redCV
Open Source Computer Vision Library

François Jouen
Human and Artificial Cognitions Laboratory
Ecole Pratique des Hautes Etudes
Université Paris Sciences et Lettres

What is redCV

redCV means Red Language Open Source Computer Vision Library. It is a collection of Red language 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 routines and functions. Except for ZLib, 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 contributors:
François Jouen (aka ldci) for the library creation and development.
Thanks to Nénad Rakocevic and Qingtian Xie for their constant help.
Thanks to Red Team for developing Red.
Thanks to Didier Cadieu for samples optimization.
Thanks to Boleslav Březovský for distance mapping.
Thanks to Toomas Vooglaid and Mel Cepstrum for 1-D FFT code.
Thanks to Bruno Anselme for ZLib binding.
Thanks to Fyodor Shchukin for illustration.

Where to get redCV

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

Table of Contents

Using redCV Library

In order to get pretty good image processing, redCV uses a lot of **Red/System routines** for faster image rendering. All redCV routines and functions can be directly called from any Red program. But, you need to compile your code. All red routines prefixed with underscore (e.g. *_rcvDotsDistance*) are *for internal use*. Both redCV routines and functions are documented. Code sample included with redCV is also documented in RedCV_Samples.pdf.

All includes for redCV library are declared in a single file (/libs/redcv.red). You just need including *redcv.red* file in your Red programs if you want use all the library (#include %libs/redcv.red, for all redCV functions). But now, **redCV is modular**. This means, that you can use only **required libraries** for your code and not all redCV library. This modular organization reduces compilation duration, reduces the size of the executable applications and, helps in maintaining redCV. As detailed below, some libraries are mandatory and other are optional according to specific applications. *All code samples included in redCV use modular library calling*.

mandatory libs

#include %core/rcvCore.red	Basic image creating and processing functions
#include %matrix/rcvMatrix.red	Matrices functions
#include %tools/rcvTools.red	Some Red tools mainly used by rcvImgProc.red
#include %imgproc/rcvImgProc.red	Basic image and matrix processing algorithms

optional libs

#include %imgproc/rcvFreeman.red	Contour detection
#include %imgproc/rcvIntegral.red	Integral image
#include %imgproc/rcvMorphology.red	Morphological operators
#include %imgproc/rcvHough.red	Hough transforms
#include %math/rcvRandom.red	Random laws for generating random images
#include %math/rcvStats.red	Statistical functions for images and matrices
#include %math/rcvMoments.red	Spatial and central moments
#include %math/rcvHistogram.red	Histograms
#include %math/rcvDistance.red	Distance algorithms for detection in images
#include %math/rcvQuickHull.red	Convex area
#include %math/rcvChamfer.red	Chamfer distance computation
#include %math/rcvComplex.red	Some operators for complex numbers
#include %math/rcvCluster.red	Data clustering (kMeans)
#include %objdetect/rcvHaarCascade.red	Object detection
#include %objdetect/rcvHaarRectangles.red	Rectangles clustering
#include %pbm/rcvPbm.red	Portable BitMap files support
#include %zLib/rcvZLib.red	ZLib compression
#include %tiff/rcvTiff.red	Tiff image reading and writing
#include %timeseries/rcvTS.red	Time Series algorithms
#include %timeseries/rcvSGF.red	Savitzky-Golay filter
#include %timeseries/rcvDTW.red	Dynamic Time Warping algorithms

#include %timeseries/rcvFFT.red	FFT algorithms
#include %highgui/rcvHighGui.red	Fast Highgui functions

Some lectures

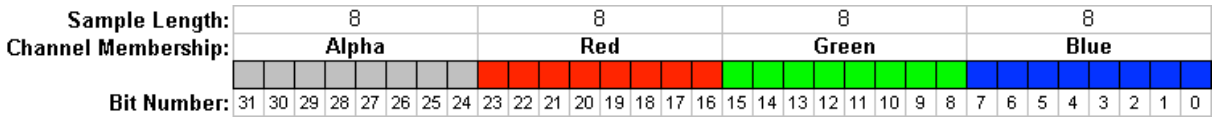
Image Processing in C, by Dwayne Phillips. The first edition of Image Processing in C (Copyright 1994, ISBN 0-13-104548-2) was published by R & D Publications
1601 West 23rd Street, Suite 200
Lawrence, Kansas 66046-0127
Algorithms for Image Processing and Computer Vision (2011) by J.R. Parker, published by Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256

Basic Structures

Image

redCV directly uses Red image! datatype. Image! values contain a series of RGBA values, which represent pixels in a 2D image. 2D is mapped to 1D space by taking first row of pixels from left to right, then second row, and so on (row-major-order). 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 RGBA 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 RGBA components of a pixel.

Internally, Red uses the ARGB (word-order) for encoding the intensity of each channel sample[1]. Red uses an integer to store ARGB values. Since the memory size of an integer is 32 bits, it's really easy to store 4 bytes (8-bit) with an integer for the 4 channels. This single 32-bit integer has the alpha channel in the highest 8 bits, followed by the red channel, the green channel and finally the blue channel in the lowest 8 bits. Consequently, an int-ptr! will be used to access pixel value.



Source:wikipedia

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

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

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

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

See https://github.com/red/docs/blob/master/en/datatypes/image.adoc for details about image datatype.

[1] tuple datatype used for represent pixel color is confusing, since transparency is the 4th value of the tuple. See https://github.com/red/red/issues/2684 and https://github.com/red/red/issues/2812.

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. Matrices are intensively used to simulate 1-channel image for faster rendering.

Vector is detailed here: https://github.com/red/docs/blob/master/en/datatypes/vector.adoc

Array

This a block! type for quick access. Basically, array is a block of vectors. Array is useful for addressing pixels by lines and columns and is very efficient for Fourier transforms for example.

```
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 in type and size.**

Images and matrices basic operators

rcvCreateImage

Creates and returns empty (black) image

```
rcvCreateImage: function [
        size      [pair!] ; -- 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!]; -- source image
]
```
Defined in /libs/core/rcvCore.red

rcvGetImageFileSize

Returns image file size as a pair!

