redCV Open Source Computer Vision Library

What is redCV

redCV means Red Language Open Source Computer Vision Library. It is a collection of Red functions and routines that give access to many popular Image Processing algorithms.

The key features

redCV provides cross-platform high level API that includes many functions. redCV has no strict dependencies on external libraries. RedCV is free for both non-commercial and commercial use.

Who created it

The list of authors and major contributors:
François Jouen for the library development.
Thanks to Nénad Rakocevic and Qingtian Xie for developing Red and their constant help.
Thanks to Didier Cadieu for samples optimization.
Thanks to Boleslav Březovský for distance mapping
Thanks to Bruno Anselme for ZLib binding.
Thanks to Fyodor Shchukin for illustration.

Where to get redCV

Go https://github.com/ldci/redCV.

redCV Reference Manual

Using redCV Library

Most of functions are calling Red/System routines for faster image rendering. All redCV routines can be directly called from a red program (not for newbies). For a more convenient access, Red/System routines are exported as red functions. All red routines are prefixed with underscore (e.g. _rcvCopy). **Only red functions are documented**. Code sample included with redCV is documented in RedCV_Samples.pdf.

All includes to redCV libraries are declared in a single file (/libs/redcv.red). You just need including *redcv.red* file in your Red programs.

```
#include %core/rcvCore.red                ; Basic image creating and processing functions
#include %highgui/rcvHighGui.red          ; Fast highgui functions
#include %matrix/rcvMatrix.red            ; Matrices functions
#include %imgproc/rcvImgProc.red          ; Image processing functions
#include %math/rcvRandom.red              ; Random laws for generating random images
#include %math/rcvStats.red               ; Statistical functions for images
#include %math/rcvDistance.red            ; Distance algorithm for detection in images
#include %ZLib/rcvZLib.red                ; ZLib compression algorithms
#include %tiff/rcvTiff.red                ; Tiff image reading and writing
#include %timeseries/rcvTS.red            ; Time Series algorithms
#include %tools/rcvTools.red              ; Some tools
```

```
; all we need for computer vision with Red
#include %../../libs/redcv.red ; for red functions
```

Some lectures

Image Processing in C, by Dwayne Phillips. The first edition of Image Processing in C (Copyright 1994, ISBN 0-13-104548-2) was published by R & D Publications
1601 West 23rd Street, Suite 200
Lawrence, Kansas 66046-0127

Algorithms for Image Processing and Computer Vision (2011) by J.R. Parker, published by Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256

Basic Structures

Image

redCV directly uses Red image! datatype. Loaded images by Red are in ARGB format (a tuple). Images are 8-bit and internally use bytes [0..255] as a binary string. Images are 4-channels and actually Red can't create 1, 2 or 3-channels images. Similarly Red can't create 16-bit (0..65536) 32-bit or 64-bit (0.0..1.0) images.

Each pixel channel ARGB is represented by a byte! The byte! datatype's purpose is to represent unsigned integers in the 0-255 range. Many libraries use a byte pointer to access ARGB components of a pixel. Red proposes an optimized way which uses an integer to store ARGB values in a single value. Since the memory size of an integer is 32 bits, is really easy to store 4 bytes (8-bit) value with an integer. Consequently, an int-ptr! will be used to access pixel value.

Now, to access to ARGB values stored in the integer, Red applies right shift operators, both unsigned right shift: >>> and signed right shift: >>

```
a: pix1/value >>> 24                ; byte 1 [0-255] Alpha (transparency) channel
r: pix1/value and 00FF0000h >> 16   ; byte 2 [0-255] Red channel
g: pix1/value and FF00h >> 8        ; byte 3 [0-255] Green channel
b: pix1/value and FFh               ; byte 4 [0-255] Blue channel
```

To write back pixel values, Red call signed left shift: << operator

pixD/value: (a << 24) OR (r << 16) OR (g << 8) OR b

Matrix

Matrix! Datatype is not yet implemented by Red. A 100 x 100 color image is nothing but an array of 100 x 100 x 3 (for each R, G, B color channel) numbers. Usually, we like to think of 100 x 100 x 3 array as a 3D array, but you can think of it as a long 1D array consisting of 30,000 elements. This is why we use vector! datatype to simulate matrices with Red. Matrices are 2-D with n lines *m columns with only one value. Matrix element can be Char!, Integer! or Float!. RedCV uses integer 8, 16 or 32-bit matrices or 32 or 64-bit float matrices.

.

Array

This a new type I use for quick access. Basically, array is a block of vectors.

```
nBins: 16
histo: copy []
append/only histo make vector! nBins
append/only histo make vector! nBins
append/only histo make vector! nBins
```

Important

Except for creating either images or matrices, redCV functions require to pass image, matrix or array as argument to get the result of processing. This avoids memory leaks if you're using a lot of structures, but developers **must control that both source and destination structures are compatible.**

Images and matrices basic operators

rcvCreateImage

Creates and returns empty (black) image

```
rcvCreateImage: function [
        size [pair!]  ;size : image size width and height as a pair
]
```

Defined in /libs/core/rcvCore.red

```
dst: rcvCreateImage 512x512
```

rcvGetImageSize

Returns Image Size as a pair!

```
rcvGetImageSize: function [src [image!]]
```

Defined in /libs/core/rcvCore.red

rcvGetImageFileSize

Returns Image File Size as a pair!

```
rcvGetImageFileSize: function [fileName [file!] return: [pair!]]
```

Defined in /libs/core/rcvCore.red

rcvCreateMat

Creates and returns 2D matrix

```
rcvCreateMat: function [ type [word!] bitSize [integer!] mSize [pair!]]
type: name of accepted datatype: char! | integer!| float!
bitSize: 8 for char!,  8 | 16 | 32 for integer!, 32 | 64 for float!
mSize: matrix size as pair
```

Defined in /libs/matrix/rcvMatrix.red

```
msize: 128x128
mat1: rcvCreateMat 'integer! 8 msize
mat2: rcvCreateMat 'integer! 16 msize
mat3: rcvCreateMat 'integer! 32 msize
```

rcvLengthMat

Returns matrix length as integer

```
rcvLengthMat: function [mat [vector!]]
```

Defined in /libs/matrix/rcvMatrix.red

rcvMakeRangeMat

Creates and returns an ordered matrix

rcvMakeRangeMat: function [a [number!] b [number!] step [number!]]

Defined in /libs/matrix/rcvMatrix.red

```
rcvMakeRangeMat -5.0 5.0 0.25 -> [-5.0 -4.75 -4.5 -4.25 -4.0 -3.75 -3.5 -3.25 -3.0 -2.75 -2.5 -2.25 -2.0 -1.75 -1.5 -1.25 -1.0 -0.75 -0.5 -0.25 0.0 0.25 0.5 0.75 1.0 1.25 1.5 1.75 2.0 2.25 2.5 2.75 3.0 3.25 3.5 3.75 4.0 4.25 4.5 4.75 5.0]
rcvMakeRangeMat 1 10 1 -> [1 2 3 4 5 6 7 8 9 10]
```

rcvMakeIndenticalMat

Creates and returns matrix with identical values

rcvMakeIndenticalMat: func [type [word!] bitSize [integer!] vSize [integer!] value [number!]]

Defined in /libs/matrix/rcvMatrix.red

```
v: rcvMakeIndenticalMat 'Integer! 32 10 1 -> [1 1 1 1 1 1 1 1 1 1]
v: rcvMakeIndenticalMat 'Integer! 32 10 5 -> [5 5 5 5 5 5 5 5 5 5]
v: rcvMakeIndenticalMat 'Float! 64 10 0.25 ->[ 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25]
```

rcvMakeBinaryMat

Makes a binary matrix [0..1]

rcvMakeBinaryMat: function [src [vector!] dst: [vector!]

Defined in /libs/matrix/rcvMatrix.red

```
Source matrix is 16 or 32-bit matrix [0..255]
```

rcvReleaseImage

Releases image data

rcvReleaseImage: function [src [image!]]

src : image to remove

Defined in /libs/core/rcvCore.red

rcvReleaseAllImages

Delete all images

rcvReleaseAllImages: function [list [block!]]

```
loaded images must be stored into a block! before releasing
```

Defined in /libs/core/rcvCore.red

rcvReleaseMat

Releases Matrix

rcvReleaseMat: function [mat [vector!]]

mat: matrix to be released

Defined in /libs/matrix/rcvMatrix.red

Release functions will be probably removed according to Red garbage collector development.

rcvLoadImage

Loads and returns image from file
rcvLoadImage: function [fileName [file!] /grayscale]
filename: name of the file to load as a Red file datatype
/grayscale: loads image as grayscale image
Defined in /libs/core/rcvCore.red

tmp: request-file if not none? tmp [img1: rcvLoadImage tmp img2: rcvLoadImage /grayscale]

rcvLoadTiffImage

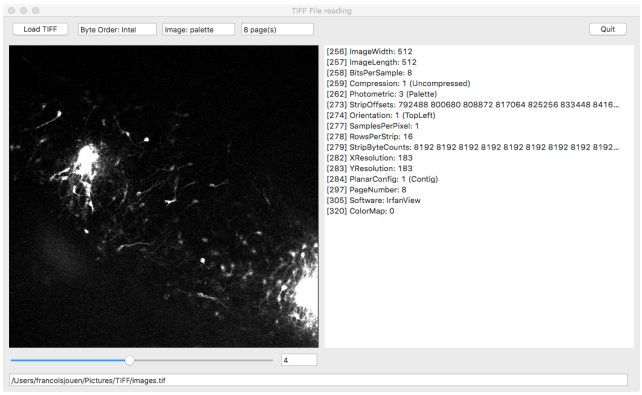
loads TIFF image
rcvLoadTiffImage: func [f [file!]]
f: name of the Tiff file to load
Defined in /libs/tiff/rcvTiff.red

tmp: request-file if not none? tmp [rcvLoadTiffImage tmp] **Attention**: you need to call rcvTiff2RedImage function in order to display the image canvas/image: rcvTiff2RedImage **Uncompressed bilevel, grayscale, palette-color images and RGB with samples per pixel up to 4 are supported.** Samples Per Pixel is usually 1 for bilevel, grayscale, and palette-color images. Samples Per Pixel is usually 3 for RGB images

rcvReadTiffImageData

Reads multiple images included in TIFF File
rcvReadTiffImageData: func [page [integer!]]
page: number of image to access
Defined in /libs/tiff/rcvTiff.red

Tiff files can include more than 1 image. You can use this function to access any image by its number. **Attention**: you need to load first the tiff file before accessing image



rcvLoadImageAsBinary

Loads image from file and returns image as binary

rcvLoadImageAsBinary: function [fileName [file!] /alpha]
filename: name of the file to load as a Red file datatype
/ alpha: loads image as 4 channels image including alpha channel
Defined in /libs/core/rcvCore.red

rcvSaveImage

Save image to file

rcvSaveImage: function [src [image!] fileName [file!]]
src: image to save
filename: name of the file to save as a Red file datatype

Actually only png codec is supported for saving image. Will be improved in future by Red Team

Defined in /libs/core/rcvCore.red

rcvSaveTiffImage

Save red image as tiff

rcvSaveTiffImage: func [anImage [image!] f [file!] mode [integer!]]
anIlmage: image to save
f: name of the file
mode : 1 little endian (Intel) / 2 big endiand (Motorola)
Defined in /libs/tiff/rcvTiff.red

rcvCloneImage

Returns a copy of source image

rcvCloneImage: function [src [image!]]
src: image to be cloned
Defined in /libs/core/rcvCore.red

img: recCreateImage 512x512 hsv: rcvCloneImage img

rcvCloneMat

Returns a copy of source matrix
rcvCloneMat: function [src [vector!]]
src: matrice to be cloned
Defined in /libs/matrix/rcvMatrix.red

rcvCopyImage

Copies source image to destination image
Source and destination image must have the same size!
rcvCopyImage: function [src [image!] dst [image!]]
src: image to be copied
dst: destination image
Defined in /libs/core/rcvCore.red

```
img: hsv : recCreateImage 512x512
hsv: rcvCopy img hsv
```

rcvCopyMat

Copy source matrix to destination matrix
rcvCopyMat: function [src [vector!] dst [vector!]]
src: matrice to be copied
dst: destimation matrix
Defined in /libs/matrix/rcvMatrix.red

rcvZeroImage

Sets all image pixels to 0
rcvZeroImage: function [src [image!]]
src: image to clear
Defined in /libs/core/rcvCore.red

rcvRandomImage

Creates and returns a random uniform color or pixel random image
rcvRandomImage: function [size [pair!] value [tuple!] /uniform /alea]
size: size of image as pair!
Value: random value as tuple!
/uniform : random uniform color
/alea : random pixels
Defined in /libs/core/rcvCore.red

rcvRandomMat

Randomizes matrix
rcvRandomMat: function [mat [vector!] value [integer!]]
mat: destinatiobn matrix
value: random value as integer!

Defined in /libs/matrix/rcvMatrix.red

```
mat1: rcvCreateMat 'integer! 8 msize
mat2: rcvCreateMat 'integer! 16 msize
mat3: rcvCreateMat 'integer! 32 msize
rcvRandomMat mat1 FFh
rcvRandomMat mat2 FFFFh
rcvRandomMat mat3 FFFFFFh
```

rcvImageNoise

Generates Gaussian noise

function [src [image!] noise [float!] t [tuple!]

src: image

noise : float!

t : color as tuple !

Defined in /libs/imgproc/rcvImgProc.red

noise is a float value between 0.0 and 1.0 used to calculate the number of pixels to be randomly assigned.
You can use color to make any kind of colored noise on image.

rcvColorImage

Set image color

function [src [image!] acolor [tuple!]]

src: image to color

acolor: required color as a tuple

Defined in /libs/core/rcvCore.red

rcvColorMat

Set matrix color

rcvColorMat: function [mat [vector!] value [integer!]]

mat: destination matrix

value: color value as integer

Defined in /libs/matrix/rcvMatrix.red

```
mat1: rcvCreateMat 'integer! 8 msize
rcvColorMat mat1 0
```

rcvSortMat

Returns ascending sort of matrix

rcvFlipMat: function [v [vector!]]

Defined in /libs/matrix/rcvMatrix.red

rcvFlipMat

Returns flip matrix

rcvFlipMat: function [v [vector!]]
Defined in /libs/matrix/rcvMatrix.red

rcvCompressRGB

Compresses rgb image values as binary
rcvCompressRGB: function [rgb [binary!] level [integer!]]
rgb: rgb binary values of the image (image/rgb)
level: compression level for ZLib compression
[0: No compression
1: Best Speed
9: Best compression
-1: default compression]
Defined in /libs/Zlib/rcvZLib.red

rcvDecompressRGB

Uncompresses rgb image values as binary
rcvDecompressRGB: function [rgb [binary!] bCount [integer!]]
rgb: previously rgb compressed values
bCount: size of non-compressed rgb values

```
rgb: copy img/rgb                    ; image rgb values
clevel: 9                            ; zLib best compression
result: copy #{}                     ; for compressed data
result2: copy #{}                    ; for uncompressed data
n: length? rgb                       ; size of uncompressed data
result: rcvCompressRGB rgb clevel    ; compress
result2: rcvDecompressRGB result n   ; uncompress
```

Defined in /libs/Zlib/rcvZLib.red

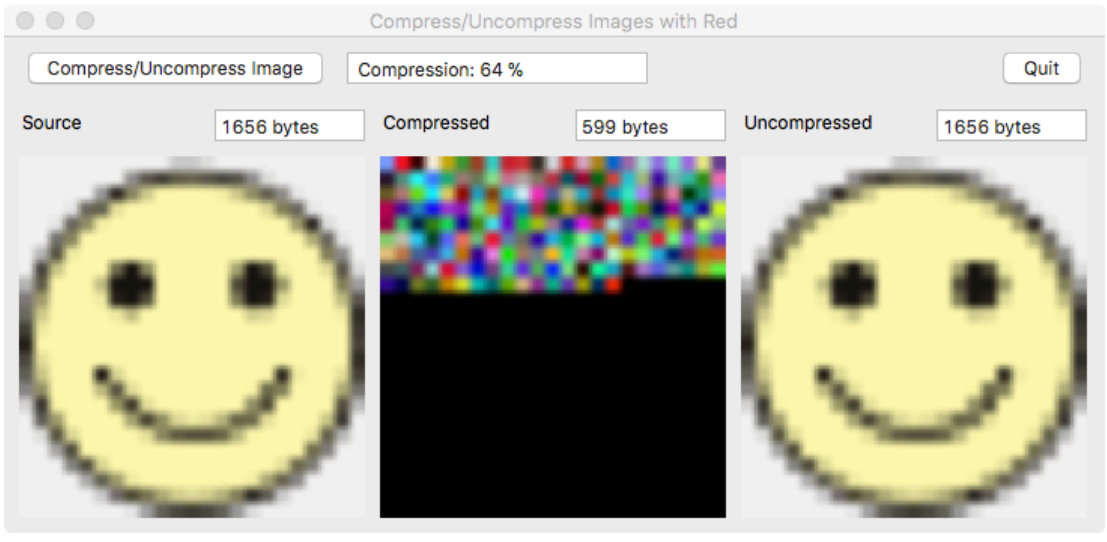


Image and matrix utilities

rcvIsAPixel

Returns true if pixel value is greater than threshold
rcvIsAPixel: function [src [image!] coordinate [pair!] threshold [integer!]]
src: image
coordinate: xy position in image as a pair
threshold: threshold value
Defined in /libs/core/rcvCore.red

rcvGetPixel

Returns pixel value at xy coordinates as tuple
rcvGetPixel: function [src [image!] coordinate [pair!]]
src: image
coordinate: xy position in image as a pair
Defined in /libs/core/rcvCore.red

rcvPickPixel

Returns pixel value at xy coordinates as tuple
rcvPickPixel: function [src [image!] coordinate [pair!]]
src: image
coordinate: xy position in image as a pair
Defined in /libs/core/rcvCore.red

