NEC

Preliminary User's Manual

IMAPCAR2 Family

Image Processing Library

Software

IMAPCAR2 series

IMAPCAR2-300

IMAPCAR2-200

IMAPCAR2-100

IMAPCAR2-50

Document No. U19879EE1V0MU00

Date Published: July 2009

Legal Notes

The information in this document is current as of July 2009. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).
  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

Table of Contents

1 Purpose of the document

The main goal of this document is to create a complete documentation for all the functions of the LIBIPL (Image Processing Library).

LIBIPL contains image processing basic functions. Image processing is the analysis, interpretation and manipulation of images and it is part of signal processing. Typical applications are filtering or feature extraction. Below are some illustrated examples of image processing applications.

Original image	Binarization	Edge detection	Median filter	Disatnce transform
				

2 Image manipulation

2.1 IxBinIm – Image Binarization

Syntax

void IxBinIm (sep void * src, sep void * dst, int bytes, int thres, int lines)

void IxBinImc(sep uchar src[], sep uchar dst[], int thres, int lines)

void IxBinImi(sep uint src[], sep uint dst[], int thres, int lines)

Description

Binarizes the initial image

- *src* and *dst* are respectively the source and destination image
- *bytes* is the number of bytes for each pixel of the source image
- *thres* is the threshold value to be used for the binarization
- *lines* is the number of lines to process

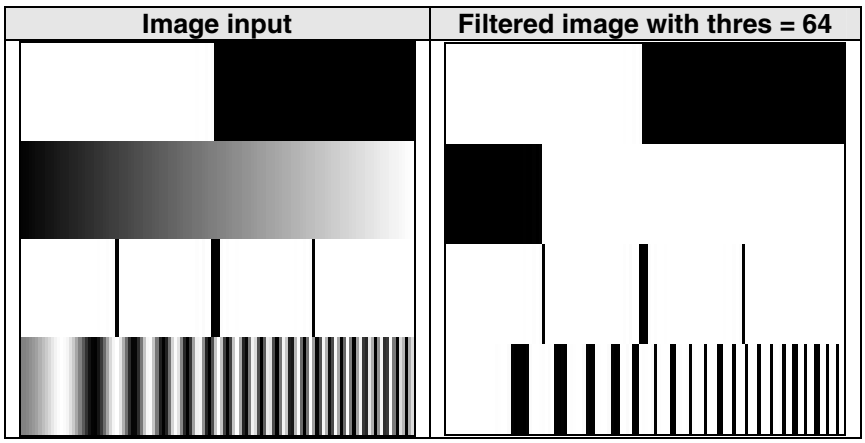
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- thres: scalar -> int
- lines: scalar -> int

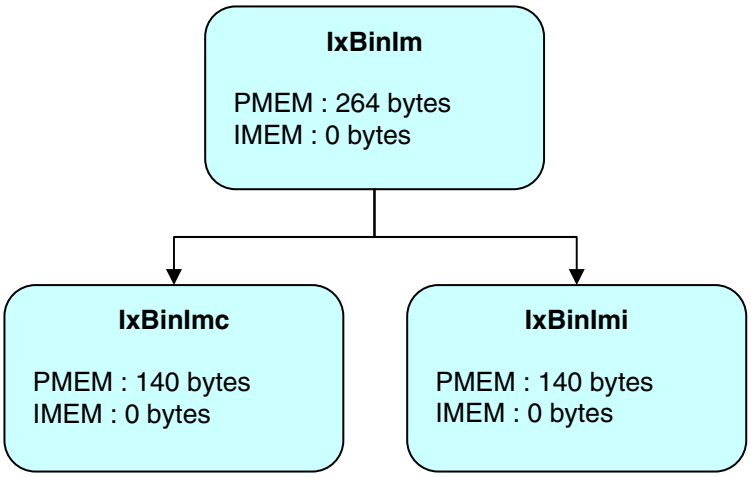
Type of output

- dst: 2-dimensional array -> sep void *

Example



Dependency diagram



Time performance

592 steps with 128 PE and for a 128*128 image

2.2 IxClearIm – Image Clear

Syntax

void IxClearIm(sep void dst[], int bytes, uchar val, int lines)

void IxClearImc(sep void dst[], uchar val, int lines)

void IxClearImi(sep void dst[], uint val, int lines)

Description

The whole image is filled with a constant value.

- *dst* is the destination image
- *bytes* is the number of bytes for each pixel of the source image
- *val* is the value to be used for the filling
- *lines* is the number of lines to process

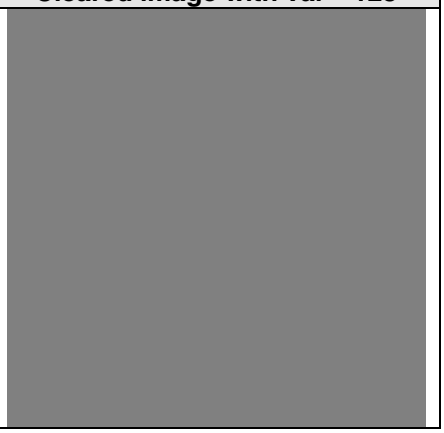
Type of inputs

- val: scalar -> uchar or uint
- bytes: scalar -> int
- lines: scalar -> int

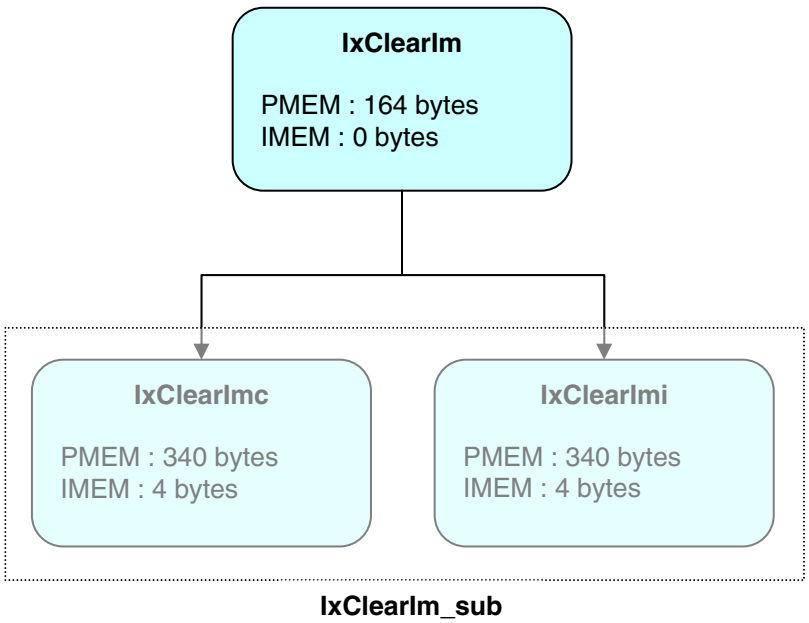
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Cleared image with val = 128
	

Dependency diagram



IxClearIm_sub

Time performance

290 steps with 128 PE and for a 128*128 image

2.3 IxCopyIm – Image Copy

Syntax

void IxCopyIm(sep void src[],sep void dst[], int bytes, int lines)

void IxCopyImc(sep void src[],sep void dst[], int lines)

void IxCopyImi(sep void src[],sep void dst[], int lines)

void IxCopyImci(sep void src[],sep void dst[], int lines)

Description

This function copies the *lines* number of lines of the source image *src* to the destination image *dst*.

bytes is the number of bytes for each pixel of the source image

- IxCopyImc copies 8-bits source image *src* to 8-bits destination image *dst*
- IxCopyImi copies 16-bits source image *src* to 16-bits destination image *dst*
- IxCopyImci copies 8-bits source image *src* to 16-bits destination image *dst*

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> int

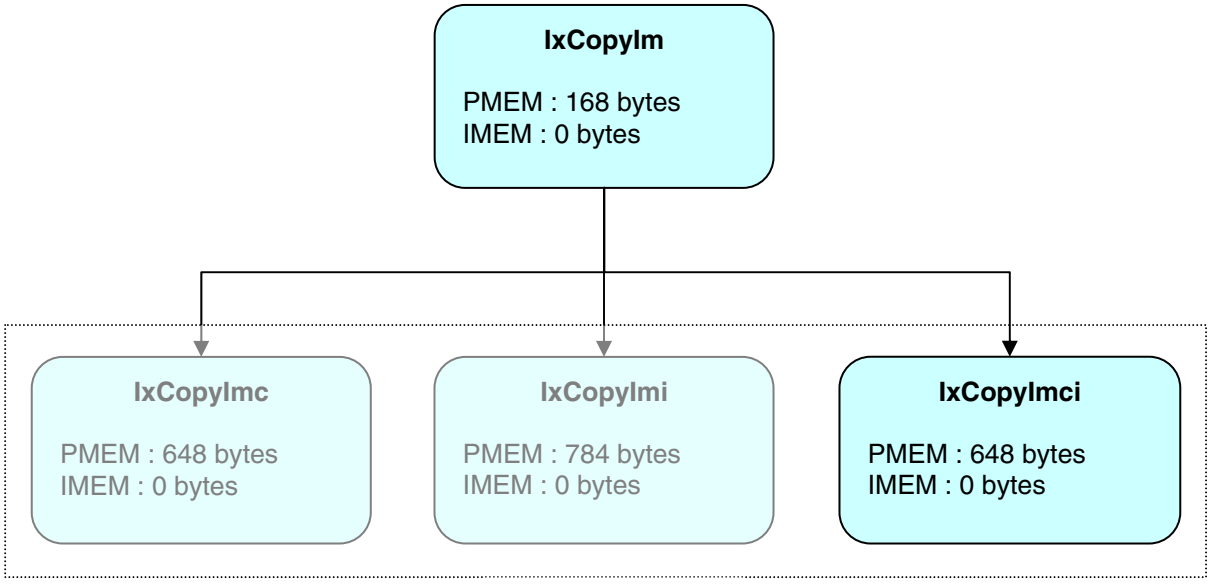
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Copied image
	

Dependency diagram



IxCopyIm_sub

Time performance

671 steps with 128 PE and for a 128*128 image

2.4 IxSubIm – Images Substraction

Syntax

void IxSubIm(sep void * src, int bytes, sep void * dst, uint lines)

void IxSubImc(sep uchar src[],sep uchar dst[], uint lines)

void IxSubImi(sep uint src[],sep uint dst[], uint lines)

Description

Calculates the absolute difference between two images. If the result is negative, absolute value is used. The result is stored in *dst*.

- *src* and *dst* are the image operands
- *dst* is also used as destination image
- *bytes* is the number of bytes for each pixel of the source image
- *lines* is the number of lines to process

Output = abs (Input1 – Input2)

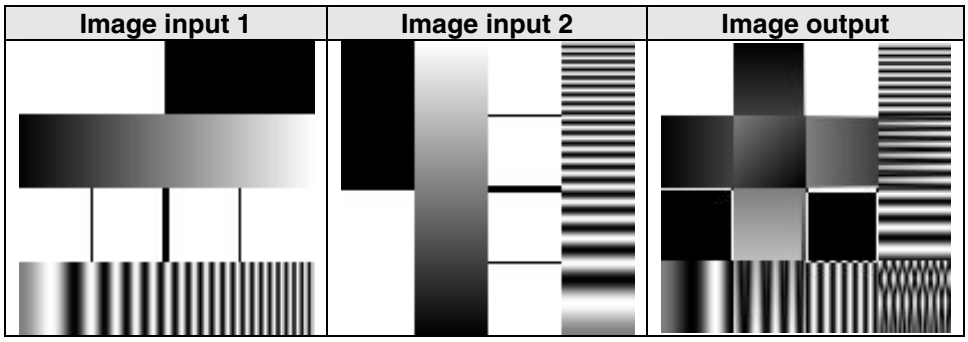
Type of inputs

- src: 2-dimensional array -> sep void *
- dst: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

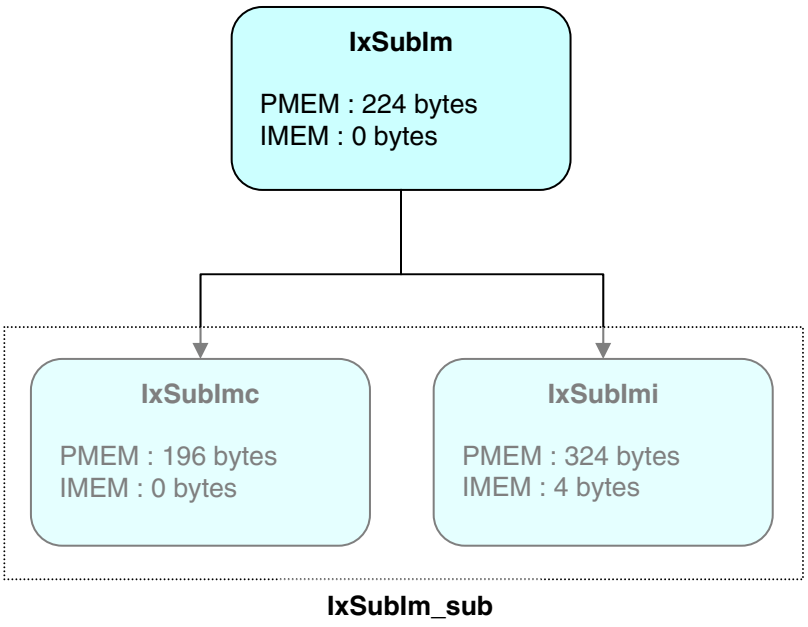
Type of output

- dst: 2-dimensional array -> sep void *

Example



Dependency diagram



Time performance

1012 steps with 128 PE and for a 128*128 image

2.5 IxSplitIm - Image Splitting

Syntax

void IxSplitIm(outside sep uchar eadr[], outside sep uchar eadr_out[], int lines)

Description

Splits the lower byte part (8 least significant bits) from the higher byte part (8 most significant bits) of an integer image (16 bits).

lines is the number of lines of the integer image.

The integer image is stored from *eadr* address (2**lines*).

Result is stored back into *eadr_out* … *eadr_out*+2**lines*.

Type of inputs

- eadr: pointer to the integer image -> outside sep uchar * (2**lines*)
- lines: scalar -> int

Type of output

- eadr_out: pointer to the two splitted images -> outside sep uchar *

Memory consumption

Memory type	Bytes
PMEM	460
IMEM	480

2.6 IxArrowDraw – Arrow Drawing

Syntax

void IxArrowDraw(osep uchar xdir[], osep uchar ydir[], sep void * img, int bytes, int rate1, int rate2, int col, int lines)

Description

Draws arrows according to direction data.

Regarding *xdir* as the disposition in the x direction, *ydir* as the disposition in the y direction, IxArrowDraw draws arrows at corresponding location on the source image *img*.

- *bytes* is the number of bytes for each pixel of the source image
- *rate1* (>=6) designates the distance between two arrows.
- *rate2* (<=*rate1*) designates the maximum arrow length.
- *col* designates the grey level of the arrow (between 0 and 255).
- *lines* is the number of lines to process

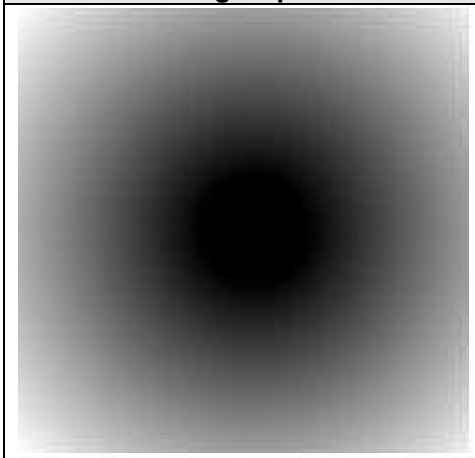
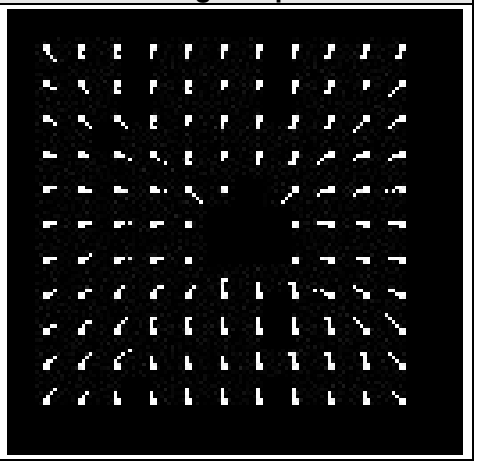
Type of inputs

- xdir: 2-dimensional array stored in external memory -> osep uchar *
- ydir: 2-dimensional array stored in external memory -> osep uchar *
- img: 2-dimensional array -> sep void *
- bytes: scalar -> int
- rate1: scalar -> int
- rate2: scalar -> int
- col: scalar -> int
- lines: scalar -> int

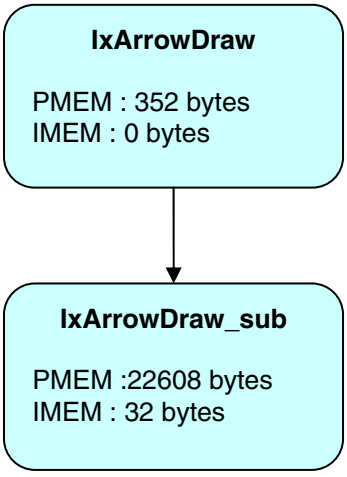
Type of output

- img: 2-dimensional array -> sep void *

Example

Image input	Image output
	

Dependency diagram



2.7 IxAffine – Affine Transform

Syntax

void IxAffine(sep void * src, int bytes, int src_x, int src_y, sep void * dst, int dst_x, int dst_y, long cof_a, long cof_b, long cof_c, long cof_d, long cof_e, long cof_f, int defval)

Description

Performs the affine transform on source image *src* returned in destination image *dst*.

bytes is the number of bytes for each pixel of the source image

src_x and *src_y* are respectively the width and height of the source image

dst_x and *dst_y* are respectively the width and height of the destination image

Transform matrix *cof_a, …, cof_f* must be fixed pointer numbers (integer part 16 bits, others 16 bits).

cof_a, …, cof_f must fulfill the following conditions:

|cof_b| < 0.5 * 65536 et |cof_d| < 0.5 * 65536

|cof_a| < 0.125 * 65536 et |cof_c| < 0.125 * 65536

For each location (dx,dy), pixel locationed at the following (sx,sy) is placed:

sx = (cof_a * dx + cof_b * dy + cof_c) / 65536

sy = (cof_d * dx + cof_e * dy + cof_f) / 65536

Or in matricial formulation:

$$sx = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} + c$$

$$sy = \begin{bmatrix} d & e \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} + f$$

Linear interpolation is performed during the above calculation.

And if (*sx*,*sy*) is beyond the range of the input image, then *defval* is used to fill the corresponding (*dx*,*dy*) location.

The error produced by this function depends on the rotation angle.

The larger the rotation angle is, the more the error may occur.