```
rcvGetImageFileSize: function [
        fileName        [file!]   ; -- Red file name
        return:         [pair!]  ;-- pair
]
```
Defined in /libs/core/rcvCore.red

rcvCreateMat

Creates and returns 2-D matrix

```
rcvCreateMat: function [
        type    [word!]    ; -- char! | integer! | float!
        bitSize [integer!] ; -- 8 for char!,  8 | 16 | 32 for integer!, 32 | 64 for float! matrix
        mSize   [pair!]      ; -- 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!]       ; -- matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMakeRangeMat

Creates and returns an ordered matrix

```
rcvMakeRangeMat: function [
        a       [number!]       ; -- starting value
        b       [number!]       ; -- ending value
        step    [number!]       ; -- Step is used for range
]
```
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!]         ; -- char! | integer!| float!
        bitSize [integer!]      ; -- 8 |16 | 32| 64
        vSize   [integer!]      ; -- matrix size
        value   [number!]       ; -- value
]
```
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 values]

```
rcvMakeBinaryMat: function [
        src     [vector!]       ; -- source matrix
        dst:    [vector!]       ; -- destination matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

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

rcvReleaseImage

Releases image data

```
rcvReleaseImage: routine [
        src     [image!]        ; Red image
]
```

Defined in /libs/core/rcvCore.red

rcvReleaseAllImages

Delete all images

```
rcvReleaseAllImages: function [
        list    [block!]        ; -- list of Red images
]
```

Defined in /libs/core/rcvCore.red

loaded or created images must be stored into a block! before releasing

rcvReleaseMat

Releases Matrix

```
rcvReleaseMat: function [
        mat     [vector!]       ; -- 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!]        ; -- name of the file to load as a Red file datatype
        /grayscale              ; -- refinement: loads image as grayscale image
]
```

Defined in /libs/core/rcvCore.red

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

rcvLoadImageAsBinary

Loads image from file and returns image as binary string

```
rcvLoadImageAsBinary: function [
        fileName        [file!]   ; -- 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!]        ; -- image to save
        fileName        [file!]         ; -- name of the file to save as a Red file datatype
]
```
Defined in /libs/core/rcvCore.red

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

rcvCloneImage

Returns a copy of source image

```
rcvCloneImage: function [
        src     [image!]        ; -- 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!]       ; -- matrice to be cloned
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvCopyImage

Copy source image to destination image

Source and destination image must have the same size!

```
rcvCopyImage: routine [
        src     [image!]        ; -- image to be copied
        dst     [image!]        ; -- destination image
]
```

Defined in /libs/core/rcvCore.red

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

rcvCopyMat

Copy source matrix to destination matrix

```
rcvCopyMat: function [
        src     [vector!]       ; -- matrice to be copied
        dst     [vector!]       ; -- destimation matrix
]
```
Defined in /libs/matrix/rcvMatrix.red.

This function calls 2 routines:
rcvCopyMatI for integer matrices copy
rcvCopyMatF for float matrices copy

rcvZeroImage

Sets all image pixels to 0

```
rcvZeroImage: function [
        src     [image!]        ; -- image to clear
]
```
Defined in /libs/core/rcvCore.red

rcvRandImage

A fast routine for random images

```
rcvRandImage: routine [
        src     [image!]        ; Red image
]
```
Defined in /libs/core/rcvCore.red

rcvRandomImage

Creates and returns a random uniform color or pixel random image

```
rcvRandomImage: function [
        size    [pair!]         ; -- size of image as pair!
        value   [tuple!]        ; -- random value as tuple!
        /uniform /alea/fast
]
refinement
```

/uniform: random uniform color
/alea: random pixels
/fast: uses rcvRandImage routine
Defined in /libs/core/rcvCore.red

rcvRandomMat

Randomizes matrix

```
rcvRandomMat: function [
        mat     [vector!]       ; -- destinatiobn matrix
        value   [integer!]      ; -- random value as integer!
]
```

Only for integer matrices
Defined in /libs/matrix/rcvMatrix.red

```
msize: 512x512
mat1: rcvCreateMat 'integer! 8 msize
mat2: rcvCreateMat 'integer! 16 msize
mat3: rcvCreateMat 'integer! 32 msize
rcvRandomMat mat1 FFh
rcvRandomMat mat2 FFFFh
rcvRandomMat mat3 FFFFFFh
```

rcvGenerateNoise

Generates Gaussian noise

```
rcvGenerateNoise: routine [
        src     [image!]        ; -- Red image
        noise   [float!]        ; -- value between 0.0 and 1.0
        t       [tuple!]        ; -- 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

```
rcvColorImage: function [
        src     [image!]        ; -- image to colorize
        acolor  [tuple!]        ; -- color as a tuple
]
```

Defined in /libs/core/rcvCore.red

rcvColorMat

Set matrix color

```
rcvColorMat: function [
        mat     [vector!]       ; -- destination matrix
        value   [integer!]      ; -- color value as integer
]
```

Only for integer matrices

Defined in /libs/matrix/rcvMatrix.red

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

rcvSortMat

Returns ascending sort of matrix

```
rcvSortMat: function [
        v       [vector!]       ; -- matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvFlipMat

Returns flip matrix

```
rcvFlipMat: function [
        v       [vector!]       ; matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvCompressRGB

Compresses rgb image values as binary

```
rcvCompressRGB: routine [
        rgb     [binary!]       ; -- rgb binary values of the image (image/rgb)
        level   [integer!]      ; -- compression level for ZLib compression
]
```

level:
0: No compression
1: Best Speed
9: Best compression
-1: default compression

Defined in /libs/Zlib/rcvZLib.red

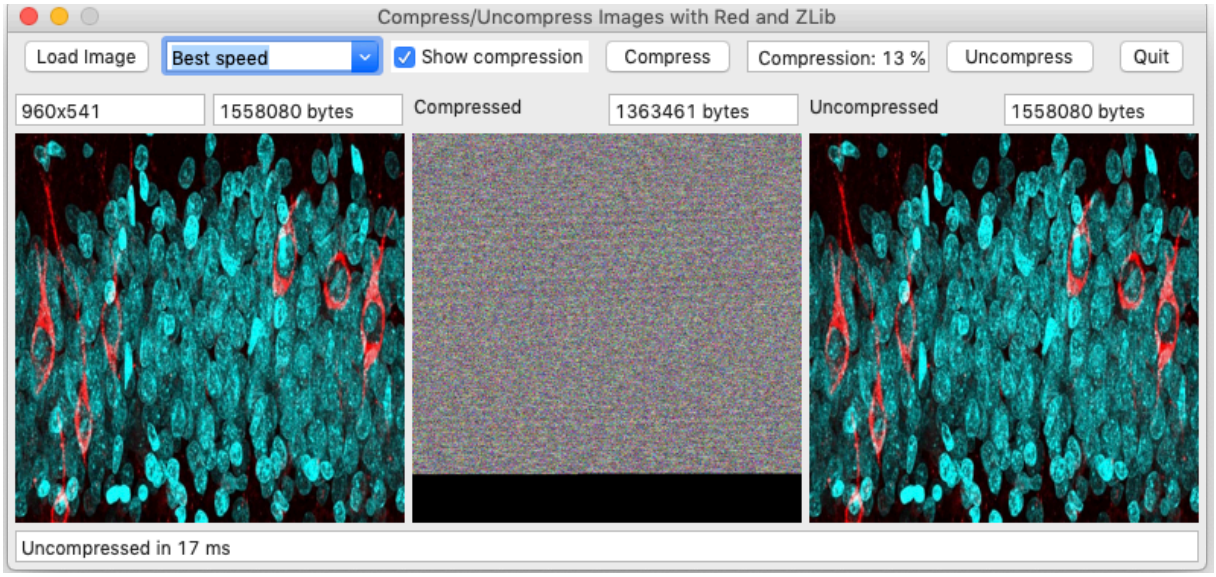
rcvDecompressRGB

Uncompresses rgb image values as binary

```
rcvDecompressRGB: routine [
        rgb             [binary!]       ; -- previously rgb compressed values
        bCount          [integer!]      ; -- size of non-compressed rgb values
]
```

Defined in /libs/Zlib/rcvZLib.red

```
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
```



TIFF images access

Red doesn't support yet reading and writing Tiff images. But, many scientific images are in Tiff format, and it was important to get a basic support for Tiff. Tiff is powerful, but rather complicated. Here are basic routines and functions for grayscale and color tiff images access. Multi images are also supported. Uncompressed bilevel, grayscale, palette-color images and RGB with samples per pixel up to 4 are supported.