rcvGetPixelAsInteger

Returns pixel value at xy coordinates as integer
rcvGetPixelAsInteger: function [src [image!] coordinate [pair!]]
src: image
coordinate: xy position in image as a pair
Defined in /libs/core/rcvCore.red

rcvGetInt2D

Returns matrix value at xy coordinates as integer
getInt2D: function [src [vector!] mSize [pair!] coordinate [pair!]]
src: source matrix
coordinate: xy position of the element as a pair
msize: matrix size as pair!
Defined in /libs/core/rcvCore.red

rcvGetReal2D

Returns matrix value at xy coordinates as float
getInt2D: function [src [vector!] mSize [pair] coordinate [pair!]]
src: source matrix
msize: matrix size as pair!
coordinate: xy position of the element as a pair
Defined in /libs/core/rcvCore.red

rcvSetPixel

Sets pixel value at xy coordinates
rcvSetPixel: function [src [image!] coordinate [pair!] val [tuple!]]
src: image
coordinate: xy position in image as a pair
val : pixel value as a tuple
Defined in /libs/core/rcvCore.red

rcvPokePixel

Set pixel value at xy coordinates
rcvPokePixel: function [src [image!] coordinate [pair!] val [tuple!]]
Defined in /libs/core/rcvCore.red

rcvSetInt2D

Sets value in integer matrix
setInt2D: function [dst [vector!] mSize [pair] coordinate [pair!] val [integer!]]
dst: destination matrix
msize: matrix size as pair!
coordinate: xy position of the element as a pair
val: value as integer
Defined in /libs/core/rcvCore.red

rcvSetReal2D

Sets value in float matrix
setInt2D: function [dst [vector!] mSize [pair] coordinate [pair!] val [float!]]
dst: destination matrix
msize: matrix size as pair
coordinate: xy position of the element as a pair
val: value as float
Defined in /libs/core/rcvCore.red

rcvMatleftPixel

Gets coordinates of first left pixel as pair

rcvMatleftPixel: function [mat [vector!] matSize [pair!] value [integer!]]
mat: Integer matrix
matSize: matrix size as pair
value : pixel value (e.g. 1 or 255)
Defined in /libs/core/rcvCore.red

rcvMatRightPixel

Gets coordinates of first right pixel as pair

rcvMatRightPixel: function [mat [vector!] matSize [pair!] value [integer!]]
mat: Integer matrix
matSize: matrix size as pair
value : pixel value (e.g. 1 or 255)
Defined in /libs/core/rcvCore.red

rcvMatUpPixel

Gets coordinates of first top pixel as pair

rcvMatRightPixel: function [mat [vector!] matSize [pair!] value [integer!]]
mat: Integer matrix
matSize: matrix size as pair
value : pixel value (e.g. 1 or 255)
Defined in /libs/core/rcvCore.red

rcvMatDownPixel

Gets coordinates of first bottom pixel as pair

rcvMatRightPixel: function [mat [vector!] matSize [pair!] value [integer!]]
mat: Integer matrix
matSize: matrix size as pair
value: pixel value (e.g. 1 or 255)
Defined in /libs/core/rcvCore.red

rcvSetAlpha

Sets image transparency

rcvSetAlpha: function [src [image!] dst [image!] alpha [integer!]
src: image remains unchanged and transparency is modified for destination image
alpha : transparency value [0..255]

```
sl: slider 256 [t: 255 - (to integer! sl/data * 255) rcvSetAlpha img1 img2 t]
```

Defined in /libs/core/rcvCore.red

rcvBlend

Computes the alpha blending of two images

rcvBlend: function [src1 [image!] src2 [image!] dst [image!] alpha [float!]]
src1: first image
src2: second image
dst: destination image
alpha: ratio of first image mixed with the second. Float value [0.0-1.0]
Defined in /libs/imgproc/rcvImgProc.red

rcvBlendMat

Computes the alpha blending of two matrices

rcvBlendMat: function [mat1 [vector!] mat2 [vector!] dst [vector!] alpha [float!]]
mat1: first matrix
mat2: second matrix
dst: destination matrix
alpha: ratio of first matrix mixed with the second. Float value [0.0-1.0]
defined in /libs/matrix/rcvMatrix.red

Format conversion

rcvImage2Mat

Converts Red Image to 8-bit 2-D Matrix
rcvImage2Mat: function [src [image!] mat [vector!]]
src: image
mat: vector
Grayscale
defined in /libs/matrix/rcvMatrix.red

rcvMat2Image

Converts 8, 16 or 32-bit integer Matrix to Red Image
rcvMat82Image: function [mat [vector!] dst [image!]]
mat: vector
dst: image
defined in /libs/matrix/rcvMatrix.red

rcvConvertMatScale

Converts Matrix Scale to another bit size
rcvConvertMatScale: function [src [vector!] dst [vector!] srcScale [number!] dstScale [number!] /fast /normal]
src: vector
dst: vector
srcScale: source range e.g 255
dstScale : destination range e.g 65535
/normal : uses a general function
/fast: uses a faster routine
defined in /libs/matrix/rcvMatrix.red

```
msize: 256x256
mat1: rcvCreateMat 'integer! 8 msize
mat2: rcvCreateMat 'integer! 16 msize
mat3: rcvCreateMat 'integer! 32 msize
rcvConvertMatScale/normal mat1 mat2 FFh FFFFh
rcvConvertMatScale/normal mat1 mat3 FFh FFFFFFh
```

rcvMatInt2Float

Converts integer matrix to float [0..1] matrix
rcvMatInt2Float: function [src [vector!] dst [vector!] srcScale [float!]]
src: source matrix
dst: destination matrix
srcScale: source range as float!
defined in /libs/matrix/rcvMatrix.red

rcvMatFloat2Int

Converts float matrix to integer [0..255] matrix

rcvMatFloat2Int: function [src [vector!] dst [vector!] dstScale [integer!]]
src: source matrix
dst: destination matrix
dstScale : dest range as integer!
defined in /libs/matrix/rcvMatrix.red

rcvSplit

Separates source image in ARGB channels. Destination contains selected source channel

rcvSplit: function [src [image!] dst [image!]/red /green /blue /alpha]
src: source image
dst: destination image
/red: red channel
/green: green channel
/blue: blue channel
/alpha: alpha channel
defined in /libs/core/rcvCore.red

rcvMerge

Combines 3 images to destination image

rcvMerge: function [src1 [image!] src2 [image!] src3 [image!] dst [image!]]
src1: source 1 image
src2: source 2 image
src3: source 3 image
dst: destination image
defined in /libs/core/rcvCore.red

This function takes r channel from image 1, g channel form image 2, and b channel from image3 to make destination image. You can change image order as you want.

rcvSplit2Mat

Separates image channels to 4 8-bit matrices

rcvSplit2Mat: function [src [image!] mat0 [vector!] mat1 [vector!] mat2 [vector!] mat3 [vector!]]
src: image
mat0: image alpha channel
mat1: image red channel
mat2: image green channel
mat3: image blue channel
defined in /libs/matrix/rcvMatrix.red

if source image is grayscale then mat1 = mat2 = mat3.

rcvMerge2Image

Merges 4 8-bit matrices to Red image

rcvMerge2Image: function [mat0 [vector!] mat1 [vector!] mat2 [vector!] mat3 [vector!] dst [image!]]

mat0: image alpha channel

mat1: image red channel

mat2: image green channel

mat3: image blue channel

dst: image

defined in /libs/matrix/rcvMatrix.red

rcvTiff2RedImage

Converts TIFF image and returns Red image

rcvTiff2RedImage: func []

defined in /libs/tiff/rcvTiff.red

Attention: you need a loaded Tiff File. See rcvLoadTiffImage documentation.

Color and color space conversion

rcvInvert

Destination image: inverted source image (Similar to NOT image)
rcvInvert: function [source [image!] dst [image!]]
src: source image
dst: destination image
defined in /libs/core/rcvCore.red

rcv2NzRGB

Normalizes the RGB values of an image
rcv2NzRGB: function [src [image!] dst [image!] /sum/sumsquare]
src: source image
dst: destination image
refinement/sum: sum of r g b values is used for normalization
refinement/ sumsquare: sqrt((power r 2.0) + (power g 2.0) + (power b 2.0))
defined in /libs/core/rcvCore.red

rcv2BW

Converts RGB image to black[0] and white [255]
rcv2BW: function [src [image!] dst [image!]]
src: source image
dst: destination image
defined in /libs/core/rcvCore.red

rcv2WB

Converts RGB image to white [255] and black[0]
rcv2BW: function [src [image!] dst [image!]]
src: source image
dst: destination image
defined in /libs/core/rcvCore.red

rcv2BW: background = 0 rcv2WB: background = 255 Internal threshold value equal to 128. For an accurate thresholding see rcv2BWFilter function.

rcv2Gray

Converts RGB image to Grayscale according to refinement

rcv2Gray: function [src [image!] dst [image!] /average /luminosity /lightness return: [image!]]
src: source image
dst: destination image
defined in /libs/core/rcvCore.red

The */average* method simply averages the values: (R + G + B) / 3. The */lightness* method averages the most prominent and least prominent colors: (max(R, G, B) + min(R, G, B)) / 2. The */luminosity* method is a more sophisticated version of the average method. It also averages the values, but it forms a weighted average to account for human perception. The formula for luminosity is 0.21 R + 0.72 G + 0.07 B.

rcv2BGRA

Converts RGBA to BGRA

rcv2BGRA: function [src [image!] dst [image!]]
src: source image
dst: destination image
defined in /libs/core/rcvCore.red

rcv2RGBA

Converts BGRA to RGBA

rcv2RGBA: function [src [image!] dst [image!]]
src: source image
dst: destination image
defined in /libs/core/rcvCore.red

rcvRGB2HSV

RBG color to HSV conversion

rcvRGB2HSV: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

rcvBGR2HSV

BGR color to HSV conversion

rcvBGR2HSV: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

The Hue/Saturation/Value model was created by A. R. Smith in 1978. The coordinate system is cylindrical. The hue value H runs from 0 to 360º. The saturation S is the degree of purity and is from 0 to 1. Purity is how much white is added to the color. S=1 makes the purest color (no white). Brightness V also ranges from 0 to 1, where 0 is the black. There is no transformation matrix for RGB or BGR to HSV conversion, but R, G and B are converted to floating-point format and scaled to fit 0..1 range.

rcvRGB2HLS

RBG color to HLS conversion
rcvRGB2HLS: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

rcvBGR2HLS

RBG color to HLS conversion
rcvBGR2HLS: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

Also a cylindrical coordinates system. There is no transformation matrix for RGB or BGR to HLS conversion, but R, G and B are converted to floating-point format and scaled to fit 0..1 range.

rcvRGB2YCrCb

RBG color to YCrCb conversion
rcvRGB2YCrCb: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

rcvBGR2YCrCb

BGR color to YCrCb conversion
rcvBGR2YCrCb: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

There is no transformation matrix.
Y <- 0.299*R + 0.587*G + 0.114*B
Cr <- (R-Y)*0.713 + delta
Cb <- (B-Y)*0.564 + delta

rcvRGB2XYZ

RGB to CIE XYZ color conversion
rcvRGB2XYZ: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

rcvBGR2XYZ

BGR to CIE XYZ color conversion
rcvBGR2XYZ: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

To transform from XYZ to RGB the matrix transform used is :
[X] [0.412453 0.357580 0.180423] [R]
[Y] = [0.212671 0.715160 0.072169] * [G]
[Z] [0.019334 0.119193 0.950227] [B]

rcvRGB2Lab

RBG color to CIE L*a*b conversion
rcvRGB2Lab: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

rcvRGB2Lab

RBG color to CIE L*a*b conversion
rcvBGR2Lab: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

R, G and B are converted to floating-point format and scaled to fit 0..1 range. R, G and B are first converted to CIE XYZ before processing. On output 0≤L≤100, -127≤a≤127, -127≤b≤127. The values are then converted to 8-bit images: L <- L*255/100, a <- a + 128, b <- b + 128.

rcvRGB2Luv

RBG color to CIE L*u*v conversion

rcvRGB2Luv: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

rcvRGB2Luv

RBG color to CIE L*u*v conversion

rcvBGR2Luv: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/core/rcvCore.red

R, G and B are converted to floating-point format and scaled to fit 0..1 range. R, G and B are first converted to CIE XYZ before processing. On output 0≤L≤100, -134≤u≤220, -140≤v≤122. The values are then converted to 8-bit images: L <- L*255/100, u <- (u + 134)*255/354, v <- (v + 140)*255/256.

rcvIRgBy

Log-opponent conversion

rcvIRgBy: function [src [image!] dst [image!] val [integer!]]
src: image
dst: image
val: integer value as parameter for color adjustment
defined in /libs/core/rcvCore.red

This transformation is useful for face detection, since the function is very efficient for skin color detection.

Arithmetic operators

rcvAdd

dst: src1 + src2
rcvAdd: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvAddMat

Returns dst: src1 + src2
rcvAddMat: function [src1 [vector!] src2 [vector!]]
src1: first matrix
src2: second matrix
defined in /libs/matrix/rcvMatrix.red

rcvAddLIP

Destination image: image 1 + image 2 (Logarithmic Image Processing)
rcvAddLIP : function [src1 [image!] src2 [image!] dst [image!]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

Computes the addition of the two input images, according to the LIP model (Logarithmic Image Processing). The LIP image addition is defined as: dest(x,y)= src1(x,y)+ src(x,y) – (src1(x,y)* src2(x,y)) / M where M is the number of gray tones (256 for byte image)

rcvSub

dst: src1 - src2
rcvSub: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvSubMat

Returns dst: src1 - src2
rcvSubMat: function [src1 [vector!] src2 [vector!]]
src1: first matrix
src2: second matrix
defined in /libs/matrix/rcvMatrix.red

rcvSubLIP

Destination image: image 1 - image 2 (Logarithmic Image Processing)
rcvSubLIP: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

Computes the difference of the two input images, according to the LIP model (Logarithmic Image Processing). The LIP image addition is defined as:
dest(x,y) = M*(src1(x,y) – src2(x,y)) / (M – src2(x,y))
where M is the number of gray tones (256 for byte image)

rcvMul

dst: src1 * src2
rcvMul: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvMulMat

Returns dst: src1 * src2
rcvMulMat: function [src1 [vector!] src2 [vector!]]
src1: first matrix
src2: second matrix
defined in /libs/matrix/rcvMatrix.red

rcvDiv

dst: src1 / src2
rcvDiv: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvDivMat

Rturns dst: src1 / src2
rcvDivMat: function [src1 [vector!] src2 [vector!]]
src1: first matrix
src2: second matrix
defined in /libs/matrix/rcvMatrix.red

rcvMod

dst: src1 // src2 (modulo)
rcvMod: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvRem

dst: src1 % src2 (remainder)
rcvRem: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvRemMat

Returns dst: src1 % src2
rcvRemMat: function [src1 [vector!] src2 [vector!]]
src1: first matrix
src2: second matrix
defined in /libs/matrix/rcvMatrix.red

rcvAbsDiff

dst: absolute difference src1 src2
rcvAbsDiff: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvMIN

dst: minimum src1 src2
rcvMIN: function [src1 [image!] src2 [image!] dst [image!]]
src1: image

src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvMAX

dst: maximun src1 src2
rcvMax: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvLSH

Left shift image by value
rcvLSH: function [src [image!] dst [image!] val [integer!]]
src: image
dst: image
val: integer
defined in /libs/core/rcvCore.red

rcvRSH

Right shift image by value
rcvLSH: function [src [image!] dst [image!] val [integer!]]
src: image
dst: image
val: integer
defined in /libs/core/rcvCore.red

rcvPow

dst: src ^integer! or Float! Value
rcvPow: function [src [image!] dst [image!] val [number!]]
src: image
dst: image
val: integer or float
defined in /libs/core/rcvCore.red

rcvSqr

Image square root
rcvSqr: function [src [image!] dst [image!] val [number!]]
src: image
dst: image
val: integer or float
defined in /libs/core/rcvCore.red

rcvExp
TBD requires float images
rcvLog
TBD requires float images
rcvMeanImages
dst: (src1 + src2) /2
rcvMeanImages: function [src1 [image!] src2 [image!] dst [image!]]
src1: image
src2: image
dst: image
defined in /libs/core/rcvCore.red

rcvMeanMats
Calculates mean values for 2 matrices and returns result matrix
rcvMeanMats: function [src1 [vector!] src2 [vector!]]
src1: matrix 1
src2: matrix 2
defined in /libs/matrix/rcvMatrix.red

rcvAddS
dst: src + integer! value
rcvAddS: function [src [image!] dst [image!] val [integer!]]
src: image
dst: image
val: integer
defined in /libs/core/rcvCore.red

rcvAddSMat
src + value
rcvAddSMat: function [src [vector!] value [integer!]]
src: matrix
value: integer
defined in /libs/matrix/rcvMatrix.red

rcvAddT
dst: src + tuple! value
rcvAddT: function [src [image!] dst [image!] val [tuple!]]
defined in /libs/core/rcvCore.red

rcvSubS
dst: src - integer! value
rcvSubS: function [src [image!] dst [image!] val [integer!]]