Also, the speed of the function depends largely on the amount of displacement along the x direction.

Type of inputs

- src: 2-dimensional array -> sep void *
- src_x: scalar -> int
- src_y: scalar -> int
- dst_x: scalar -> int
- dst_y: scalar -> int
- bytes: scalar -> int
- cof_a: scalar -> long
- cof_b: scalar -> long
- cof_c: scalar -> long
- cof_d: scalar -> long
- cof_e: scalar -> long
- cof_f: scalar -> long
- defval: scalar -> int

Type of output

- dst: 2-dimensional array -> sep void *

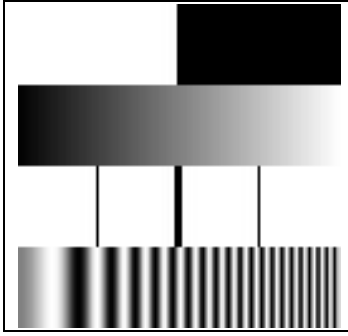
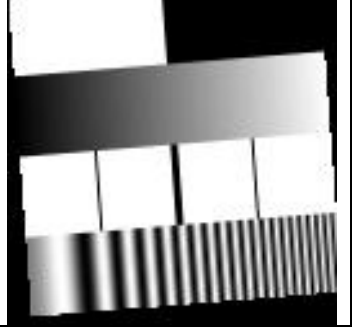
Example

- Chosen parameter values:

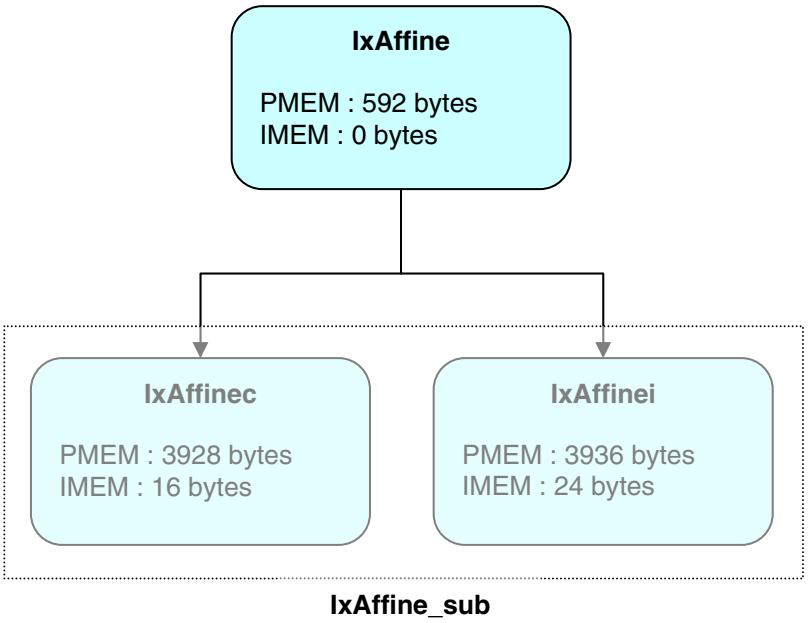
Parameter	Value
src_x	128
src_y	128
dst_x	100
dst_y	100

Parameter	Value
cof_a	68680
cof_b	-5025
cof_c	0
cof_d	5397
cof_e	66774
cof_f	0

- Result:

Image input	Image output
	

Dependency diagram



Time performance

10038 steps with 128 PE and for a 128*128 image

2.8 IxRotation90 – 90° Image Rotation

Syntax

void IxRotation90(sep void * src, int bytes, int flag, int lines)

Description

Rotates a PENO * lines image over 90 degrees.

- src is the source image
- flag is used to choose the type of rotation: 1 if clockwise, 0 if anticlockwise
- bytes represents the number of bytes per pixel

Type of inputs

- src: 2-dimensional array -> sep void *
- flag: scalar -> int
- bytes: scalar -> int
- lines: scalar -> int

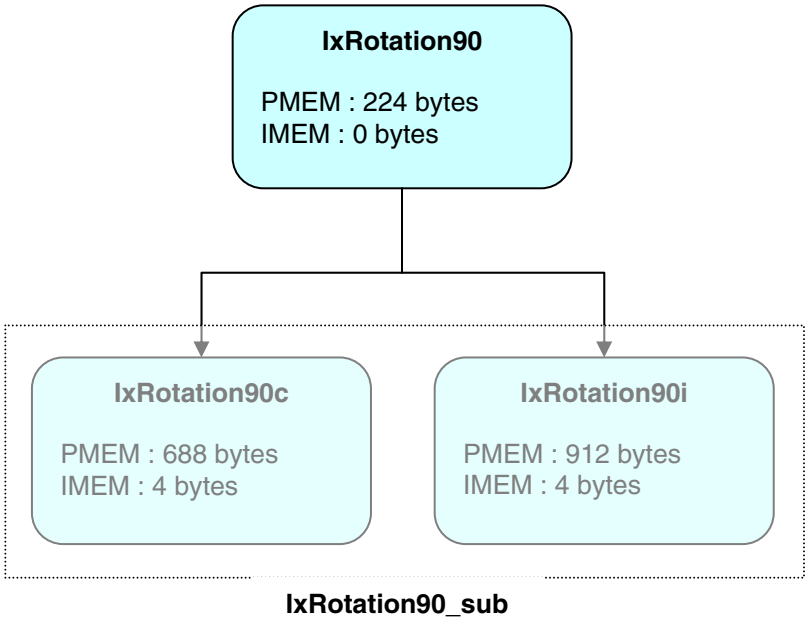
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Result for IxRotation90
	

Dependency diagram



Time performance

- 2944 steps for a 128*128 image with 128 PE

2.9 IxRotation - Image Rotation

Syntax

- void IxRotation(sep void * src, sep void * tmp, int bytes, uint degree, uint lines)

Description

Rotates the *src* source image depending on the value of *degree*.

The angle of rotation in degrees correspond to 45/16 times the *degree* value

0	**-90°**
16	-45°
32	0
48	45°
64	90°

bytes is the number of bytes for each pixel of the source image

The destination image is temporary computed in *tmp* and finally returned in *src*.

lines is the number of lines to process

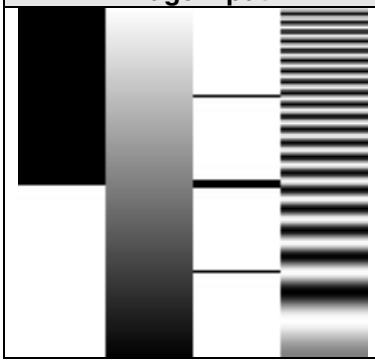
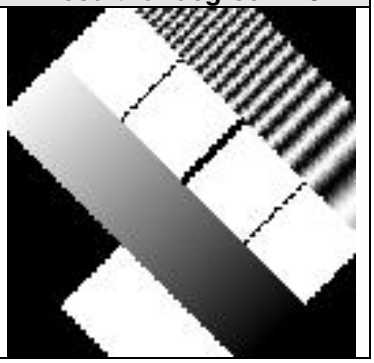
Type of inputs

- src: 2-dimensional array -> sep void *
- degree: scalar -> int
- bytes: scalar -> int
- lines: scalar -> uint

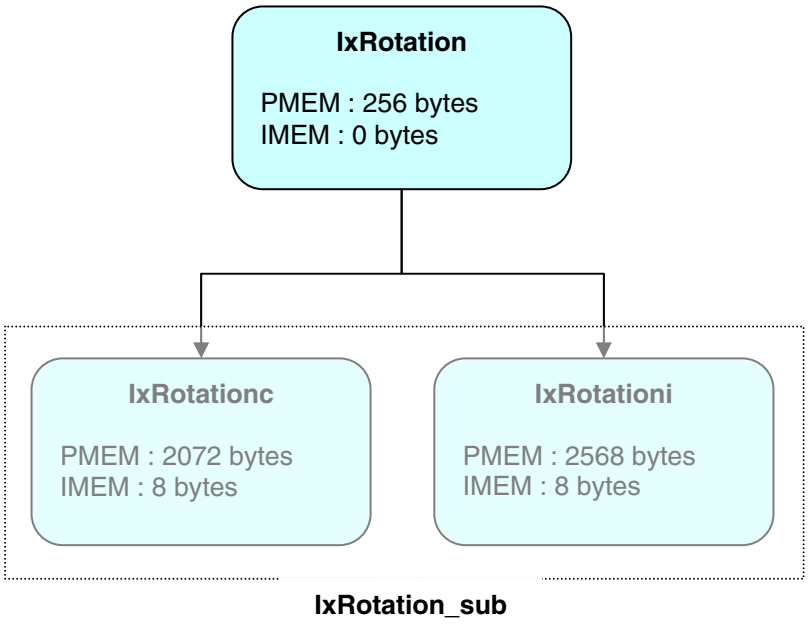
Type of output

- src: 2-dimensional array -> sep void *
- tmp: 2-dimensional array -> sep void *

Example

Image input	**Result for degree = 16**
	

Dependency diagram



Time performance

32047 steps for a 128*128 image with 128 PE

3 Image filtering

3.1 Edge detection

3.1.1 IxKirsch – Kirsch filter

Syntax

void IxKirsch(sep void * src, int bytes, sep uchar dir[], int thre, uint lines)

Description

The Kirsch Filter detects edges using 8 oriented filters.

All eight filters are applied to the image with the maximum being retained for the final image.

The convolution filters are of the form:

$$\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix}$$

$$\begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix} \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix}$$

- *src* is used as source and destination image
- *bytes* is the number of bytes per pixel for the source image
- *thre* is the threshold value for edge detection
- *lines* is the number of lines of the source image
- *dir* is the output direction information

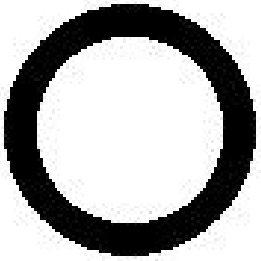
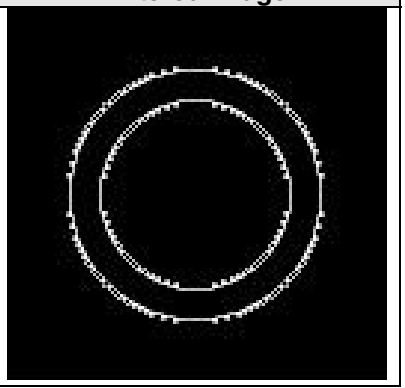
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- thre: scalar -> int
- lines: scalar -> uint

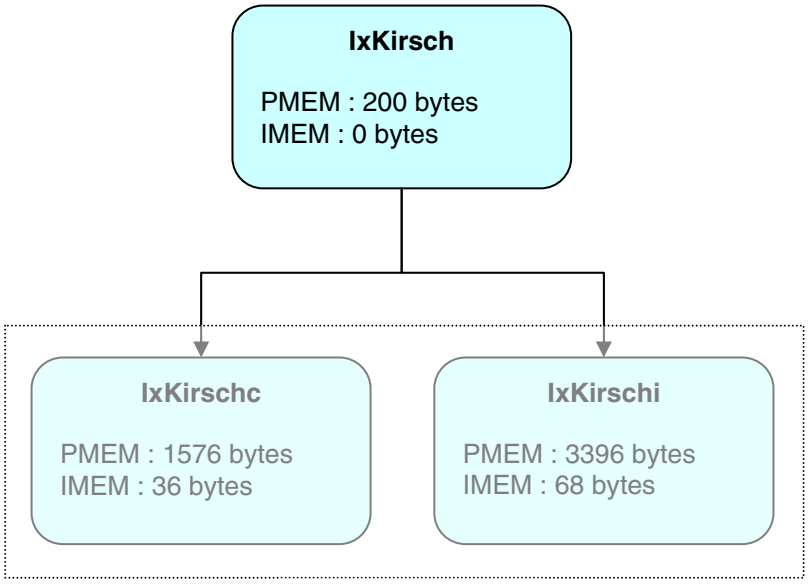
Type of output

- src: 2-dimensional array -> sep void *
- dir: 2-dimensional array -> sep uchar *

Example

Image input	Filtered image
	

Dependency diagram



IxKirsch_sub

Time performance

52132 steps with 128 PE and for a 128*128 image

3.1.2 IxPrewitt – Prewitt filter

Syntax

void IxPrewitt(sep void * src, int bytes, sep uchar dir[], int thre, uint lines)

Description

The Prewitt filter detects edges using 8 oriented filters. All eight filters are applied to the image with the maximum being retained for the final image. The convolution filters are of the form:

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

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

- *src* is used as source and destination image
- *bytes* is the number of bytes per pixel for the source image
- *thre* is the threshold value for edge detection
- *lines* is the number of lines of the source image
- *dir* is the output direction information

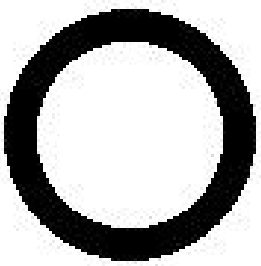
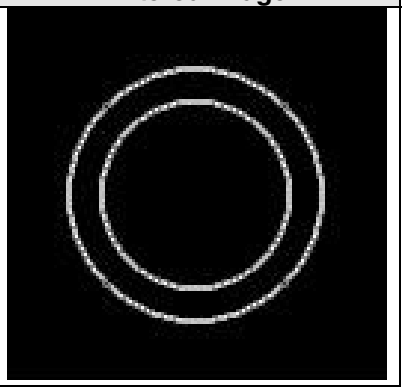
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- thre: scalar -> int
- lines: scalar -> uint

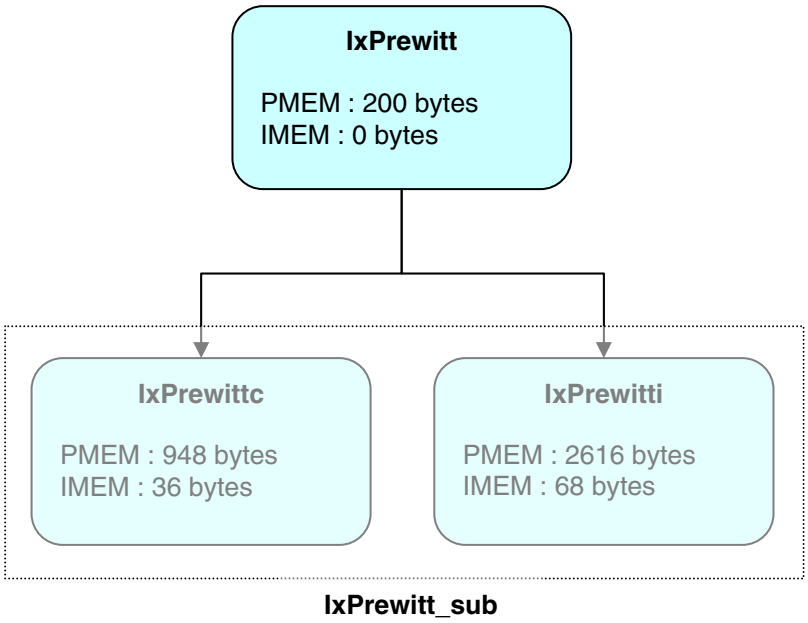
Type of output

- src: 2-dimensional array -> sep void *
- dir: 2-dimensional array -> sep uchar *

Example

Image input	Filtered image
	

Dependency diagram



Time performance

40389 steps with 128 PE and for a 128*128 image

3.1.3 IxRobinson – Robinson filter

Syntax

void IxRobinson(sep void * src, int bytes, sep uchar dir[], int thre, uint lines)

Description

The Robinson filter detects edges using 8 oriented filters.

All eight filters are applied to the image with the maximum being retained for the final image.

The convolution filters are of the form:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$$

- *src* is used as source and destination image
- *bytes* is the number of bytes per pixel for the source image
- *thre* is the threshold value for edge detection
- *lines* is the number of lines of the source image
- *dir* is the output direction information

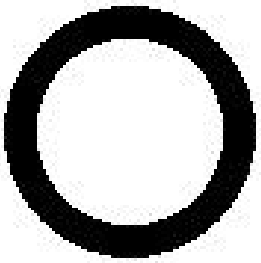
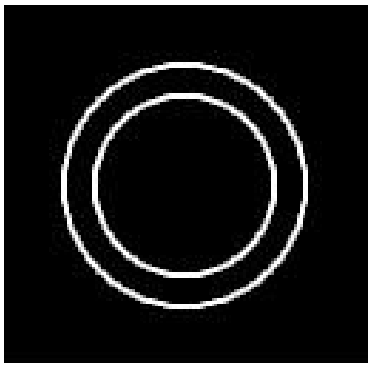
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- thre: scalar -> int
- lines: scalar -> uint

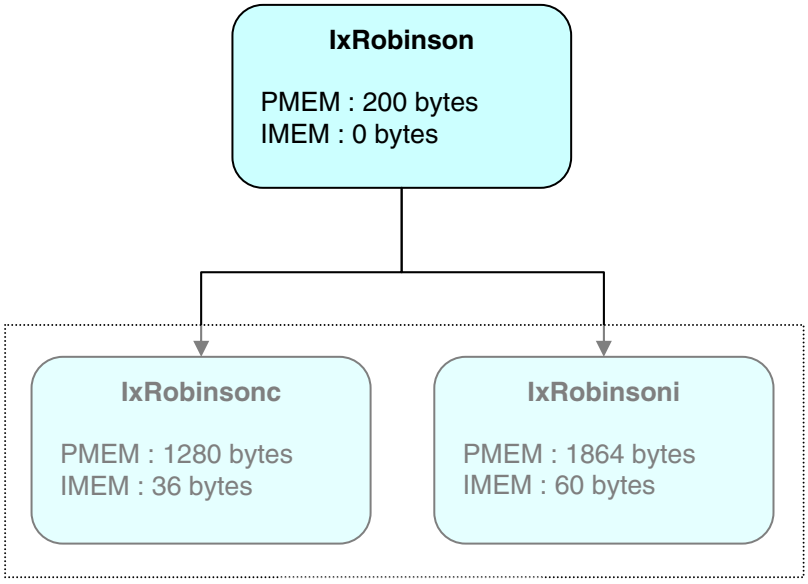
Type of output

- src: 2-dimensional array -> sep void *
- dir: 2-dimensional array -> sep uchar *

Example

Image input	Filtered image
	

Dependency diagram



IxRobinson_sub

Time performance

39129 steps with 128 PE and for a 128*128 image

3.1.4 IxVanderbrug – Vanderbrug filter

Syntax

void IxVanderbrug(sep void * src, int bytes, sep uint dir[], int thres, uint lines)

Description

The Vanderbrug operator detects edges using 16 oriented filters.

All 16 filters are applied to the image with the maximum being retained for the final image.