All objects we need to decode and encode Tiff files are defined in /libs/tiff/rcvTiffObject.red

rcvTiff2Image

Converts Tiff image to Red image

```
rcvTiff2Image: routine [
        bin     [binary!]       ; -- tiff image as binary string
        dst     [image!]        ; -- Red image
]
```

Defined in /libs/tiff/rcvTiff.red

rcvAssertTiffFile

Tiff file or not?

```
rcvAssertTiffFile: func []
```

Defined in /libs/tiff/rcvTiff.red

rcvReadTiffHeader

Reads Tiff File header (8 bytes)

```
rcvReadTiffHeader:  func []
```

Defined in /libs/tiff/rcvTiff.red

```
Tiff Header object
TIFFHeader: make object! [
        tiffBOrder:     integer!        ; 2 bytes 0-1 byte order
        tiffVersion:    integer!        ; 2 bytes 2-3 Tiff version number (42)
        tiffFIFD:       integer!        ; 4 bytes 4-7 offset of the first Image File Directory
```

rcvmakeTiffIFDList

Makes the list of Image File Directory (IFD 12 bytes)

```
rcvmakeTiffIFDList: func []
```

Defined in /libs/tiff/rcvTiff.red

```
TImgFDEntry: make object![
	tiffTag:		integer!; byte 0-1 TIFF Field Tag 2 bytes 0-1 see TIFF Tag Definitions
	tiffDataType:	integer!; byte 2-3  Field data type  2 bytes 2-3 see TIFFDataType
	tiffDataLength:	integer!; byte 4-7  number of values of the indicated type; length in spec  4 bytes 4-7
	tiffOffset: integer!; byte 8-11 offset to field data 4 bytes 8-11 or value of the field if length < 4 byte
	redValue: string!; supplementary red field to get the "real" value
]
```

rcvGetTiffImageType

Returns the image type

```
rcvGetTiffImageType: func  [
	pageNumber [integer!]		; -- Page number. By default 1
]
```

Defined in /libs/tiff/rcvTiff.red

Page number is used for multi images tiff file.

rcvGetTiffTagValue

Reads tag value

```
rcvGetTiffTagValue: func []
```

Defined in /libs/tiff/rcvTiff.red

rcvProcessTiffTag

Processes tag value

```
rcvProcessTiffTag: func []
```

Defined in /libs/tiff/rcvTiff.red

rcvReadTiffFileDirectory

Get the image description for each subfile included in the file

```
rcvReadTiffFileDirectory: func [
	index [integer!]		; -- index is the page number (by default 1)
]
```

Defined in /libs/tiff/rcvTiff.red

Next functions are easy-to-use funcs for reading and writing Tiff files.

rcvLoadTiffImage

loads TIFF image

```
rcvLoadTiffImage: func [
        f       [file!]   ; -- 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.

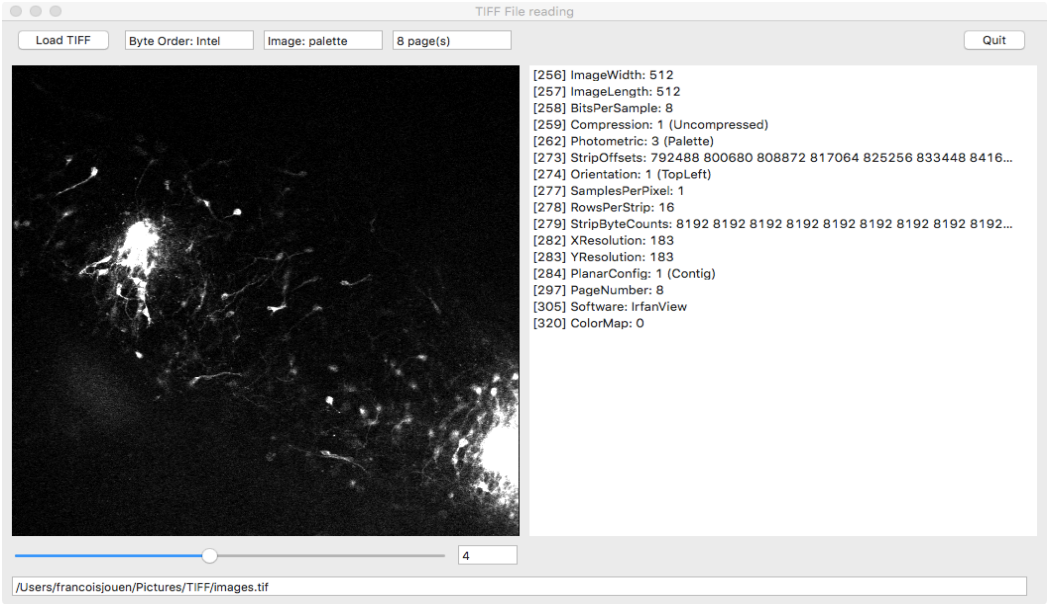
rcvReadTiffImageData

Reads multiple images included in TIFF File

```
rcvReadTiffImageData: func [
        page [integer!]         ; page number (by default 1)
]
```

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



rcvSaveTiffImage

Save red image as tiff

```
rcvSaveTiffImage: func [
RedImage        [image!]        ; -- Red image to save
        f               [file!]         ; -- name of the file
        mode            [integer!]      ; -- 1: little endian (Intel) | 2: big endian (Motorola)
]
```

Defined in /libs/tiff/rcvTiff.red

Portable BitMap images access

Portable BitMap (PMB) format is a very old image format. The PBM format was invented by Jef Poskanzer in the 1980s as a format that allowed monochrome bitmaps to be transmitted within a message as plain ASCII text. Nevertheless, this basic format is sometimes useful for fast computation on image, and the redCV supports reading, processing, and writing PBM images. PBM files are also interesting since we can play with the color max value, and thus create image with higher or lower definition.

rcvGetMagicNumberPBM

Returns PBM file magic number

```
rcvGetMagicNumberPBM: function [
        fName           [file!]         ;--PBM File
        return:         [binary!]       ;--PBM Magic Number
]
```

Defined in /libs/pbm/rcvPbm.red

PMB format supports a lot of images which are identified by file extension and a magic number.
P1: Portable Bit Map ASCII: 0–1 (white & black) ;pbm
P2: Portable Gray Map ASCII: 0–255 (gray scale) or 0–65535 (gray scale);pgm
P5: Portable Gray Map Byte: 0–255 (gray scale) or 0–65535 (gray scale);pgm
P3: Portable Pixel Map ASCII: 16777216 (0–255 for each RGB channel) ;ppm
P6: Portable Pixel Map Byte: 16777216 (0–255 for each RGB channel);ppm
for P1, P2 and P3 color maximal value is > 0 and < 65535
Magic Numbers:

```
MAGIC_P1:       #{5031}; "P1"
MAGIC_P2:       #{5032}; "P2"
MAGIC_P3:       #{5033}; "P3"
MAGIC_P5:       #{5035}; "P5"
MAGIC_P6:       #{5036}; "P6"
```

rcvReadPBMAsciiFile

Reads ASCII PBM and PGM Files

```
rcvReadPBMAsciiFile: func [
        fName           [file!]         ;--PBM file
        magic           [binary!]       ;--PBM magic number
        return:         [image!]        ;--Red image
]
```

Defined in /libs/pbm/rcvPbm.red

This function allows to read PBM and PGM files (P1, P2, and P3)

rcvWritePBMAsciiFile

Writes ASCII pbm file

```
rcvWritePBMAsciiFile: func [
        fName       [file!]       ;--Destination file name
        magic       [binary!]     ;--PBM magic number
        src         [image!]      ;--Red source image
        colorMax    [integer!]    ;-- Max color range
]
```

Defined in /libs/pbm/rcvPbm.red

This function allows to write PBM and PGM Files (P1, P2, and P3)

rcvReadPBMByteFile

Reads binary PPM Files

```
rcvReadPBMAsciiFile: func [
        fName       [file!]       ;--PBM file
        magic       [binary!]     ;--PBM magic number
return:         [image!]        ;--Red image
]
```

Defined in /libs/pbm/rcvPbm.red

This function allows to read PPM files (P5 and P6)

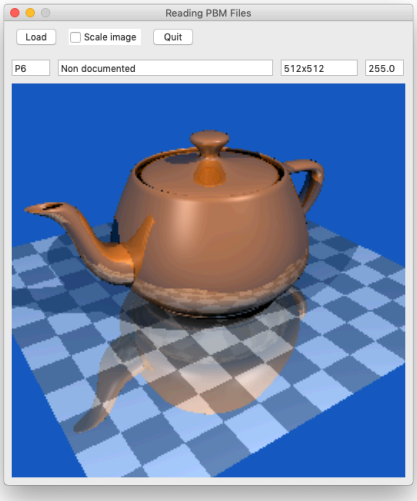
rcvWritePBMByteFile

Writes binary PPM Files

```
rcvWritePBMAsciiFile: func [
        fName       [file!]       ;--Destination file name
        magic       [binary!]     ;--PBM magic number
        src         [image!]      ;--Red source image
        colorMax    [integer!]    ;-- Max color range
]
```

Defined in /libs/pbm/rcvPbm.red

This function allows to read PPM files (P5 and P6)


Reading PBM Files
Load
Scale image
Quit
P6
Non documented
512x512
255.0

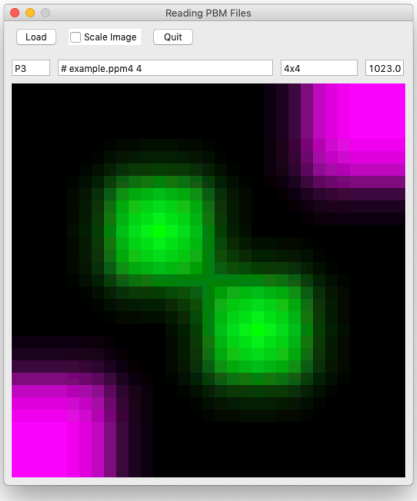

Reading PBM Files
Load
Scale Image
Quit
P3
example.ppm4 4
4x4
1023.0

Image and matrix utilities

rcvIsAPixel

Returns true if pixel value is greater than threshold

```
rcvIsAPixel: routine [
        src             [image!]        ; -- Red image
        coordinate      [pair!]         ; -- pixel xy position in image as a pair
        threshold       [integer!]      ; -- threshold value (e.g. 127)
]
```
Defined in /libs/core/rcvCore.red

rcvGetPixel

Returns pixel value at xy coordinates as tuple

```
rcvGetPixel: routine [
        src             [image!]        ; -- Red image
        coordinate      [pair!]         ; -- pixel 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!]        ; -- Red image
        coordinate      [pair!]         ; -- pixel xy position in image as a pair
]
```
Defined in /libs/core/rcvCore.red

rcvGetPixelAsInteger

Returns pixel value at xy coordinates as integer

```
rcvGetPixelAsInteger: routine [
        src             [image!]        ; -- Red image
        coordinate      [pair!]         ; -- pixel xy position in image as a pair
]
```
Defined in /libs/core/rcvCore.red

rcvGetMatType

Returns matrix type (integer or float)

```
rcvGetMatType: routine [
        mat     [vector!]       ; -- matrix
```

```
        return: [integer!]
]
return value: 1: integer matrix 2: float matrix
```
Defined in /libs/matrix/rcvMatrix.red

rcvGetMatBitSize

Returns matrice bit size

```
rcvGetMatBitSize: routine [
        mat     [vector!]       ; -- matrix
        return: [integer!]
]
return value
1: 8-bit integer
2:16-bit integer
4:32-bit integer or 32-bit float
8: 64-bit float
```
Defined in /libs/matrix/rcvMatrix.red

get and set integer matrix element value
pointer address (p)must be passed as integer! since red routine doesn't know byte-ptr!

rcvGetIntValue

```
rcvGetIntValue:  routine [
        p               [integer!]      ; -- address of mat element as integer
        unit            [integer!]      ; -- size of integer 8 16 32 [1 2 4]
        return:         [integer!]
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvSetIntValue

```
rcvSetIntValue: routine [
        p               [integer!]      ; -- address of mat element as integer
        value           [integer!]      ; -- integer value
        unit            [integer!]      ; -- size of integer 8 16 32 [1 2 4]
]
```
Defined in /libs/matrix/rcvMatrix.red

get and set float matrix element value
pointer address (p)must be passed as integer! since red routine doesn't know byte-ptr!

rcvGetFloatValue

```
rcvGetFloatValue: routine [
        p               [integer!]      ; -- address of mat element as integer
        return:         [float!]        ; -- float value
```

]
Defined in /libs/matrix/rcvMatrix.red

rcvGetFloat32Value

```
rcvGetFloat32Value: routine [
        p               [integer!]      ; -- address of mat element as integer
        return:         [float!]
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvSetFloatValue

```
rcvSetFloatValue: routine [
        p               [integer!]      ; -- address of mat element as integer address
        f               [float!]        ; -- 32 or 64-bit float
        unit            [integer!]      ; -- [4 8]: size of float 32 64
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvGetInt2D

Returns matrix value at xy coordinates as integer

```
rcvGetInt2D: routine [
        src     [vector!]       ; -- source matrix
        width   [integer!]      ; -- matrix width
        x       [integer!]      ; -- x coordinate
        y       [integer!]      ; -- y coordinate
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvGetReal2D

Returns matrix value at xy coordinates as float 64

```
rcvGetReal2D: routine [
        src     [vector!]       ; -- source matrix
        width   [integer!]      ; -- matrix width
        x       [integer!]      ; -- x coordinate
        y       [integer!]      ; -- y coordinate
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvGetReal322D