src: image
dst: image
val: integer
defined in /libs/core/rcvCore.red

rcvSubSMat

src - value
rcvSubSMat: function [src [vector!] value [integer!]]
src: matrix
value: integer
defined in /libs/matrix/rcvMatrix.red

rcvSubT

dst: src - tuple! value
rcvSubT: function [src [image!] dst [image!] val [tuple!]]
defined in /libs/core/rcvCore.red

rcvMulS

dst: src * integer! value
rcvMulS: function [src [image!] dst [image!] val [integer!]]
src: image
dst: image
val: integer
defined in /libs/core/rcvCore.red

rcvMulSMat:

dst: src * integer! value
rcvMulSMat: function [src [vector!] value [integer!]
src: matrix
val: integer

rcvMulT

dst: src * tuple! value
rcvMulT: function [src [image!] dst [image!] val [tuple!]]
defined in /libs/core/rcvCore.red

rcvDivS

dst: src / integer! value
rcvDivS: function [src [image!] dst [image!] val [integer!]]
src: image
dst: image
val: integer
defined in /libs/core/rcvCore.red

rcvDivSMat

src / value

rcvDivSMat: function [src [vector!] value [integer!]]
src: matrix
value: integer
defined in /libs/matrix/rcvMatrix.red

rcvDivT

dst: src / tuple! value

rcvDivT: function [src [image!] dst [image!] val [tuple!]]
defined in /libs/core/rcvCore.red

rcvModS

dst: src // integer! Value (modulo)

rcvModS: function [src [image!] dst [image!] val [integer!]]
src: image
dst: image
val: integer
defined in /libs/core/rcvCore.red

rcvModT

dst: src // tuple! Value (modulo)

rcvModT: function [src [image!] dst [image!] val [tuple!]]
defined in /libs/core/rcvCore.red

rcvRemS

dst: src % integer! Value (remainder)

rcvRemS: function [src [image!] dst [image!] val [integer!]]
src: image
dst: image
val: integer
defined in /libs/core/rcvCore.red

rcvRemT

dst: src % tuple! Value (remainder)

rcvRemT: function [src [image!] dst [image!] val [tuple!]]
defined in /libs/core/rcvCore.red

rcvRemSMat

src % value (remainder)

rcvRemSMat: function [src [vector!] value [integer!]]
src: matrix
value: integer
defined in /libs/matrix/rcvMatrix.red

Logic operators

rcvAND

dst: src1 AND src2
rcvAND: function [src1 [image!] src2 [image!] dst [image!]]
src1: first image
src2: second image
dst: src1 and src2
defined in /libs/core/rcvCore.red

rcvANDMat

Returns source1 AND source2
rcvAndMat: function [src1 [vector!] src2 [vector!]]
src1: first matrix
src2: second matrix
defined in /libs/matrix/rcvMatrix.red

rcvOR

dst: src1 OR src2
rcvOR: function [src1 [image!] src2 [image!] dst [image!]]
src1: first image
src2: second image
dst: src1 or src2
defined in /libs/core/rcvCore.red

rcvORMat

Returns source1 OR source2
rcvORMat: function [src1 [vector!] src2 [vector!]]
src1: first matrix
src2: second matrix
defined in /libs/matrix/rcvMatrix.red

rcvXOR

dst: src1 XOR src2
rcvXOR: function [src1 [image!] src2 [image!] dst [image!]]
src1: first image
src2: second image
dst: src1 xor src2
defined in /libs/core/rcvCore.red

rcvXORMat

Returns source1 XOR source2
rcvORMat: function [src1 [vector!] src2 [vector!]]
src1: first matrix
src2: second matrix
defined in /libs/matrix/rcvMatrix.red

rcvNAND

dst: src1 NAND src2
rcvNAND: function [src1 [image!] src2 [image!] dst [image!]]
src1: first image
src2: second image
dst: src1 nand src2
defined in /libs/core/rcvCore.red

rcvNOR

dst: src1 NOR src2
rcvNOR: function [src1 [image!] src2 [image!] dst [image!]]
src1: first image
src2: second image
dst: src1 nor src2
defined in /libs/core/rcvCore.red

rcvNXOR

dst: src1 NXOR src2
rcvNOR: function [src1 [image!] src2 [image!] dst [image!]]
src1: first image
src2: second image
dst: src1 nxor src2
defined in /libs/core/rcvCore.red

rcvNOT

dst: src1 NOT src2
rcvNOR: function [src1 [image!] src2 [image!] dst [image!]]
src1: first image
src2: second image
dst: src1 not src2
defined in /libs/core/rcvCore.red

rcvANDS

Tuple value is used to create a colored image which is ANDed to source image. Result is copied to destination

rcvANDS: function [src [image!] dst [image!] value [tuple!]
src: source image
dst: image
value: tuple!
defined in /libs/core/rcvCore.red

```
rcvANDS img1 dst 255.0.0.0; dst: add red color to img1
```

rcvORS

Tuple value is used to create a colored image which is ORed to source image. Result is copied to destination

rcvORS: function [src [image!] dst [image!] value [tuple!]
src: source image
dst: image
value: tuple!
defined in /libs/core/rcvCore.red

rcvXORS

Tuple value is used to create a colored image which is XORed to source image. Result is copied to destination

rcvXORS: function [src [image!] dst [image!] value [tuple!]
src: source image
dst: image
value: tuple!
defined in /libs/core/rcvCore.red

rcvANDSMat

And integer value to all element in source matrix

rcvANDSMat: function [src [vector!] value [integer!]
src: matrice
value: integer!
defined in /libs/matrix/rcvMatrix.red

rcvORSMat

OR integer value to all element in source matrix

rcvANDSMat: function [src [vector!] value [integer!]
src: matrice
value: integer!
defined in /libs/matrix/rcvMatrix.red

rcvXORSMat

XOR integer value to all element in source matrix

rcvANDSMat: function [src [vector!] value [integer!]
src: matrice
value: integer!
defined in /libs/matrix/rcvMatrix.red

Statistics and image features extraction

rcvCountNonZero

Returns number of non-zero values in image or matrix
rcvCountNonZero: function [arr [image! vector!]]
arr: image or vector
defined in /libs/math/rcvStats.red

rcvSum

Returns sum value of image or matrix as a block of rgb values
rcvSum: function [arr [image! vector!] /argb]
arr: image or vector
/argb: includes alpha channel
defined in /libs/math/rcvStats.red

rcvSumMat

Returns matrix sum as a float value
rcvSumMat: function [mat [vector!]]
defined in /libs/matrix/rcvMatrix.red

rcvMean

Returns mean value of image or matrix as a tuple of rgb values
rcvMean: function [arr [image! vector!] /argb]
arr: image or vector
/argb: includes alpha channel
defined in /libs/math/rcvStats.red

rcvMeanMat

Returns matrix mean as float value
rcvMeanMat: function [mat [vector!]]
defined in /libs/matrix/rcvMatrix.red

rcvSTD

Returns standard deviation value of image or matrix as tuple
rcvSTD: function [arr [image! vector!] /argb]
arr: image or vector
/argb: includes alpha channel
defined in /libs/math/rcvStats.red

rcvMedian

Returns median value of image or matrix as tuple
rcvMedian: function [arr [image! vector!] /argb]
arr: image or vector
/argb: includes alpha channel
defined in /libs/math/rcvStats.red

rcvProdMat

Return matrix product as a float value
rcvProdMat: function [mat [vector!]
defined in /libs/matrix/rcvMatrix.red

rcvMinValue

Returns minimal value of image or matrix as tuple
rcvMinValue: function [arr [image! vector!]]
arr: image or vector
defined in /libs/math/rcvStats.red

rcvMaxValue

Returns maximum value of image or matrix as tuples
rcvMaxValue: function [arr [image! vector!]]
arr: image or vector
defined in /libs/math/rcvStats.red

rcvMaxMat

Return maximum of matrix as a number
rcvMaxMat: function [mat [vector!]]
defined in /libs/matrix/rcvMatrix.red

cvMinMat

Return minimum of matrix as a number
rcvMinMat: function [mat [vector!]]
defined in /libs/matrix/rcvMatrix.red

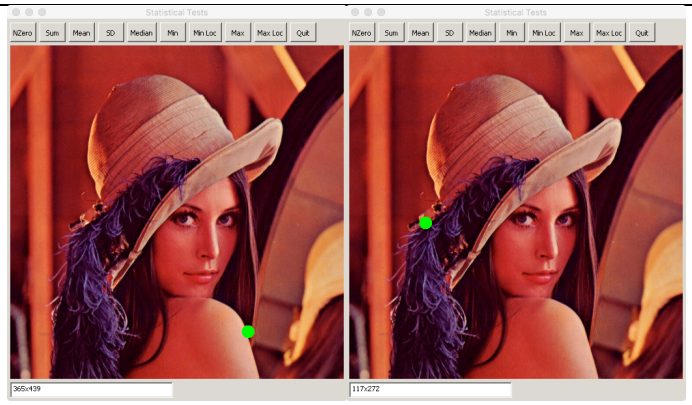
rcvMinLoc

Finds global minimum location in array and returns as pair
rcvMinLoc: function [arr [image! vector!] arrSize [pair!]
arr: image or vector
arrSize: array size as pair
defined in /libs/math/rcvStats.red

rcvMaxLoc

Finds global maximum location in array and returns as pair

rcvMaxLoc: function [arr [image! vector!] arrSize [pair!]]
arr: image or vector
arrSize: array size as pair
defined in /libs/math/rcvStats.red



Min Location Max Location

rcvHistogram

Calculates array histogram

rcvHistogram: function [arr [image! vector!] return: [vector!] /red /green /blue]
arr: image or vector
/red: histogram for red channel
/green: histogram for green channel
/blue: histogram for blue channel
defined in /libs/math/rcvStats.red

rcvRGBHistogram

Calculates array histogram according to the number of bins

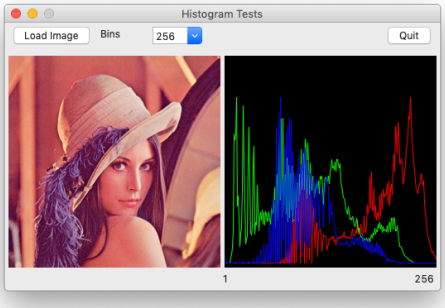
rcvRGBHistogram: function [img [image!] array [block!]]
img: source image
array: block
defined in /libs/math/rcvStats.red

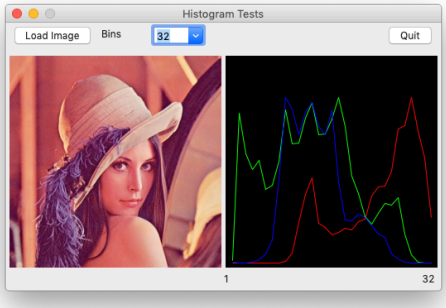
rcvMeanShift

Mean Shift filter on image

rcvMeanShift: function [src [image!] dst [image!] array [block!] colorBW [float!] converg[float!]
src: source image
dst: destination image
array: block of vectors (typically rgb histogram)
colorBW: color bandwidth
converg: mean convergence factor
defined in /libs/math/rcvStats.red

ATTENTION: these functions use array type: a block of vectors.
Vectors are used since they are faster than blocks.
nBins: 256
vect1: make vector! nBins
vect2: make vector! nBins
vect3: make vector! nBins
histo: copy []
append/only histo vect1
append/only histo vect2
append/only histo vect3

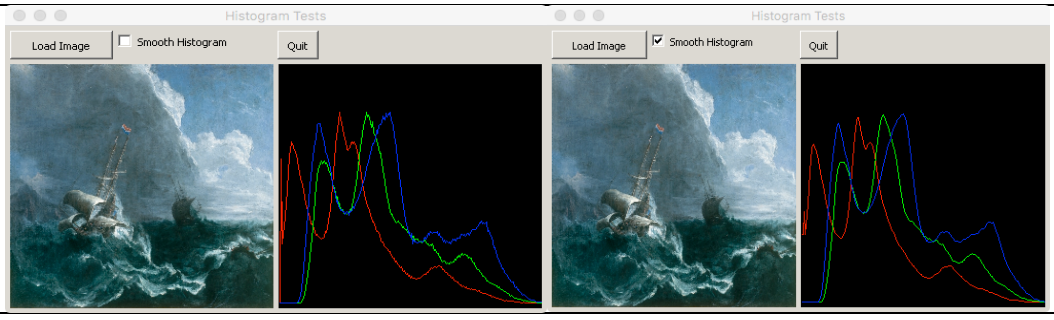




rcvSmoothHistogram

This function filters the input histogram by 3 points mean moving average and returns filtered vector

rcvSmoothHistogram: function [arr [vector!]]
arr: input histogram as vector!
defined in /libs/math/rcvStats.red



rcvRangeImage

Gives range value in image as a tuple

```
rcvRangeImage: function [source [image!]]
source: image
```

defined in /libs/math/rcvStats.red

rcvGetMatSpatialMoment

Returns the spatial moment of the matrix as float

```
rcvGetMatSpatialMoment: function [
        mat [vector!]
        matSize [pair!]
        p  [float!]
        q  [float!]
]
```

defined in /libs/matrix/rcvMatrix.red

p - the order of the moment q - the repetition of the moment p: q: 0.0 -> moment order 0 -> form area

rcvGetMatCentralMoment

Returns the central moment of the matrix as float

```
rcvGetMatCentralMoment: function [
        mat     [vector!]
        matSize [pair!]
        p       [float!]
        q       [float!]
]
```

defined in /libs/matrix/rcvMatrix.red

rcvGetNormalizedCentralMoment

Return the scale invariant moment of the image as float

```
rcvGetNormalizedCentralMoment: function [
        mat     [vector!]
        matSize [pair!]
        p       [float!]
        q       [float!]
]
```

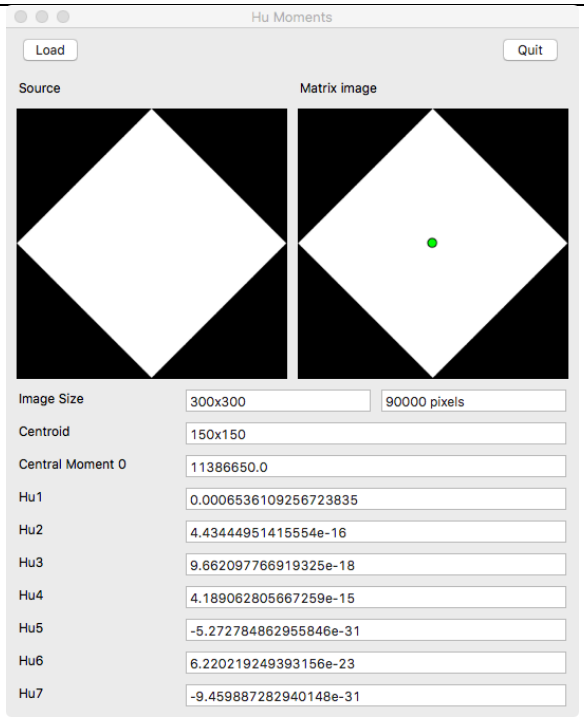
defined in /libs/matrix/rcvMatrix.red

rcvGetMatHuMoments

Returns Hu moments of the image as block

```
rcvGetMatHuMoments: function [
"Returns Hu moments of the image"
        mat     [vector!]
        matSize [pair!]
]
```

defined in /libs/matrix/rcvMatrix.red



Hu Moments are normally extracted from the outline of an object in an image. An **image moment** is a particular weighted average of the image pixels' intensities, or a function of such moments, usually chosen to have some attractive property.

rcvSortImage

Ascending image sorting

```
rcvSortImage: function [source [image!] dst [image!]]
source: image
dst: image
```

defined in /libs/math/rcvStats.red

rcvXSortImage

Image sorting by line

```
rcvXSortImage: function [src [image!] dst[image!] flag [logic!]]
src: image
dst: image
```

flag : logic. If true reverse sorting.
defined in /libs/math/rcvStats.red

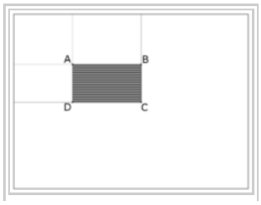
rcvYSortImage

Image sorting by column
rcvXSortImage: function [src [image!] dst[image!] flag [logic!]]
src: image
dst: image
flag : logic. If true reverse sorting.
defined in /libs/math/rcvStats.red

rcvIntegral

Calculates integral images
rcvIntegral: function [src [image!] sum [image!] sqsum [image!]
src: image or integer matrice
sum: image or integer matrice for summed area table
sqsum: image or integer matrice for square summed area table
mSize : image or integer matrice size as a pair!
defined in /libs/imgproc/rcvImgProc.red

Using these integral images, one may calculate sum, mean, standard deviation over arbitrary up-right rectangular region of the image in a constant time with only 4 points ABCD.



Using integral image, it is also possible to do variable-size image blurring, block correlation etc.
Integral image *ii* is defined according to :

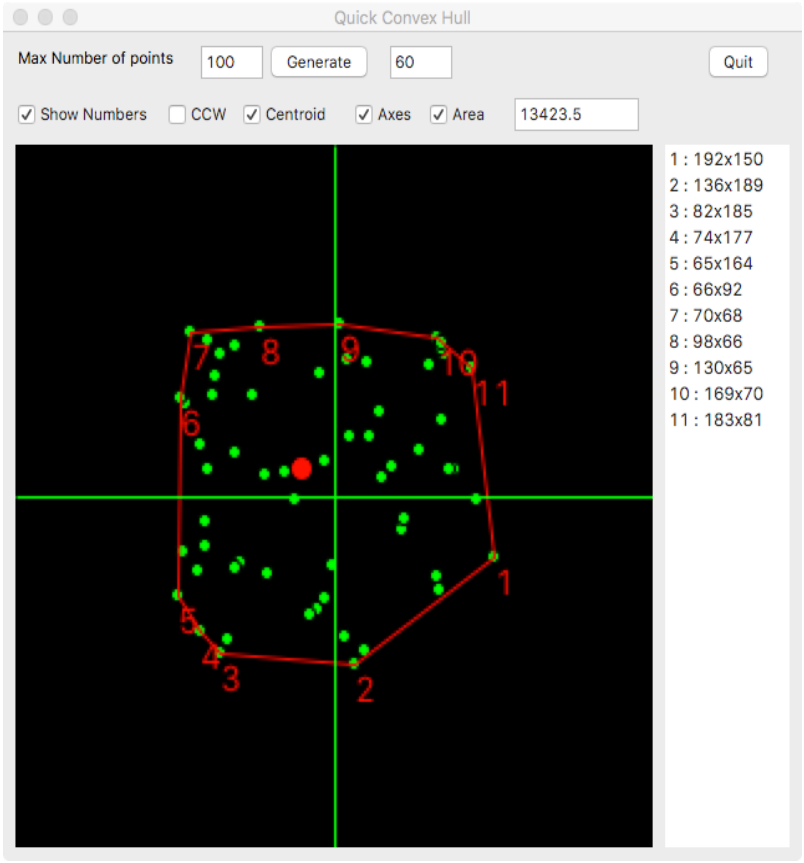
$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y')$$

rcvQuickHull

Finds the convex hull of a point set and returns as block

rcvQuickHull: function [points [block!] /cw/ccw]
points: Input 2D point set as pair!
Returns output convex hull as a block of pair
/cw/ccw : Orientation flag. If cw, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise. The assumed coordinate system has its X axis pointing to the right, and its Y axis pointing upwards.
defined in /libs/math/rcvStats.red

The convex Hull problem in geometry tries to find the smallest convex set containing the points.