- The result is stored in *src*, and the output direction information is stored in *dir*.
- *bytes* is the number of bytes per pixel for the source image
- *thres* is the threshold value used for edge detection.
- *lines* is the number of lines of the source image

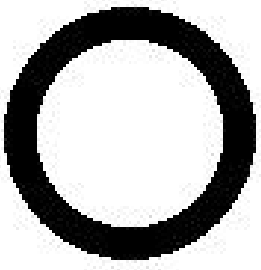
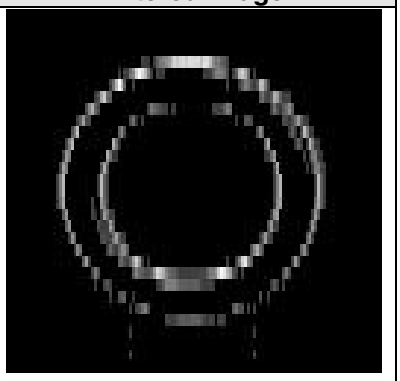
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- thres: scalar -> int
- lines: scalar -> uint

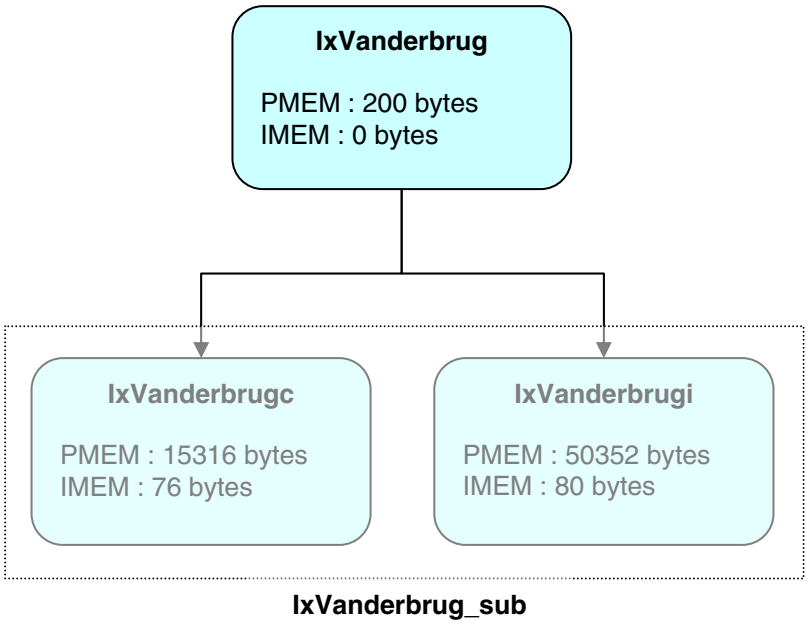
Type of output

- src: 2-dimensional array -> sep void *
- dir: 2-dimensional array -> sep uint *

Example

Image input	Filtered image
	

Dependency diagram



Time performance

94312 steps with 128 PE and for a 128*128 image

3.1.5 IxLaplace – Laplace filter

Syntax

void IxLaplace(sep void * src, int bytes, uint lines)

Description

It performs the laplacian filter between the current pixel and its 8 neighbors by convoluting the image with the matrix below.

- *src* is used as source and destination image
- *lines* is the number of lines to process
- *bytes* is the number of bytes per pixel for the source image

[0 -1 0

Output = abs (-1 4 -1 * Input)

0 -1 0]

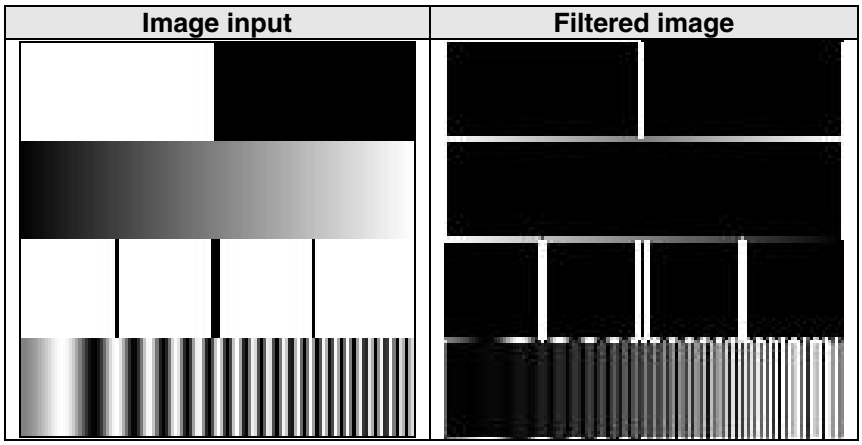
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

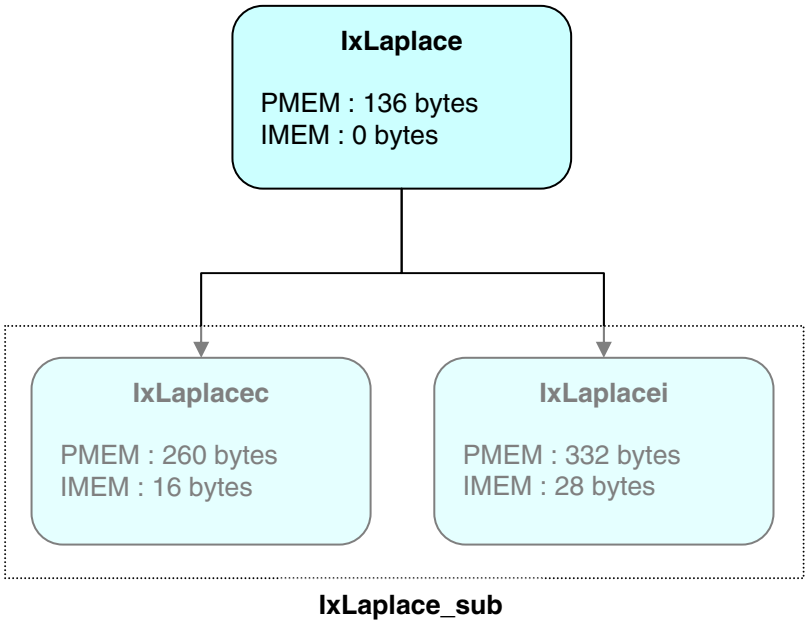
Type of output

- src: 2-dimensional array -> sep void *

Example



Dependency diagram



IxLaplace_sub

Time performance

2383 steps with 128 PE and for a 128*128 image

3.1.6 IxSobelx, IxSobely – 3-dimensional Sobel filter

Syntax

void IxSobelx(sep void * src, int bytes, uint lines)

void IxSobely(sep void * src, int bytes, uint lines)

Description

Performs the edge-detecting Sobel filter which is a special high-pass filter.

- *src* is used as source and destination image
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines of the source image
  - o IxSobelx(src,lines) returns in *src* the result of the convolution between the source image and the vertical filter below:

$$\text{Output} = \text{abs}\left(\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * \text{Input} \right)$$

  - o In the same way, IxSobely(src,lines) returns in *src* the result of the convolution between the source image and the horizontal filter below:

$$\text{Output} = \text{abs}\left(\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \text{Input} \right)$$

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

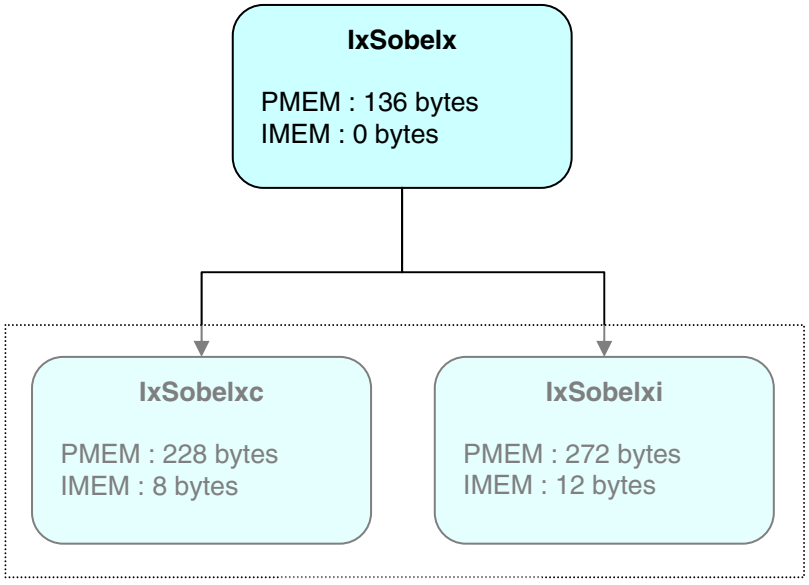
Type of output

- src: 2-dimensional array -> sep void *

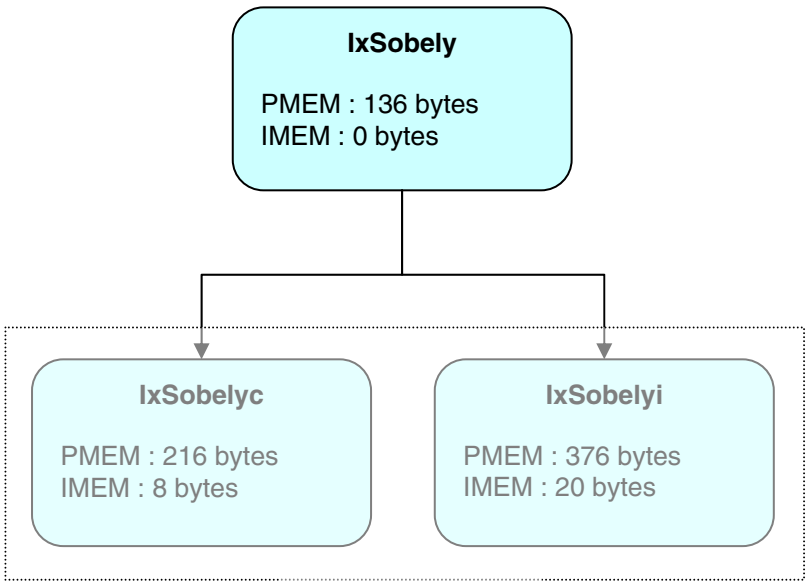
Examples

Image input	Sobelx result	Sobely result
		

Dependency diagrams



IxSobelx_sub



IxSobely_sub

Time performance

Number of steps with 128 PE and for a 128*128 image:

Function	Steps
IxSobelx	2382
IxSobely	2361

3.1.7 IxSobel5x, IxSobel5y – 5-dimensional Sobel filter

Syntax

void IxSobel5x(sep void * src, sep void * dst, int bytes, uint lines)

void IxSobel5y(sep void * src, sep void * dst, int bytes, uint lines)

Description

Performs the edge-detecting Sobel filter which is a special high-pass filter

- *src* and *dst* are respectively the source and destination image
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines of the source image
  - o IxSobel5x(src,dst,lines) returns in *src* the result of the convolution between the source image and the vertical filter below:

$$\begin{bmatrix} 31 & 48 & 0 & -48 & -31 \\ 24 & 60 & 0 & -60 & -24 \\ 60 & 120 & 0 & -120 & -60 \\ 24 & 60 & 0 & -60 & -24 \\ 31 & 48 & 0 & -48 & -31 \end{bmatrix}$$

  - o In the same way, IxSobel5y(src,dst,lines) returns in *src* the result of the convolution between the source image and the horizontal filter below:

$$\begin{bmatrix} 31 & 24 & 60 & 24 & 31 \\ 48 & 60 & 120 & 60 & 48 \\ 0 & 0 & 0 & 0 & 0 \\ -48 & -60 & -120 & -60 & -48 \\ -31 & -24 & -60 & -24 & -31 \end{bmatrix}$$

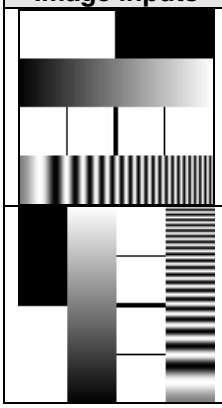
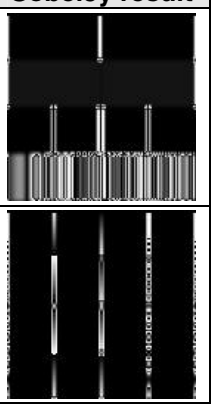
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

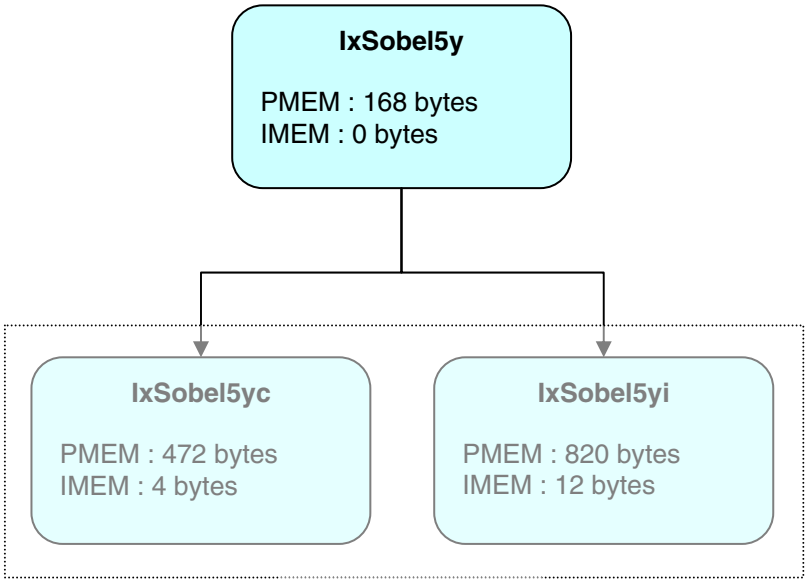
Type of output

- dst: 2-dimensional array -> sep void *

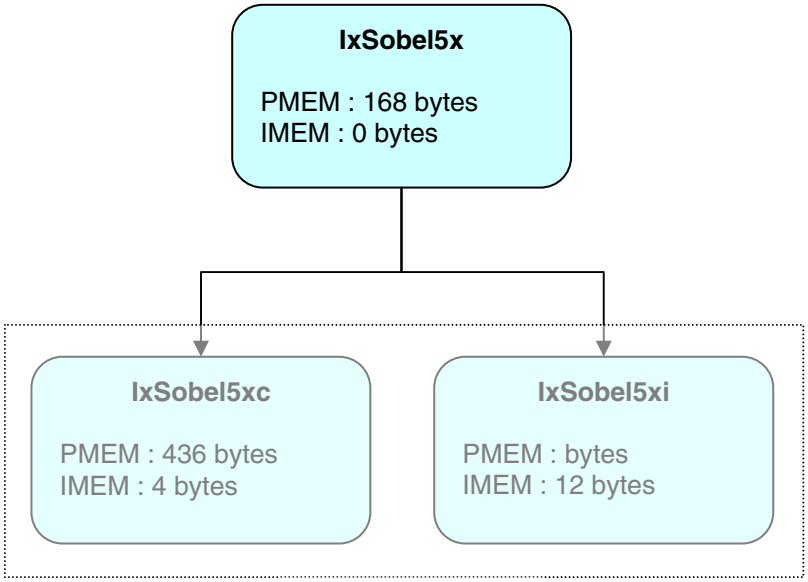
Examples

Image inputs	Sobel5x result	Sobel5y result
		

Dependency diagrams



IxSobel5y_sub



IxSobel5x_sub

Time performance

Number of steps with 128 PE and for a 128*128 image:

Function	Steps
IxSobel5x	7204
IxSobel5y	6901

3.1.8 IxSobel3x3

Syntax

void IxSobel3x3(sep void * src, int bytes, uchar thre, uint lines)

Description

Performs the edge-detecting Sobel filter which uses 2 special high-pass filters.

- *src* is the source image
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines of the source image
- *dst* is the output image

IxSobel(src,dst,lines) returns in *dst* the result of the following operation:

$$\text{Output} = \left| \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \times \text{Input} \right| + \left| \begin{bmatrix} 1 & 2 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \times \text{Input} \right|$$

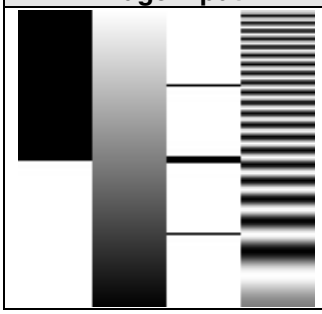
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

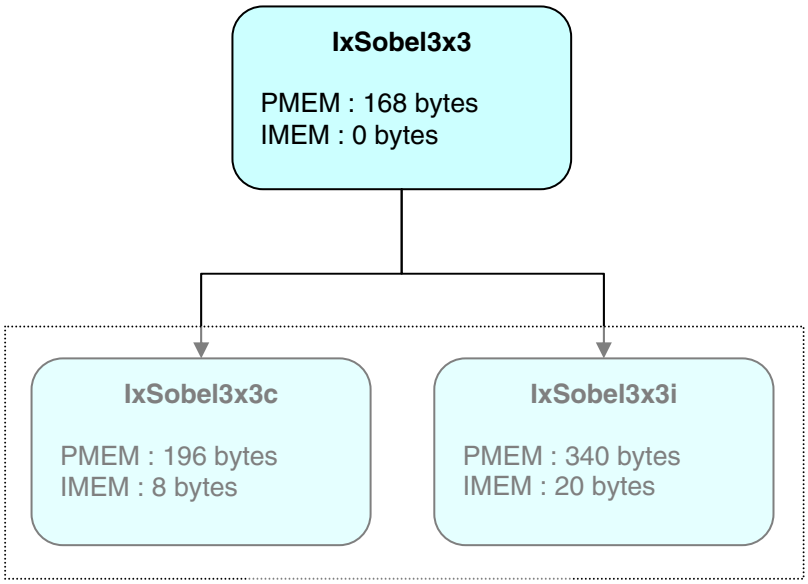
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Filtered image
	

Dependency diagram



IxSobel3x3_sub

Time performance

2136 steps with 128 PE and for a 128*128 image

3.1.9 IxSobel5 – 5x5 Sobel filter

Syntaxes

void IxSobel5(sep void * src, sep void * dst, int bytes, uint thre, uint lines)

Description

Performs the edge-detecting Sobel filter which uses two special high-pass filters.

