTOR, INVERS, IFAIL)

      INTEGER*3 DATA_IN(,,NS_SIZE,WE_SIZE,2)
     &          DATA_OUT(,,NS_SIZE,WE_SIZE,2)

      REAL*3 WORK(,,NS_SIZE,WE_SIZE,2)

      INTEGER*4 NS_SIZE, WE_SIZE, SCALE_FACTOR, IFAIL

      LOGICAL INVERS
```

3 Description

The two dimensional transform is calculated by performing independent row and column transforms on the image. The forward Fourier transform is defined to be:

$$\tilde{X}(k_1,k_2) = \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1} exp\left\{\frac{-2\pi i}{N_1 N_2}(k_1 j_1 + k_2 j_2)\right\} X(j_1,j_2)$$

where:

$N_1 = NS\_SIZE \times ES$
$N_2 = WE\_SIZE \times ES$

If INVERS is .TRUE. then the complex conjugate of the exponential is used. Normalisation is not performed for either the forward or the inverse transform.

4 References

None.

5 Arguments

DATA_IN

On entry DATA_IN contains the image to be transformed. The dimensions of the image must be specified in NS_SIZE and WE_SIZE. The image is assumed to be stored in DATA_IN using a crinkled mapping where the third and fourth co-ordinates of DATA_IN are the vertical and horizontal co-ordinates of each crinkled tile. The real part of the image is stored in the first half of DATA_IN, and the imaginary part in the second half. Unchanged on exit, unless DATA_IN has been EQUIVALENCEd to DATA_OUT or WORK.

DATA_OUT

On exit DATA_OUT contains the two dimensional Fourier transform of DATA_IN.

WORK

Used as work space.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor by which the input data has been scaled. (positive if multiplied by; negative if divided by). On exit SCALE_FACTOR contains the factor by which the result has been scaled.

INVERS

If INVERS is set to .TRUE. the inverse discrete Fourier transform is performed. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

This routine calls the Image Processing Library subroutines COMP_FFT_2D_REAL_3, X05LOG2 and X05PI, and the parallel data transform library is also used.

8 Accuracy

Not available.

9 Further Comments

The transform is calculated in place; DATA_IN, DATA_OUT and WORK may all occupy disjoint areas of DAP store or any two may be EQUIVALENCEd or all three may be EQUIVALENCEd to start at the same address. For example:

```
EQUIVALENCE (DATA_IN,DATA_OUT)
EQUIVALENCE (DATA_IN,DATA_OUT,WORK)
```

would both work. But

```
EQUIVALENCE (DATA_IN,WORK),(DATA_OUT,WORK(,,NS_SIZE,3))
```

would not work.

10 Keywords

Fast Fourier Transform, Image Processing

5.12 COMP_FFT_2D_REAL_3

1 Purpose

COMP_FFT_2D_REAL_3 calculates the two dimensional fast Fourier transform of a 24-bit floating point complex image producing a 24-bit floating-point complex image. The input data may be scaled; the output transform is scaled automatically.

2 Specification

```
SUBROUTINE COMP_FFT_2D_REAL_3 (DATA, NS_SIZE, WE_SIZE,
&                              INVERS, IFAIL)

REAL*3 DATA(,,NS_SIZE,WE_SIZE,2)

INTEGER*4 NS_SIZE, WE_SIZE, SCALE_FACTOR, IFAIL

LOGICAL INVERS
```

3 Description

The two dimensional transform is calculated by performing independent row and column transforms on the image. The forward Fourier transform is defined to be:

$$\tilde{X}(k_1,k_2) = \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1} exp\left\{\frac{-2\pi i}{N_1 N_2}(k_1 j_1 + k_2 j_2)\right\} X(j_1,j_2)$$

where:

$N_1 = NS\_SIZE \times ES$
$N_2 = WE\_SIZE \times ES$

If INVERS is .TRUE. then the complex conjugate of the exponential is used. Normalisation is not performed for either the forward or the inverse transform.

4 References

None.

5 Arguments

DATA

On entry DATA contains the image to be transformed. The dimensions of the image must be specified in NS_SIZE and WE_SIZE. The image is assumed to be stored using a crinkled mapping where the third and fourth co-ordinates of DATA are the vertical and horizontal co-ordinates of each crinkled tile. The real part of the image is stored in the first half of DATA, and the imaginary part in the second half. On exit, DATA contains the transformed image.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32 and a power of two. Unchanged on exit.

INVERS

If INVERS is set to .TRUE. the inverse discrete Fourier transform is performed. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

This routine calls the Image Processing Library subroutines X05LOG2 and X05PI, and the parallel data transform library is also used.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Fast Fourier Transform, Image Processing

5.13 CONVOLVE_8

1 Purpose

CONVOLVE_8 calculates the convolution of an 8-bit per pixel two's complement image with a square mask. The result may be scaled to avoid overflow.

2 Specification