There are many approaches for handling this problem, but for redCV we focused on the *Quick Hull algorithm*, which is one of the easiest to implement and has a reasonable expected running time of O(n log n). A clear explanation of the algorithm can be found here: http://www.ahristov.com/tutorial/geometry-games/convex-hull.html. Thanks to Alexander Hristov for the original Java code.

rcvContourArea

Calculates and returns the area of polygon generated by rcvQuickHull function as float

rcvContourArea: function [hull [block!] /signed]

hull: list of coordinates (as pair!) generated by the rcvQuickHull function

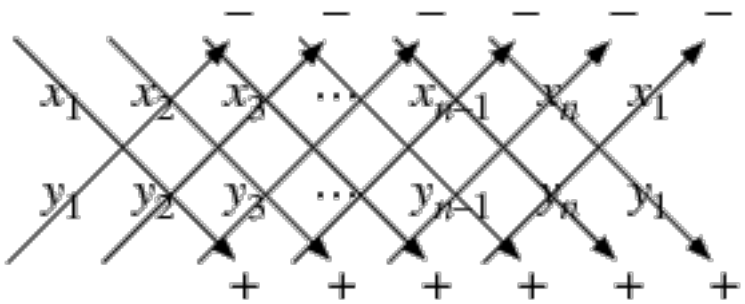
Return: area as float!

If signed refinement is used returns the signed area

defined in /libs/math/rcvStats.red

This function returns the (signed or not) area (A) of a planar **non-self-intersecting** polygon with vertices (x_1,y_1), ..., (x_n,y_n) according to the formula:

$$A = \frac{1}{2}(x_1 y_2 - x_2 y_1 + x_2 y_3 - x_3 y_2 + \ldots + x_{n-1} y_n - x_n y_{n-1} + x_n y_1 - x_1 y_n),$$



See Weisstein, Eric W. "Polygon Area." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/PolygonArea.html

Geometrical transformations

rcvFlip

Returns Left/Right, Up/Down or both directions image flip
rcvFlip: function [src [image!] dst [image!] /horizontal /vertical /both]
src: source image
dst: destination image
refinement for direction
defined in /libs/imgproc/rcvImgProc.red

source Left/Right Up/Down Both

rcvResizeImage

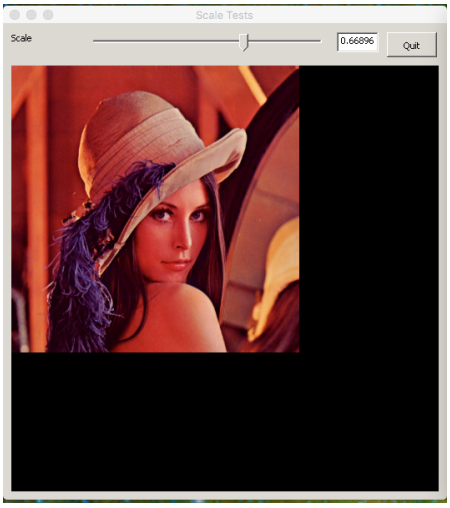
Resizes image, applies filter for Gaussian pyramidal up or downsizing if required, and returns image
rcvResizeImage: function [src [image!] iSize [pair!] /Gaussian]
src: source image
iSize: New size of the image as pair
/Gaussian: applies a 5x5 kernel Gaussian filter on image
defined in /libs/imgproc/rcvImgProc.red

```
img1: rcvLoadImage %../../images/lena.jpg
dst:  rcvCreateImage img1/size
iSize: 256x256
canvas: base iSize dst
nSize: 512x512
dst: rcvResizeImage dst nSize
canvas/size: nSize
```

rcvScaleImage

Sets the scale factors: Returns a Draw block
rcvScaleImage: function [factor [float!]]
factor: scale factor as float! Default value : 1.0 original size
defined in /libs/imgproc/rcvImgProc.red

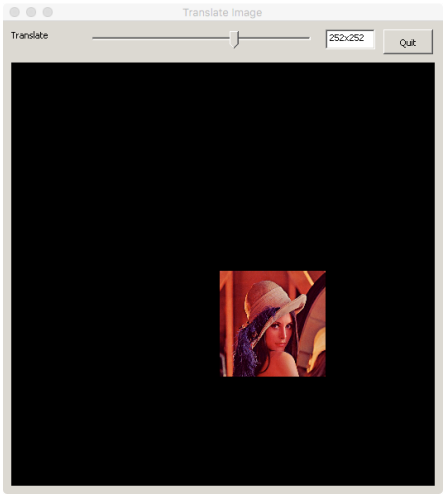
```
This function uses Draw Dialect and you have to add
the image instance to the draw block.
img1: rcvLoadImage %../../images/lena.jpg
factor: 1.0
drawBlk: rcvScaleImage factor
append drawBlk [img1]
...
```



rcvTranslateImage

Sets the origin for drawing commands: Returns a Draw block
rcvTranslateImage: function [scaleValue [float!] translateValue [pair!]]
scaleValue: float! value to reduce or increase image size
translateValue : pair to translate image in X and Y direction
defined in /libs/imgproc/rcvImgProc.red

```
This function uses Draw Dialect and you have to
add the image instance to the draw block.
img1: rcvLoadImage %../../images/lena.jpg
factor: 0x0
drawBlk: rcvTranslateImage 0.25 factor
append drawBlk [img1]
...
```



rcvRotateImage

Sets the clockwise rotation about a given point, in degrees : Returns a Draw block

rcvRotateImage: function [scaleValue [float!] translateValue [pair!] angle [float!] center [pair!]]

scaleValue: float! value to reduce or increase image size

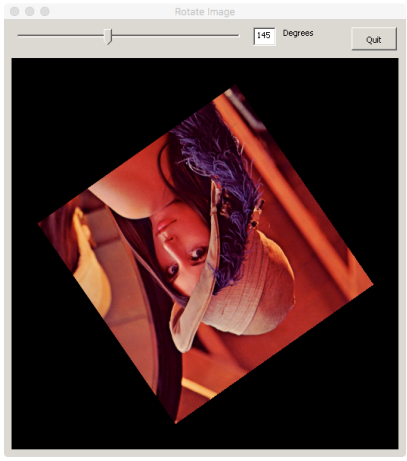
translateValue : pair to translate image in X and Y direction

angle: rotation of image in degrees

center: center of image rotation as pair! Default value 0x0

defined in /libs/imgproc/rcvImgProc.red

```
This function uses Draw Dialect and you
have to add the image instance to the draw
block.
img1: rcvLoadImage %../../images/lena.jpg
iSize: img1/size
centerXY: iSize / 2
rot: 0.0
drawBlk: rcvRotateImage 0.625 96x96 rot
centerXY
append drawBlk [img1]
...
```

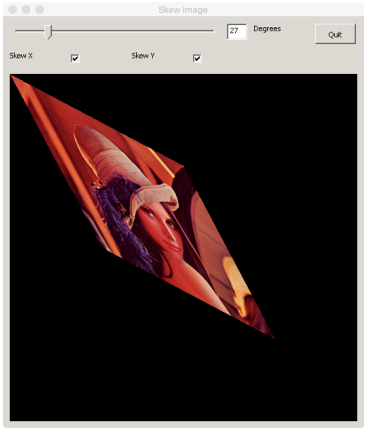


rcvSkewImage

Sets a coordinate system skewed from the original by the given number of degrees

rcvSkewImage: function [scaleValue [float!] translateValue [pair!] x [number!] y [number!]]
scaleValue: float! value to reduce or increase image size
translateValue : pair to translate image in X and Y direction
x: skew along the x-axis in degrees (integer! float!).
y: skew along the y-axis in degrees (integer! float!).
defined in /libs/imgproc/rcvImgProc.red

```
This function uses Draw Dialect and you have to
add the image instance to the draw block.
img1: rcvLoadImage %../../images/lena.jpg
x: 0
y: 0
drawBlk: rcvSkewImage 0.5 0x0 x y
append drawBlk [img1]
```



Distances Functions

rcvGetEuclidianDistance

Returns Euclidian distance between 2 points as float
rcvGetEuclidianDistance: function [a [pair!] b [pair!]]
a: first point as pair!
b: second point as pair!
defined in /libs/math/rcvDistance.red

rcvGetEuclidian2Distance

Returns Squared Euclidian distance between 2 points as float
rcvGetEuclidian2Distance: function [a [pair!] b [pair!]]
a: first point as pair!
b: second point as pair!
defined in /libs/math/rcvDistance.red

rcvGetManhattanDistance

Returns Manhattan distance between 2 points as float
rcvGetManhattanDistance: function [a [pair!] b [pair!]]
a: first point as pair!
b: second point as pair!
defined in /libs/math/rcvDistance.red

rcvGetChessboardDistance

Returns Chessboard distance between 2 points as float
rcvGetChessboardDistance: function [a [pair!] b [pair!]]
a: first point as pair!
b: second point as pair!
defined in /libs/math/rcvDistance.red

rcvGetChebyshevDistance

Returns Chebyshev distance between 2 points as float
rcvGetChebyshevDistance: function [a [pair!] b [pair!]]
a: first point as pair!
b: second point as pair!
defined in /libs/math/rcvDistance.red

rcvGetMinkowskiDistance

Returns Minkowski distance between 2 points as float
rcvGetMinkowskiDistance: function [a [pair!] b [pair!] p [float!]]
a: first point as pair!
b: second point as pair!

p : power value
defined in /libs/math/rcvDistance.red

if p = 1.0 same as rcvGetManhattanDistance if p = 2.0 same as rcvGetEuclidianDistance

rcvGetCamberraDistance

Returns Camberra fractional distance between 2 points as float
rcvGetCamberraDistance: function [a [pair!] b [pair!]]
a: first point as pair!
b: second point as pair!
defined in /libs/math/rcvDistance.red

rcvGetSorensenDistance

Returns Sorensen or Bray Curtis fractional distance between 2 points as float
rcvGetSorensenDistance: function [a [pair!] b [pair!]]
a: first point as pair!
b: second point as pair!
defined in /libs/math/rcvDistance.red

rcvDistance2Color

Returns tuple value modified by distance
rcvDistance2Color: function [dist [float!] t [tuple!]]
dist as float !
t as tuple !
defined in /libs/math/rcvDistance.red

rcvGetAngle

Returns angle in degrees from 2 points coordinates as float
rcvGetAngle: function [p [pair!] cg [pair!]]
p: first point as pair!
cg: second point as pair!
Returned value is in degrees
defined in /libs/math/rcvDistance.red

rcvGetAngleRadian

Returns angle in radian from p coordinates as float
rcvGetAngleRadian: function [p [pair!]]
defined in /libs/math/rcvDistance.red

Attention: needs a coordinate translation p - shape centroid. The function is useful for shape signature detection and polar coordinates transformation.

rcvRhoNormalization

Returns normalized block [0.0..1.0] of distances values

rcvRhoNormalization: function [b [block!]]

b: block of distance (rho) value to normalize

rcvVoronoiDiagram

Creates Voronoï diagram

rcvVoronoiDiagram: function [peaks [block!] peaksC [block!] img [image!] param1 [logic!] param2 [integer!] param3 [float!]]

defined in /libs/math/rcvDistance.red

peaks: block of pairs

peaksC: block of tuples

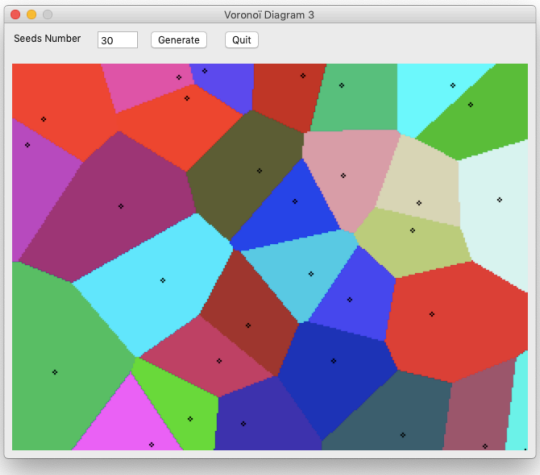
img: image for render

param1: logic! for seeds visualization

parm2: kind of distance used for render : 1: Euclidian 2: Manhattan 3: Minkowski 4: Chebyshev

param3: p value for Minkowski

param3 only for Minkowski distance with 3.0 default value



rcvDistanceDiagram

Creates Distance diagram based on Boleslav Březovský's sample

rcvDistanceDiagram: function [peaks [block!] peaksC [block!] img [image!] param1 [logic!] param2 [integer!] param3 [float!]]

peaks: block of pairs

peaksC: block of tuples

img: image for render

param1: logic! for seeds visualization

parm2: kind of distance used for render: 1: Euclidian 2: Manhattan 3: Minkowski 4: Chebyshev
param3: p value for Minkowski
defined in /libs/math/rcvDistance.red

param3 only for Minkowski distance with 3.0 default value

rcvKMInitData

Creates block data of centroid array
rcvKMInitData: function [count [integer!]]
count: number of elements of the array
defined in /libs/math/rcvDistance.red

rcvKMGenCentroid

Generates centroids initial values
rcvKMGenCentroid: function [array [block!]]
array: block generated by rcvKMInitData

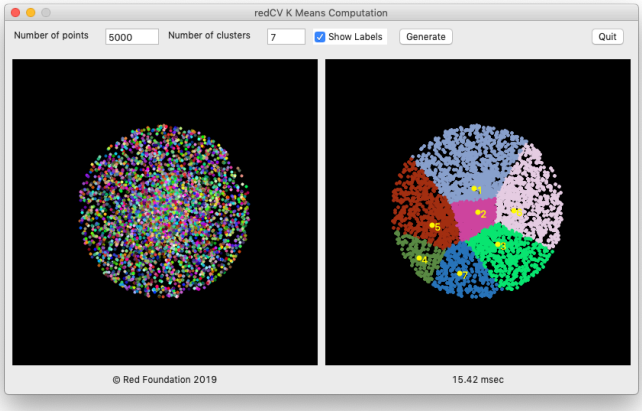
rcvKMInit

k-means first initialization
rcvKMInit: function [points [block!] centroid [block!] tmpblk [block!]]
points: data block
centroid: centroid block
tmpblk: temporary block used for sum computation
defined in /libs/math/rcvDistance.red

rcvKMCompute

Lloyd K-means clustering with convergence
rcvKMCompute: function [points [block!] centroid [block!]]
points: data block
centroid: centroid block
defined in /libs/math/rcvDistance.red

Attention: **k-means functions require redCV specific array datatype**. This array contains n vectors of 3 float values used to calculate x and y point values and group value. Group value is used for clustering points in clusters as illustrated below.



rcvMakeGradient

Makes a gradient matrix for contour detection (similar to Sobel) and returns max gradient value as integer

rcvMakeGradient: function [src [vector!] dst [vector!] mSize [pair!]]
src: integer matrix
dst: integer matrix
mSize: matrix size as a pair!
defined in /libs/math/rcvDistance.red

rcvMakeBinaryGradient

Makes a binary [0 1] matrix for contour detection

rcvMakeBinaryGradient: function [src [vector!] mat [vector!] maxG [integer!] threshold [integer!]]
src: integer matrix
mat: destination integer matrix
maxG : max gradient value
threshold : integer value for binary thresholding
defined in /libs/math/rcvDistance.red

rcvFlowMat

Returns the distance map to binarized gradient as float

rcvFlowMat: function [input [vector!] output[vector!] scale [float!]]
input: float source matrix
output: destination matrix including distances
returns max distance
defined in /libs/math/rcvDistance.red

rcvnormalizeFlow

Normalizes distance into 0..255 range according to scale value

rcvnormalizeFlow: function [input [vector!] factor [float!]]
input: integer matrix
factor: value used for normalization such as max gradient value
defined in /libs/math/rcvDistance.red

rcvGradient&Flow

Creates an image including flow and gradient values

rcvGradient&Flow: function [input1 [vector!] input2 [vector!] dst [image!]]
input1: flow integer matrix
input2: gradient integer matrix
dst: red image for mixing flow and gradient
defined in /libs/math/rcvDistance.red

rcvChamferDistance

Selects a pre-defined chamfer kernel

rcvChamferDistance: function [chamferMask [block!]]
Kernels calculated by Verwer, Borgefors and Thiel
cheessboard: copy [1 0 1 1 1 1]
chamfer3: copy [1 0 3 1 1 4]
chamfer5: copy [1 0 5 1 1 7 2 1 11]
chamfer7: copy [1 0 14 1 1 20 2 1 31 3 1 44]
chamfer13: copy [1 0 68 1 1 96 2 1 152 3 1 215 3 2 245 4 1 280 4 3 340 5 1 346 6 1 413]
defined in /libs/math/rcvDistance.red

rcvChamferCreateOutput

Creates a distance map (float!). Returns vector

rcvChamferCreateOutput: function [mSize [pair!]]
mSize: matrix size as a pair value
defined in /libs/math/rcvDistance.red

rcvChamferInitMap

Initializes distance map
rcvChamferInitMap: function [input [vector!] output [vector!]]
input: a binary [0/1] matrix
output: must be a vector of float!
defined in /libs/math/rcvDistance.red