- *src* is the source image
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines of the source image
- *thre* is the threshold value used for edge detection
- *dst* is the output image

IxSobel5(src,dst,thre,lines) returns the sum of the images filtered by the two 5-dimensional filters described below:

$$\text{Output} = \text{abs}\left[\begin{bmatrix} 31 & 48 & 0 & -48 & -31 \\ 24 & 60 & 0 & -60 & -24 \\ 60 & 120 & 0 & -60 & -120 \\ 24 & 60 & 0 & -60 & -24 \\ 31 & 48 & 0 & -48 & -31 \end{bmatrix} * \text{Input}\right] + \text{abs}\left[\begin{bmatrix} 31 & 24 & 60 & 24 & 31 \\ 48 & 60 & 120 & 60 & 48 \\ 0 & 0 & 0 & 0 & 0 \\ -48 & -60 & -120 & -60 & -48 \\ -31 & -24 & -60 & -24 & -31 \end{bmatrix} * \text{Input}\right]$$

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint
- thre: scalar -> uint

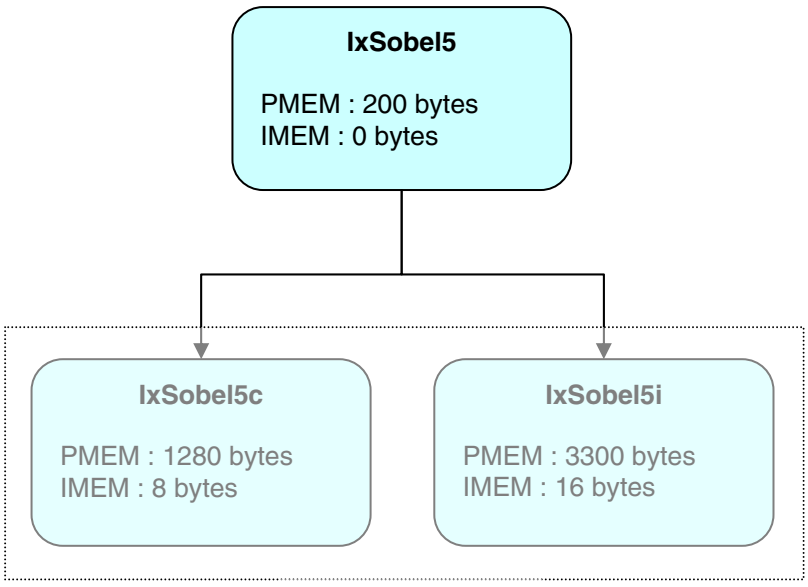
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Image output
	

Dependency diagram



Time performance

13579 steps with 128 PE and for a 128*128 image

3.1.10 IxSobelxy – 3x3 Sobel filter

Syntaxes

void IxSobelxy(sep void * src, int bytes, uint thre, uint lines)

Description

Performs the edge-detecting Sobel filter which uses two special high-pass filters.

The two oriented filters are applied to the image, then the two resulting images are binarized and the OR is taken between them to provide the final image.

- *src* is used as source and destination image
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines of the source image
- *thre* is the threshold value used for binarization.

The two convolution filters are of the form:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint
- thre: uchar -> uint

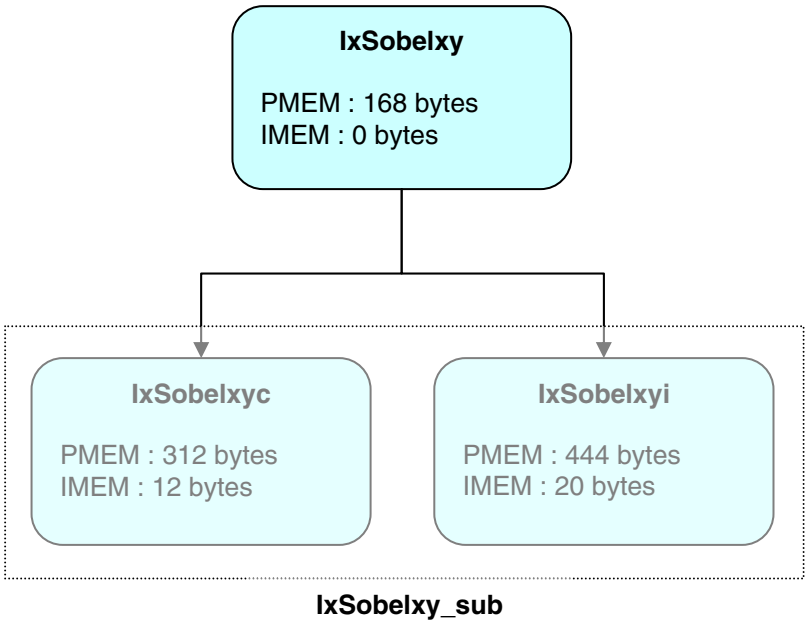
Type of output

- src: 2-dimensional array -> sep void *

Example

Image input	Result (thre = 120)
	

Dependency diagram



Time performance

3662 steps with 128 PE and for a 128*128 image

3.1.11 IxSobelDegree – Sobel Degree filter

Syntax

void IxSobelDegree(sep void * src, int bytes, sep uchar deg[], uchar sel, uint lines)

Description

This function calculates the sobel magnitude and angle.

Both are returned respectively in *src* and *deg*

- *bytes* is the number of bytes per pixel for the source image
- *sel* is the parameter used to choose the Kernel size: 3*3 if sel = 0, 5*5 if sel = 1
- *lines* is the number of lines to process

Therefore, the magnitude array is an approximation (the 9 lowest significant bits are deleted).

The angle is returned in a value between 0 and 255 corresponding to the range 0 up to 180 degree.

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- sel: scalar -> uchar
- lines: scalar -> uint

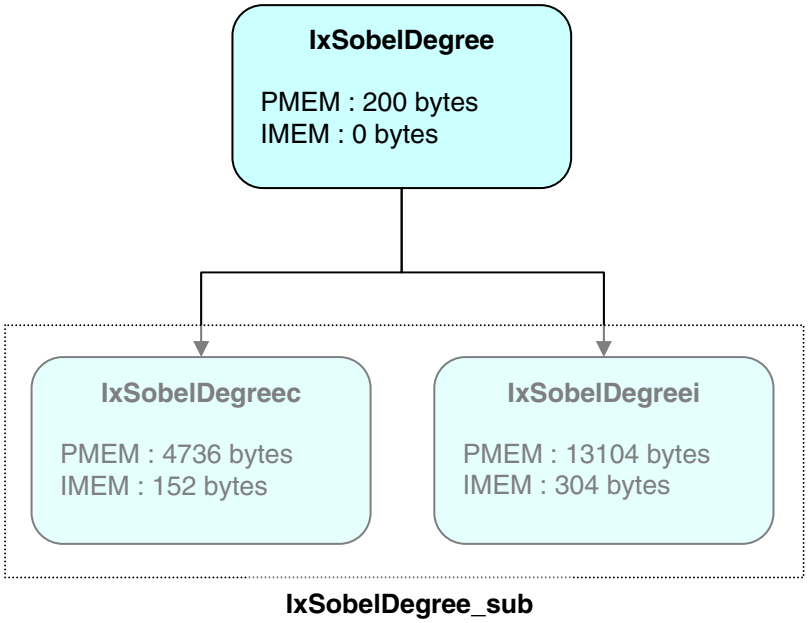
Type of output

- src: 2-dimensional array -> sep void *
- deg: 2-dimensional array -> sep uchar *

Example

Image input	Result
	

Dependency diagram



Time performance

12472 steps with 128 PE and for a 128*128 image

3.1.12 IxPercentileFilter – Percentile filter

Syntax

void IxPercentileFilter(sep void * src, int bytes, int percent, uint lines)

Description

Performs percentile filter upon src.

For each pixel, the intensity level of all the 3x3 neighbourhood pixels are first sorted.

Then the difference in intensity of the pixel at the "percent" % and "100-percent" % replaces the intensity of the central pixel.

- src is used as source and destination image
- bytes is the number of bytes per pixel for the source image
- lines is the number of lines of the source image

“percent” can take the values: 10, 20, 30 or 40.

For src = [0 1 2 , PercentileFilter(src,20,3) returns [0 0 0

3 4 5 0 6 0

6 7 8] 0 0 0]

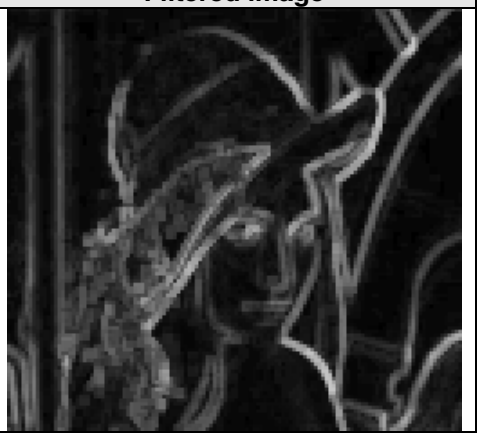
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- percent: scalar -> int
- lines: scalar -> uint

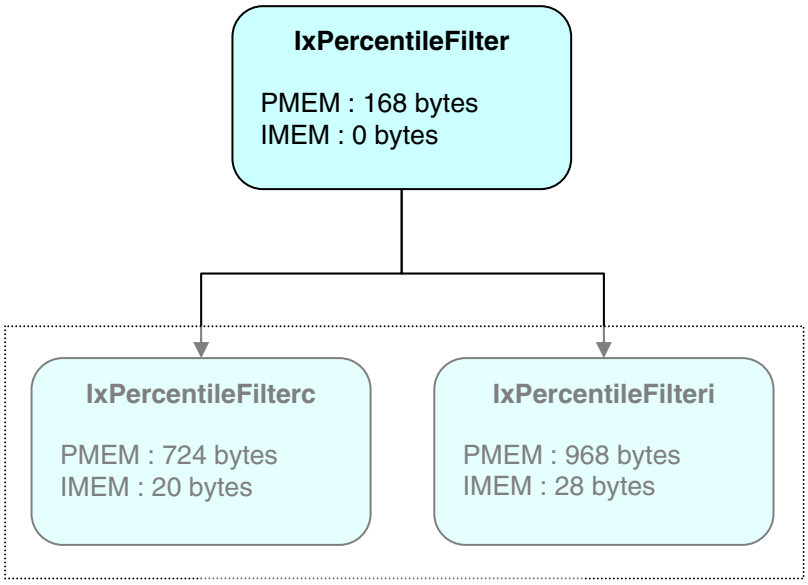
Type of output

- src: 2-dimensional array -> sep void *

Example

Image input	Filtered image
	

Dependency diagram



IxPercentileFilter_sub

Time performance

50281 steps with 128 PE and for a 128*128 image

3.1.13 IxCanny – Canny filter

Syntax

void IxCanny(sep void * src, sep void * dst, int bytes, uint zeros, uint thres, uint lines)

Description

Performs the Canny operator, namely, it calculates the ratio between the second directional derivative and the gradient magnitude.

- *src* and *dst* are respectively the source and destination images
- *bytes* is the number of bytes per pixel of the source image
- *thres* is the threshold used for edge detection
- if equal to zero, *zeros* provides the second directional derivative only
- else *zeros* calculate edges
- *lines* is the number of lines to process

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint
- thres: scalar -> uint
- zeros: scalar -> uint

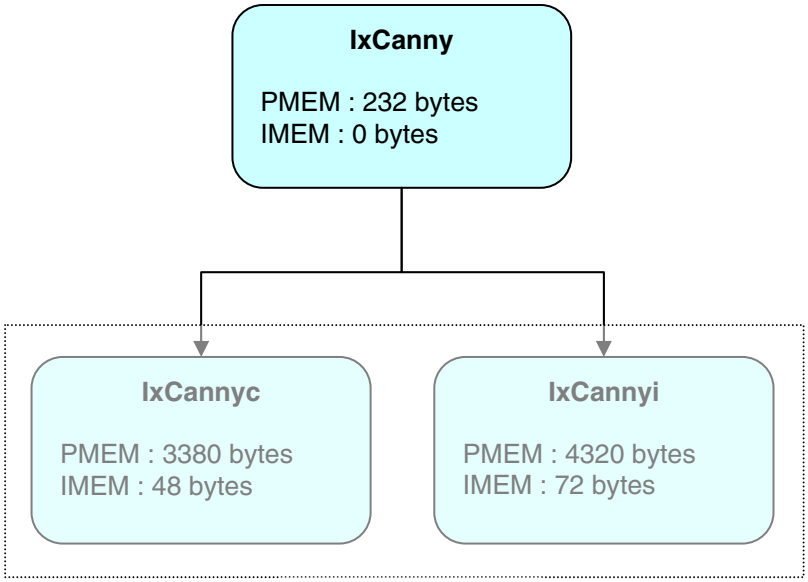
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Filtered image
	

Dependency diagram



Time performance

36341 steps with 128 PE and for a 128*128 image

3.2 Noise reduction

3.2.1 IxAverageFilter – Averaging filter

Syntax

void IxAverageFilter(sep void * src, int bytes, uint lines)

Description

It performs the average filter between the current pixel and its 8 neighbors by convoluting the image with the matrix below.

- *src* is the source image
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines.

$$\text{Output} = 1/16 * \text{abs} \left(\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \text{Input} \right)$$

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

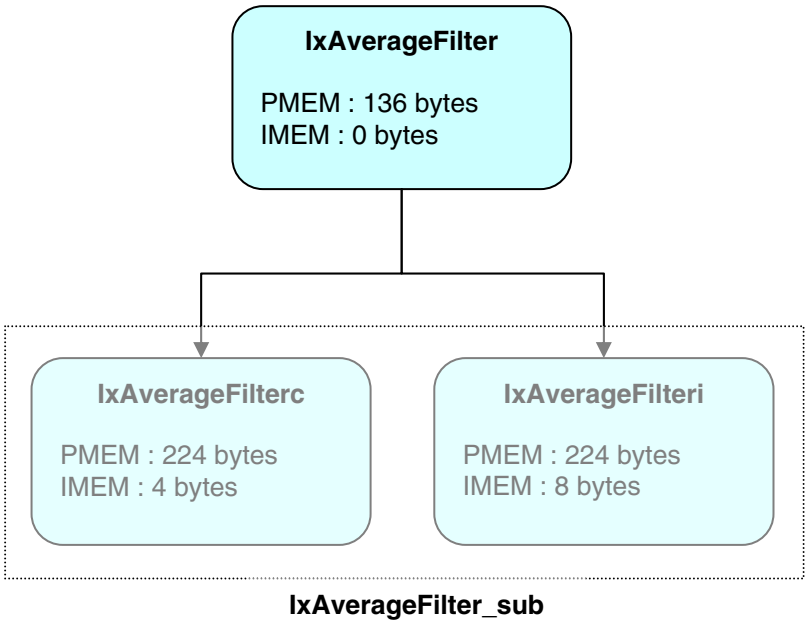
Type of output

- src: 2-dimensional array -> sep void *

Example

Image input	Filtered image
	

Dependency diagram



Time performance

1766 steps with 128 PE and for a 128*128 image

3.2.2 IxMedian3x3 – 3-dimensional median filter

Syntax

void IxMedian3x3(sep void * src, sep void * dst, int bytes, uint lines)

Description

Performs 3-dimensional median filter.

Example: Input = [4 2 8 / 9 1 7 / 3 6 5] => Output = [0 0 0 / 0 5 0 / 0 0 0]

- *src* is the source image
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines of the source image
- *dst* is the destination image

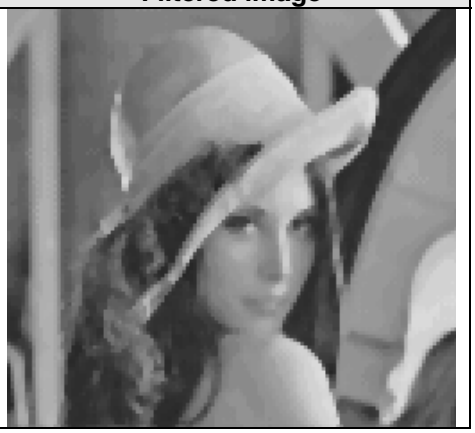
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

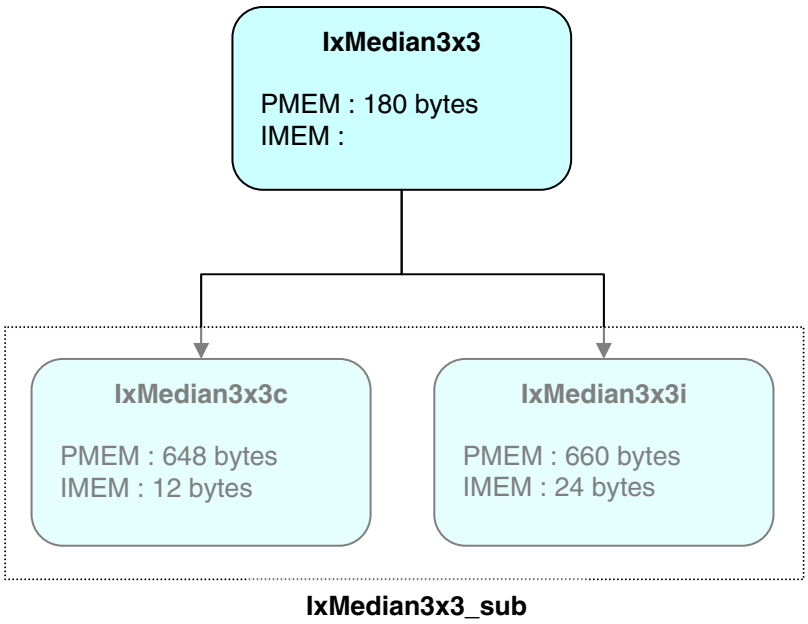
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Filtered image
	

Dependency diagram



IxMedian3x3_sub

Time performance

3197 steps with 128 PE and for a 128*128 image

3.2.3 IxSmoothing – Smoothing filter

Syntax

void IxSmoothing(sep void * src, sep void * dst, int bytes, uint lines, uchar mask)

Description

Smooth an image using a masksize * masksize neighborhood averaging, with edge preserving

- *src* and *dst* are respectively the source and destination images
- *bytes* is the number of bytes per pixel for the images
- *lines* is the number of lines to process
- *mask* is the mask size (equal to 1, 3 or 5, …)

Type of inputs

- src: 2-dimensional array -> sep void *
- lines: scalar -> uint
- mask: scalar -> uchar

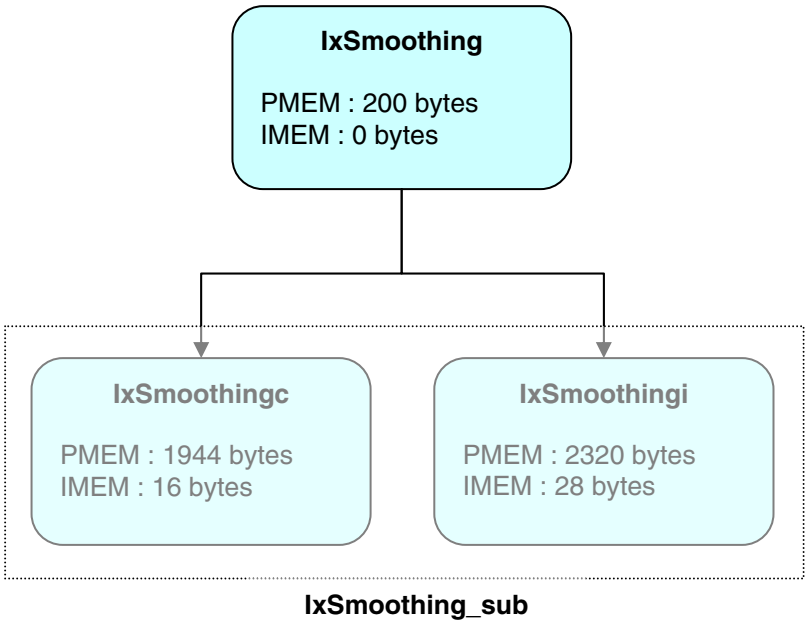
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Filtered image
	