Returns matrix value at xy coordinates as float 32

```
rcvGetReal2D: routine [
        src     [vector!]       ; -- source matrix
        width   [integer!]      ; -- matrix width
        x       [integer!]      ; -- x coordinate
        y       [integer!]      ; -- y coordinate
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvSetPixel

Sets pixel value at xy coordinates

```
rcvSetPixel: routine [
        src             [image!]        ; -- Red image
        coordinate      [pair!]         ; -- pixel coordinate
        val             [tuple!]        ; -- color
]
```
Defined in /libs/core/rcvCore.red

rcvPokePixel

Set pixel value at xy coordinates

```
rcvPokePixel: function [
        src             [image!]        ; -- Red image
        coordinate      [pair!]         ; -- pixel coordinate
        val             [tuple!]        ; -- color
]
```
Defined in /libs/core/rcvCore.red

rcvSetInt2D

Sets value in integer matrix

```
rcvSetInt2D: routine [
        dst             [vector!]       ; -- destination matrix
        mSize           [pair!]         ; -- matrix size as pair!
        coordinate      [pair!]         ; -- pixel coordinate
        val             [integer!]      ; -- integer value
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvSetReal2D

Sets value in float matrix

```
rcvSetInt2D: routine [
        dst             [vector!]       ; -- destination matrix
        mSize           [pair!]         ; -- matrix size as pair!
        coordinate      [pair!]         ; -- pixel coordinate
```

```
        val             [float!]        ; -- float value
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvGetPoints

Gets coordinates from a binary matrix as pair values

```
rcvGetPoints: routine [
        binMatrix       [vector!]       ; -- matrix
        mSize           [pair!]         ; -- matrix size
        points          [vector!]       ; -- to store the result
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvGetPairs

Gets coordinates from a binary mat as pair value

```
rcvGetPairs: routine [
        binMatrix       [vector!]       ; -- matrix
        mSize           [pair!]         ; -- matrix size
        points          [block!]        ; -- to store the result
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvGetMatCentroid

Returns the centroid of the matrix

```
rcvGetMatCentroid: routine [
        mat             [vector!]       ; -- matrix
        mSize           [pair!]         ; -- matrix size
        return:         [pair!]
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMatleftPixel

Gets coordinates of first left non-zero pixel as pair

```
rcvMatleftPixel: routine [
        mat             [vector!]       ; -- matrix
        matSize         [pair!]         ; -- matrix size
        value           [integer!]      ; -- pixel value (e.g. 1 or 255)
]
```
Defined in /libs/imgproc/rcvFreeman.red

rcvMatRightPixel

Gets coordinates of first right non-zero pixel as pair

```
rcvMatRightPixel: routine [
        mat             [vector!]       ; -- matrix
        matSize         [pair!]         ; -- matrix size
        value           [integer!]      ; -- pixel value (e.g. 1 or 255)
]
```

mat: Integer matrix
matSize: matrix size as pair
value: pixel value (e.g. 1 or 255)
Defined in /libs/imgproc/rcvFreeman.red

rcvMatUpPixel

Gets coordinates of first top non-zero pixel as pair

```
rcvMatRightPixel: routine [
        mat             [vector!]       ; -- matrix
        matSize         [pair!]         ; -- matrix size
        value           [integer!]      ; -- pixel value (e.g. 1 or 255)
]
```

Defined in / libs/imgproc/rcvFreeman.red

rcvMatDownPixel

Gets coordinates of first bottom non-zero pixel as pair

```
rcvMatRightPixel: routine [
        mat             [vector!]       ; -- matrix
        matSize         [pair!]         ; -- matrix size
        value           [integer!]      ; -- pixel value (e.g. 1 or 255)
]
```

Defined in libs/imgproc/rcvFreeman.red

rcvSetAlpha

Sets image transparency

```
rcvSetAlpha: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        alpha   [integer!]      ; -- transparency value [0..255]
]
```

Defined in /libs/core/rcvCore.red

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

rcvBlend

Computes the alpha blending of two images

```
rcvBlend: routine [
        src1    [image!]        ; -- first image
        src2    [image!]        ; -- second image
        dst     [image!]        ; -- destination image
        alpha   [float!]        ; -- ratio of first image mixed with the second [0.0-1.0]
]
```

Defined in /libs/core/rcvCore.red

rcvBlendWin

Computes the alpha blending of two images. For Windows users

```
rcvBlendWin: routine [
        src1    [image!]        ; -- first image
        src2    [image!]        ; -- second image
        dst     [image!]        ; -- destination image
        alpha   [float!]        ; -- ratio of first image mixed with the second [0.0-1.0]
]
```

Defined in /libs/core/rcvCore.red

rcvBlendMat

Computes the alpha blending of two matrices

```
rcvBlendMat: routine [
        mat1    [vector!]       ; -- first matrix
        mat2    [vector!]       ; -- second matrix
        dst     [vector!]       ; -- destination matrix
        alpha   [float!]        ; -- ratio of first matrix mixed with the second [0.0-1.0]
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvChannel

Separates source image in RGBA channels

```
rcvChannel: routine [
   src   [image!]       ; -- source image
   dst   [image!]       ; -- destination image
   op    [integer!]     ; -- channel selection
]
op:
1: red channel
2: green channel
3: blue channel
4: alpha channel
```

Defined in /libs/core/rcvCore.red

rcvSplit

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

```
rcvSplit: function [
        src     [image!]            ; -- source image
        dst     [image!]            ; -- destination image
        /red /green /blue /alpha    ; -- selected channel
]
```
Defined in /libs/core/rcvCore.red

rcvSplit2

Separates source image in RGBA channels.

```
rcvSplit: function [
        src     [image!]            ; -- source image
        return: [block!]            ; -- block with r g b a images
]
```
Defined in /libs/core/rcvCore.red

Similar to rcvSplit, but for easier access returns a block with 4 images

rcvMerge

Combines 3 images to destination image

```
rcvMerge: routine [
        src1            [image!]    ; -- source 1 image (r)
        src2            [image!]    ; -- source 2 image (g)
        src3            [image!]    ; -- source 3 image (b)
        src4            [image!]    ; -- source R image (a)
        return:         [image!]    ; -- result image
]
```
Defined in /libs/core/rcvCore.red

This function takes r channel from image 1, g channel form image 2, b channel from image3 and alpha from image 4 to make destination image. You can change image order as you want.

rcvMerge2

Combines 4 images to destination image

```
rcvMerge: routine [
        src1            [image!]    ; -- source 1 image (r)
        src2            [image!]    ; -- source 2 image (g)
        src3            [image!]    ; -- source 3 image (b)
```

```
        src4            [image!]        ; -- source R image (a)
        return:         [image!]        ; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvSplit2Mat

Separates image channels to 4 8-bit matrices

```
rcvSplit2Mat: routine [
        src     [image!]        ; -- source image
        mat0    [vector!]       ; -- image alpha channel
        mat1    [vector!]       ; -- image red channel
        mat2    [vector!]       ; -- image green channel
        mat3    [vector!]       ; -- 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: routine [
        mat0    [vector!]       ; -- image alpha channel
        mat1    [vector!]       ; -- image red channel
        mat2    [vector!]       ; -- image green channel
        mat3    [vector!]       ; -- image blue channel
        dst     [image!]        ; -- result image
]
```
Defined in /libs/matrix/rcvMatrix.red