If input value= 0, the point belongs to the object and thus the distance is 0.0 If input value= 1, the point is outside and the distance (-1.0) must be calculated

rcvChamferCompute

Calculates the distance map to binarized gradient

rcvChamferCompute: function [output [vector!] chamfer [block!] mSize [pair!]]
output: float matrix created by rcvChamferInitMap function is used
chamfer: selected pre-defined kernel used for distance calculation (e.g. chamfer5)
defined in /libs/math/rcvDistance.red

rcvChamferNormalize

Normalizes distance map

rcvChamferNormalize: function [output [vector!] value [integer!]]
defined in /libs/math/rcvDistance.red

Image enhancement

rcvMakeTranscodageTable

Creates a transcoding 256 vector for affine enhancement

rcvMakeTranscodageTable: function [n [percent!]]

n: percent of values to exclude

defined in /libs/math/rcvStats.red

This function is used by rcvContrastAffine method. See below.

rcvContrastAffine

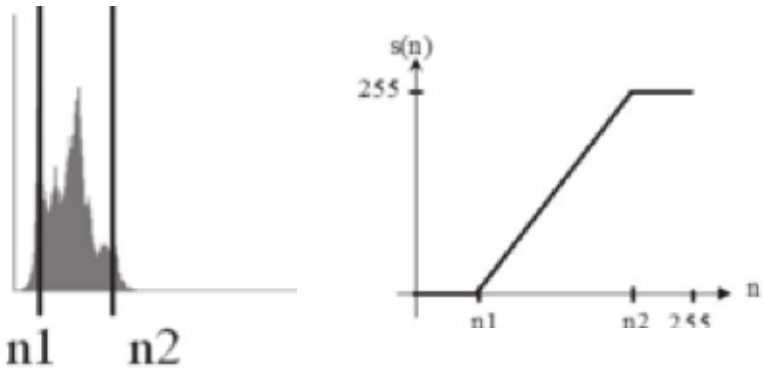
Enhances image contrast with affine function

rcvContrastAffine: function [image [vector!] n [percent!]]

image: a 8-bit matrice

n: percent of values to exclude

defined in /libs/math/rcvStats.red



S(n)= 0 if n<=n1, s(n)= 255 * (n-n1)/(n2-n1) if n1<n<n2 and s(n)=255 if n >= n2

rcvHistogramEqualization

This function performs histogram equalization on the input image array

rcvHistogramEqualization: function [image [vector!] gLevels [integer!]]

image: a 8-bit matrice

gLevels: number of gray levels in the new image

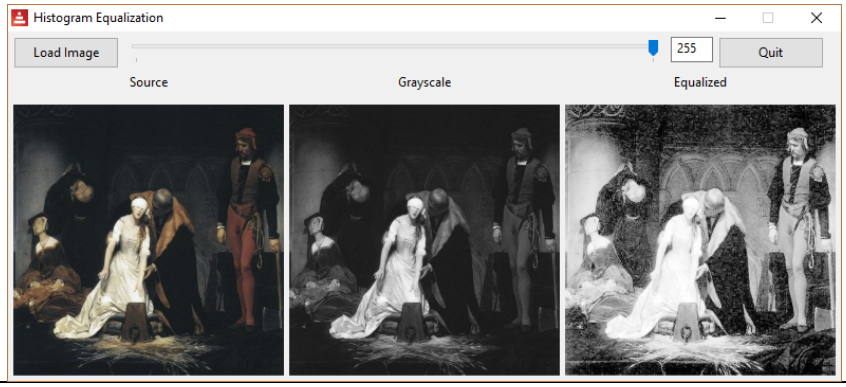
defined in /libs/math/rcvStats.red

This algorithm performs first the histogram of the source image and then calculates the probability-density function of a pixel value n: $p_1(n)$ is the probability of finding a pixel with the value n in the image. Then the following equation is applied

$$f(a) = D_m \frac{1}{Area_1} \sum_{i=0}^{a} H_c(a)$$

$H_c(a)$ is the histogram of the original image c and D_m is the number of gray levels in the new image b. Area is the size of the image. f(a) simply takes the probability density function for the values in image b and multiplies this by the cumulative density function of the values in image c.

This function is useful for improving contrast in low-contrasted images or simply modifying the contrast of image.



Thresholding

rcv2BWFilter

Binarization of RGB image according to threshold value
rcv2BWFilter: function [src [image!] dst [image!] thresh [integer!]]
src: image
dst: image
thresh: threshold integer value
Defined in /libs/core/rcvCore.red

rcvThreshold

Applies fixed-level threshold to array elements. Images are processed as grayscale.
rcvThreshold: function [src [image!] dst [image!] thresh [integer!] mValue [integer!] /binary /binaryInv /trunc /toZero /toZeroInv
src: image
dst: image
thresh: threshold integer value
mValue: maximal integer value
refinements are used for thresholding type
Defined in /libs/core/rcvCore.red

binary:dst(x,y) = mValue, if src(x,y)>threshold, 0, otherwise
binarryInv: dst(x,y) = 0, if src(x,y)>threshold, mVvalue, otherwise
trunc: dst(x,y) = threshold, if src(x,y)>threshold , src(x,y), otherwise
toZero: dst(x,y) = src(x,y), if src(x,y)>threshold, 0, otherwise
toZeroInv: dst(x,y) = 0, if src(x,y)>threshold, src(x,y), otherwise

rcvInRange

Extracts sub array from image according to lower and upper rgb values
rcvInRange: function [src [image!] dst [image!] lower [tuple!] upper [tuple!] op [integer!]]
src: source image
dst: destination image
lower: lower tuple
upper: lower tuple
op: if op = 0 image is binarized else colors are extracted
Defined in /libs/core/rcvCore.red

rcvInRangeMat

Extracts sub array from matrix according to lower and upper values

rcvInRangeMat: function [src [vector!] dst [vector!] lower [integer!] upper [integer!] op [integer!]

src: source matrix

dst: destination matrix

lower: lower value

upper: lower upper

op: if op = 0 image is binarized else gray values are extracted

Defined in /libs/matrix/rcvMatrix.red

Spatial filtering

Many filters are based on 2-D convolution. The 2-D convolution operation isn't extremely fast, unless you use small (3x3 or 5x5) filters. There are a few rules about the filter. Its size has to be generally uneven, so that it has a center, for example 3x3, 5x5, 7x7 or 9x9 are ok. Apart from using a kernel matrix, convolution operation also has a multiplier factor and a bias. After applying the filter, the factor will be multiplied with the result, and the bias added to it. So, if you have a filter with an element 0.25 in it, but the factor is set to 2, all elements of the filter are multiplied by two so that element 0.25 is actually 0.5. The bias can be used if you want to make the resulting image brighter.

rcvMakeGaussian

Creates a Gaussian uneven kernel

rcvMakeGaussian: function [kSize [pair!] sigma[float!]]
kSize: uneven pair for kernel e.g 3x3
sigma: required variance e.g. 1.0
defined in /libs/imgproc/rcvImgProc.red

Creates a Gaussian uneven kernel with the following equation

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and σ is the standard deviation of the distribution.

rcvGaussianFilter

Fast Gaussian 2D filter

rcvGaussianFilter: function [src [image!] dst [image!] kSize [pair!] sigma [float!]]
src: image
dst: image
kSize: kernel size
sigma: required variance e.g. 1.0
defined in /libs/imgproc/rcvImgProc.red

rcvDoGFilter:

Difference of Gaussian

```
function [
        src     [image!] ;source image
        dst     [image!] ;destination image
        kSize   [pair!]  ;kernel size
        sig1    [float!] ;variance 1
        sig2    [float!] ;variance 2
        factor  [float!] ; normalization
]
```

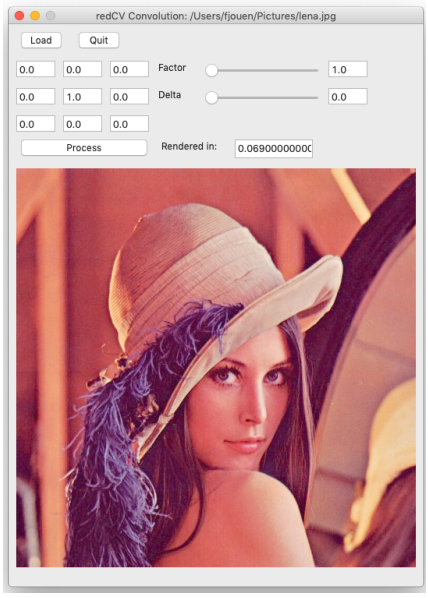
rcvConvolve

Convolves an image with the kernel

rcvConvolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]]
src: source image
dst: destination image
kernel: kernel matrix as block!
factor: multiplier factor as float!
delta: bias for image brightness
defined in /libs/imgproc/rcvImgProc.red

This function is a general convolution function that can be used for creating a lot of image filters.

```
img1: rcvLoadImage %../../images/lena.jpg
dst:  rcvCreateImage img1/size
gaussian: [0.0 0.0 0.0
           0.0 1.0 0.0
           0.0 0.0 0.0]
rcvConvolve img1 dst gaussian 1.0 0.0
```



rcvConvolveMat

Convolves a 2-D matrix with the kernel

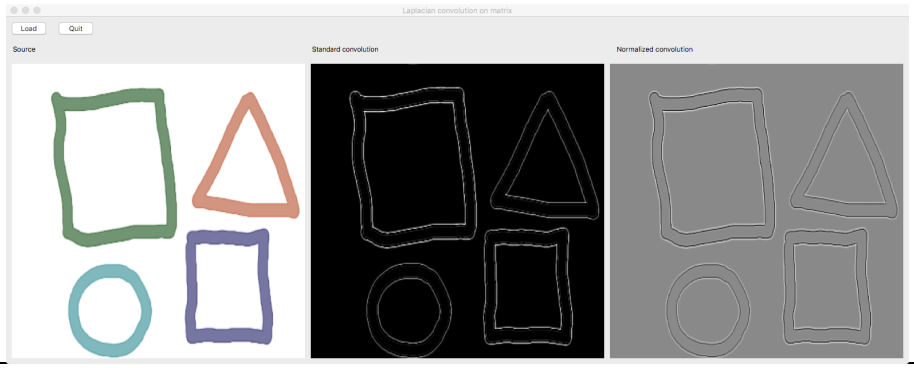
rcvConvolveMat: function [src [vector!] dst [vector!] mSize[pair!] kernel [block!] factor [float!] delta [float!]]
src: source matrix
dst: destination matrix
mSize: matrix size as a pair!
kernel: kernel matrix as block!
factor: multiplier factor as float!
delta: bias for image brightness
defined in /libs/matrix/rcvMatrix.red

rcvConvolveNormalizedMat

Convolves a 2-D matrix with the kernel and applies a scale to result

rcvConvolveNormalizedMat: function [src [vector!] dst [vector!] mSize[pair!] kernel [block!] factor [float!] delta [float!]]
src: source matrix
dst: destination matrix
mSize: matrix size as a pair!
kernel: kernel matrix as block!
factor: multiplier factor as float!
delta: bias for image brightness
defined in /libs/matrix/rcvMatrix.red

This function is two-pass: First, calculates minimal and maximal weighted sums resulting from the convolution process. This allows to calculate a scale equal to 255 / (maximal – minimal). Then each matrix convoluted value is rescaled by (value – minimal) * scale. This means that whatever the sign of convoluted values, values are transformed into bytes [0..255] values.



rcvFastConvolve

Convolves 8-bit and 1-channel image with the kernel

rcvFastConvolve: function [src [image!] dst [image!] channel [integer!] kernel [block!] factor [float!] delta [float!]]
src: source image
dst: destination image
channel: image channel to process (RGB)

kernel: kernel matrix as block!
factor: multiplier factor as float!
delta: bias for image brightness
defined in /libs/imgproc/rcvImgProc.red

rcvFilter2D

Basic convolution filter
rcvFilter2D: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [integer!]]
src: image
dst: image
kernel: kernel matrix as block!
factor: multiplier factor as float!
delta: bias for image brightness
defined in /libs/imgproc/rcvImgProc.red

Similar to convolution but the sum of the weights is computed during the summation, and used to scale the result.

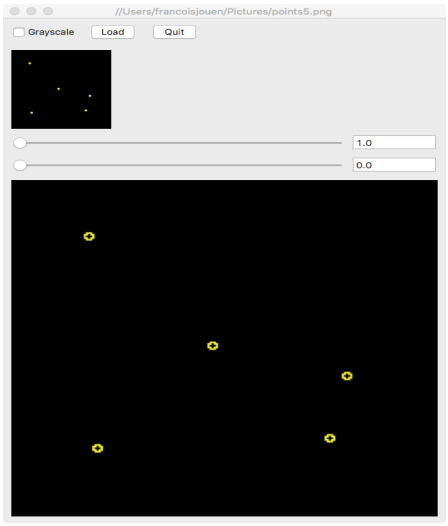
rcvFastFilter2D

Fast convolution filter
rcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]]
src: image
dst: image
kernel: kernel matrix as block!
defined in /libs/imgproc/rcvImgProc.red

A faster version without controls on pixel value! Basically for 1 channel gray scaled image. The sum of the weights is computed during the summation, and used to scale the result

rcvPointDetector

Convolution allowing to find dots in image or matrix
rcvPointDetector: function [src [image! vector!] dst [image! vector!] param1 [float!] param2 [float!]]
src: source image or matrix
dst: destination image or matrix
param1 : threshold value
param2 : luminance value
defined in /libs/imgproc/rcvImgProc.red


//Users/francoisjouen/Pictures/points5.png
Grayscale
Load
Quit
1.0
0.0

Fast edge detection

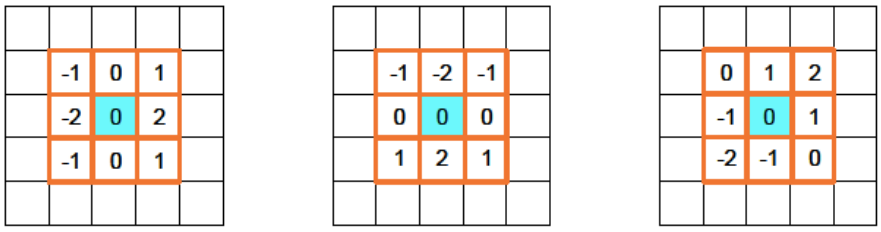
You can, of course, build your own edges detectors by convolution (see rcvConvolve function). Here are included a set of classical and predefined filters which are fast and easy to use. They are two components in derivative filters. The x derivative extracts vertical edges and the y derivative extracts horizontal edges.

First derivative filters

rcvSobel

Direct Sobel edges detection for image or matrix
function [src [image! vector!] dst [image! vector!] iSize [pair!] direction [integer!] op [integer!]]
src: image or matrix as vector
dst: image or matrix as vector
iSize: image or matrix size as pair
direction:
1: returns vertical gradient direction (Gx)
2: returns horizontal gradient direction (Gy)
3: both gradient directions by G= Gx + Gy
4: both gradients estimated by G= Sqrt (Gx^2 +Gy^2)
5: G direction
op: for kernel inversion [1, 2] and for kernel combination [3,4]
defined in /libs/imgproc/rcvImgProc.red

Used Hx, Hy and Ho (oblique) kernels




```
img1: rcvLoadImage %../../images/lena.jpg
img2: rcvCreateImage img1/size
img3: rcvCreateImage img1/size
rcv2Gray/average img1 img2                  ; Grayscaled image
rcvSobel img2 img3 img1/size        4       ; Direct Sobel on image
```

```
img1: rcvLoadImage %../../images/lena.jpg
img2: rcvCreateImage img1/size
mat1: rcvCreateMat 'integer! intSize img1/size
mat2: rcvCreateMat 'integer! intSize img1/size
```

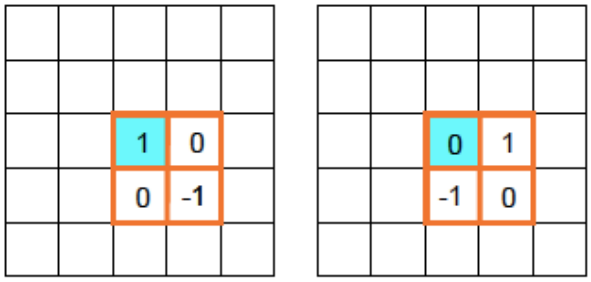
```
rcvImage2Mat img1 mat1          ; Converts image to 1 Channel matrix [0..255]
rcvSobel mat1 mat2 img1/size    ; Sobel detector on Matrix
```

rcvRoberts

Robert's cross edges detection for image or matrix

rcvRoberts: function [src [image! vector!] dst [image! vector!] iSize [pair!] direction [integer!]
src: image or matrix as vector
dst: image or matrix as vector
iSize: image or matrix size as pair
factor: multiplier factor as float!
delta: bias for image brightness
direction:
1: returns vertical gradient direction (Gx)
2: returns horizontal gradient direction (Gy)
3: both gradient directions by G= Gx + Gy
4: both gradients estimated by G= Sqrt (Gx^2 +Gy^2)
defined in /libs/imgproc/rcvImgProc.red

Used Hx and Hy kernels

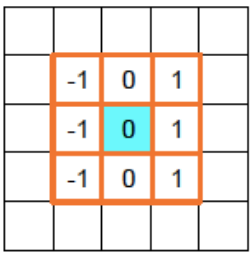


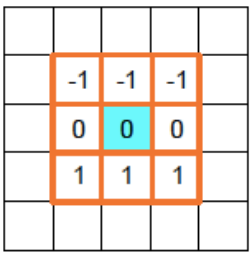
rcvPrewitt

Computes an approximation of the gradient magnitude of the input image

rcvPrewitt: function [src [image! vector!] dst [image! vector!] iSize [pair!] direction [integer!] op [integer!]
src: image or matrix
dst: image or matrix
iSize: size of the image or the matrix
direction:
1: returns vertical gradient direction (Gx)
2: returns horizontal gradient direction (Gy)
3: both gradient directions by G= Gx + Gy
4: both gradients estimated by G= Sqrt (Gx^2 +Gy^2)
5: G direction (angles)
op: for kernel inversion [1, 2] and for kernel combination [3,4]
defined in /libs/imgproc/rcvImgProc.red

Used Hx and Hy kernels. Hx and Hy can be inverted.





rcvMDIF

Computes an approximation of the gradient magnitude of the input image

rcvPrewitt: function [src [image! vector!] dst [image! vector!] iSize [pair!] direction [integer!]]

src: image or matrix

dst: image or matrix

iSize: size of the image or the matrix

direction:

1: returns vertical gradient direction (Gx)

2: returns horizontal gradient direction (Gy)

3: both gradient directions by G= Gx + Gy

4: both gradients estimated by G= Sqrt (Gx^2 +Gy^2)

5: G direction (angles)

rcvGradientMasks

Fast gradient mask filter

rcvGradientMasks: function [src [image!] dst [image!] direction [integer!]]