Dependency diagram



IxSmoothing_sub

Time performance

17620 steps with 128 PE and for a 128*128 image

3.2.4 IxGuassian3 – 3-dimensional Guassian filter

Syntax

void IxGuassian3(sep void * src, sep void * dst, int bytes, uint lines)

Description

Performs the 3*3 Guassian filter.

$$\text{Output} = \left(\begin{bmatrix} 19 & 32 & 19 \\ 32 & 51 & 32 \\ 19 & 32 & 19 \end{bmatrix} * \text{Input} \right) / 256$$

- *src* and *dst* are respectively the source and destination images
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines to process

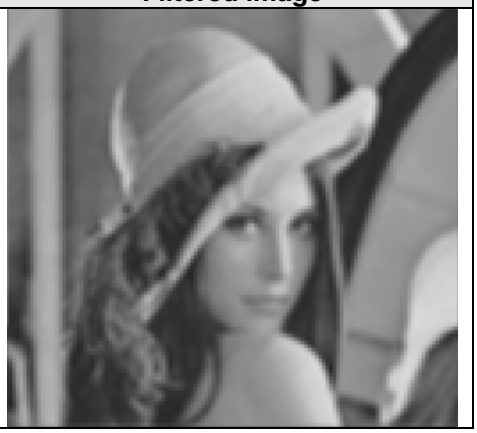
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

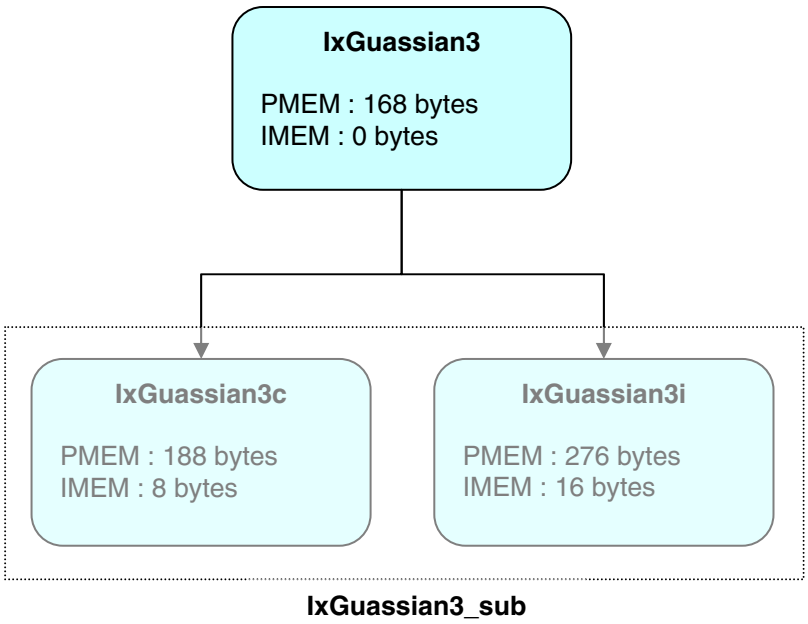
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Filtered image
	

Dependency diagram



IxGuassian3_sub

Time performance

2132 steps with 128 PE and for a 128*128 image

3.2.5 IxGuassian5 – 5-dimensional filter

Syntax

void IxGuassian5(sep void * src, sep void *dst, int bytes, uint lines)

Description

Performs the 5*5 Guassian filter.

$$\text{Output} = \left(\begin{bmatrix} 2 & 4 & 6 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 6 & 12 & 15 & 12 & 6 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 6 & 4 & 2 \end{bmatrix} * \text{Input} \right) / 163$$

- *src* and *dst* are respectively the source and destination images
- *bytes* is the number of bytes per pixel for the source image
- *lines* is the number of lines to process

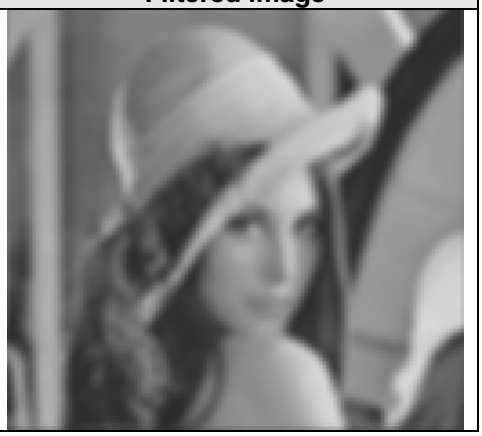
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

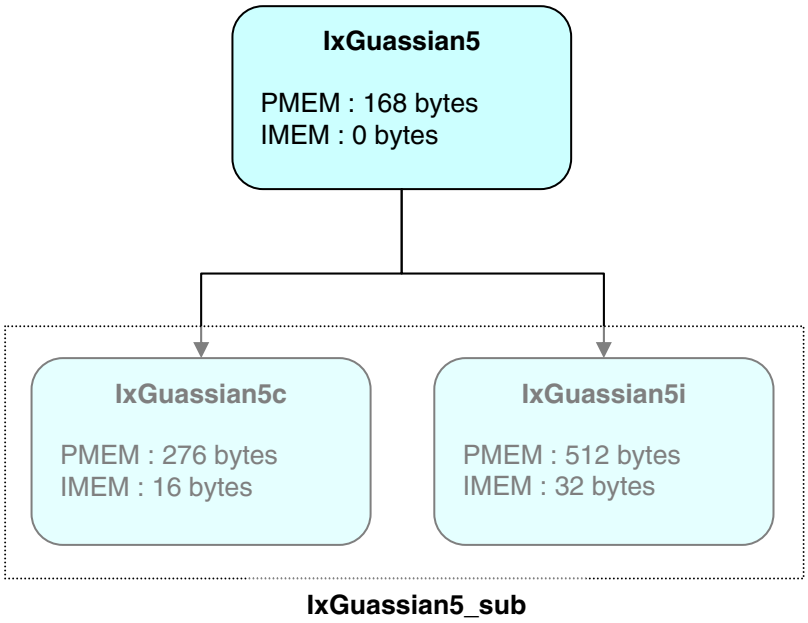
Type of output

- dst: 2-dimensional array -> sep void *

Example

Image input	Filtered image
	

Dependency diagram



Time performance

3733 steps with 128 PE and for a 128*128 image

3.2.6 IxPnSRemoving – Pepper and Salt removing filter

Syntax

void IxPnSRemoving(sep void * img, int bytes, uchar lim1, uchar lim2, uint lines)

Description

Removes pepper and salt noise from an image

The function supposes a binary (0/255) input image *img* of *lines* number of lines.

bytes is the number of bytes per pixel for the source image

It looks for every pixel how many are non-zero:

- if this number is smaller than *lim1* the pixel is zeroed, thus removing salt noise (white pixels in black area).
- if it is more than *lim2*, the pixel is set to 255 too, thus removing pepper noise (black pixels in a white area).

Limitations

This function works well with binary image only

Type of inputs

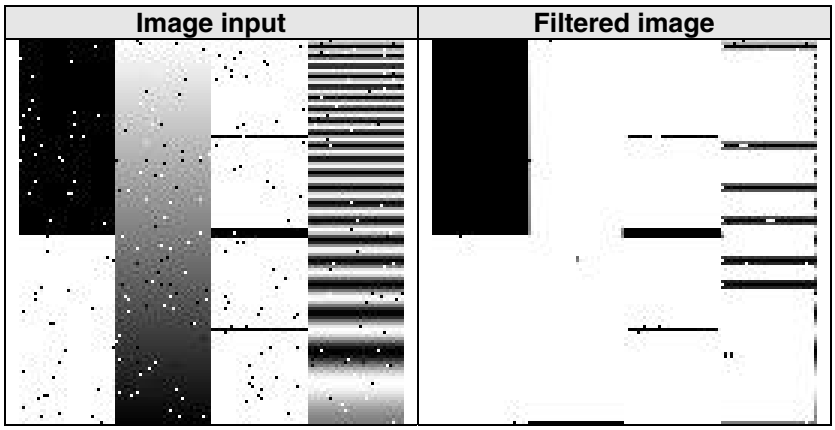
- img: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint
- lim1: scalar -> uchar
- lim2: scalar -> uchar

Type of output

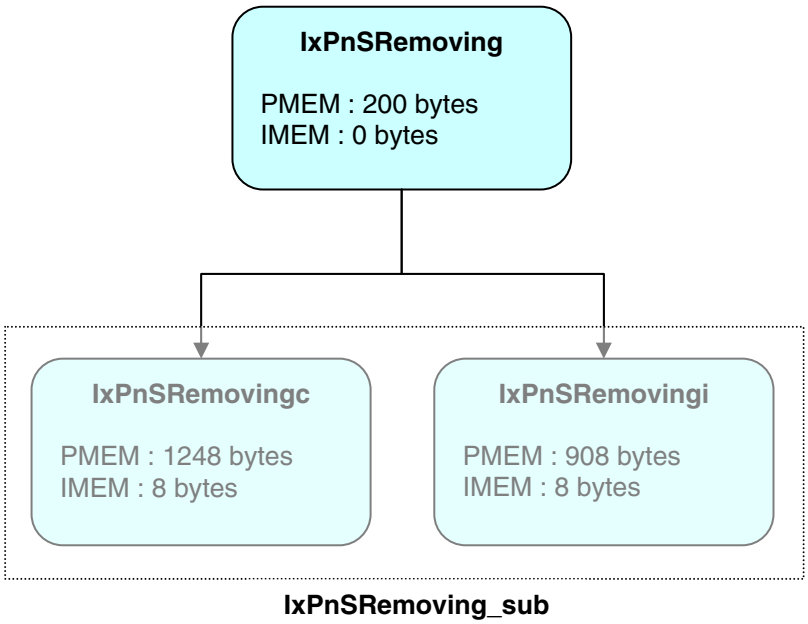
- img: 2-dimensional array -> sep void *

Example

For lim1 = 2 and lim2 = 6:



Dependency diagram



IxPnSRemoving_sub

Time performance

5981 steps with 128 PE and for a 128*128 image

3.2.7 IxErosion – Image Erosion

Syntax

void IxErosion(sep void * src, int bytes, uint lines)

Description

Performs erosion operation upon binary source image src.

Every white pixel that is touching a black pixel is changed into a black pixel

- The result is overwritten into *src*
- *bytes* is the number of bytes per pixel for the source image.
- *lines* is the number of lines to process

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

Type of output

- src: 2-dimensional array -> sep void *

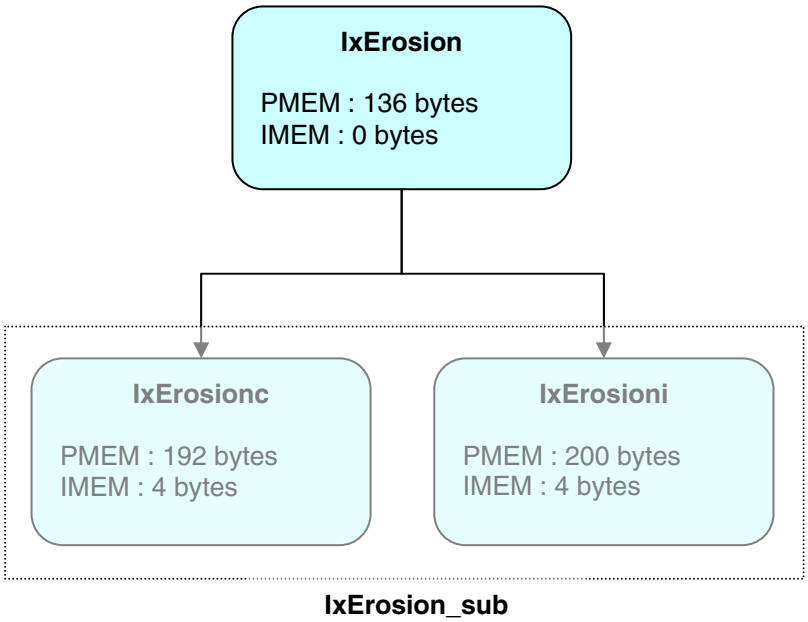
Limitations

This function works well on binary image input only

Example

Image input	Filtered image
	

Dependency diagram



Time performance

1671 steps with 128 PE and for a 128*128 image

3.2.8 IxDilation – Image Dilation

Syntax

void IxDilation(sep void * src, int bytes, uint lines)

Description

Performs dilation operation upon binary source image *src*.

Every black pixel that is touching a white pixel is changed into a white pixel

- The result is overwritten into *src*
- *bytes* is the number of bytes per pixel for the source image.
- *lines* is the number of lines to process

Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

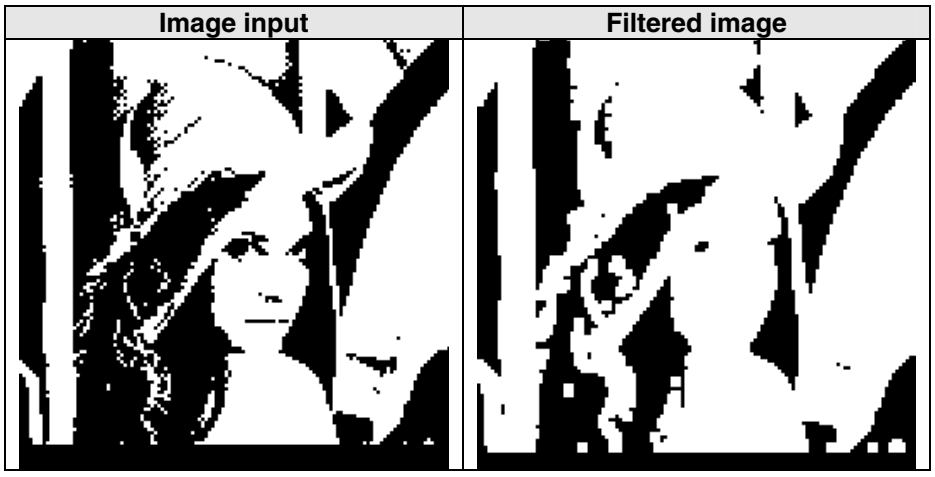
Type of output

- src: 2-dimensional array -> sep void *

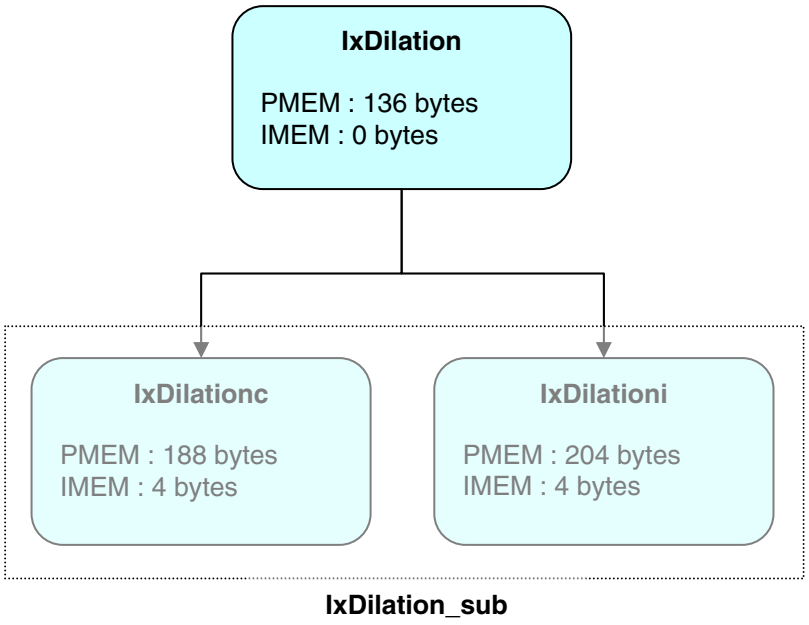
Limitations

This function works well on binary image input only

Example



Dependency diagram



Time performance

7042 steps with 128 PE and for a 128*128 image

4 Color space conversions

4.1 IxRGB2HSI, IxHSI2RGB – RGB and HSI conversions

Syntax

void IxRGB2HSI(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

void IxHSI2RGB(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

Description

Note that HSI color space is also known as HSV (hue, saturation, value).