```
      SUBROUTINE CONVOLVE8(IMAGE,NS_SIZE, WE_SIZE, CONV,
     &                     SCALE_FACTOR,MASK_SIZE,MASK,IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE), CONV(,,NS_SIZE,WE_SIZE)
     &          MASK(MASK_SIZE,MASK_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, MASK_SIZE, SCALE_FACTOR, IFAIL
```

3 Description

CONVOLVE_8 calculates the convolution of a square user-supplied mask with an image. Here, by convolution we mean in the image processing sense, that is a correlation. The image is assumed to be extended with the grey-scale value 0 for calculating the convolution. Each value returned in CONV is calculated the mask centred over the corresponding pixel.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

CONV

On exit CONV contains the result of convolving IMAGE with MASK divided by SCALE_FACTOR.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor which the result is scaled down by to avoid overflow. SCALE_FACTOR must be in the range -2^{15} to $2^{15}-1$ for CONVOLVE_8 and in the range -2^{31} to $2^{31}-1$ for CONVOLVE_16. Unchanged on exit.

MASK_SIZE

On entry MASK_SIZE contains the size of the convolving mask. MASK_SIZE must be one of 3,5,7,...31. Unchanged on exit.

MASK

On entry MASK contains the mask that is to be convolved with IMAGE. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	MASK_SIZE out of range
IFAIL=4	Overflow in 16-bit to 8-bit conversion

7 Auxiliary Routines

The Image Processing Library routines ADD_16, SCAL_MULT_8_16, SCAL_DIV_16, SHIFT_COL_NORTH_P, SHIFT_COL_SOUTH_P, SHIFT_COL_WEST_P, SHIFT_COL_EAST_P, and CONTRACT_16_TO_8 are called.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Convolution, Image Processing

5.14 DIFF_OF_GAUSS_8

1 Purpose

DIFF_OF_GAUSS_8 convolves one of a set of nine difference-of-Gaussian masks with an 8-bit two's complement image.

2 Specification

```
SUBROUTINE DIFF_OF_GAUSS_8 (IMAGE,NS_SIZE, WE_SIZE, DOG,
&                           CHANNEL, IFAIL)

INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE), DOG(,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE, WE_SIZE, CHANNEL, IFAIL
```

3 Description

DOG is twice the convolution (in the image processing sense) of IMAGE with the selected difference-of-Gaussian mask which is specified in CHANNEL. IMAGE is assumed to be extended by a field of zeros for the convolution. The difference-of-Gaussian mask is:

$$\frac{1}{2\pi\sigma^2}\{\exp\left[-\frac{(x^2+y^2)}{2\sigma^2}\right] - .5 \quad \exp\left[-\frac{(x^2+y^2)}{4\sigma^2}\right]\}$$

where σ takes the following values depending on the value of CHANNEL:

CHANNEL	=	1	2	3	4	5	6	7	8	9
σ	=	$\sqrt{2}$	2	$2\sqrt{2}$	4	$4\sqrt{2}$	8	$8\sqrt{2}$	16	$16\sqrt{2}$

The mask is taken to be zero for $x^2 + y^2 > (3\sigma)^2$; thus a square mask of linear size $6\sigma + 1$ is used. For the large mask sizes IMAGE is scaled down by averaging 2x2, 4x4 and 8x8 blocks of pixels so that the largest mask size actually used in the computation is 19.

4 References

[1] Marr, D. and Hildreth, E.
1980 Theory of Edge Detection.
Proc. R. Soc. Lond. B 207 187-217

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved with the difference-of-Gaussian mask. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

DOG

On exit DOG contains twice the convolution of IMAGE with the difference-of-Gaussian mask selected by CHANNEL. DOG stores the convolution using a crinkled mapping (see IMAGE above).

CHANNEL

On entry CHANNEL contains an integer in the range 1 to 9 which selects the difference-of-Gaussian mask which is to be used. If CHANNEL is 4 or 5, then NS_SIZE and WE_SIZE must be divisible by 2, if CHANNEL is 6 or 7 NS_SIZE and WE_SIZE must be divisible by 4, or if CHANNEL is 8 or 9, NS_SIZE and WE_SIZE must be divisible by 8. Unchanged on exit.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below.)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=6	CHANNEL out of range
IFAIL=19	NS_SIZE incompatible with CHANNEL
IFAIL=20	WE_SIZE incompatible with CHANNEL

7 Auxiliary Routines

This routine uses the Image Processing Library routines APPLY_DOG_8, EXPAND_IMAGE_8, REDUCE_IMAGE_8 and SHIFT_IMAGE_8.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Convolution, Difference-of-Gaussians, Image Processing

5.15 F01_G_MM

1 Purpose

F01_G_MM performs a general matrix multiply of two matrices A and B where A is a P x Q matrix and B is a Q x R matrix with P,Q and R in the range 1 to ES.

2 Specification

```
REAL MATRIX FUNCTION F01_G_MM ( A , B , P , Q , R , IFAIL )

REAL A(,) , B(,)

INTEGER P , Q , R , IFAIL
```

3 Description

The routine is an optimised general matrix multiply using one of the following three procedures,depending on the relative sizes of P,Q and R (see [1]).

Procedure 1

```
   F01GMM=0.0
   DO 10 I=1,Q
10 F01GMM=F01GMM+MATC(A(,I))*MATR(B(I,))
```

Procedure 2