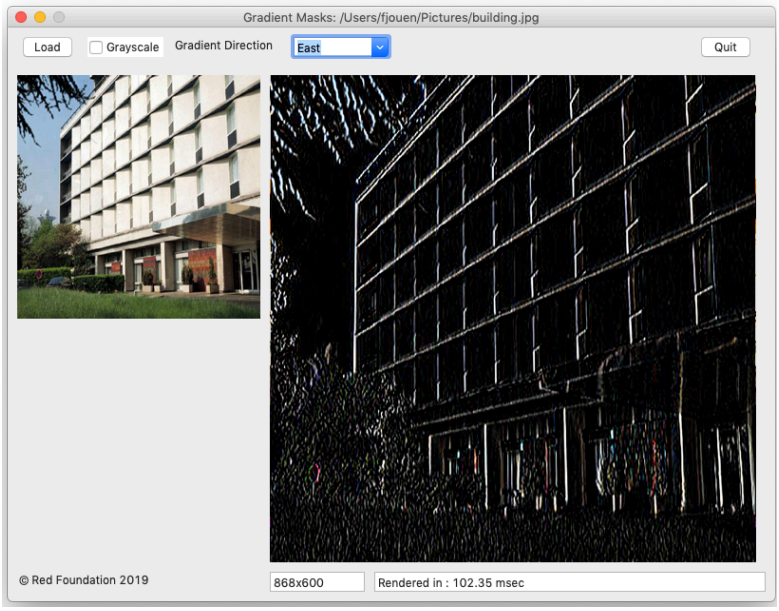
src: image

dst: image

direction: integer

defined in /libs/imgproc/rcvImgProc.red

Very efficient use of Prewitt's Filter for line detection.
1: North 2: Northeast 3: East 4: Southeast 5: South 6: Southwest 7: West 8: Northwest



rcvKirsch

Computes an approximation of the gradient magnitude of the input image

rcvKirsch: function [src [image! vector!] dst [image! vector!] iSize [pair!] direction [integer!] op [integer!]]

src: image or matrix

dst: image or matrix

iSize: size of the image or the matrix

direction:

1: returns vertical gradient direction (Gx)

2: returns horizontal gradient direction (Gy)

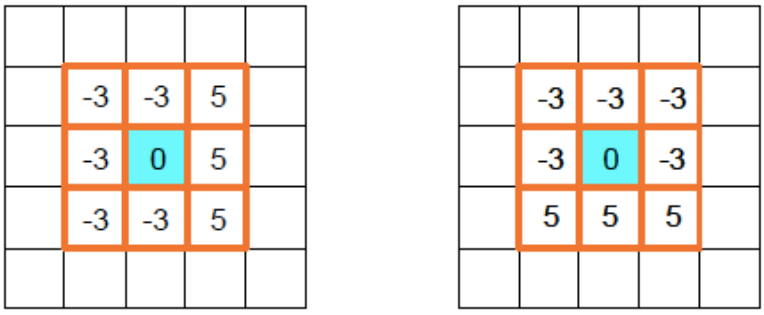
3: both gradient directions by G= Gx + Gy

4: both gradients estimated by G= Sqrt (Gx^2 +Gy^2)

op: for kernel inversion [1, 2] and for kernel combination [3,4]

defined in /libs/imgproc/rcvImgProc.red

Used Hx and Hy kernels



rcvGradNeumann

Computes the discrete gradient by forward finite differences and Neumann boundary conditions

rcvGradNeumann: function [src [image!] d1 [image!] d2 [image!]]
src: source image
d1: image for derivative along the x axis
d2: image derivative along the y axis
defined in /libs/imgproc/rcvImgProc.red

rcvDivNeumann

Computes the divergence by backward finite differences

rcvDivNeumann: function [src [image!] d1 [image!] d2 [image!]]
src: source image
d1: image for derivative along the x axis
d2: image derivative along the y axis
defined in /libs/imgproc/rcvImgProc.red

rcvRobinson:

Robinson Filter

rcvRobinson: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/imgproc/rcvImgProc.red

Second derivative filters

These filters use partial second derivative of an image or a matrix according to the equations

In x direction: $(\frac{\partial^2 I(x,y)}{\partial x^2})$ and in Y direction: $(\frac{\partial^2 I(x,y)}{\partial y^2})$

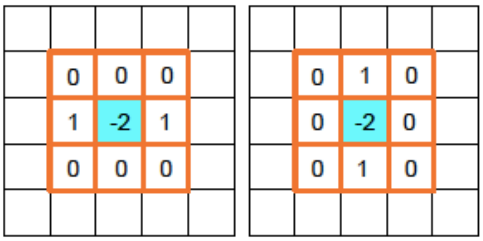
rcvDerivative2

A fast approximation of the second derivative of an image

rcvDerivative2: function [src [image! vector!] dst [image! vector!] iSize [pair!] factor [float!] direction [integer!]
src: image or matrix
dst: image or matrix
iSize: size of the image or the matrix
Factor: multiplier factor as float!

direction:
1: returns vertical gradient direction (Gx)
2: returns horizontal gradient direction (Gy)
3: both gradient directions by G= Gx + Gy
defined in /libs/imgproc/rcvImgProc.red

Used Hx and Hy kernels

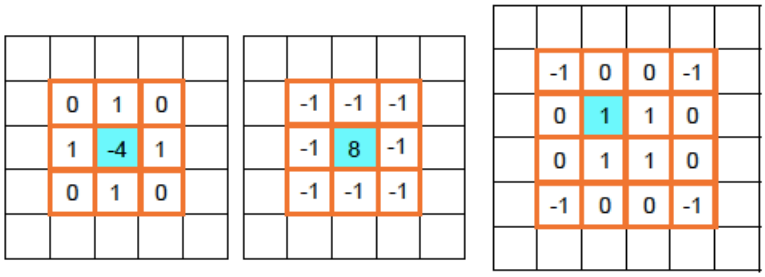


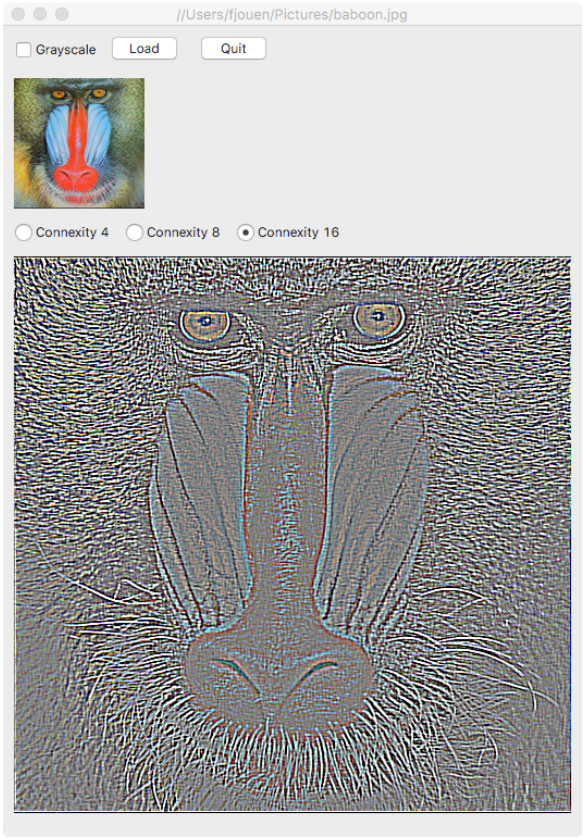
rcvLaplacian

Computes the Laplacian of an image or matrix. The Laplacian is an approximation of the second derivative of an image

rcvLaplacian: function [src [image! vector!] dst [image! vector!] iSize [pair!] connexity [integer!]
src: image or matrix
dst: image or matrix
iSize: size of the image or the matrix
connexity: neighbor pixels (4, 8 16)
defined in /libs/imgproc/rcvImgProc.red

Used kernels for 4, 8 or 16 connexity




//Users/fjouen/Pictures/baboon.jpg
Grayscale
Load
Quit
Connexity 4
Connexity 8
Connexity 16

rcvDiscreteLaplacian

Discrete Laplacian Filter
rcvDiscreteLaplacian: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/imgproc/rcvImgProc.red

rcvLaplacianOfRobinson

Laplacian of Robinson Filter
rcvLaplacianOfRobinson: function [src [image!] dst [image!]]
src: image
dst: image
defined in /libs/imgproc/rcvImgProc.red

Canny Filter

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny filter requires grayscale images. redCV includes a series of functions for Canny filter.

rcvEdgesGradient

Image gradients with hypot function
rcvEdgesGradient: function [srcX [image!] srcY [image!] mat[vector!]]
srcX: X Sobel derivative image
srcY: Y Sobel derivative image
mat: result gradient float matrix
defined in /libs/imgproc/rcvImgProc.red

rcvEdgesDirection

Angles in degrees with atan2 functions
rcvEdgesDirection: function [srcX [image!] srcY [image!] matA[vector!]]
srcX: X Sobel derivative image
srcY: Y Sobel derivative image
matA: result angle float matrix
defined in /libs/imgproc/rcvImgProc.red

rcvEdgesSuppress

Non-maximum suppression
rcvEdgesDirection: function [matA [vector!] matG[vector!] matS [vector!] mSize[pair!]]
matA: angle float matrix
matG : gradient float matrix
matS: result float matrix
mSize : matrix size as a pair
defined in /libs/imgproc/rcvImgProc.red

rcvDoubleThresh

Double thresholding

rcvDoubleThresh: function [gradS [vector!] doubleT [vector!]
lowT [integer!] highT [integer!] lowV [integer!] highV [integer!]]
matG : gradient float matrix
doubleT: result integer matrix
lowT: low threshold value
highT: high threshold value
lowV: low value associated to low threshold
highV: high value associated to high threshold
defined in /libs/imgproc/rcvImgProc.red

```
if v < lowT [_setIntValue as integer! mDTValue 0 unit2]
f all [v >= lowT v <= highT] [_setIntValue as integer! mDTValue weak unit2]
if v >= highT [_setIntValue as integer! mDTValue strong unit2]
```

rcvHysteresis

non-maximum suppression to thin out the edges

function [doubleT [vector!] edges [vector!] iSize [pair!] weak [integer!] strong [integer!]]
doubleT: integer matrix
edges: result integer matrix
iSize: matrix size as pair
weak: low value associated to low threshold
strong: high value associated to high threshold
defined in /libs/imgproc/rcvImgProc.red

Lines detection

redCV includes Hough transform operator. The Hough transformation is a great way to detect lines in an image and it is quite useful for a number of computer vision tasks (see http://en.wikipedia.org/wiki/Hough_transform).
You can find here http://www.keymolen.com/2013/05/hough-transformation-c-implementation.html a very clear and detailed explanation. Thanks a lot to Bruno Keymolen for his original C++ code.

rcvMakeHoughAccumulator

Creates Hough accumulator as vector
rcvMakeHoughAccumulator: func [w [integer!] h [integer!]]
w: source image width
h: source image height
defined in /libs/imgproc/rcvImgProc.red

rcvGetAccumulatorSize

Gets Hough space accumulator size as pair
rcvGetAccumulatorSize: function [acc [vector!]]
defined in /libs/imgproc/rcvImgProc.red

rcvHoughTransform

Makes Hough transform
rcvHoughTransform: function [mat [vector!] accu [vector!] w [integer!] h [integer!]]
defined in /libs/imgproc/rcvImgProc.red

rcvGetHoughLines

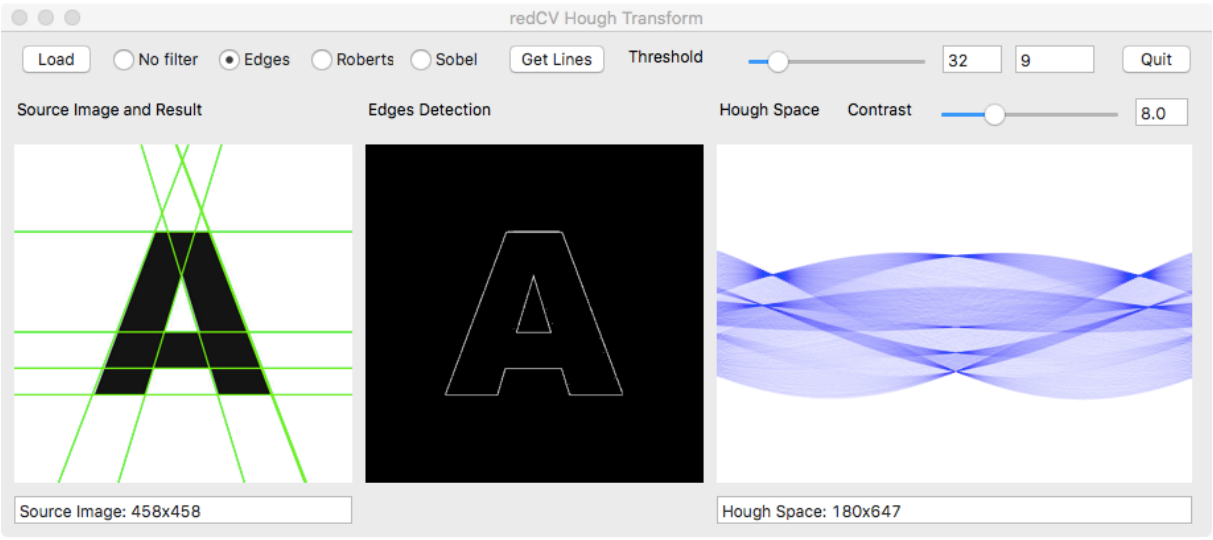
Gets lines in the accumulator according to threshold
rcvGetHoughLines: func [accu [vector!] img [image!] threshold [integer!] lines [block!]]
defined in /libs/imgproc/rcvImgProc.red

rcvHough2Image

Makes Hough space as red image
rcvHough2Image: function [mat [vector!] dst [image!] contrast [float!]]
defined in /libs/imgproc/rcvImgProc.red

Code Sample

```
rcv2BW edges bw                                    ; B&W image [0 255]
rcvImage2Mat bw mat                                ; B&W image to mat
acc: rcvMakeHoughAccumulator imgW imgH             ; makes Hough accumulator
rcvHoughTransform mat acc imgW imgH 128            ; performs Hough transform
hSpace: rcvCreateImage rcvGetAccumulatorSize acc   ; creates Hough space image
rcvHough2Image acc hSpace contrast                 ; shows Hough space
```



rcvLineDetection

Fast line detection

function [src [image!] dst [image!] direction [integer!]]

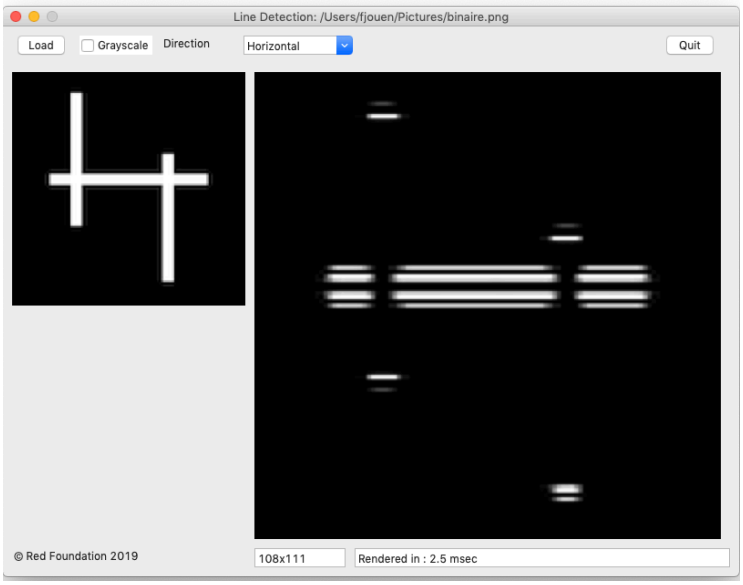
src: source image

dst: destination image

direction : orientation

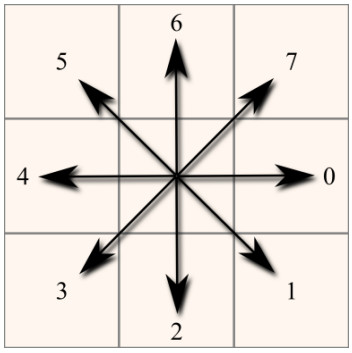
1 : horizontal 2: vertical 3: left diagonal 4: right diagonal

defined in /libs/imgproc/rcvImgProc.red


Line Detection: /Users/fjouen/Pictures/binaire.png
Load
Grayscale
Direction
Horizontal
Quit
© Red Foundation 2019
108x111
Rendered in : 2.5 msec

Shape detection

redCV includes Freeman chain code operator. The basic principle of chain codes is to separately encode each connected component, or "blob", in the image. For each such region, a point on the boundary is selected and its coordinates are transmitted. The encoder then moves along the boundary of the region and, at each step, transmits a number representing the direction of this movement.