Format conversion

rcvImage2Mat

Converts Red image to 8-bit 2-D matrix

```
rcvImage2Mat: routine [
        src     [image!]        ; -- Red image
        mat     [vector!]       ; -- destination integer matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMat2Image

Converts 8, 16 or 32-bit integer matrix to Red image

```
rcvMat2Image: routine [
        mat     [vector!]       ; -- integer matrix
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMakeBinaryMat

Makes a 0..1 matrix

```
rcvMakeBinaryMat: routine [
        src     [vector!]       ; -- integer matrix
        dst     [vector!]       ; -- result matrix [0..1]
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMat2Binary

Matrix to binary values

```
rcvMat2Binary: function [
        mat     [vector!]       ; -- integer or float matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvConvertMatIntScale

Fast integer matrix scale conversion

```
rcvConvertMatIntScale: routine [
        src             [vector!]       ; -- integer source matrix
        dst             [vector!]       ; -- integer destination matrix
        srcScale        [float!]        ; -- source scale eg FFh
        dstScale        [float!]        ; -- destination scale eg FFFFh
```

]
Defined in /libs/matrix/rcvMatrix.red

rcvConvertMatScale

Converts matrix to another bit size

```
rcvConvertMatScale: function [
        src             [vector!]       ; -- integer or float matrix
        dst             [vector!]       ; -- integer or float matrix
        srcScale        [number!]       ; -- source scale (integer or float)
        dstScale        [number!]       ; -- destination scale (integer or float)
        /fast /normal]
]
refinement
/normal: uses a general function
/fast: uses a fast 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!]       ; -- integer source matrix
        dst             [vector!]       ; -- integer destination matrix
        srcScale        [float!]        ; -- source range as float!
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMatFloat2Int

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

```
rcvMatFloat2Int: function [
        src             [vector!]       ; -- float matrix
        dst             [vector!]       ; -- integer matrix
        dstScale        [integer!]      ; -- destination range as integer
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvLogMatFloat

Applies log transform

```
rcvLogMatFloat: function [
        src     [vector!]       ; -- float source matrix
        dst     [vector!]       ; -- float destination matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

This transform is really useful with FFT algorithms.

rcvMat2Array

Matrice to array (block of vectors)

```
rcvMat2Array: routine [
        mat             [vector!]       ; -- integer or float matrix
        matSize         [pair!]         ; -- matrix size as a pair
]
```

Defined in /libs/matrix/rcvMatrix.red

cvArray2Mat

Block of vectors (array) to matrix (vector)

```
cvArray2Mat: routine [
        array           [block!]        ; -- array of integer or float vectors
        return:         [vector!]       ; -- integer or float vector
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvImg2Array

Red image to array

```
rcvImg2Array: routine [
        src             [image!]        ; -- Red image
        op              [integer!]      ; -- channel selection
        return:         [block!]        ; -- array
]
```

op:
1 red channel
2 green channel
3 blue channel
4 alpha channel
5 rgba
6 grayscale

Defined in /libs/matrix/rcvMatrix.red

Color and color space conversion

rcvConvert

General image color conversion routine

```
rcvConvert: routine [
        src1    [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        op      [integer!]      ; -- for conversion
]
```

op allows a lot of conversions. See routine code for detail.

Defined in /libs/core/rcvCore.red

rcvInvert

Destination image: inverted source image (Similar to NOT image)

```
rcvInvert: function [
        source          [image!]        ; -- source image
        dst             [image!]        ; -- destination image
]
```

Defined in /libs/core/rcvCore.red

rcv2NzRGB

Normalizes the RGB values of an image

```
rcv2NzRGB: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        /sum/sumsquare      ; -- refinement
]
```

refinement

sum: sum of r g b values is used for normalization

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] image

```
rcv2BW: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```

Defined in /libs/core/rcvCore.red

rcv2WB

Converts RGB image to white [255] and black [0] image

```
rcv2WB: function [
        src     [image!]        ; -- source image
        dst [   image!]         ; -- destination image
]
```
Defined in /libs/core/rcvCore.red

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

rcv2Gray

Converts RGB image to Grayscale according to refinement

```
rcv2Gray: function [
src     [image!]                ; -- source image
dst      [image!]               ; -- destination image
/average /luminosity /lightness ; -- refinement
return: [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!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/core/rcvCore.red

rcv2RGBA

Converts BGRA to RGBA

```
rcv2RGBA: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/core/rcvCore.red

rcvHSV

RGB<=>HSV

```
rcvHSV: routine [
        src             [image!]        ; -- source image
         dst            [image!]        ; -- destination image
        op              [integer!]      ; -- op: 1 to RGB, 2 to BGR
]
```
Defined in /libs/core/rcvCore.red

rcvRGB2HSV

RBG color to HSV conversion

```
rcvRGB2HSV: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/ imgproc/rcvImageProc.red

rcvBGR2HSV

BGR color to HSV conversion

```
rcvBGR2HSV: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/ imgproc/rcvImageProc.red

The Hue/Saturation/Value (HSV) 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.

rcvHLS

RGB<=>HLS

```
rcvHLS: routine [
   src          [image!]        ; -- source image
```

```
    dst             [image!]        ; -- destination image
    op              [integer!]      ; -- op: 1: RGB, 2: BGR
]
```
Defined in /libs/ imgproc/rcvImageProc.red

rcvRGB2HLS

RBG color to HLS conversion

```
rcvRGB2HLS: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/ imgproc/rcvImageProc.red

rcvBGR2HLS

BGR color to HLS conversion

```
rcvBGR2HLS: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/ imgproc/rcvImageProc.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.

rcvYCrCb

RGB<=>YCrCb JPEG (a.k.a. YCC)

```
rcvYCrCb: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        op      [integer!]      ; -- op 1: RGB, 2: BGR
]
```
Defined in /libs/ imgproc/rcvImageProc.red

rcvRGB2YCrCb

RBG color to YCrCb conversion

```
rcvRGB2YCrCb: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/ imgproc/rcvImageProc.red

rcvBGR2YCrCb

BGR color to YCrCb conversion

```
rcvBGR2YCrCb: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/ imgproc/rcvImageProc.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

rcvXYZ

RGB<=>CIE XYZ.Rec 709 with D65 white point

```
rcvXYZ: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        op      [integer!]      ; -- op 1: to BGR 2: to RGB
]
```
Defined in /libs/ imgproc/rcvImageProc.red

rcvRGB2XYZ

RGB to CIE XYZ color conversion

```
rcvRGB2XYZ: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/ imgproc/rcvImageProc.red

rcvBGR2XYZ

BGR to CIE XYZ color conversion

```
rcvBGR2XYZ: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/ imgproc/rcvImageProc.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]

rcvXYZ2RGB

CIE XYZ to BGR color conversion