```
   DO 10 I=1,P
10 F01GMM(I,)=SUMR(MATC(A(I,))*B)
```

Procedure 3

```
   DO 10 I=1,R
10 F01GMM(,I)=SUMC(A*MATR(B(,I)))
```

If P/Q > 0.75 and R/Q > 0.75 procedure 1 is used, otherwise if P $\geq$ R procedure 3 is used or if P< R procedure 2 is used; the number 0.75 was determined empirically.

4 References

[1] MCKEOWN J.J.
Multiplication of non-standard matrices on DAP.
DAP newsletter no. 7.

5 Arguments

A – REAL MATRIX

On entry A contains the first of the two matrices to be multiplied together - array elements outside the matrix to be multiplied must be set to zero. The contents of A are unchanged on exit.

B – REAL MATRIX

On entry B contains the second of the two matrices to be multiplied together - array elements outside the matrix to be multiplied must be set to zero. The contents of B are unchanged on exit.

P – INTEGER

The number of rows in the first matrix. Unchanged on exit.

Q – INTEGER

The number of columns in the first matrix and the number of rows in the second matrix. Unchanged on exit.

R – INTEGER

The number of columns in the second matrix. Unchanged on exit.

IFAIL - INTEGER

Unless the routine detects an error (see 6 below) IFAIL contains zero on exit.

6 Errors

Errors detected by the routine:

IFAIL = 1	At least one of P, Q or R is not in the range 1 to ES.

7 Auxiliary Routines

None.

8 Accuracy

You can expect six significant figures.

9 Further Comments

None.

10 Keywords

Matrix multiply.

5.16 F01_M_INV

1 Purpose

F01_M_INV calculates, in place, the inverse of a given N x N matrix with N in the range 1 to ES.

2 Specification

```
SUBROUTINE F01_M_INV( A , N , IFAIL )

REAL A(,)

INTEGER N , IFAIL
```

3 Description

The matrix is inverted using Gauss-Jordan elimination with full pivoting.

4 References

None.

5 Arguments

A – REAL MATRIX

On entry A contains the matrix to be inverted,which is assumed to be located in the top left of A and array elements outside the input matrix must be set to zero. On exit A contains the inverse of that matrix.

N – INTEGER

On entry N must be set to the order of the matrix to be inverted. N is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see 6 below) IFAIL contains zero on exit.

6 Errors

Errors detected by the routine:

IFAIL=1	N is not in the range 1 to ES.
IFAIL=2	A pivot element is equal to zero – the matrix is singular

7 Auxiliary Routines

None.

8 Accuracy

You can expect five or six significant figures for well conditioned problems.

9 Further Comments

None.

10 Keywords

Matrix inversion, Gauss-Jordan elimination.

5.17 F04_QR_GIVENS_SOLVE

1 Purpose

F04_QR_GIVENS_SOLVE solves the linear system $Ax = b$ for x where A is an n x n matrix with 2<n<33. The routine may be used to simultaneously solve for up to ES different right hand side vectors b.

2 Specification

SUBROUTINE F04_QR_GIVENS_SOLVE(A , X , B , N , NB , IFAIL)

INTEGER N , NB , IFAIL

REAL A(,) , X(,) , B(,)

3 Description

The routine factorizes the given n x n matrix A as:

$$QA = R$$

where Q is an orthogonal matrix and r is upper triangular.

Givens method of plane rotations is used to annihilate elements of A below the leading diagonal until the matrix R remains. This leaves an upper triangular system which is solved by back substitution. Row i of A is used to annihilate the element in position $(i+1, j)$ by pre-multiplying A by a matrix of the form:

$$p^{j}_{i,i+1} = diag\,(I_{(i-1)}, U_{(i,i+1)}, I_{(n-i-1)}) \quad 1 \leq j \leq n-1$$

where $U_{i,i+1} = \begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix}$, with $c_i^2 + s_i^2 = 1$

In the usual serial application, these rotations are applied sequentially, but on the DAP you can perform up to $\frac{n}{2}$ rotations simultaneously [1].

4 References

[1] SAMEH A H and KUCK D J
On stable parallel linear system solvers
Journal of the Association of Computing Machinery, Vol 25, No 1, pp 81–91.

5 Arguments

A - REAL MATRIX

On entry, elements $A_{(i,j)}$ $(i = 1, 2, \ldots, N;\ j = 1, 2, \ldots, N)$ must be set to the elements of the matrix defining the linear system. A is unchanged on exit.

X – REAL MATRIX

On exit, column i of X will contain the solution of the system corresponding to the i^{th} column of B.

B – REAL MATRIX

On entry, columns 1 to NB must give the NB right hand side vectors. B is unchanged on exit.

N – INTEGER

On entry, N must be set to the order of the matrix A. N is unchanged on exit.

NB – INTEGER

On entry, NB must be set to the number of right hand side vectors for which the system is to be solved. NB is unchanged on exit.

IFAIL – INTEGER

Unless the routine detects an error (see section 6) IFAIL contains zero on exit.

6 Errors

Errors detected by the routine:

IFAIL = 1	N is not in the range 3 to ES or NB is not in the range 1 to ES
IFAIL = 2	A zero pivot has been found during the back substitution process, that is, the matrix is singular
IFAIL = 3	A very small pivot has been found during the back substitution process and the matrix is probably singular. Computation proceeds anyway, but you should treat the results with caution

7 Auxiliary Routines

This routine calls library routines Z_F04_BACK_SUBST, Z_FO4_SPREAD_LMAT_EAST, Z_FO4_SPREAD_RMAT_EAST and Z_FO4_UPDATE.

8 Accuracy

Empirical results indicate that errors may be expected in the 6^{th} or 7^{th} significant digit. The routine will return IFAIL = 3 (see 6 below) if the condition:

$$\frac{MAX_{i,j}\,|R_{ij}|}{MIN_i\,|R_{ii}|} > 5 \times 10^5$$

is satisfield, where R_{ij} is the upper triangular matrix defined in section 3.

9 Further Comments

You must not use common blocks with the names:

C_F04_QR1 and C_F04_QR2

10 Keywords

Givens' Rotation, Linear Equations.

5.18 FILL_IN_1

1 Purpose

FILL_IN_1 performs a region growing process on a logical image starting from input seed points. The region growing is constrained by the input image – it can only grow where the image is .TRUE..

2 Specification