You need a binary matrix [0..1] or [0..255] corresponding to your source image. See rcvMakeBinaryMat function.

rcvMatGetBorder

Gets pixels that belong to shape border
function [mat [vector!] matSize [pair!] value [integer!] border [block!]]
mat: source binary matrix
matSize: matrix size as pair
value: pixel value (default 1)
border: a block to store pixels direction
defined in /libs/matrix/rcvMatrix.red

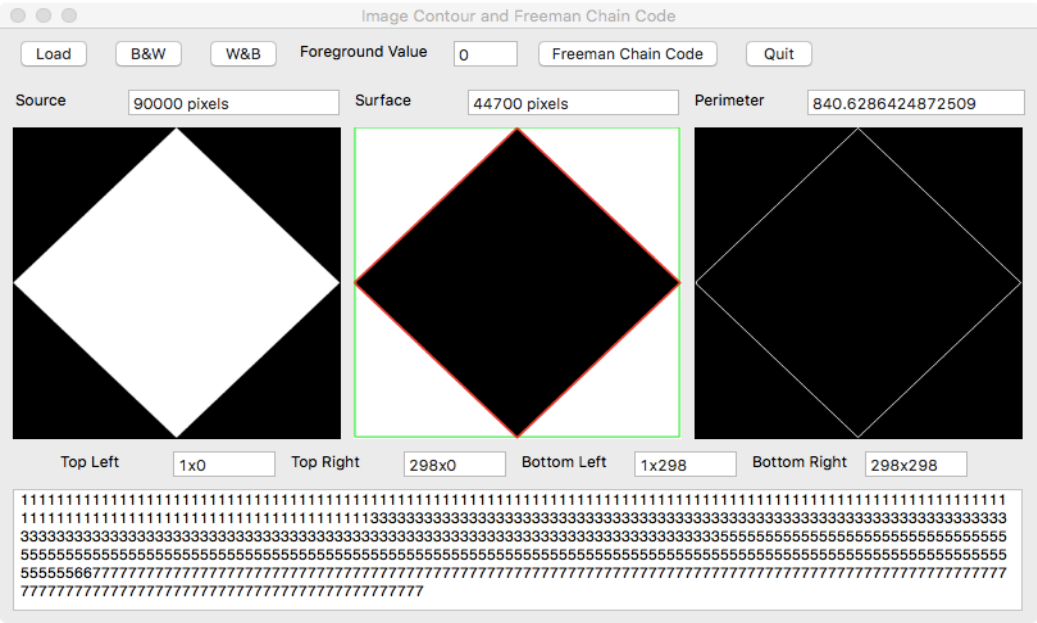
rcvMatGetChainCode

Gets Freeman Chain code (integer)
function [mat [vector!] matSize [pair!] coord [pair!] value [integer!]!
mat: source binary matrix
matSize: matrix size as pair
coord: pixel x and y coordinates
value: pixel value (default 1)
defined in /libs/matrix/rcvMatrix.red

The function uses the current pixel and explores pixel neighbors in order to get the direction of the next pixel. Top-left pixel (first value of the border block) is used as starting coordinate, then the encoder clockwise processes next pixels.

ATTENTION: you have to write a function to store chain codes such as (see imagecontour.red sample. Visited is a temporary matrix used to store visited pixels)

```
getCodes: does [
	visited: rcvCreateMat 'integer! 32 matSize
	border: copy []
	rcvMatGetBorder bmat matSize 1 border
	foreach p border [rcvSetInt2D visited matSize p 255]
	count: length? border
	p: first border
	i: 1
	s: copy ""
	clear r/text
	perimeter: 0.0
	while [i < count] [
		d: rcvMatGetChainCode visited matSize p 255
		rcvSetInt2D visited matSize p 0 ; pixel is visited
		if d >= 0 [append s form d]; only external pixels -1: internal
		switch d [
			0	[p/x: p/x + 1 perimeter: perimeter + 1.0]			; east
			1	[p/x: p/x + 1 p/y: p/y + 1 perimeter: perimeter + sqrt 2]	; southeast
			2	[p/y: p/y + 1 perimeter: perimeter + 1.0]			; south
			3	[p/x: p/x - 1 p/y: p/y + 1 perimeter: perimeter + sqrt 2]	; southwest
			4	[p/x: p/x - 1 perimeter: perimeter + 1.0]			; west
			5	[p/x: p/x - 1 p/y: p/y - 1 perimeter: perimeter + sqrt 2]	; northwest
			6	[p/y: p/y - 1 perimeter: perimeter + 1.0]			; north
			7	[p/x: p/x + 1 p/y: p/y - 1 perimeter: perimeter + sqrt 2]	; northeast
		]
		i: i + 1
	]
]
```



By using Freeman chain code and distance functions you can also make shape signature analysis with redCV.

Mathematical morphology

rcvCreateStructuringElement

The function allocates, fills, and returns a block, which can be used as a structuring element in the morphological operations

cvCreateStructuringElement: function [kSize [pair!] /rectangle /cross]
kSize: Kernel size (e.g. 3x3)
/refinements
defined in /libs/imgproc/rcvImgProc.red

Refinement is used to create a cross-shaped element or a rectangular element

rcvErode

Erodes image by using structuring element

rcvErode: function [src [image!] dst [image!] kSize [pair!] kernel [block!]]
src: image
dst: image
kSize: kernel size as pair!
Kernel: block generated by cvCreateStructuringElement or customed structuring element
defined in /libs/imgproc/rcvImgProc.red

The function rcvErode erodes the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the minimum is taken:

$$\texttt{dst=erode(src,element):}\quad \texttt{dst(x,y)=}\min_{\texttt{((x',y') in element)}}\texttt{src(x+x',y+y')}$$

rcvErodeMat:

Erodes matrice by using structuring element

rcvErodeMat: function [src [vector!] dst [vector!] mSize [pair!] kSize [pair!] kernel [block!]]
src: matrix
dst: matrix
mSize: matrix size as pair!
kSize: kernel size as pair!
defined in /libs/imgproc/rcvImgProc.red

rcvDilate

Dilates image by using structuring element

rcvDilate: function [src [image!] dst [image!] kSize [pair!] kernel [block!]]
src: image
dst: image
kSize: kenel size as pair!
kernel: block generated by cvCreateStructuringElement or customed structuring element
defined in /libs/imgproc/rcvImgProc.red

The function rcvDilate dilates the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the maximum is taken:

$$dst=dilate(src,element): \quad dst(x,y)=\max_{((x',y') \text{ in element})} src(x+x',y+y')$$

rcvDilateMat

Dilates matrix by using structuring element

rcvDilateMat: function [src [vector!] dst [vector!] mSize [pair!] kSize [pair!] kernel [block!]]
src: matrix
dst: matrix
mSize: matrix size as pair!
kSize: kernel size as pair
defined in /libs/imgproc/rcvImgProc.red

rcvOpen

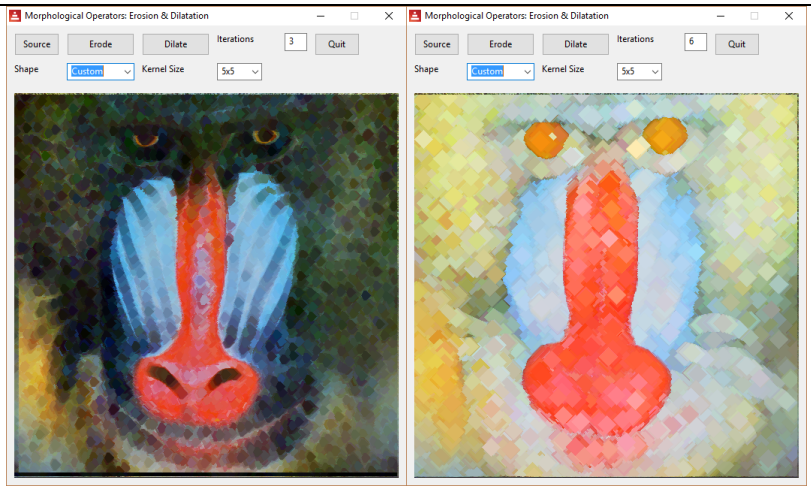
Erodes and Dilates image by using structuring element

rcvOpen: function [src [image!] dst [image!] kSize [pair!] kernel [block!]]
src: image
dst: image
kSize: kenel size as pair!
kernel: block generated by cvCreateStructuringElement or customed structuring element
defined in /libs/imgproc/rcvImgProc.red

rcvClose

Dilates and Erodes image by using structuring element

rcvClose: function [src [image!] dst [image!] kSize [pair!] kernel [block!]]
src: image
dst: image
kSize: kenel size as pair!
kernel: block generated by cvCreateStructuringElement or customed structuring element
defined in /libs/imgproc/rcvImgProc.red



rcvMGradient

Performs advanced morphological transformations using erosion and dilatation as basic operations

rcvMGradient: function [src [image!] dst [image!] kSize [pair!] kernel [block!] /reverse]
src: image
dst: image
kSize: kenel size as pair!
kernel: block generated by cvCreateStructuringElement or customed structuring element
defined in /libs/imgproc/rcvImgProc.red

dst=dilate src – erode src
/reverse : dst=erode src – dilate src

rcvTopHat

Performs advanced morphological transformations

rcvTopHat: function [src [image!] dst [image!] kSize [pair!] kernel [block!]]
src: image
dst: image
kSize: kenel size as pair!
kernel: block generated by cvCreateStructuringElement or customed structuring element
defined in /libs/imgproc/rcvImgProc.red

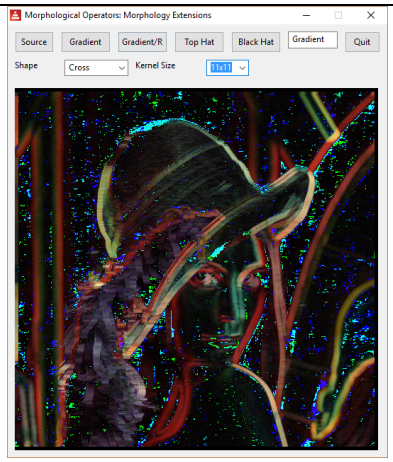
dst =src – open src dst

rcvBlackHat

Performs advanced morphological transformations

rcvTopHat: function [src [image!] dst [image!] kSize [pair!] kernel [block!]]
src: image
dst: image
kSize: kenel size as pair!
kernel: block generated by cvCreateStructuringElement or customed structuring element
defined in /libs/imgproc/rcvImgProc.red

dst = open src dst - src



rcvMMean

Means image by using structuring element

rcvMMean: function [src [image!] dst [image!] kSize [pair!] kernel [block!]]

src: image

dst: image

kSize: kernel size

kernel: block generated by cvCreateStructuringElement or customized structuring element

defined in /libs/imgproc/rcvImgProc.red

Image denoising and image smoothing

redCV can be used for image denoising. A lot of functions are included for helping image restoration. Basically, a 3x3 kernel is used to calculate the pixel neighbors' value and replace the pixel value by the result. Of course, kernel size can be changed. According to the noise included in image you can use different parametric filters. These filters can be also used for image smoothing.

rcvMeanFilter

Mean Filter for images
rcvMeanFilter: function [src [image!] dst [image!] kSize [pair!] op [integer!]]
src: image source
dst: destination image
kSize: kernel size
op: parameter for mean computing
defined in /libs/imgproc/rcvImgProc.red

Central pixel value is replaced by mean of neighbors' values according to kernel size and to op parameter. n is the size of the kernel.
op = 0 arithmetic mean: 1/n * (x1+ x2 + ….xn)
op = 1 harmonic mean: n / (1/x1 + 1/x2 + … 1/xn)
op = 2 geometric mean: power (x1 * x2 * … xn) 1/n
op= 3 Quadratic mean: sqrt (x1*x1 + x2 * x2 + … xn * xn / n)
op= 4 Cubic mean: power (x1*x1 + x2 * x2 + … xn * xn / n) (1.0 / 3.0)
op= 5 Root mean square: sqrt (1/n * (x1*x1 + x2 * x2 + … xn * xn))

rcvMedianFilter

Median Filter for images
rcvMedianFilter: function [src [image!] dst [image!] kSize [pair!]]
src: image source
dst: destination image
kSize: kernel size
defined in /libs/imgproc/rcvImgProc.red

Central pixel value is replaced by the median value of neighbors.

rcvMinFilter

Minimum Filter for images
rcvMinFilter: function [src [image!] dst [image!] kSize [pair!]]
src: image source
dst: destination image
kSize: kernel size
defined in /libs/imgproc/rcvImgProc.red

Central pixel value is replaced by the minimum value of neighbors.

rcvMaxFilter

Maximum Filter for images

rcvMaxFilter: function [src [image!] dst [image!] kSize [pair!]]
src: image source
dst: destination image
kSize: kernel size
defined in /libs/imgproc/rcvImgProc.red

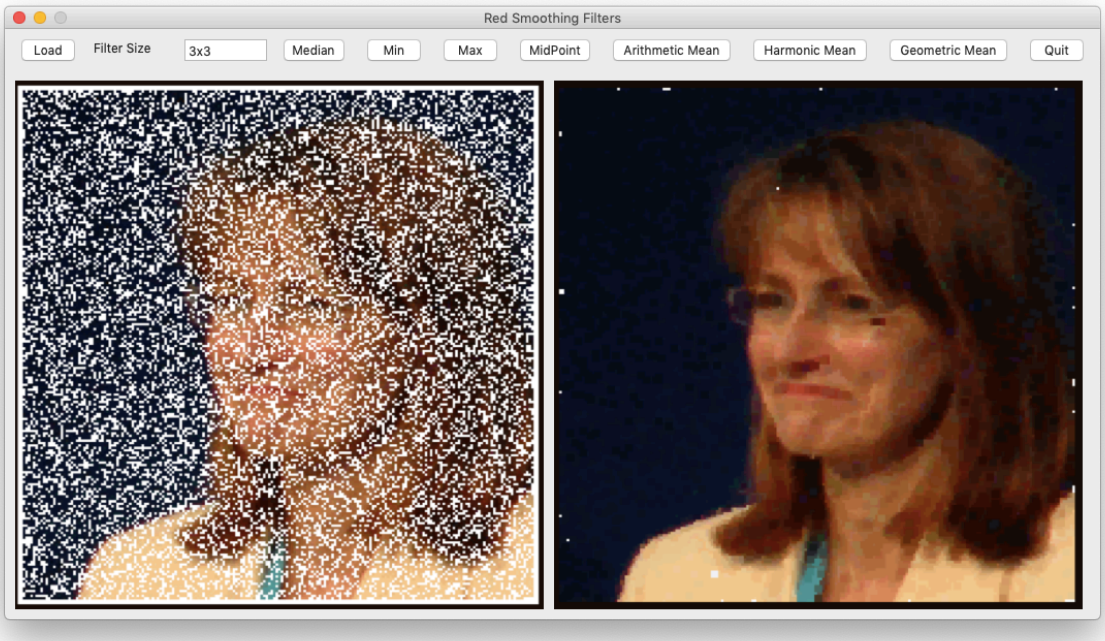
Central pixel value is replaced by the maximum value of neighbors

rcvMidPointFilter

Midpoint Filter for images

rcvMidPointFilter: function [src [image!] dst [image!] kSize [pair!]]
src: image source
dst: destination image
kSize: kernel size
defined in /libs/imgproc/rcvImgProc.red

Central pixel value is replaced by minimum+ maximum values of neighbors divided by 2



Time series and signal processing

All functions are defined in */libs/timeseries/rcvTS.red*. These functions can be associated to Freeman code chain and contour signature identification. 1-D series are vector! Datatype for faster computation.

1-D Series Filtering

rcvTSCopySignal

Makes a copy of original signal

rcvTSCopySignal: function [signal [vector!]]
signal: 1-D matrix of integer or float
returns a copy of the original signal

rcvTSStatSignal

Return mean, sd, minimal and maximal values of the signal serie

rcvTSStatSignal: function [signal [vector!]]
signal: 1-D matrix of integer or float

rcvTSSDetrendSignal

Removes linear trend in the signal by removing mean value of the series

rcvTSSDetrendSignal: function [signal [vector!] filter [vector!]]
signal: 1-D matrix of integer or float
Detrended values are stored in filter matrix

rcvTSSNormalizeSignal

Normalize data by replacing each value by a normalized value

rcvTSSNormalizeSignal: function [signal [vector!] filter [vector!]]
signal: 1-D matrix of integer or float
Normalized values are stored in filter matrix. Each value = (value -mean)/sd

rcvTSMMFilter

Calculates a mobile mean according to the number of points given by filterSize

rcvTSMMFilter: function [signal [vector!] filter [vector!] filterSize [integer!]]
signal: 1-D matrix of integer or float
Filtered values are stored in filter matrix
filterSize is the number of points used for calculating mobile mean

rcvSGFilter

Calculates second order polynomial Savitzky-Golay filter

rcvSGFilter: function [signal [vector!] filter [vector!] opSG [integer!]]
signal: 1-D matrix of integer or float
filter: predefined coefficients for faster computation
opSG: flag allowing cubic , quartic or quintic polynomials filtering

rcvSGCubicFilter

Calculates second order polynomial Savitzky-Golay filter

rcvSGCubicFilter: function [signal [vector!] filter [vector!] opSG [integer!]]
signal: 1-D matrix of integer or float
filter: predefined coefficients for faster computation
opSG: flag allowing cubic kernels for filtering;

rcvSGQuarticFilter

Calculates second order polynomial Savitzky-Golay filter

rcvSGQuarticFilter: function [signal [vector!] filter [vector!] opSG [integer!]]
signal: 1-D matrix of integer or float
filter: predefined coefficients for faster computation
opSG: flag allowing quartic kernels for filtering

rcvSGDerivative1

Calculates derivative Savitzky-Golay filter

rcvSGDerivative1: function [signal [vector!] filter [vector!] opSG [integer!]]
signal: 1-D matrix of integer or float
filter: predefined coefficients for faster computation
opSG: flag allowing quadratic kernels for filtering

Dynamic Time Warping

Quoting wikipedia:
"In time series analysis, dynamic time warping (DTW) is an algorithm for measuring similarity between two temporal sequences which may vary in time or speed. For instance, similarities in walking patterns could be detected using DTW, even if one person was walking faster than the other, or if there were accelerations and decelerations during the course of an observation."

