Math-c Documentation

Image functions

The image function to load and save only supports 8 bit per component. If the image only has one component the image is represented as matrix with scalars integers, if the image has 2 or more components are loaded as a matrix with each element of the matrix is a vector.

example:

structure of a gray image 3x4 pixels

grayimg = [42 224 182 196; 167 254 34 78;152 215 57 75]

structure of a ARGB image 3x4 pixels

colorimg = [[255 173 225 113] [255 188 9 124] [255 130 253 194] [255 173 2 191];

[255 215 226 51] [255 129 40 64] [255 77 60 255] [255 164 196 202];

[255 122 247 184] [255 136 161 121] [255 33 54 160] [255 197 168 246]]

The image handle up to 4 channels; 1 channel(Gray), 2 channel(Alpha and Gray) 3 channels (Red,Green and Blue) and 4 channels(Alpha, Red,Green and Blue).

Those values are possible to display on screen but almost all the processing functions only handle 1 or 4 channels.

Om = imgaffinetr(Im,tr)

Apply a affine transform to image pixel position

Im -> Image value

tr -> 2x3 matrix with the affine values.

returns

Om -> an image with the the transform, the pixel values outside the image a refilled with 0.

the new pixel values are calculated as:

[x' ; y'] = tr * [x ; y ; 1]

where tr=[rotXX rotXY translationX; rotYX rotYY translationY]

Om = imgAYCbCrToARGB(Im)

Om = imgAYCbCrToARGB(Im,mode)

Convert an image in AYCbCr(4:4:4) format to ARGB.

Im -> Image value

mode -> String value the only supported value is "601-4"

returns

Om -> return a ARG image,if mode is not defined, convert the image using the standard ITU-Recommendation BT.709-2, if mode is set to "601-4" then use the ITU-Recommendation BT.601-4.

Om = imgchannel(Im)

Om = imgchannel(Im,channel)

Split the channels of image

Im -> Image value

channel -> Integer, channel number (default -1)

returns

Om -> if channel is -1, returns all channels in a vector; or if channel is set, returns the matrix with the scalar values of the channel selected.

Om = imgclip(Im,clip)

Split the channels of image

Im -> Image value

clip -> vector 1x2, where clip(0) is the low value and clip(1) is the high value

returns

Om -> the image within range clip(0) to clip(1). do the following operation.

if(Ipx>clip(1)){

Opx = clip(1)

}else if(Ipx<clip(0)){

Opx = clip(0)

}else{

Opx = Ipx

}

Om = imgclose(Im,kernel)

image close morphing operation

Im -> Image value

kernel -> matrix of integers with odd size example:3x1 or 5x7 etc.

returns

Om -> the image with the applied operation.(equivalent to dilate and then erode)

Om = imgconv(Im,kernel,bias,d)

image convolution

Im -> image ARGB or G

kernel -> must be integer, kernel, the size must be odd

bias -> must be integer, value added to each element in a convolution result(default 0)

d -> must be integer, divisor of kernel (default, width*height of kernel)

Om -> result image of convolution

Note: Alpha channel is not modified

examples:

b=imgconv(a,[1 1 1; 1 1 1; 1 1 1])

b=imgconv(a,[1 1 1; 1 1 1; 1 1 1],128)

b=imgconv(a,[255 255 255; 255 255 255; 255 255 255],0,2295)

Om = imgdeconvRLucy(Im,k1,d1,it,k2,d2)

Deconvolve the image

Im -> image ARGB or G

k1 -> must be integer, kernel 1, the size must be odd

d1 -> must be integer, divisor of kernel 1 (default, sizex*sizey of kernel 1)

it -> must be integer, num. of iterations (default 10)

k2 -> must be integer, kernel 2, the size must be odd, useful only if kernel1 is asymmetrical (default, this is ignored)

d2 -> must be integer, divisor of kernel 2(default, sizex*sizey of kernel 2, only if kernel 2 is set, otherwise is ignored)

Om -> result image

Note: Alpha channel is not modified

More info in vImageRichardsonLucyDeConvolve_ARGB8888

examples:

```
b=imgdeconvRLucy(a,[1 1 1; 1 1 1; 1 1 1])

b=imgdeconvRLucy(a,[1 1 1; 1 1 1; 1 1 1],20)

b=imgdeconvRLucy(a,[1 1 1; 1 1 1; 1 1 1],9,5)

b=imgdeconvRLucy(a,[1 1 0; 1 1 1; 1 1 0],9,5,[0 1 0; 1 1 1; 1 1 0],9)
```

Om = imgdilate(Im,kernel)

image dilate morphing operation

Im -> Image value

kernel -> matrix of integers with odd size example:3x1 or 5x7 etc.

returns

Om -> the image with the applied operation.