The first function converts RGB image to HSI image using the following equations:

$$h_i = \left\lfloor \frac{h}{60} \right\rfloor \mod 6$$
$$f = \frac{h}{60} - \left\lfloor \frac{h}{60} \right\rfloor$$
$$p = v \times (1 - s)$$
$$q = v \times (1 - f \times s)$$
$$t = v \times (1 - (1 - f) \times s)$$

$$(r, g, b) = \begin{cases} (v, t, p), & \text{if } h_i = 0 \\ (q, v, p), & \text{if } h_i = 1 \\ (p, v, t), & \text{if } h_i = 2 \\ (p, q, v), & \text{if } h_i = 3 \\ (t, p, v), & \text{if } h_i = 4 \\ (v, p, q), & \text{if } h_i = 5 \end{cases}$$

The second function converts HSI image to RGB image using the following equations:

$$h = \begin{cases} 0 & \text{if } \max = \min \\ (60° \times \frac{g-b}{\max - \min} + 0°) \bmod 360°, & \text{if } \max = r \\ 60° \times \frac{b-r}{\max - \min} + 120°, & \text{if } \max = g \\ 60° \times \frac{r-g}{\max - \min} + 240°, & \text{if } \max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } \max = 0 \\ \frac{\max - \min}{\max} = 1 - \frac{\min}{\max}, & \text{otherwise} \end{cases}$$

$$v = \max$$

- *img1* is used as R- and H-components
- *img2* is used as G- and S-components
- *img3* is used as B- and I-components
- *lines* is the number of lines to process

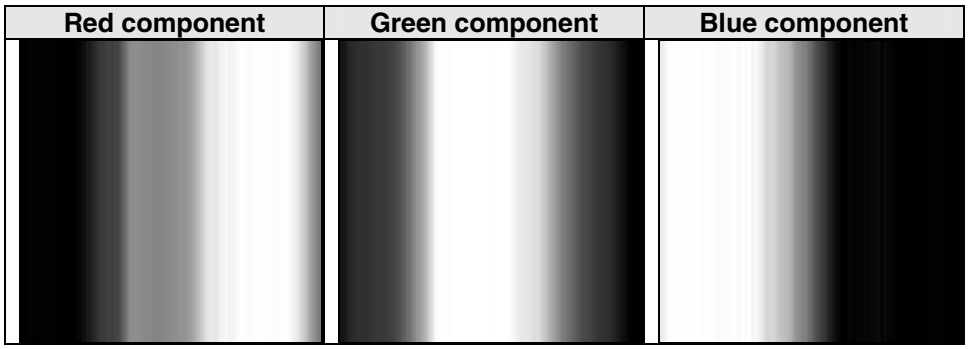
Type of inputs

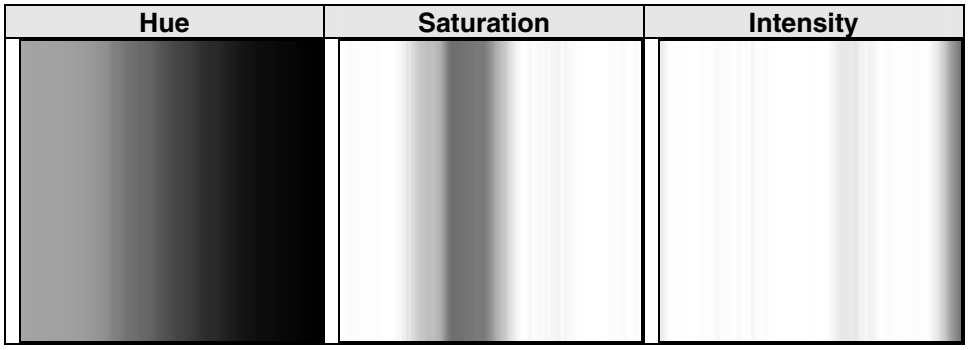
- img1: 2-dimensional array -> sep uchar *
- img2: 2-dimensional array -> sep uchar *
- img3: 2-dimensional array -> sep uchar *
- lines: scalar -> uint

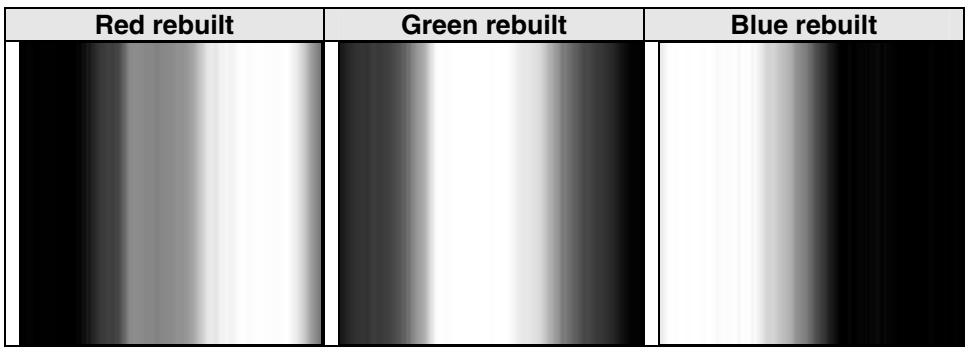
Type of outputs

- img1: 2-dimensional array -> sep uchar *
- img2: 2-dimensional array -> sep uchar *
- img3: 2-dimensional array -> sep uchar *

Example







Memory consumption

	PMEM	IMEM
RGB to HSI	852 bytes	8 bytes
HSI to RGB	760 bytes	28 bytes

Time performance

Number of steps for a 128*128 image with 128 PE:

Function	Steps
RGB to HSI	15301
HSI to RGB	6728

4.2 IxRGB2XYZ, IxXYZ2RGB – RGB and XYZ conversions

Syntax

void IxRGB2XYZ(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

void IxXYZ2RGB(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

Description

The first function converts RGB image to XYZ image using the following transformation matrix:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{0,17967} * \begin{bmatrix} 0,49 & 0,31 & 0,20 \\ 0,17697 & 0,81240 & 0,01063 \\ 0,00 & 0,01 & 0,99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The second function converts XYZ image to RGB image using the inverse of the matrix above.

- *img1* is used for R and X components.
- *img2* is used for G and Y components
- *img3* is used for B and (X+Y+Z) components
- *lines* is the number of lines to process

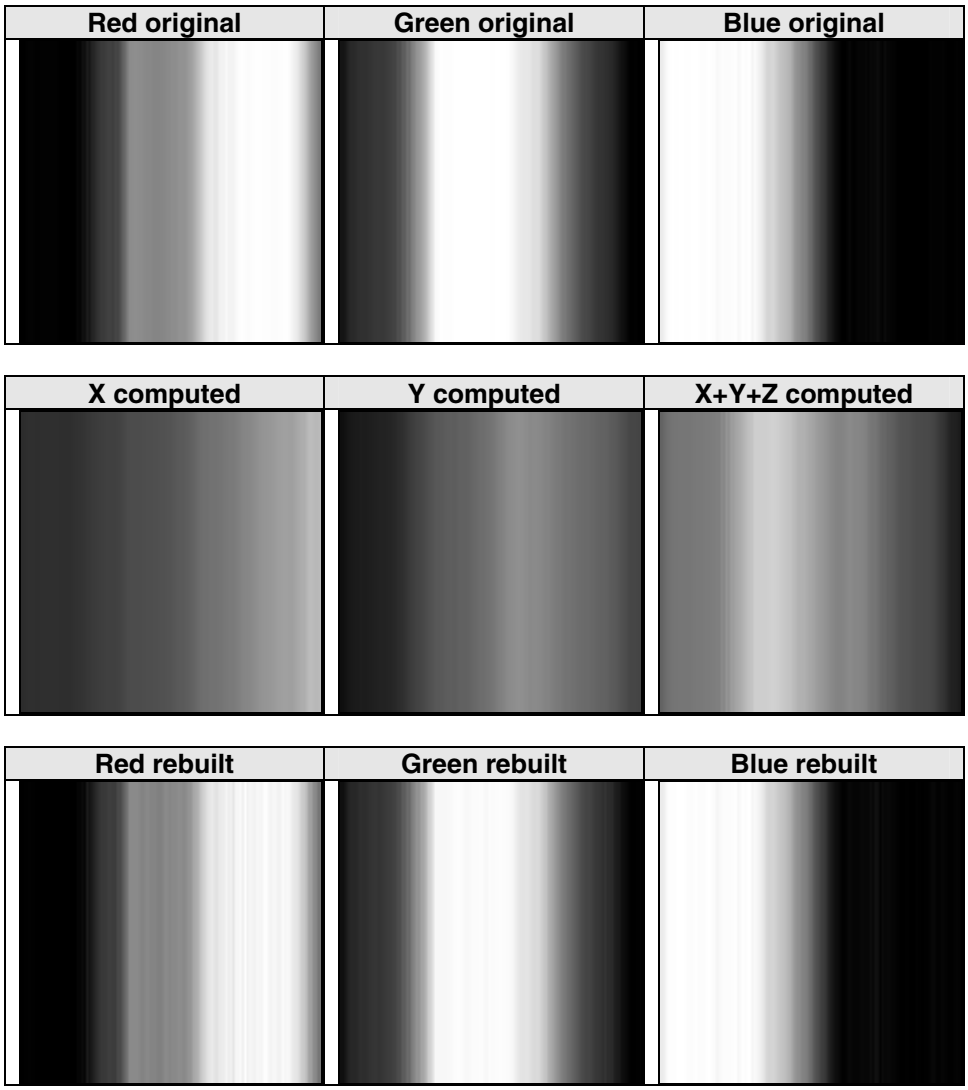
Type of inputs

- img1: 2-dimensional array -> sep uchar *
- img2: 2-dimensional array -> sep uchar *
- img3: 2-dimensional array -> sep uchar *
- lines: scalar -> uint

Type of outputs

- img1: 2-dimensional array -> sep uchar *
- img2: 2-dimensional array -> sep uchar *
- img3: 2-dimensional array -> sep uchar *

Example



Memory consumption

	PMEM	IMEM
RGB to XYZ	392 bytes	8 bytes
XYZ to RGB	552 bytes	12 bytes

Time performance

Number of steps for a 128*128 image with 128 PE:

Function	Steps
RGB to XYZ	11664
XYZ to RGB	6849

4.3 IxRGB2YC, IxYC2RGB, IxYC2R, IxYC2G, IxYC2B – RGB and Luma, Chroma conversions

Syntax

void IxRGB2YC(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

void IxYC2RGB(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

void IxYC2R(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

void IxYC2G(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

void IxYC2B(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

Description

The first function converts RGB image to YC image using the following equations:

Y = 0.299 * R + 0.587 * G + 0.114 * B

Cb = -0.1687 * R - 0.3313 * G + 0.5 * B

Cr = 0.5 * R - 0.4187 * G - 0.0813 * B

Then, C is coded so that Cr is on odd lines and Cb on even lines.

The other functions convert YC image to RGB image using the following equations:

R = Y + 1.402 * Cr

G = Y - 0.34414 * Cb - 0.71414 * Cr

B = Y + 1.772 * Cb

- *img1* is used for R and Y components.
- *img2* is used for G and C components
- *img3* is used for B component
- *lines* is the number of lines to process

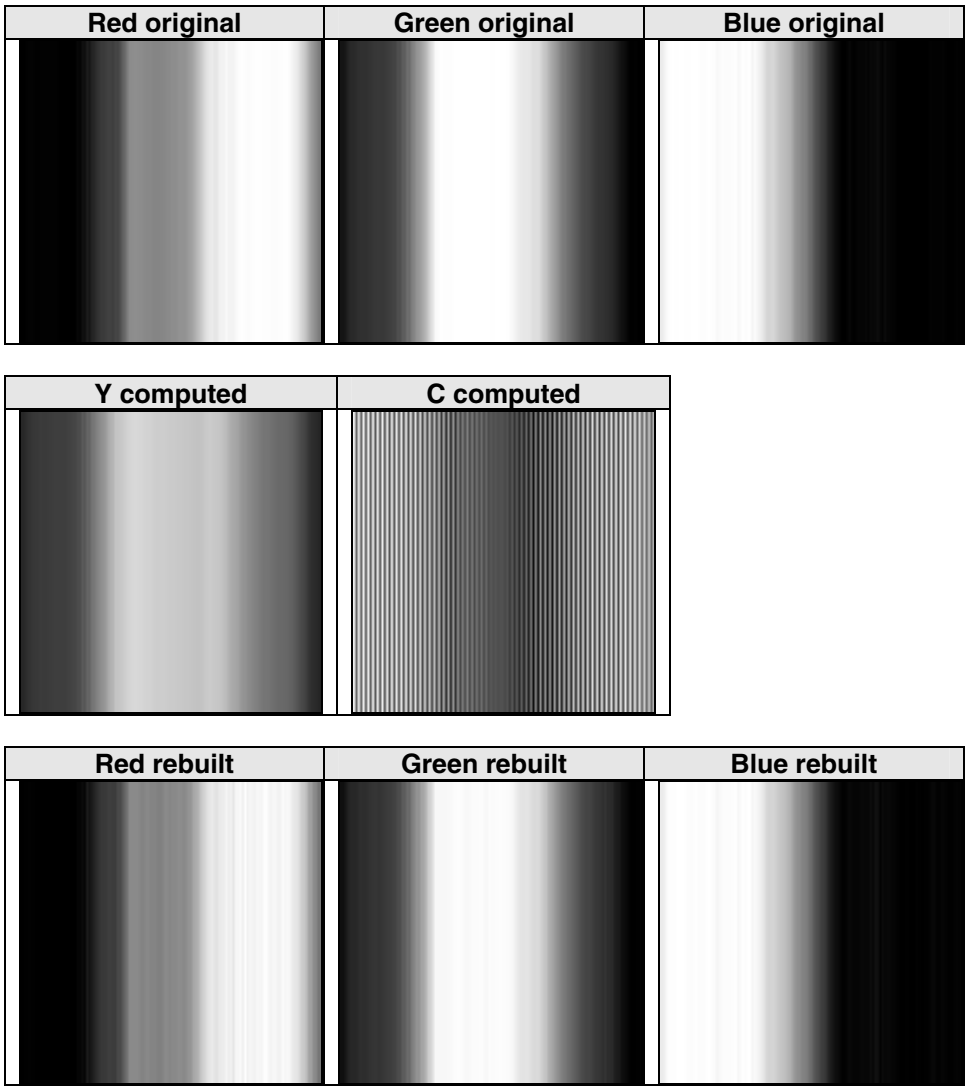
Type of inputs

- img1: 2-dimensional array -> sep uchar *
- img2: 2-dimensional array -> sep uchar *
- img3: 2-dimensional array -> sep uchar *
- lines: scalar -> uint

Type of outputs

- img1: 2-dimensional array -> sep uchar *
- img2: 2-dimensional array -> sep uchar *
- img3: 2-dimensional array -> sep uchar *

Example



Memory consumption

	PMEM	IMEM
RGB to YC	404 bytes	8 bytes
YC to RGB	596 bytes	12 bytes
YC to R	272 bytes	8 bytes
YC to G	320 bytes	8 bytes
YC to B	272 bytes	8 bytes

Time performance

Number of steps for a 128*128 image with 128 PE:

Function	Steps
RGB to YC	3969
YC to RGB	7596
YC to R	3191
YC to G	3576
YC to B	3170

4.4 IxYC2YCrCb, IxYCrCb2YC – Luma, Chroma conversions

Syntax

void IxYCrCb2YC(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

void IxYC2YCrCb(sep uchar * img1, sep uchar * img2, sep uchar * img3, uint lines)

Description

The first function converts (Y,Cr,Cb) components to (Y,C) components: C is coded with Cr on odd lines and Cb on even lines.

The second function does the inverse operation

- *img1* is used for Y component.
- *img2* is used for Cr and C components
- *img3* is used for Cb component
- *lines* is the number of lines to process

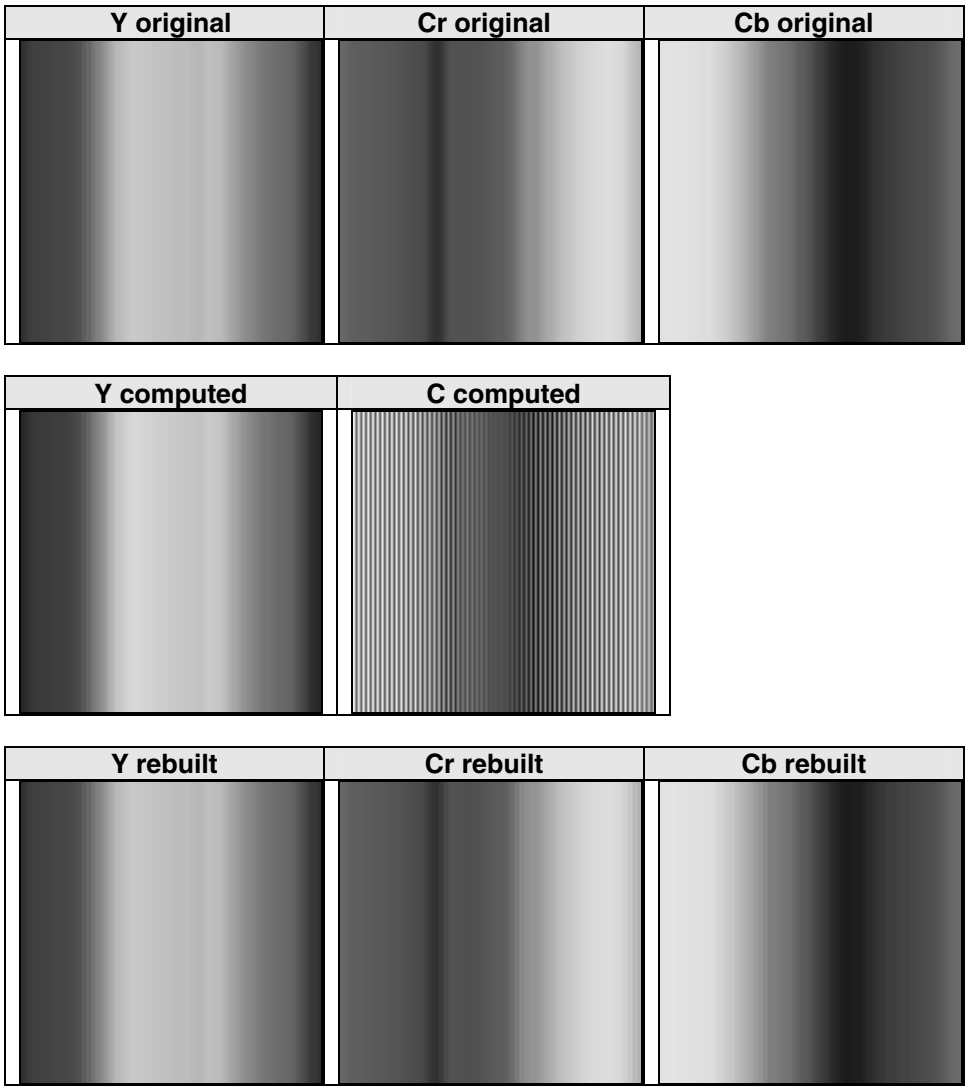
Type of inputs

- img1: 2-dimensional array -> sep uchar *
- img2: 2-dimensional array -> sep uchar *
- img3: 2-dimensional array -> sep uchar *
- lines: scalar -> uint

Type of outputs

- img1: 2-dimensional array -> sep uchar *
- img2: 2-dimensional array -> sep uchar *
- img3: 2-dimensional array -> sep uchar *

Example



Memory consumption

	PMEM	IMEM
Y,Cr,Cb to Y,C	196 bytes	4 bytes
Y,C to Y,Cr,Cb	212 bytes	8 bytes

Time performance

Number of steps for a 128*128 image with 128 PE:

Function	Steps
Y,Cr,Cb to Y,C	2020
Y,C to Y,Cr,Cb	2148

5 Image analysis

5.1 IxHistogram – Histogram computation

Syntax

int IxHistogram(sep uchar * src, sep ulong * hist, uint lines, uint off)

int IxHistogrami(sep uint * img, sep ulong * hist, uint maxlabel, uint lines, uint off)

Description

These functions calculate the grey level distribution of the source image.