```
      SUBROUTINE FILL_IN_1(BLOBS,NS_SIZE, WE_SIZE, SEED,
     &                     FILLED, PASSES, IFAIL)

      LOGICAL BLOBS(,,NS_SIZE,WE_SIZE), SEED(,,NS_SIZE,WE_SIZE)
     &        FILLED(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, PASSES, IFAIL
```

3 Description

FILL_IN_1 grows out a .TRUE. region from each .TRUE. point in SEED. The nearest neighbours of each .TRUE. point in the growing region are also set to be .TRUE. provided the equivalent point in BLOBS is also .TRUE.. This process is iterated until no further growth occurs (or PASSES iterations have been made). Thus FILLED is identical to BLOBS for all 'blobs' with seed points in them marked in SEED (provided no more than PASSES iterations are required to do this).

4 References

None.

5 Arguments

BLOBS

On entry BLOBS contains logical bit-map image. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of BLOBS are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

SEED

On entry SEED contains a logical bit-map of the same size as BLOBS. SEED must be set to .TRUE. at each seed point to be grown from. SEED is assumed to be stored using a crinkled mapping (see BLOBS above). Unchanged on exit.

FILLED

On exit FILLED is identical to BLOBS for all 'blobs' with .TRUE. seeds in them. FILLED is stored using a crinkled mapping (see BLOBS above).

PASSES

On entry PASSES contains the number of iterations of the growing process allowed. Unchanged on exit.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below.)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=8	PASSES not positive
IFAIL=10	Area fill incomplete after PASSES iterations

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Area Fill, Image Processing

5.19 HISTOGRAM_C_8

1 Purpose

HISTOGRAM_C_8 calculates the histogram for an 8-bit two's complement crinkle mapped image or tiles thereof.

2 Specification

```
      SUBROUTINE HISTOGRAM_C_8(IMAGE, NS_SIZE, WE_SIZE, HISTS,
     &                         HIST_SIZE, NO_HISTS, BINS, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, HISTS(BINS,NO_HISTS), HIST_SIZE,
     &          NO_HISTS, BINS, IFAIL
```

3 Description

HISTS is divided into a number of bins, each of which will contain the number of pixels with grey-scale values within its range. The number of bins is specified in BINS, and the number of grey-scale values in each range is 256 divided by BINS. For example, if BINS is 32, pixels with grey-scale values from −128 to −121 will be counted in BIN_1, those with grey-scale values from −120 to −113 will be counted in BIN_2, etc.

The sub-images histogrammed are square tiles with sides of ES x HIST_SIZE.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be histogrammed. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. *If local histograms are required then only powers of two are allowed.* Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32. *If local histograms are required then only powers of two are allowed.* Unchanged on exit.

HISTS

On exit HISTS contains the histogram results for each sub-image.

HIST_SIZE

On entry HIST_SIZE contains the size of side of the square sub-images to be histogrammed, divided by ES. HIST_SIZE must be a power of two unless a global histogram is required. Unchanged on exit.

NO_HISTS

On entry NO_HISTS contains the number of histograms to be calculated. NO_HISTS must be equal to $(NS\_SIZE \times WE\_SIZE)/HIST\_SIZE^2$, unless NO_HISTS is 1, in which case a global histogram is calculated. Unchanged on exit.

BINS

On entry BINS is the number of ranges of grey-scale values into which the image is to be divided. BINS must be one of 32, 64, 128 or 256 for both HISTOGRAM_C_8 and HISTOGRAM_C_16. Unchanged on exit.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	Invalid NS_SIZE
IFAIL=2	Invalid WE_SIZE
IFAIL=14	Invalid HIST_SIZE
IFAIL=15	HIST_SIZE incompatible with NO_HISTS
IFAIL=16	Invalid BINS

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Histogram, Image Processing

5.20 HISTOGRAM_S_8

1 Purpose

HISTOGRAM_S_8 calculates the histogram for an 8-bit two's complement sheet mapped image or tiles thereof.

2 Specification