Applied to computer vision DTW is really useful if we want to compare shapes and decide if shapes are similar or not.
In redCV we use a basic DTW algorithm which is documented here: https://nipunbatra.github.io/blog/2014/dtw.html. Thanks to Nipun Batra for writing a clear python code.
The objective is to find a mapping between all points of x and y series. In the first step, we will find out the distance between all pair of points in the two signals. Then, in order to create a mapping between the two signals, we need to create a path. The path should start at (0,0) and want to reach (M,N) where (M, N) are the lengths of the two signals. To do this, we thus build a matrix similar to the distance matrix. This matrix would contain the minimum distances to reach a specific point when starting from (0,0). DTW value corresponds to (M,N) sum value.

rcvDTWMin

Returns inimal value between 3 values
rcvDTWMin: function [x [number!] y [number!] z [number!]]

rcvDTWDistances

Making a 2d matrix to compute distances between all pairs of x and y series
rcvDTWDistances: function [x [block!] y [block!] dMatrix [vector!]]
x: a block! of number!
y: a block! of number!
dMatrix: matrix to store the distances between x and y series

rcvDTWCosts

Making a 2d matrix to compute minimal distance cost
rcvDTWCosts: function [x [block!] y [block!] dMat [vector!] cMat [vector!]
x: a block! of number!
y: a block! of number!
dMat: a matrix of distances
cMat: a matrix to store the minimal distances

rcvDTWGetDTW

Returns DTW value
rcvDTWGetDTW: function [cMat [vector!] return: [number!]] [
cMat: Minimal distances matrix
returns DTW value

rcvDTWGetPath

Finds the path minimizing the distance

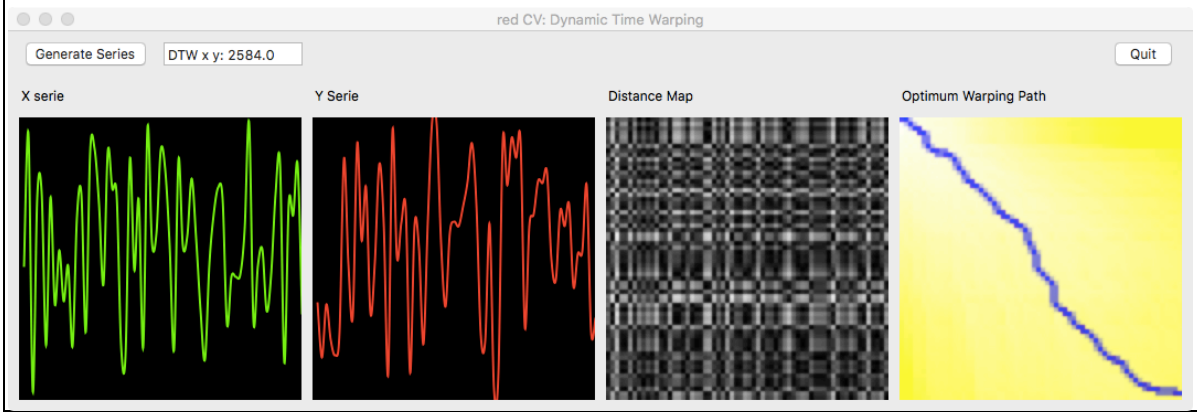
rcvDTWGetPath: function [x [block!] y [block!] cMat [vector!] xPath: [block!]
x: a block! of number!
y: a block! of number!
xPath: a blocks to store optimal distance path

rcvDTWCompute

Short-cut to get DTW value if you don't need distance and cost matrices

rcvDTWCompute: function [x [block!] y [block!] return: [number!]
x: a block! of number!
y: a block! of number!
Returns DTW value

Code sample
dMatrix: rcvDTWDistances x y
cMatrix: rcvDTWRun x y dMatrix
dtw: rcvDTWGetDTW cMatrix
xPath: rcvDTWGetPath x y cMatrix



GUI Functions

Some functions for RedCV quick test. Functions are pure Red code. Routines are not required.
These functions can also be used for displaying temporary images.
All functions are defined *in /libs/highgui/rcvHighGui.red.*

rcvNamedWindow

Creates and returns a window
rcvNamedWindow: function [title [string!]]
title: Windows title as a string
window is returned a face datatype!

rcvDestroyWindow

Destroys a created window
rcvDestroyWindow: function [window [face!]]
window: Points to a window created by rcvNamedWindow

rcvDestroyAllWindows

Destroys all windows
rcvDestroyAllWindows: function []

rcvResizeWindow

Sets window size
rcvResizeWindow: function [window [face!] wSize [pair!]]

rcvMoveWindow

Sets window position
rcvMoveWindow: function [window [face!] position [pair!]]

rcvShowImage

Shows image in window
rcvShowImage: function [window [face!] image [image!]]

```
#include %../../libs/redcv.red
img1: rcvLoadImage %../../images/lena.jpg
s1: rcvNamedWindow "Source"
rcvShowImage s1 img1 wait 2
rcvMoveWindow s1 20x60 wait 2
rcvResizeWindow s1 512x512 wait 2
rcvDestroyWindow s1
do-events
```

Random generator

redCV includes a lot of random generators with continuous and discrete laws. These functions are for machine learning and neural networks. All functions are defined in */libs/math/rcvRandom.red.*

Continuous Laws

randFloat

Returns a decimal value beween 0 and 1. Base 16 bit
randFloat: function[]

randUnif

Uniform law
randUnif: function [i [float!] j [float!]]
i: float value

randExp

Exponential law
randExp: function []

randExpm

Exponential law with a l degree
randExpm: function [l [float!]]
l: float value (e.g. 1.0)

randNorm

Normal law
randNorm : function [A [float!]]
A: float value (e.g. 1.0)

randLognorm

Lognormal law
randLognorm: function [a [float!] b [float!] z [float!]]
a: float value
b: float value
z: float value

randGamma

Gamma law
randGamma: func [k [integer!] l [float!] i]
k: integer value
l: float value

randDisc

Geometric law in a disc
randDisc: function []

randRect

Geometric law in a rectangle
randRect: function [a [float!] b [float!] c [float!] d [float!]]
a: float value
b: float value
c: float value
d: float value

randChi2

Chi square law
randChi2: function [v [integer!]]
v: integer value (e.g. 2)

randErlang

Erlang law
randErlang: function [n [integer!]]
n: integer value (e.g. 2)

randStudent

Student law
randStudent: function [n [integer!] z [float!]]
n: integer value (e.g. 3)
z: float value (e.g. 1.0)

randFischer

Fisher law (e.g 1 1)
randFischer: function [n [integer!] m [integer!]]
n: integer value (e.g. 1)
m: integer value (e.g. 1)

randLaplace

Laplace law
randLaplace: function [a [float!] /local u1 u2]
a: float value (e.g. 1.0)

randBeta

Beta law
randBeta: function [a [integer!] b [integer!]]
a: integer value (e.g. 1)
b: integer value (e.g. 1)

randWeibull

Weibull law
randWeibull: function [a [float!] l [float!]]
a: float value (e.g. 1.0)
l: float value (e.g. 1.0)

randRayleigh

Rayleigh law
randRayleigh: function []

Discrete Laws

randBernouilli

Bernouilli law
randBernouilli: function [p [float!]]
p: float value (e.g. 0.5)

randBinomial

Binomial law
randBinomial: function [n [integer!] p [float!]]
n: integer value (e.g. 1)
p: float value (e.g. 0.5)

randBinomialneg

Binomial negative law (e.g. 1 0.5)
randBinomialneg: function [n [integer!] p [float!]]
n: integer value (e.g. 1)
p: float value (e.g. 0.5)

randGeo

Geometric law
randGeo: func [p [float!]]

p: float value (e.g. 0.25)

randPoisson

Poisson law
randPoisson: function [l [float!]]
l: float value (e.g. 1.5)

Misc routines and functions

Defined in /libs/tools/rcvTools.red

Routines are Red/System compatible.

Min and Max routines

```
minInt: routine [
        a               [integer!]
        b               [integer!]
        return:         [integer!]
]
minFloat: routine [
        a               [float!]
        b               [float!]
        return:         [float!]
]

maxInt: routine [
        a               [integer!]
        b               [integer!]
        return:         [integer!]
]

maxFloat: routine [
        a               [float!]
        b               [float!]
        return:         [float!]
]
```

Round routine

```
rcvRound: routine [
        f       [float!]
        return: [float!]
]
```

```
[ either (f - floor f) > 0.5 [ceil f] [floor f] ]
```

Hypot routine

Hypot is a mathematical function defined to calculate the length of the hypotenuse of a right-angle triangle. It was designed to avoid errors arising due to limited-precision calculations performed on computers.

```
rcvHypot: routine [
        a               [float!]
        b               [float!]
        return:         [float!]
]
```

rcvNSquareRoot

Returns the nth root of Num

rcvNSquareRoot: function [num [number!] nroot [number!] return: [float!]]

general square root function used by Minkowski Distance

defined in /libs/math/rcvDistance.red

rcvElapsed

Calculates elapsed time in ms. Requires time/now/precise

rcvElapsed: function [t1 [time!] t2 [time!] return: [float!]]

Functions index

TOTAL: 339 FUNCTIONS

rcvCore: 70 functions
defined in /libs/core/rcvCore.red

rcvAddLIP
rcvSubLIP
rcvAddS
rcvSubS
rcvMulS
rcvDivS
rcvModS
rcvRemS
rcvLSH
rcvRSH
rcvPow
rcvSQR
rcvAddT
rcvSubT
rcvMulT
rcvDivT
rcvModT
rcvRemT
rcvAND
rcvOR
rcvXOR
rcvNAND
rcvNOR
rcvNXOR
rcvNot
rcvANDS
rcvORS
rcvXORS
rcvMeanImages
rcvSplit
rcvMerge
rcvInRange

rcvImageProc: 80 functions
defined in /libs/imgproc/rcvImageProc.red

rcvRGB2XYZ
rcvBGR2XYZ
rcvXYZ2RGB
rcvRGB2HSV
rcvBGR2HSV
rcvRGB2YCrCb
rcvBGR2YCrCb
rcvRGB2HLS
rcvBGR2HLS
rcvRGB2Lab
rcvBGR2Lab

rcvRGB2Luv
rcvBGR2Luv
rcvIRgBy
rcvFlip
rcvSetIntensity
rcvBlend
rcvBlendWin
rcvConvolve
rcvFastConvolve
rcvFilter2D
rcvFastFilter2D
rcvPointDetector
rcvSharpen
rcvBinomialFilter
rcvLowPass
rcvBinomialLowPass
rcvHighPass
rcvHighPass2
rcvBinomialHighPass
rcvMakeGaussian
rcvGaussianFilter
rcvDoGFilter
rcvMedianFilter
rcvMinFilter
rcvMaxFilter
rcvMidPointFilter
rcvMeanFilter
rcvKirsch
rcvSobel
rcvPrewitt
rcvRoberts
rcvRobinson
rcvDiscreteLaplacian
rcvLaplacianOfRobinson
rcvGradientMasks
rcvLineDetection
rcvGradNeumann
rcvDivNeumann
rcvNLFilter
rcvDerivative2
rcvLaplacian
rcvIntegral
rcvProcessIntegralImage
rcvResizeImage
rcvScaleImage
rcvRotateImage
rcvTranslateImage

rcvSkewImage
rcvClipImage
rcvCreateStructuringElement
rcvErode
rcvDilate
rcvOpen
rcvClose
rcvMGradient
rcvTopHat
rcvBlackHat
rcvMMean
rcvMakeHoughAccumulator
rcvGetAccumulatorSize
rcvHoughTransform
rcvHough2Image
rcvGetHoughLines
rcvImageNoise
rcvEdgesGradient
rcvEdgesDirection
rcvEdgesSuppress
rcvDoubleThresh
rcvHysteresis

rcvDistance: 28 functions
defined in /libs/math/rcvDistance.red
rcvGetEuclidianDistance
rcvGetManhattanDistance
rcvGetChessboardDistance
rcvGetChebyshevDistance
rcvGetMinkowskiDistance
rcvGetCamberraDistance
rcvGetSorensenDistance
rcvDistance2Color
rcvGetAngle
rcvGetAngleRadian
rcvRhoNormalization
rcvVoronoiDiagram
rcvDistanceDiagram
rcvMakeGradient
rcvMakeBinaryGradient
rcvFlowMat
rcvnormalizeFlow
rcvGradient&Flow
rcvChamferDistance
rcvChamferCreateOutput
rcvChamferInitMap
rcvChamferCompute

rcvChamferNormalize
rcvKMInitData
rcvKMGenCentroid
rcvKMInit
rcvKMCompute

rcvStats: 27 functions
defined in /libs/math/rcvStats.red

rcvCountNonZero
rcvSum
rcvMean
rcvSTD
rcvMedian
rcvMinValue
rcvMaxValue
rcvMinLoc
rcvMaxLoc
rcvHistogram
rcvSmoothHistogram
rcvRGBHistogram
rcvMeanShift
rcvHistogramEqualization
rcvMakeTranscodageTable
rcvContrastAffine
rcvRangeImage
rcvSortImage
rcvXSortImage
rcvYSortImage
rcvCross
rcvPointDistance
rcvFindExtrema
rcvSeparateSets
rcvHullSet
rcvQuickHull
rcvContourArea

rcvMatrix: 69 Functions
defined in /libs/matrix/rcvMatrix.red

rcvCreateMat
rcvReleaseMat
rcvCloneMat
rcvCopyMat
rcvMakeBinaryMat
rcvMakeRangeMat
rcvMakeIndenticalMat

rcvCreateRangeMat
rcvSortMat
rcvFlipMat
rcvLengthMat
rcvSumMat
rcvMeanMat
rcvProdMat
rcvMaxMat
rcvMinMat
rcvRandomMat
rcvColorMat
rcvGetInt2D
rcvGetReal2D
rcvSetInt2D
rcvSetReal2D
rcvGetPairs
rcvGetPoints
rcvMatleftPixel
rcvMatRightPixel
rcvMatUpPixel
rcvMatDownPixel
rcvMatGetBorder
rcvMatGetChainCode
rcvGetContours
rcvGetMatCentroid
rcvGetMatSpatialMoment
rcvGetMatCentralMoment
rcvGetNormalizedCentralMoment
rcvGetMatHuMoments
rcvImage2Mat
rcvMat2Image
rcvMat2Binary
rcvSplit2Mat
rcvMerge2Image
rcvConvolveMat
rcvConvolveNormalizedMat
rcvConvertMatScale
rcvMatInt2Float
rcvMatFloat2Int
rcvMatFastSobel
rcvMatrixMedianFilter
rcvAddMat
rcvSubMat
rcvMulMat
rcvDivMat
rcvRemMat
rcvMeanMats

rcvAddSMat
rcvSubSMat
rcvMulSMat
rcvDivSMat
rcvRemSMat
rcvANDMat
rcvORMat
rcvXORMat
rcvANDSMat
rcvORSMat
rcvXORSMat
rcvErodeMat
rcvDilateMat
rcvBlendMat
rcvInRangeMat

rcvTiff: 4
defined in /libs/tiff/rcvTiff.red

rcvLoadTiffImage
rcvReadTiffImageData
rcvTiff2RedImage
rcvSaveTiffImage

rcvTS: 16
defined in /libs/timeseries/rcvTS.red

rcvTSCopySignal
rcvTSStatSignal
rcvTSSDetrendSignal
rcvTSSNormalizeSignal
rcvTSMMFilter
rcvSGFilter
rcvSGCubicFilter
rcvSGQuarticFilter
rcvSGDerivative1
rcvDTWMin
rcvDTWDistances
rcvDTWCosts
rcvDTWGetPath
rcvDTWGetPath1 [OLD]
rcvDTWGetDTW
rcvDTWCompute

rcvZLib: 2 functions
defined in /libs/ZLib/rcvZLib.red

rcvCompressRGB
rcvDecompressRGB

rcvHighGui: 7 functions
defined in /libs/highgui/rcvHighGui.red

rcvNamedWindow
rcvDestroyWindow
rcvDestroyAllWindows
rcvResizeWindow
rcvMoveWindow
rcvShowImage
rcvDrawPlot

rcvtools: 8 functions and routines
defined in /libs/tools/rcvtools.red

minInt (routine)
minFloat (routine)
maxInt (routine)
MaxFloat (routine)
rcvRound (routine)
rcvHypot (routine)
rcvElapsed
rcvNSquareRoot

rcvCapture: 8 Functions [EXPERIMENTAL]
defined in /libs/video/rcvCapture.red

rcvCreateCam
rcvSetCamSize
rcvsetCamWidth
rcvsetCamHeight
rcvgetCamSize
rcvgetCameraFPS
rcvGetCamImage
rcvGetMovieFile

rcvRandom: 23 Functions (including 1 routine)
defined in /libs/math/rcvRandom.red

randf (routine)
randFloat
randUnif
randExp
randExpm

randNorm
randLognorm
randGamma
randDisc
randRect
randChi2
randErlang
randStudent
randFischer
randLaplace
randBeta
randWeibull
randRayleigh
randBernouilli
randBinomial
randBinomialneg
randGeo
randPoisson