- In case of 8-bits image, use IxHistogram:

grey levels from 0 to 127: histogram result is stored in the PE value of hist[0] which has the same PE number.

grey levels from 128 to 255: histogram result is stored in the PE value of hist[1] which has the PE number equal to “grey level substracted by 128”

- In case of more than 8-bits per pixel, use IxHistogrami:

For grey levels from 128*i to 128*i+127, histogram result is stored in the PE value of hist[i] which has the PE number equal to “grey level substracted by 128*i”

- *src* is the source image
- *hist* is the resulting histogram
- *lines* represents the number of lines to process
- maxlabel represents the maximum greylevel of the source image
- off is an offset

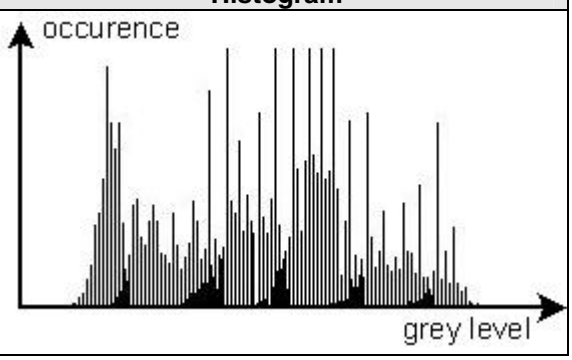
Type of inputs

- src: 2-dimensional array -> sep uchar *
- lines: scalar -> uint

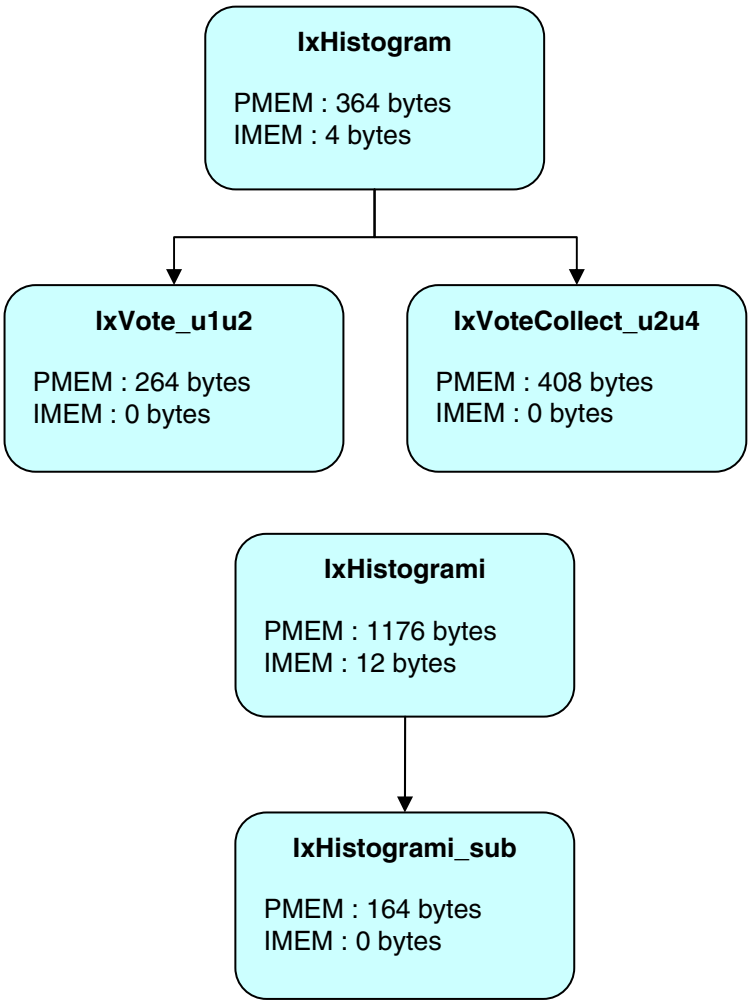
Type of output

- hist: 2-dimensional array -> sep ulong *

Example

Initial Image	Histogram
	

Dependency diagrams



Time performance

3560 steps with 128 PE and for a 128*128 image

5.2 IxHistNorm – Histogram equalization

Syntax

void IxHistNorm(sep uchar * src, uint lines)

Description

This function corrects the input image by normalizing its histogram.

This type of histogram spreading uses the following formula:

$$y = \mathrm{Sum}_x(x)$$

where x and y represents respectively the old and new pixel values.

The Sum function is defined as:

$$\mathrm{Sum}_x(i) = \sum_{j=0}^{i} p_x(j)$$

Where Px(i) represents the probability of occurrence of the grey level i:

$$p_x(i) = \frac{n_i}{n}$$

where n_i is the number of occurrence of the grey level i and n the total number of pixel in the image

- *src* is used as source and destination image
- *lines* represents the number of lines to process

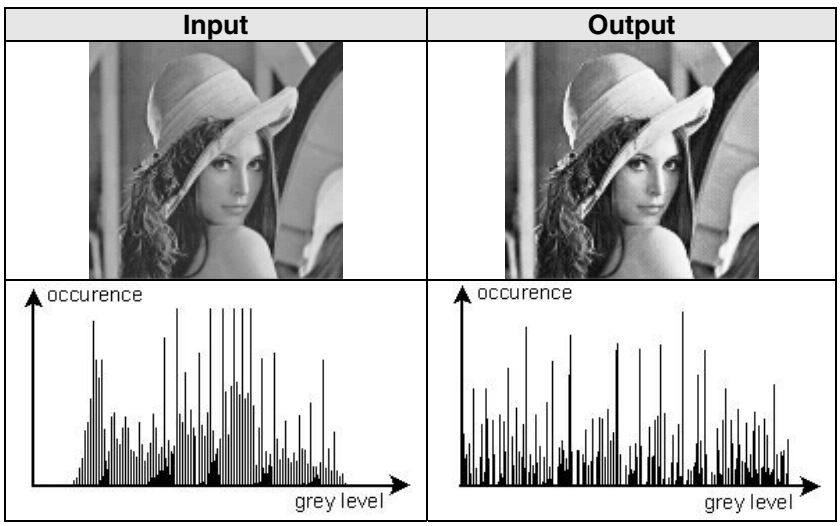
Type of inputs

- src: 2-dimensional array -> sep uchar *
- lines: scalar -> uint

Type of output

- src: 2-dimensional array -> sep uchar *

Example



Memory consumption

Memory type	Bytes
PMEM	900 bytes
IMEM	8 bytes

Time performance

12646 steps with 128 PE and for a 128*128 image

5.3 IxHistStretch – Histogram stretching

Syntax

void IxHistStretch(sep uchar * src, sep uchar * wrk, uint lines)

Description

This function performs histogram stretching.

Histogram is first computed, then histogram values are stretched according to the following formula:

$$New_{greylevel} = \frac{Old_{greylevel} - min}{max - min}$$

where min and max are respectively the minimum and maximum grey level values of the old histogram

- *src* is used as source and destination image
- *wrk* is the temporary workspace to compute the new grey level values
- *lines* represents the number of lines to process

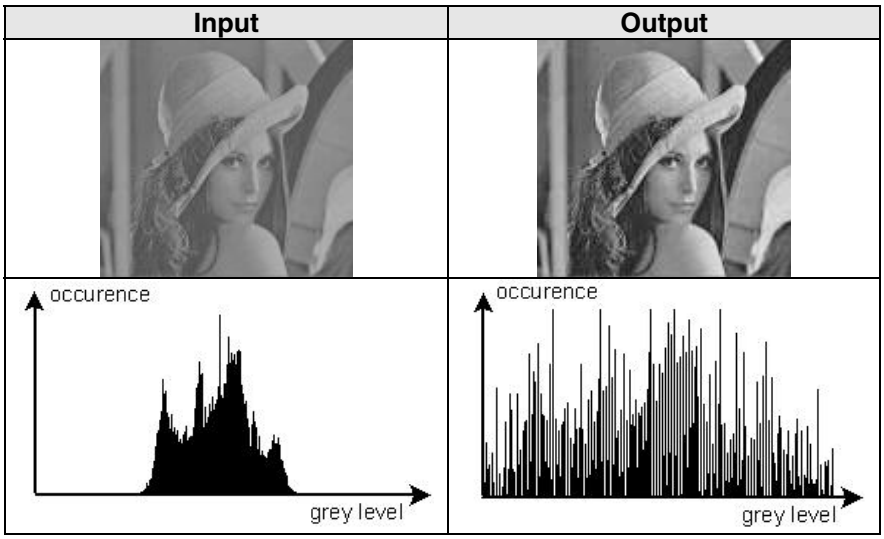
Type of inputs

- src: 2-dimensional array -> sep uchar *
- lines: scalar -> uint

Type of output

- src: 2-dimensional array -> sep uchar *
- wrk: 2-dimensional array -> sep uchar *

Example



Memory consumption

Memory type	Bytes
PMEM	1292 bytes
IMEM	16 bytes

Time performance

19867 steps with 128 PE and for a 128*128 image

5.4 IxFFT – Fast Fourier Transform

Syntax

void IxFFT(int inverse, sep long * real, sep long * imag, int lines)

Description

Performs N elements 1D-FFT or 1D-IFFT defined by the following formulas:

- FFT:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}$$

where k = 0, …, N-1

- IFFT:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{-\frac{2\pi i}{N} kn}$$

where n = 0, …, N-1

N represents the number of coefficients to compute and is equal to the number *lines* of lines to process.

real and *imag* are used respectively as input/resulting real and imaginary part of the image

inverse is a flag equal to 0 if FFT should be performed, 1 if IFFT should be performed

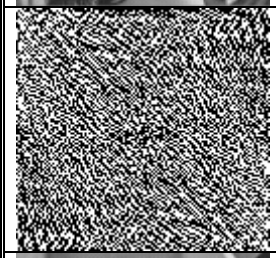
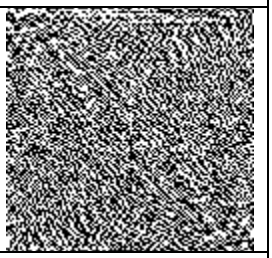
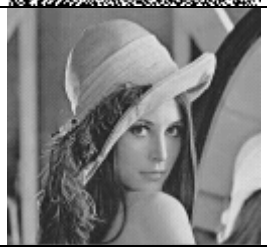
Type of inputs

- real: 2-dimensional array -> sep long *
- imag: 2-dimensional array -> sep long *
- inverse: scalar -> int
- lines: scalar -> int

Type of output

- real: 2-dimensional array -> sep long *
- imag: 2-dimensional array -> sep long *

Examples

	Real part	Imaginary part
Input		none
Output after 2D-FFT		
Output after 2D-IFFT		none

Memory consumption

Memory type	Bytes
PMEM	18240 bytes
IMEM	100 bytes

Time performance

116473 steps with 128 PE and for a 128*128 image

5.5 IxDistanceTransform – Distance Transform

Syntax

void IxDistanceTransform(sep void * src, int bytes, uint xsize, uint lines, uint diagonal, uint straight)

Description

IxDistanceTransform performs Distance Transform (see references below) based on a Recursive Neighborhood Operation (RNO)

Maximum resulting distance value is always below 256.

- *src* is used as source and destination image (*src* must be a binary image)
- *bytes* is the number of bytes per pixel for the source image.
- *xsize* represents the width of the source image.
- *lines* is the number of lines of the source image.
- *straight* represents the 4-connected neighborhood distance
- *diagonal* represents the 8-connected neighborhood distance

Type of inputs

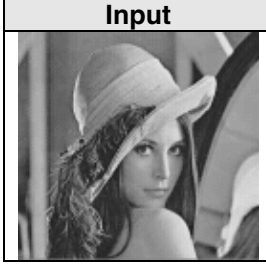
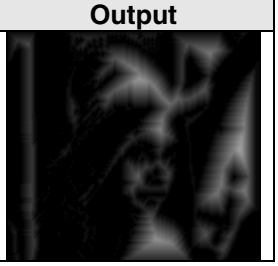
- src: 2-dimensional array -> sep uchar * or sep uint *
- bytes: scalar -> int
- xsize: scalar -> uint
- lines: scalar -> uint
- straight: scalar -> uint
- diagonal: scalar -> uint

Type of output

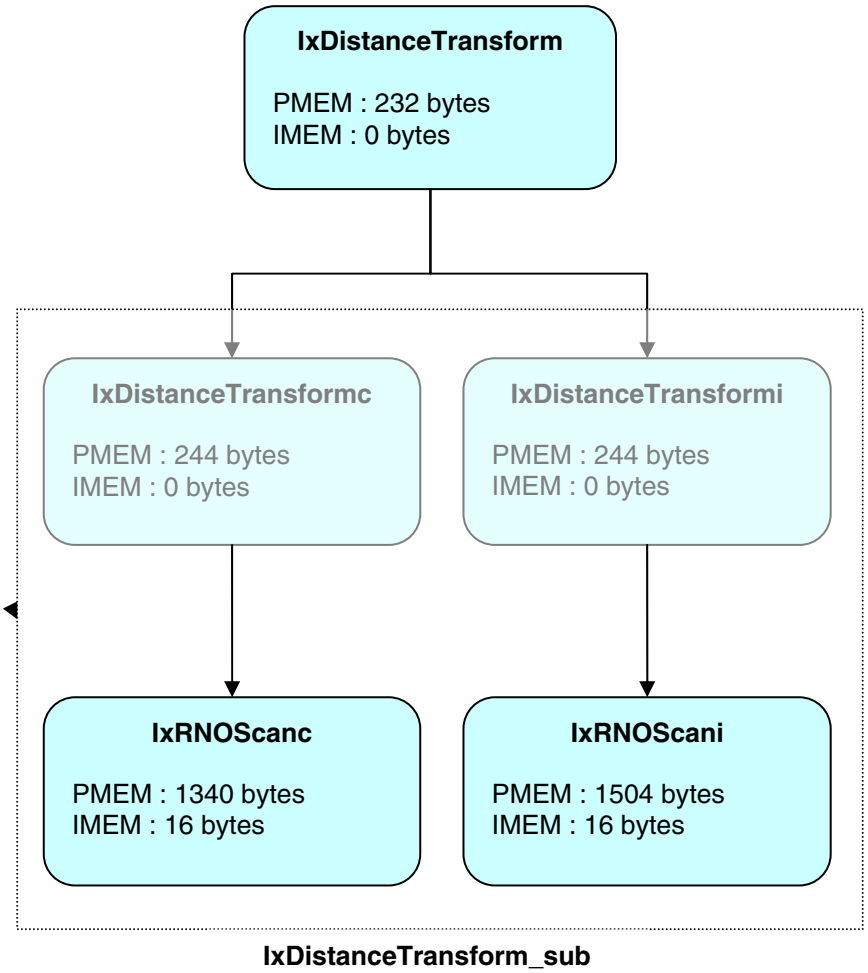
- src: 2-dimensional array -> sep uchar * or sep uint *

Example

With straight = 20 and diagonal =10, we obtain:

Input	Output
	

Dependency diagram



Time performance

39410 steps with 128 PE and for a 128*128 image

5.6 IxThinning – Line thinning

Syntax

void IxThinning(sep void * src, sep void * stack, sep void * mtbl, int bytes, uint lines)

Description

This function performs thinning on a binary image, i.e. any line wider than one pixel becomes one pixel wide.

- *src* is used as source and destination image (*src* must be a binary image)
- *stack[lines]* and *mtbl[256]* are used as work areas.
- *bytes* is the number of bytes per pixel for the source image.
- *lines* is the number of lines of the source image.

Type of inputs

- src: 2-dimensional array -> sep uchar * or sep uint *
- bytes: scalar -> int
- lines: scalar -> uint

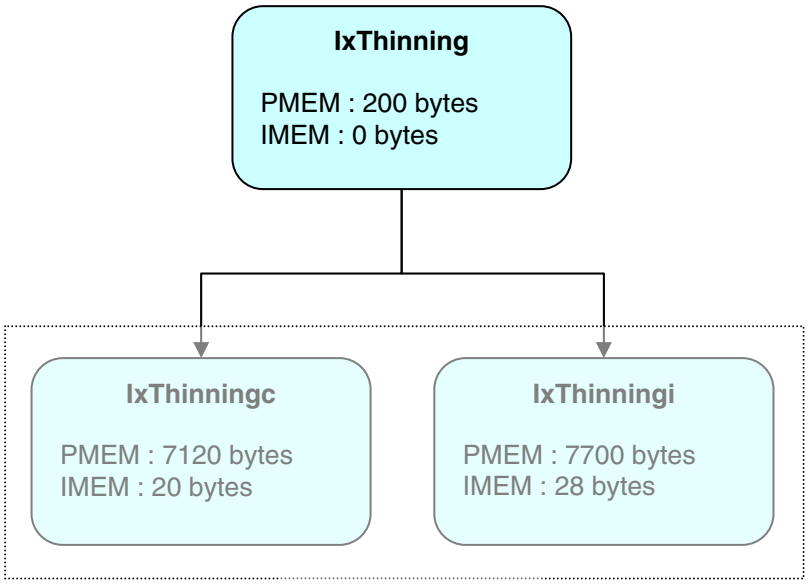
Type of output

- src: 2-dimensional array -> sep uchar * or sep uint *
- stack: 2-dimensional array -> sep uchar * or sep uint *
- mtbl: 2-dimensional array -> sep uchar * or sep uint *

Example

Input	After a Sobel filter	After thinning
		

Dependency diagram



IxThinning_sub

Time performance

143642 steps with 128 PE and for a 128*128 image

6 Image segmentation

6.1 IxColorReduction – Color Reduction

Syntax

void IxColorReduction (sep uchar * img, uchar nr_levels, uchar method, uint lines)

Description

This function reduces the number of gray values in an image.

It uses one of the two methods described below depending on the value of *method*:

- o if *method* = 0 -> use histogram information to put each accumulated continuous grey level pixels into same label
- o if *method* = 1 -> Simply let 256/*nr_levels* grey values get same level (quantization)

The '+2' in the new levels is to allow for labeling, which does not work on 1 value pixels.

0 pixels stay 0.

- *img* is used as source and destination image
- *nr_levels* represents the number of destination levels
- *lines* is the number of lines to process

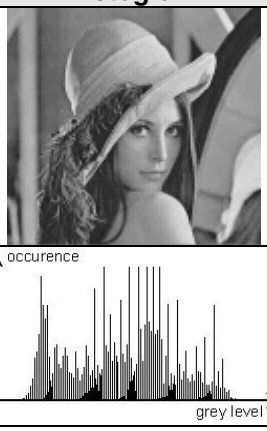
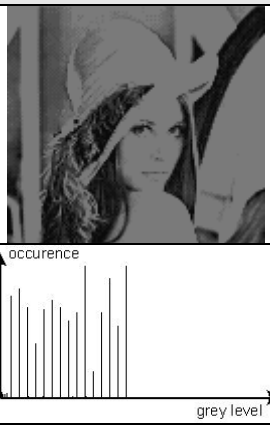
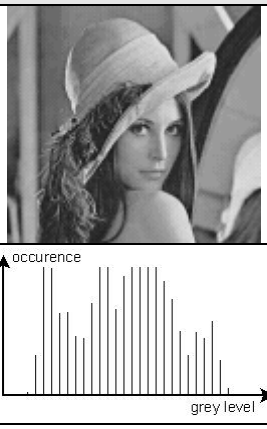
Type of inputs

- img: 2-dimensional array -> sep uchar *
- nr_levels: scalar -> uchar
- method: scalar -> uchar
- lines: scalar -> uint

Type of output

- img: 2-dimensional array -> sep uchar *

Examples

Input image and histogram	Results with 1st method	Result with 2nd method
		

Memory consumption

Memory type	Bytes
PMEM	2032 bytes
IMEM	256 bytes

Time performance

Number of steps for a 128*128 image with 128 PE:

Chosen method	Steps
1st method	20256
2nd method	10787

6.2 IxConnectCheck8, IxConnectCheck16 – Connection detection

Syntax

void IxConnectCheck8(sep void * src, int bytes, sep uchar * dir, uint lines)

void IxConnectCheck16(sep void * src, int bytes, sep uint * dir, uint lines)

Description

Check the connectivity of edge pixels and delete edge pixels which do not satisfied the following conditions:

- There exists at least one edge pixel:
  - in the 3x3 neighborhood whose direction is within 45 degrees of the central pixel's direction for IxConnectCheck8
  - in the 5x5 neighborhood whose direction is within 22.5 degrees of the central pixel's direction for IxConnectCheck16
- The central pixel possesses the largest value among edge pixels:
  - in the 3x3 neighborhood which has the same direction with the central pixel for IxConnectCheck8
  - in the 5x5 neighborhood which has the same direction with the central pixel for IxConnectCheck16
- *src* is used as source and destination image and represents the power of each edge pixel.
- *lines* is the number of lines to process
- *bytes* is the number of bytes of the source image.
- *dir* represents the direction information of each edge pixel, where each bit (from LSB to MSB) represents respectively:
  - n, ne, e, se, s, sw, w, and nw directions for IxConnectCheck8
  - n, nne, ne, nee, e, see, se, sse, s, ssw, sw, ssw, w, wwn, nw and nnw directions for IxConnectCheck16

Note that the mentioned direction is not the direction of the edge, but the direction of the intensity (from dark to bright).