```
      SUBROUTINE HISTOGRAM_S_8 (IMAGE, NS_SIZE, WE_SIZE, HISTS,
     &                          HIST_SIZE, NO_HISTS, BINS, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, HISTS(BINS,NO_HISTS), HIST_SIZE,
     &          NO_HISTS, BINS, IFAIL
```

3 Description

HISTS is divided into a number of bins, each of which will contain the number of pixels with grey-scale values within its range. The number of bins is specified in BINS, and the number of grey-scale values in each range is 256 divided by BINS. For example, if BINS is 32, pixels with grey-scale values from –128 to –121 will be counted in BIN_1, those with grey-scale values from –120 to –113 will be counted in BIN_2, etc.

The sub-images histogrammed are square tiles with sides of ES x HIST_SIZE.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be histogrammed. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a sheet mapping where the third and fourth co-ordinates of IMAGE label the sheets in the vertical and horizontal directions. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

HISTS

On exit HISTS contains the histogram results for each sub-image.

HIST_SIZE

On entry HIST_SIZE contains the size of side of the square sub-images to be histogrammed, divided by ES. HIST_SIZE must be a factor of NS_SIZE and WE_SIZE unless a global histogram is required. Unchanged on exit.

NO_HISTS

On entry NO_HISTS contains the number of histograms to be calculated. NO_HISTS must be equal to $(NS\_SIZE \times WE\_SIZE)/HIST\_SIZE^2$, unless NO_HISTS is 1, in which case a global histogram is calculated. Unchanged on exit.

BINS

On entry BINS is the number of ranges of grey-scale values into which the image is to be divided. BINS must be one of 32, 64, 128 or 256 for both HISTOGRAM_S_8 and HISTOGRAM_S_16. Unchanged on exit.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below.)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=14	Invalid HIST_SIZE
IFAIL=15	NS_SIZE not divisible by HIST_SIZE
IFAIL=16	WE_SIZE not divisible by HIST_SIZE
IFAIL=17	HIST_SIZE incompatible with NO_HISTS
IFAIL=18	Invalid BINS

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Histogram, Image Processing

5.21 KIRSCH_8

1 Purpose

KIRSCH_8 applies a user-selected Kirsch compass gradient mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

2 Specification

```
      SUBROUTINE KIRSCH_8(IMAGE,NS_SIZE, WE_SIZE, MASK_NO,
     &                    KIRSCH, SCALE_FACTOR, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE),
     &          KIRSCH(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, MASK_NO, SCALE_FACTOR, IFAIL
```

3 Description

The possible Kirsch compass gradient masks to be convolved (in the image processing sense) with IMAGE are:

$MASK\_NO = 1$:	3	3	3	$MASK\_NO = 2$:	3	3	3
(*North*)	3	0	3	(*Northeast*)	−5	0	3
	−5	−5	−5		−5	−5	3
$MASK\_NO = 3$:	−5	3	3	$MASK\_NO = 4$:	−5	−5	3
(*East*)	−5	0	3	(*Southeast*)	−5	0	3
	−5	3	3		3	3	3
$MASK\_NO = 5$:	−5	−5	−5	$MASK\_NO = 6$:	3	−5	−5
(*South*)	3	0	3	(*Southwest*)	3	0	−5
	3	3	3		3	3	3
$MASK\_NO = 7$:	3	3	−5	$MASK\_NO = 8$:	3	3	3
(*West*)	3	0	−5	(*Northwest*)	3	0	−5
	3	3	−5		3	−5	−5

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved with the Kirsh compass gradient mask. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

KIRSCH

On exit KIRSCH contains the convolution of IMAGE with the Kirsch compass gradient mask selected by MASK_NO divided by SCALE_FACTOR.

MASK_NO

On entry MASK_NO contains an integer in the range 1 to 8 which selects the mask to be used. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor which the result is scaled down by to avoid overflow. SCALE_FACTOR must be in the range -2^{15} to $2^{15}-1$ for KIRSCH_8, and in the range -2^{31} to $2^{31}-1$ for KIRSCH_16. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	MASK_NO out of range

7 Auxiliary Routines

Not available.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Kirsch, Compass Gradient, Edge Detection, Image Processing

5.22 LAPLACE_8

1 Purpose

LAPLACE_8 applies a user-selected Laplacian mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

2 Specification

```
SUBROUTINE LAPLACE_8(IMAGE,NS_SIZE, WE_SIZE, MASK_NO,
&                    LAP,SCALE_FACTOR, IFAIL)

INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE), LAP(,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE, WE_SIZE, MASK_NO, SCALE_FACTOR, IFAIL
```

3 Description

The possible Laplacian masks to be convolved (in the image processing sense) with IMAGE are:

$$MASK\_NO = 1: \begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix} \qquad MASK\_NO = 2: \begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

$$MASK\_NO = 3: \begin{matrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{matrix}$$

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved with the Laplacian masks. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

LAP

On exit LAP contains the convolution of IMAGE with the Laplacian mask selected by MASK_NO divided by SCALE_FACTOR.

MASK_NO

On entry MASK_NO contains 1, 2, or 3 which selects the Laplacian mask to be used. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor which the result is scaled down by to avoid overflow. SCALE_FACTOR must be in the range -2^{15} to $2^{15}-1$ for LAPLACE_8, and in the range -2^{31} to $2^{31}-1$ for LAPLACE_16. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	MASK_NO out of range

7 Auxiliary Routines

Not available.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Laplacian Smoothing, Filtering, Image Processing

5.23 LINE_DET_8

1 Purpose

LINE_DET_8 applies a user-selected line detection mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

2 Specification