```
rcvBGR2XYZ: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```

Defined in /libs/ imgproc/rcvImageProc.red

rcvLab

RGB<=>CIE L*a*b*

```
rcvLab: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        op      [integer!]      ; -- op 1: RGB 2: BGR
]
```

Defined in /libs/ imgproc/rcvImageProc.red

rcvRGB2Lab

RBG color to CIE L*a*b conversion

```
rcvRGB2Lab: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```

Defined in /libs/ imgproc/rcvImageProc.red

rcvRGB2Lab

RBG color to CIE L*a*b conversion

```
rcvBGR2Lab: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```

Defined in /libs/ imgproc/rcvImageProc.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.

rcvLuv

RBG color to CIE L*u*v conversion

```
rcvLuv: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        op      [integer!]      ; -- op 1: RGB, 2: BGR
]
```

Defined in /libs/ imgproc/rcvImageProc.red

rcvRGB2Luv

RBG color to CIE L*u*v conversion

```
rcvRGB2Luv: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```

Defined in /libs/ imgproc/rcvImageProc.red

rcvRGB2Luv

RBG color to CIE L*u*v conversion

```
rcvBGR2Luv: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```

Defined in /libs/ imgproc/rcvImageProc.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!]        ; -- source image
        dst     [image!]        ; -- destination image
        val     [integer!]      ; -- integer value as parameter for color adjustment
]
```

Defined in /libs/ imgproc/rcvImageProc.red

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

rcvIR2RGB

Pseudo-color to RGB image

```
rcvIR2RG: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        mat     [vector!]       ; -- array of float values for color adjustment
]
```

Defined in /libs/ imgproc/rcvImageProc.red

This transformation is useful for processing infrared thermal images.

Arithmetic operators on image and matrix

rcvMath

General Routine for math operators on image

```
rcvMath: routine [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- destination image
        op      [integer!]      ; -- op is used for math operator such as +, - …
]
```

Defined in /libs/core/rcvCore.red

rcvLIP

Logarithmic Image Processing Model

```
rcvLIP: routine [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- destination image
        op      [integer!]      ; -- op is used for LIP operator such as +, - …
]
```

Defined in /libs/core/rcvCore.red

rcvMathS

General routine for scalar on image

```
rcvMathS: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        v       [integer!]      ; -- integer value
        op      [integer!]      ; -- op is used for scalar operator such as +, - …
]
```

Defined in /libs/core/rcvCore.red

rcvMathF

General routine for float

```
rcvMathF: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        v       [float!]        ; -- float value
        op      [integer!]      ; -- op is used for scalar operator such as +, - …
]
```

Defined in /libs/core/rcvCore.red

rcvMathT

General routine for tuple

```
rcvMathT: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        t       [tuple!]        ; -- color as tuple
        op      [integer!]      ; -- op is used for tuple operator such as +, - …
        flag    [logic!]        ; -- if true, tuple values are in range 0..255
]
```

Defined in /libs/core/rcvCore.red

rvcLogical

General routine for logical operators on image

```
rvcLogical: routine [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- destination image
        op      [integer!]      ; -- op is used for scalar operator such as and, or …
]
```

Defined in /libs/core/rcvCore.red

rcvAdd

dst: src1 + src2

```
rcvAdd: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```

Defined in /libs/core/rcvCore.red

rcvAddMat

Returns dst: src1 + src2

```
rcvAddMat: function [
        src1    [vector!]       ; -- first source matrix
        src2    [vector!]       ; -- second source matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvAddLIP

Destination image: image 1 + image 2 (Logarithmic Image Processing)

```
rcvAddLIP: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result 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!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```

Defined in /libs/core/rcvCore.red

rcvSubMat

Returns dst: src1 - src2

```
rcvSubMat: function [
        src1    [vector!]       ; -- first source matrix
        src2    [vector!]       ; -- second source matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvSubLIP

Destination image: image 1 - image 2 (Logarithmic Image Processing)

```
rcvSubLIP: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result 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!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```

Defined in /libs/core/rcvCore.red

rcvMulMat

Returns dst: src1 * src2

```
rcvMulMat: function [
        src1    [vector!]       ; -- first source matrix
        src2    [vector!]       ; -- second source matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvDiv

dst: src1 / src2

```
rcvDiv: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```

Defined in /libs/core/rcvCore.red

rcvDivMat

Rturns dst: src1 / src2

```
rcvDivMat: function [
        src1    [vector!]       ; -- first source matrix
        src2    [vector!]       ; -- second source matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvMod

dst: src1 // src2 (modulo)

```
rcvMod: function [
	src1	[image!]	; -- first source image
	src2	[image!]	; -- second source image
	dst	[image!]	; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvRem

dst: src1 % src2 (remainder)

```
rcvRem: function [
	src1	[image!]	; -- first source image
	src2	[image!]	; -- second source image
	dst	[image!]	; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvRemMat

Returns dst: src1 % src2

```
rcvRemMat: function [
	src1	[vector!]	; -- first source matrix
	src2	[vector!]	; -- second source matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvAbsDiff

dst: absolute difference src1 src2

```
rcvAbsDiff: function [
	src1	[image!]	; -- first source image
	src2	[image!]	; -- second source image
	dst	[image!]	; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvMIN

dst: minimum src1 src2

```
rcvMIN: function [
	src1	[image!]	; -- first source image
	src2	[image!]	; -- second source image
	dst	[image!]	; -- result image
```

]
Defined in /libs/core/rcvCore.red

rcvMAX

dst: maximun src1 src2

```
rcvMax: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvLSH

Left shift image by value

```
rcvLSH: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; --destination image
        val     [integer!]      ; -- shift value
]
```
Defined in /libs/core/rcvCore.red

rcvRSH

Right shift image by value

```
rcvRSH: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        val     [integer!]      ; -- shift value
]
```
Defined in /libs/core/rcvCore.red

rcvPow

dst: src ^integer! or Float! Value

```
rcvPow: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; --destination image
        val     [integer!]      ; -- power value
]
```
Defined in /libs/core/rcvCore.red

rcvSqr

Image square root

```
rcvSqr: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[integer!]	; -- sqr value
]
```
Defined in /libs/core/rcvCore.red

rcvMeanImages

dst: (src1 + src2) /2

```
rcvMeanImages: function [
	src1	[image!]	; -- first source image
	src2	[image!]	; -- second source image
	dst	[image!]	; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvMeanMats

Calculates mean values for 2 matrices and returns result matrix

```
rcvMeanMat: function [
	src1	[vector!]	; -- matrix 1
	src2	[vector!]	; -- matrix 2
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMeanMat

Matrix mean as float value

```
rcvMeanMat: function [
	mat	[vector!]	; -- matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvAddS

dst: src + integer! or float! value

```
rcvAddS: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[number!]	; --value
]
```
Defined in /libs/core/rcvCore.red

rcvAddSMat

src + value

```
rcvAddSMat: function [
	src	[vector!]	; -- matrix
	value	[integer!]	; -- integer value
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvAddT

dst: src + tuple! value

```
rcvAddT: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[tuple!]	; -- color as tuple
]
```
Defined in /libs/core/rcvCore.red

rcvSubS

dst: src - integer! or float! value

```
rcvSubS: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[number!]	; -- value
]
```
Defined in /libs/core/rcvCore.red

rcvSubSMat

src - value

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

rcvSubT

dst: src - tuple! value

```
rcvSubT: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[tuple!]	; -- color as tuple
]
```