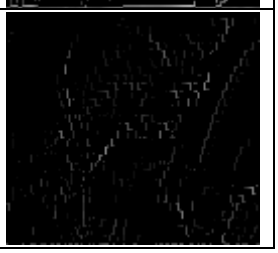
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- dir: 2-dimensional array -> sep uchar * or sep uint *
- lines: scalar -> uint

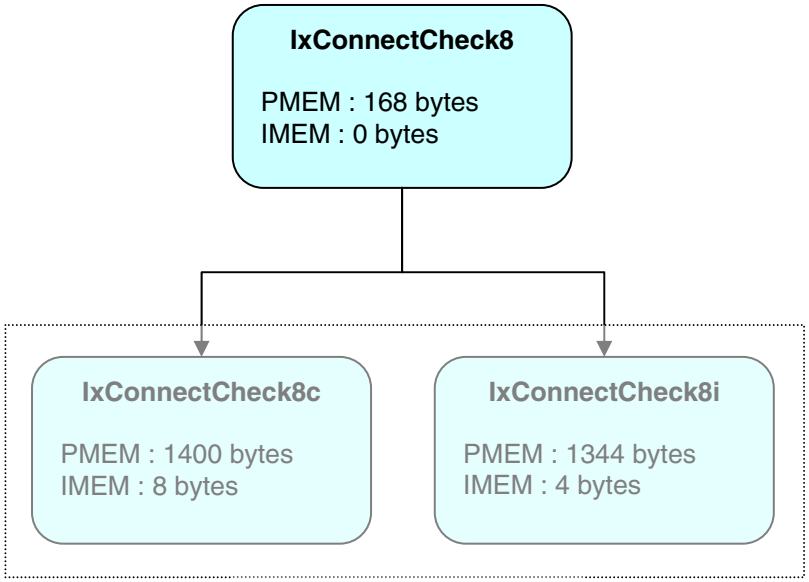
Type of output

- src: 2-dimensional array -> sep void *
- dir: 2-dimensional array -> sep uchar * or sep uint *

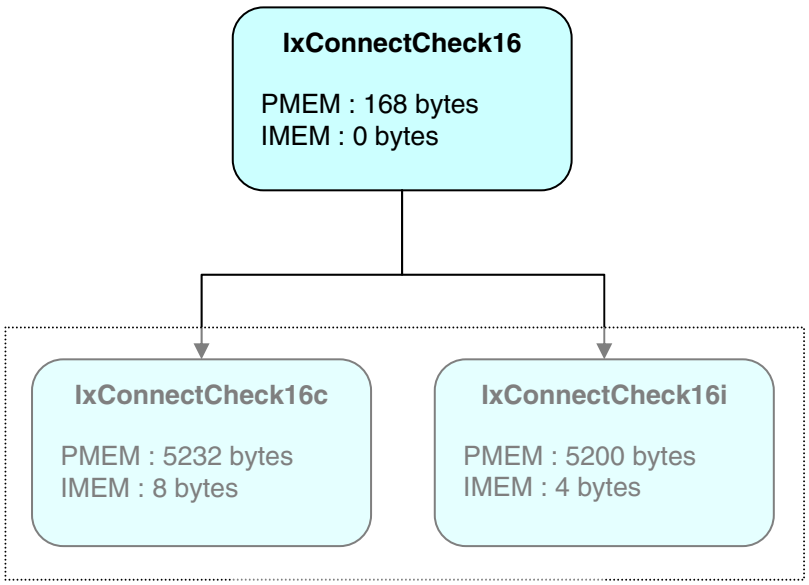
Examples

	Orignal image	Input edges	Final result
IxConnectCheck8			
IxConnectCheck16			

Dependency diagrams



IxConnectCheck8_sub



IxConnectCheck16_sub

Time performance

Number of steps for a 128*128 image with 128 PE

Function	Steps
IxConnectCheck8	4212
IxConnectCheck16	39024

6.3 IxDownBorder – Down border detection

Syntax

void IxDownBorder(sep void * src, int bytes, uint lines)

Description

This function extracts down borders of a binary image.

Pixels which do not belonging to down borders are erased.

- *src* is used as source and destination image
- *bytes* is the number of bytes of the source image
- *lines* is the number of lines to process

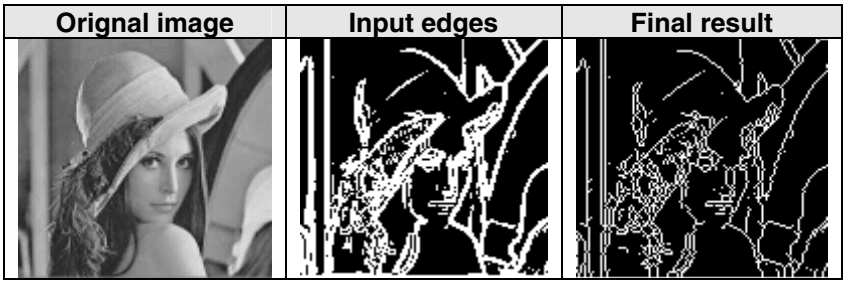
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

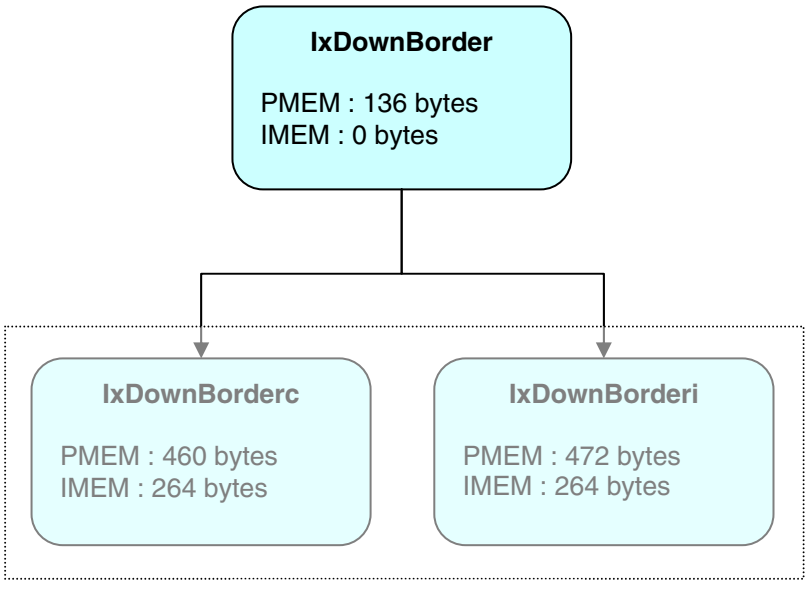
Type of output

- src: 2-dimensional array -> sep void *

Examples



Dependency diagram



IxDownBorder_sub

Time performance

4212 steps for a 128*128 image with 128 PE

6.4 IxLeftEnd – Left end detection

Syntax

void IxLeftEnd(sep void * src, int bytes, uint lines)

Description

This function selects left-end pixel of white areas from a binary image.

Selected pixels are set to 255 while remaining non-zero pixel are set to 128.

- *src* is used as source and destination image
- *bytes* is the number of bytes of the source image
- *lines* is the number of lines to process

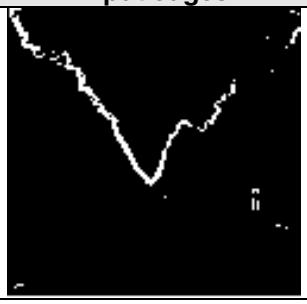
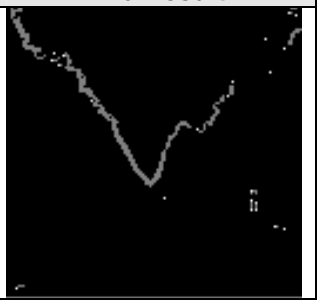
Type of inputs

- src: 2-dimensional array -> sep void *
- bytes: scalar -> int
- lines: scalar -> uint

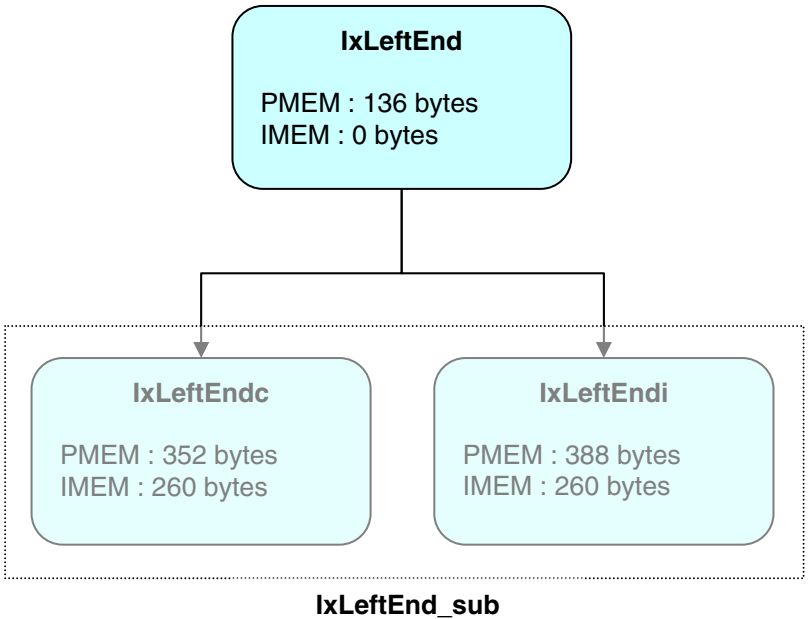
Type of output

- src: 2-dimensional array -> sep void *

Examples

Orignal image	Input edges	Final result
		

Dependency diagram



IxLeftEnd_sub

Time performance

2844 steps for a 128*128 image with 128 PE

6.5 IxGrowA, IxGrowB – Propagation functions

Syntax

void IxGrowA(sep uchar * src, sep uchar * dst, sep uchar * stack, int lines int * seedx, int * seedy, int nr_seed)

void IxGrowB(sep uchar * src, sep uchar * dst, sep uchar * stack, int lines int * seedx, int * seedy, int nr_seed)

Description

These functions propagate the value of seeds in an image.

- *src* and *dst* are respectively the source and destination images.
- *stack* is used as workspace.
- *lines* is the number of lines to process (should be < 256)
- *seedx* and *seedy* are arrays containing the x and y coordinates of *nr_seeds* starting points in the source image

For each of these seeds, the pixop function is called on PE number seedx, passing the source image and seedy.

The pixop function should return a byte, indicating in which of the 8 directions the seed should be propagated (i.e. which of its neighbors should be set to the seed value and become a new seed), according to the following table:

nw	n	ne	w	e	sw	s	se
8	7	6	5	4	3	2	1

Note that the correspondence between the bits and the pixels is like in pack8.

IxGrowA and IxGrowB propagates seeds to pixels in the destination image which have value 1 only.

For other values the result from pixop function is ignored.

Only pixels which have higher label than the seed are propagated for IxGrowA, while pixels with lower label are propagated for IxGrowB.

The returned value of the function is the number of time propagation which has taken place.

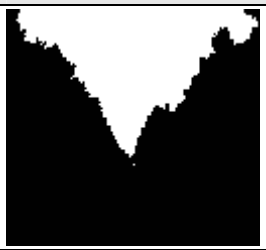
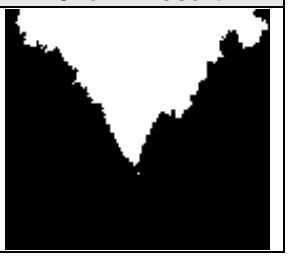
Type of inputs

- src: 2-dimensional array -> sep uchar *
- lines: scalar -> uint
- seedx: vertical array -> int *
- seedy: vertical array -> int *
- nr_seed: scalar -> int

Type of outputs

- dst: 2-dimensional array -> sep uchar *
- stack: 2-dimensional array -> sep uchar *

Examples

Original input	GrowA result	GrowB result
		

Memory consumption

	PMEM	IMEM
GrowA	3204 bytes	12 bytes
GrowB	8376 bytes	36 bytes

Time performance

Number of steps for a 128*128 image with 128 PE:

Functions	Steps
GrowA	20125
GrowB	32204

6.6 IxLabeling, IxPropLabel – Labeling function

Syntax

uint IxLabeling(sep uchar * src, uint lines)

int IxPropLabel(sep uchar * src, uint lines)

Description

Perform connected component labeling from a binary image and return 8 bit labels.

The different steps are as follows:

- extract down borders of each object
- extract left ends of the previous borders
- assign label to each left end
- propagate these labels to the down borders of each object using the second function IxPropLabel
- fill up the whole object with the corresponding labels

Note that, pixels on PE0 and PE255, and the one pixel thick border of every processing area specified by using mif ... will be ignored and set to zero.

- *src* is used as source and destination image
- *lines* is the number of lines to process
- The returned value corresponds to the total number of labeled objects

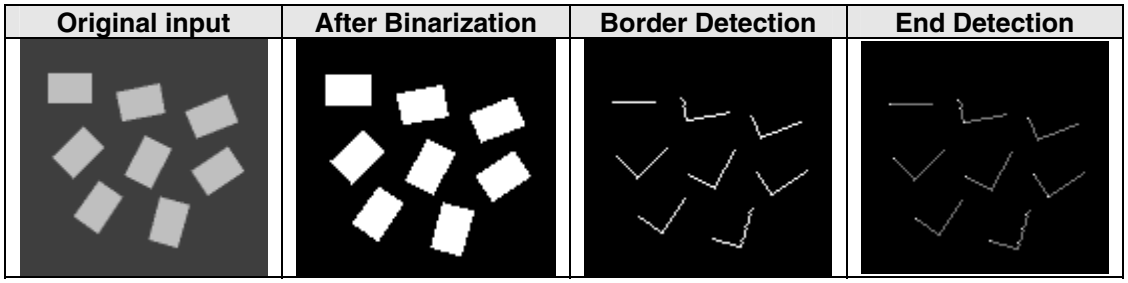
Type of inputs

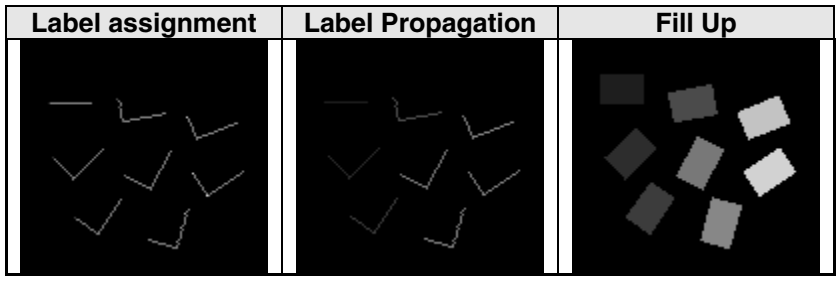
- src: 2-dimensional array -> sep uchar *
- lines: scalar -> uint

Type of output

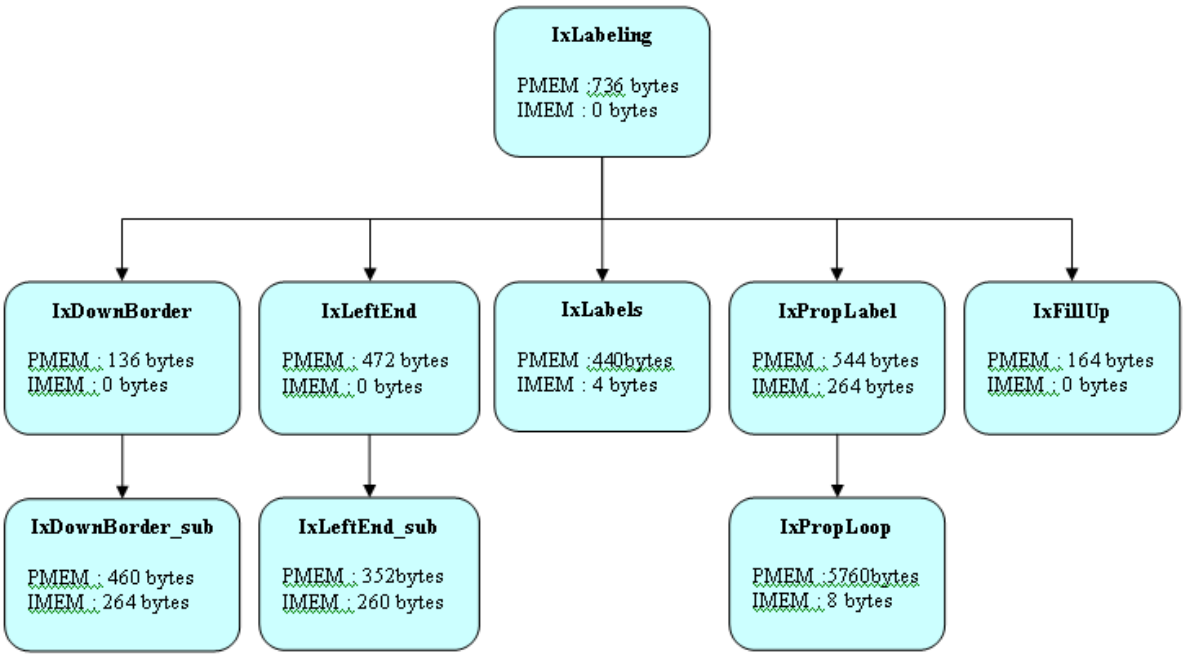
- src: 2-dimensional array -> sep uchar *

Examples





Dependency diagram



Time performance

Number of steps for a 128*128 image with 128 PE:

Function	Steps
IxLabeling	28728
IxPropLabel	11012