```
      SUBROUTINE LINE_DET_8(IMAGE,NS_SIZE, WE_SIZE, MASK_NO,
     &                      LINE_DET, SCALE_FACTOR, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE),
     &          LINE_DET(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, MASK_NO, SCALE_FACTOR, IFAIL
```

3 Description

The possible line detection masks that can be convolved (in the image processing sense) with IMAGE are:

$$MASK\_NO = 1: \begin{matrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{matrix} \qquad MASK\_NO = 2: \begin{matrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{matrix}$$

$$MASK\_NO = 3: \begin{matrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{matrix} \qquad MASK\_NO = 4: \begin{matrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{matrix}$$

4. References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved with the line detection mask. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

LINE_DET

On exit LINE_DET contains the convolution of IMAGE with the line detection mask selected by MASK_NO divided by SCALE_FACTOR.

MASK_NO

On entry MASK_NO contains an integer in the range 1 to 4 which selects the mask to be used. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor which the result is scaled down by to avoid overflow. SCALE_FACTOR must be in the range -2^{15} to $2^{15}-1$ for LINE_DET_8, and in the range -2^{31} to $2^{31}-1$ for LINE_DET_16. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	MASK_NO out of range

7 Auxiliary Routines

Not available.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Line Detection, Image Processing

5.24 NORMALIZE_8

1 Purpose

NORMALIZE_8 expands the range of grey-scale values used within an 8-bit two's complement image to the full range.

2 Specification

SUBROUTINE NORMALIZE_8(IMAGE,NS_SIZE, WE_SIZE, NORM, IFAIL)

INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE), NORM(,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE, WE_SIZE, IFAIL

3 Description

The grey-scale range of IMAGE is expanded by scaling linearly to fill the entire range -128 to 127.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be normalized. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

NORM

On exit NORM contains the normalization of IMAGE. NORM stores the convolution using a crinkled mapping (see IMAGE above).

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

None.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Normalization, Image Processing

5.25 PERC_THRESH_8

1 Purpose

PERC_THRESH_8 performs thresholding on an 8-bit two's complement image.

2 Specification

```
      SUBROUTINE PERC_THRESH_8(IMAGE, NS_SIZE, WE_SIZE, P_THRESH,
     &                         HIGH_TIDE, PERCENTILE, FLIP, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE)


      INTEGER*4 NS_SIZE, WE_SIZE, HIGH_TIDE, PERCENTILE, IFAIL

      LOGICAL P_THRESH(,,NS_SIZE,WE_SIZE), FLIP
```

3 Description

P_THRESH is a logical bit-map of the image which has .TRUE. values where the IMAGE grey-scale values exceed HIGH_TIDE and .FALSE. otherwise. (This is reversed if FLIP is .FALSE. instead of .TRUE..) HIGH_TIDE is determined by the requirement that PERCENTILE per cent of the values in P_THRESH must be .TRUE..

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be thresholded. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

P_THRESH

On exit P_THRESH contains a logical bit-map of from the thresholding of IMAGE. P_THRESH stores the bit-map using a crinkled mapping (see IMAGE above).

HIGH_TIDE

On exit HIGH_TIDE is the threshold with which IMAGE is compared.

PERCENTILE

HIGH_TIDE is adjusted so PERCENTILE percent of the pixel grey-scale values exceed HIGH_TIDE (for FLIP .TRUE.).

FLIP

If FLIP is .TRUE., P_THRESH is .TRUE. when IMAGE is greater than HIGH_TIDE.
If FLIP is .FALSE., P_THRESH is .FALSE. when IMAGE is greater than HIGH_TIDE.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below.)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=17	PERCENTILE out of range 0 to 100

7 Auxiliary Routines

ABS_THRESH_8 from the Image Processing Library is called.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Thresholding, Image Processing

5.26 PREWITT_8

1 Purpose

PREWITT8 applies a user-selected Prewitt compass gradient mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

2 Specification

```
      SUBROUTINE PREWITT_8(IMAGE,NS_SIZE, WE_SIZE, MASK_NO,
     &                     PREW, SCALE_FACTOR, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE), PREW(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, MASK_NO, SCALE_FACTOR, IFAIL
```

3 Description

The possible Prewitt compass gradient masks that can be convolved (in the image processing sense) to IMAGE are:

$MASK\_NO = 1$ (*North*):

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{matrix}$$

$MASK\_NO = 2$ (*Northeast*):

$$\begin{matrix} 1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & -1 & 1 \end{matrix}$$

$MASK\_NO = 3$ (*East*):

$$\begin{matrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{matrix}$$

$MASK\_NO = 4$ (*Southeast*):

$$\begin{matrix} -1 & -1 & 1 \\ -1 & -2 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$MASK\_NO = 5$ (*South*):

$$\begin{matrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$MASK\_NO = 6$ (*Southwest*):

$$\begin{matrix} 1 & -1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & 1 \end{matrix}$$

$MASK\_NO = 7$ (*West*):

$$\begin{matrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{matrix}$$

$MASK\_NO = 8$ (*Northwest*):

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -2 & -1 \\ 1 & -1 & -1 \end{matrix}$$

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved with the Prewitt compass gradient mask. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

PREW

On exit PREW contains the convolution of IMAGE with the Prewitt compass gradient mask selected by MASK_NO divided by SCALE_FACTOR.

MASK_NO

On entry MASK_NO contains an integer in the range 1 to 8 which selects the mask to be used. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor which the result is scaled down by to avoid overflow. SCALE_FACTOR must be in the range -2^{15} to $2^{15}-1$ for PREWITT_8, and in the range -2^{31} to $2^{31}-1$ for PREWITT_16. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	MASK_NO out of range

7 Auxiliary Routines

Not available.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Prewitt, Compass Gradient, Image Processing

5.27 PSEUDO_MEDIAN_8

1 Purpose

PSEUDO_MEDIAN_8 calculates the pseudo-median at each pixel in an 8-bit two's complement image for a rectangular neighbourhood centred on the image.

2 Specification