Om = imgerode(Im,kernel)

image dilate morphing operation

Im -> Image value

kernel -> matrix of integers with odd size example:3x1 or 5x7 etc.

returns

Om -> the image with the applied operation.

y = imghist(Im)

image histogram

Im -> Image value

returns

Om -> vectors with size 256 with the image histogram.

Om = imgload(file)

load a image from file

file -> String value

returns

Om -> loaded image, if the "file" has "iCloud:" prefix when the image is load from the iCloud.

Om = imgmatrixmult(Im,M)

Om = imgmatrixmult(Im,M,div)

Om = imgmatrixmult(Im,M,div,prebias)

Om = imgmatrixmult(Im,M,div,prebias,postbias)

Multiply each pixel by matrix M

Im -> ARGB image

M -> matrix 4x4 integers values

div -> integer value, divisor of multiplication (default 3).

prebias -> integer value, offset applied before the matrix multiplication.

postbias -> integer value, offset applied after the matrix multiplication.

returns

Om -> image

Om = imgmax(Im, [height width])

image max morphing operation

Im -> Image value

height -> integer value, of kernel to apply the operation.

width -> integer value, of kernel to apply the operation.

returns

Om -> the image with the applied operation.

Om = imgmin(Im, [height width])

image max morphing operation

Im -> Image value

height -> integer value, of kernel to apply the operation.

width -> integer value, of kernel to apply the operation.

returns

Om -> the image with the applied operation.

Om = imgopen(Im,kernel)

image open morphing operation

Im -> Image value

kernel -> matrix of integers with odd size example:3x1 or 5x7 etc.

returns

Om -> the image with the applied operation.(equivalent to erode and then dilate)

Om = imgrotate(Im,angle)

image rotation

Im -> Image value

angle -> scalar value.

returns

Om -> the rotated image.

y = imgsave(file)

save a image from file

file -> String value

returns

y -> 0, if the "file" has "iCloud:" prefix when the image is saved in iCloud.

Om = imgscale(Im,scale)

image scale

Im -> Image value

scale -> scalar value.

returns

Om -> the scaled image.

Om = imgsetvalue(Im,channel,value)

image with value to selected channel

Im -> Image value

channel -> integer value.

value -> integer value or matrix of size of Im.

returns

Om -> image with value to selected channel. example:

to fill an image with alpha 255

>>> Om = imgsetvalue(Im,0,255)

to replace the Red channel with other 1 channel image

>>> Om = imgsetvalue(Im,1,Sm)

y = imgshow(Im)

displays image

Im -> Image value

returns

y -> 0.

Om = imgthr(Im,level)

Split the channels of image

Im -> Image value

level -> integer value.

returns

Om -> the image within range 0 to 255. do the following operation.

if(Ipx>level){

Opx = 255

}else{

Opx = 0

}

Om = imgToAYCbCr(Im)

Om = imgToAYCbCr(Im,mode)

Convert in image inARGB to AYCbCr(4:4:4) format.

Im -> Image value

mode -> String value the only supported value is "601-4"

returns

Om -> return a AYCbCr image, if mode is not defined, convert the image using the standard ITU-Recommendation BT.709-2, if mode is set to "601-4" then use the ITU-Recommendation BT.601-4.

a = matjoin(a)

y = matjoin(a,b)

join two matrix elements (useful with images)

a -> matrix value

b -> matrix value (same size of a)

returns

y -> a new matrix with vectors containing the elements of each matrix, example:

a = [7 9 8 0 ;

2 0 9 4 ;

9 1 1 0]

b = [0 6 1 9 ;

9 10 6 5 ;

7 3 10 2]

>>> y = matjoin(a,b)

y = [[7 0] [9 6] [8 1] [0 9] ;

[2 9] [0 10] [9 6] [4 5] ;

[9 7] [1 3] [1 10] [0 2]]

y = matsplit(a)

split in matrices the sub elements of a (useful with images)

a -> matrix value with sub elements(must have the same number of sub-elements in the matrix), example a = [[5 6 7] [8 5 5] [4 3 5]]

returns

y -> a nested matrix a containing an array of matrices containing the elements, example with two sub-elements:

y = [[7 0] [9 6] [8 1] [0 9] ;

[2 9] [0 10] [9 6] [4 5] ;

[9 7] [1 3] [1 10] [0 2]]

>>> [a b] = matsplit(y);

>>> a

a = [7 9 8 0 ;

2 0 9 4 ;

9 1 1 0]

>>> b

b = [0 6 1 9 ;

9 10 6 5 ;

7 3 10 2]