```
      SUBROUTINE PSEUDO_MEDIAN_8 (IMAGE,NS_SIZE, WE_SIZE, PS_MED,
     &                            NS_MED, WE_MED, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE),
     &          PS_MED(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, NS_MED, WE_MED, IFAIL
```

3 Description

The pseudo-median is calculated by first calculating the median of the rows or columns of the rectangular neighbourhood about each point defined by NS_MED and WE_MED and then calculating the median of the results. The longer dimension is calculated first.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image for which the local pseudo-medians are to be calculated. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

PS_MED

On exit PS_MED contains the pseudo-median for a rectangular region about each pixel. PS_MED is stored using a crinkled mapping (see IMAGE above).

NS_MED

On entry NS_MED contains the height of the pseudo-median rectangle which must be one of 1, 3, 7, 15, 31. Unchanged on exit.

WE_MED

On entry WE_MED contains the width of the pseudo-median rectangle which must be one of 1, 3, 7, 15, 31. Unchanged on exit.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=11	Invalid NS_MED
IFAIL=12	Invalid WE_MED

7 Auxiliary Routines

The Image Processing Library routines PURE_MEDIAN_8, SHIFT_IMAGE_8 are called.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Pseudo-median, Median, Image Processing

5.28 PURE_MEDIAN_8

1 Purpose

PURE_MEDIAN_8 calculates the true median at each pixel in an 8-bit two's complement image for a rectangular neighbourhood centred on the image.

2 Specification

```
      SUBROUTINE PURE_MEDIAN_8 (IMAGE,NS_SIZE, WE_SIZE, MED,
     &                          NS_MED, WE_MED, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE),
     &          MED(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, NS_MED, WE_MED, IFAIL
```

3 Description

MED is the median grey-scale value of the pixels in the rectangular neighbourhood about each point defined by NS_MED and WE_MED.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image for which the local medians are to be calculated. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

MED

On exit MED contains the median of a rectangular region about each pixel. MED is stored using a crinkled mapping (see IMAGE above).

NS_MED

On entry NS_MED contains the height of the median rectangle which must be one of 1, 3, 7, 15, 31. Unchanged on exit.

WE_MED

On entry WE_MED contains the width of the median rectangle which must be one of 1, 3, 7, 15, 31. Unchanged on exit.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below.)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=11	Invalid NS_MED
IFAIL=12	Invalid WE_MED

7 Auxiliary Routines

The Image Processing Library routine SHIFT_IMAGE_8 is called.

8 Accuracy

Not applicable.

9 Further Comments

15 x 31, 31 x 15 and 31 x 31 medians cannot be performed by PURE_MEDIAN_16.

10 Keywords

Median, Image Processing

5.29 ROBERTS_8

1 Purpose

ROBERTS_8 applies a user-selected Roberts edge detection mask to an 8-bit per pixel two's complement image. The result may be scaled to avoid overflow.

2 Specification

```
      SUBROUTINE ROBERTS_8 (IMAGE,NS_SIZE, WE_SIZE, MASK_NO,
     &                      ROB, SCALE_FACTOR, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE), ROB(,,NS_SIZE,WE_SIZE)

      INTEGER*4 NS_SIZE, WE_SIZE, MASK_NO, SCALE_FACTOR, IFAIL
```

3 Description

The possible Roberts masks that can be convolved (in the image processing sense) with IMAGE are:

$$MASK\_NO = 1: \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} \qquad MASK\_NO = 2: \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix}$$

The masks are extended by zeros to the left and up and then applied as centred 3 x 3 masks.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved with the Roberts edge detection mask. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height in pixels of the image divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

ROB

On exit ROB contains the convolution of IMAGE with the Roberts

edge detection mask selected by MASK_NO divided by SCALE_FACTOR.

MASK_NO

On entry MASK_NO contains an integer in the range 1 to 2 which selects the mask to be used. Unchanged on exit.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor which the result is scaled down by to avoid overflow. SCALE_FACTOR must be in the range -2^{15} to $2^{15}-1$ for ROBERTS_8, and in the range -2^{31} to $2^{31}-1$ for ROBERTS_16. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	MASK_NO out of range

7 Auxiliary Routines

Not available.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Roberts, Edge Detection, Image processing

5.30 SOBEL_8

1 Purpose

SOBEL_8 applies a user-selected Sobel edge detection mask to an 8-bit per pixel two's complement image. If MASK_NO= 1 the horizontal edge detection mask is used and if MASK_NO= 2 the vertical edge detection mask is used. The result may be scaled to avoid overflow.

2 Specification

```
      SUBROUTINE SOBEL_8(IMAGE, NS_SIZE, WE_SIZE, MASK_NO, SOB,
     &                   SCALE_FACTOR, IFAIL)

      INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE), SOB(,,NS_SIZE,WE_SIZE)

      INTEGER NS_SIZE, WE_SIZE, MASK_NO, SCALE_FACTOR, IFAIL
```

3 Description

The vertical and horizontal masks convolved (in the image processing sense) with image are:

Horizontal :	1	0	−1		*Vertical* :	1	2	1
	2	0	−2			0	0	0
	1	0	−1			−1	−2	−1

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be convolved with the edge detection masks. The image is assumed to be stored using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image in pixels divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

MASK_NO

On entry MASK_NO, contains 1 if the horizontal edge detection mask is to be used and MASK_NO, contains 2 if the vertical edge detection mask is to be used. Unchanged on exit.

SOB

On exit SOB contains the the convolution of IMAGE with either the vertical or horizontal Sobel edge detection mask divided by SCALE_FACTOR.

SCALE_FACTOR

On entry SCALE_FACTOR contains the scale-factor which the result is scaled down by to avoid overflow. SCALE_FACTOR must be in the range -2^{15} to $2^{15}-1$ for SOBEL_8, and in the range -2^{31} to $2^{31}-1$ for SOBEL_16. Unchanged on exit.

IFAIL

IFAIL equals 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=3	MASK_NO not in range

7 Auxiliary Routines

Not available.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Sobel, Edge Detection, Image Processing

5.31 ZERO_X_8

1 Purpose

ZERO_X_8 finds the zero crossings in an 8-bit two's complement image.

2 Specification

```
SUBROUTINE ZERO_X_8(IMAGE,NS_SIZE, WE_SIZE, ZERO, IFAIL)

INTEGER*1 IMAGE(,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE, WE_SIZE, IFAIL

LOGICAL ZERO(,,NS_SIZE,WE_SIZE)
```

3 Description

ZERO is a logical bit-map of the image which has .TRUE. values where the IMAGE changes between positive and negative values.

4 References

None.

5 Arguments

IMAGE

On entry IMAGE contains the image to be searched for zero crossings. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a crinkled mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each crinkled tile. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be in the range 1 to 32. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to 32. Unchanged on exit.

ZERO

On exit ZERO contains a logical bit-map of zero crossings in IMAGE. ZERO stores the bit-map using a crinkled mapping (see IMAGE above).

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below.)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range
IFAIL=7	No zero crossings found

7 Auxiliary Routines

None.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Zero Crossing, Image Processing

Chapter 6

Image Analysis Routines

These image analysis routines allow the interesting features on an image to found and identified automatically. SEGMENT_16 and FEATURES_16 are available for analysing 16-bit images.

6.2 LABEL_16

1 Purpose

LABEL_16 labels the distinct 'blobs' in a binary image, returning a 16-bit two's complement image with all the pixels in a blob having the same value, but each blob is given a different value.

2 Specification

```
SUBROUTINE LABEL_16(BIN_IMAGE, NS_SIZE, WE_SIZE, LAB_IMAGE,
&                   IFAIL)

LOGICAL BIN_IMAGE(,,NS_SIZE,WE_SIZE)

INTEGER*2 LAB_IMAGE(,,NS_SIZE,WE_SIZE)

INTEGER*4 NS_SIZE, WE_SIZE, IFAIL
```

3 Description

A unique value is assigned to each .TRUE. pixel in BIN_IMAGE. These values are copy-propagated to .TRUE. nearest neighbours. When two or more values are assigned to a pixel it is the highest value which is assigned to that pixel.

4 References

None.

5 Arguments

BIN_IMAGE

On entry IMAGE contains the image on which the blob labelling is to be performed. Each dimension of the image size must be in the range ES to $32ES$ in multiples of ES. The image is assumed to be stored in IMAGE using a sheet mapping where the third and fourth co-ordinates of IMAGE are the vertical and horizontal co-ordinates of each sheet. Unchanged on exit.

NS_SIZE

On entry NS_SIZE contains the height of the image in pixels divided by ES. NS_SIZE must be in the range 1 to ES. Unchanged on exit.

WE_SIZE

On entry WE_SIZE contains the width of the image divided by ES. WE_SIZE must be in the range 1 to ES. Unchanged on exit.

LAB_IMAGE

On exit LAB_IMAGE contains a sheet-mapped (see IMAGE above) 16-bit image labelling the blobs in BIN_IMAGE.

IFAIL

IFAIL has the value 0 on exit unless an error occurs. (See 6 below)

6 Errors

IFAIL=0	Successful exit
IFAIL=1	NS_SIZE out of range
IFAIL=2	WE_SIZE out of range

7 Auxiliary Routines

The Image Processing Library routine LABEL_16_SUB is called.

8 Accuracy

Not applicable.

9 Further Comments

None.

10 Keywords

Image Feature Generation, Labelling, Image Processing

7 Auxiliary Routines

None.

8 Accuracy

Not available.

9 Further Comments

None.

10 Keywords

Image Feature Generation, Image Processing

Table 1

The following formulae are extracted from Digital Image Processing by Gonzalez and Wintz.

1. Invariant moment number 2

$$im_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

2. Invariant moment number 5

$$\begin{aligned} im_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left\{(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right\} \\ &\quad +(3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left\{3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right\} \end{aligned}$$

3. Invariant moment number 6

$$\begin{aligned} im_6 &= (\eta_{20} - \eta_{02})\{(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\} \\ &\quad +4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \end{aligned}$$

4. Invariant moment number 7

$$\begin{aligned} im_7 &= (3\eta_{21} - \eta_{30})(\eta_{30} + \eta_{12})\{(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\} \\ &\quad +(3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03})\{3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\} \end{aligned}$$

Where:

$$\eta_{pq} = \frac{u_{pq}}{a^{(p+q+2)/2}}$$

and $a = u_{00}$ so that the η_{pq} are invariant to scale, translation and intensity.
And where:

$$u_{pq} = \sum[(x - \bar{x})^p (y - \bar{y})^q f(x, y)]dxdy$$

$f(x, y)$ is the grey-scale value at position x , y.
$\bar{x}, \bar{y}$ is the position of the centroid of the blob, as defined by:

$$\bar{x} = \frac{\sum x f(x, y)dxdy}{\sum f(x, y)dxdy}$$

$$\bar{y} = \frac{\sum y f(x, y)dxdy}{\sum f(x, y)dxdy}$$