Defined in /libs/core/rcvCore.red

rcvMulS

dst: src * integer! or float! value

```
rcvMulS: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[number!]	; -- value
]
```

Defined in /libs/core/rcvCore.red

rcvMulSMat:

dst: src * integer! value

```
rcvMulSMat: function [
	src	[vector!]	; -- matrix
	value	[integer!]	; -- integer value
]
```

Defined in /libs/core/rcvCore.red

rcvMulT

dst: src * tuple! value

```
rcvMulT: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[tuple!]	; -- color as tuple
]
```

Defined in /libs/core/rcvCore.red

rcvDivS

dst: src / integer! or float! value

```
rcvDivS: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[number!]	; -- value
]
```

Defined in /libs/core/rcvCore.red

rcvDivSMat

src / value

```
rcvDivSMat: function [
	src	[vector!]	; -- matrix
```

```
        value   [integer!]      ; -- integer value
]
        src: matrix
        value: integer
```
Defined in /libs/matrix/rcvMatrix.red

rcvDivT

dst: src / tuple! value

```
rcvDivT: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        val     [tuple!]        ; -- color as tuple
]
```
Defined in /libs/core/rcvCore.red

rcvModS

dst: src // integer! value (modulo)

```
rcvModS: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        val     [integer!]      ; -- integer value
]
```
Defined in /libs/core/rcvCore.red

rcvModT

dst: src // tuple! Value (modulo)

```
rcvModT: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        val     [tuple!]        ; -- color as tuple
]
```
Defined in /libs/core/rcvCore.red

rcvRemS

dst: src % integer! Value (remainder)

```
rcvRemS: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        val     [tuple!]        ; -- integer value
]
```
Defined in /libs/core/rcvCore.red

rcvRemT

dst: src % tuple! Value (remainder)

```
rcvRemT: function [
	src	[image!]	; -- source image
	dst	[image!]	; -- destination image
	val	[tuple!]	; -- color as tuple
]
```

Defined in /libs/core/rcvCore.red

rcvRemSMat

src % value (remainder)

```
rcvRemSMat: function [
	src	[vector!]	; -- matrix
	value	[integer!]	; -- integer value
]
```

Defined in /libs/matrix/rcvMatrix.red

Logical operators on image and matrix

rcvAND

dst: src1 AND src2

```
rcvAND: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```

Defined in /libs/core/rcvCore.red

rcvANDMat

Returns source1 AND source2

```
rcvAndMat: function [
        src1    [vector!]       ; -- first matrix
        src2    [vector!]       ; -- second matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvOR

dst: src1 OR src2

```
rcvOR: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```

Defined in /libs/core/rcvCore.red

rcvORMat

Returns source1 OR source2

```
rcvORMat: function [
        src1    [vector!]       ; -- first matrix
        src2    [vector!]       ; -- second matrix
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvXOR

dst: src1 XOR src2

```
rcvXOR: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvXORMat

Returns source1 XOR source2

```
rcvORMat: function [
        src1    [vector!]       ; -- first matrix
        src2    [vector!]       ; -- second matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvNAND

dst: src1 NAND src2

```
rcvNAND: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvNOR

dst: src1 NOR src2

```
rcvNOR: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```
Defined in /libs/core/rcvCore.red

rcvNXOR

dst: src1 NXOR src2

```
rcvNOR: function [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```

Defined in /libs/core/rcvCore.red

rcvNOT

dst: src1 NOT src2

```
rcvNOR: routine [
        src1    [image!]        ; -- first source image
        src2    [image!]        ; -- second source image
        dst     [image!]        ; -- result image
]
```

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!]        ; -- source image
        dst     [image!]        ; -- destination image
        value   [tuple!]        ; -- color value as a 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!]        ; -- source image
        dst     [image!]        ; -- destination image
        value   [tuple!]        ; -- color value as a 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!]        ; -- source image
        dst     [image!]        ; -- destination image
        value   [tuple!]        ; -- color value as a tuple
]
```

Defined in /libs/core/rcvCore.red

rcvANDSMat

And integer value to all elements in source matrix

```
rcvANDSMat: function [
        src     [vector!]       ; -- matrix
        value   [integer!]      ; -- integer value
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvORSMat

OR integer value to all elements in source matrix

```
rcvANDSMat: function [
        src     [vector!]       ; -- matrix
        value   [integer!]      ; -- integer value
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvXORSMat

XOR integer value to all element in source matrix

```
rcvANDSMat: function [
        src     [vector!]       ; -- matrix
        value   [integer!]      ; -- integer value
]
```

Defined in /libs/matrix/rcvMatrix.red

Complex numbers

Since Red doesn't support complex numbers, we use vectors (e.g. a: make vector! [1.0 0.0]).

rcMathComplex

is a general routine used for complex operators.

rcvAddComplex

Adds 2 complex numbers

```
rcvAddComplex: function [
        a               [vector!]       ; -- complex number as 2 float vector
        b               [vector!]       ; -- complex number as 2 float vector
        return:         [vector!]       ; -- returns real and imaginary values as vector
]
```

Defined in /libs/math/rcvComplex.red

rcvSubComplex

Substracts 2 complex numbers

```
rcvAddComplex: function [
        a               [vector!]       ; -- complex number as 2 float vector
        b               [vector!]       ; -- complex number as 2 float vector
        return:         [vector!]       ; -- returns real and imaginary values as vector
]
```

Defined in /libs/math/rcvComplex.red

rcvMultiplyComplex

Multiplies 2 complex numbers

```
rcvAddComplex: function [
        a               [vector!]       ; -- complex number as 2 float vector
        b               [vector!]       ; -- complex number as 2 float vector
        return:         [vector!]       ; -- returns real and imaginary values as vector
]
```

Defined in /libs/math/rcvComplex.red

rcvDivComplex

Divides 2 complex numbers

```
rcvAddComplex: function [
        a               [vector!]       ; -- complex number as 2 float vector
        b               [vector!]       ; -- complex number as 2 float vector
```

```
        return:         [vector!]       ; -- returns real and imaginary values as vector
]
```
Defined in /libs/math/rcvComplex.red

rcvMakeComplexArray

Makes an array of complex numbers

```
rcvAddComplex: function [
        input           [vector!]       ; -- array of real values as float
        return:         [block!]        ; -- returns a block of real and imaginary values as vector
]
```
Defined in /libs/math/rcvComplex.red

Statistics and image features extraction

General routines and functions

rcvCount

Returns the number of non-zero values in image

```
rcvCount: routine [
        src             [image!]        ; -- source image
        return:         [integer!]
]
```
Defined in /libs/math/rcvStats.red

rcvCountMat

Returns number of non-zero values in matrix

```
rcvCountMat: routine [
        mat             [vector!]       ; -- matrix
        return:         [integer!]
]
```
Defined in /libs/math/rcvStats.red

rcvCountNonZero

Returns number of non-zero values in image or matrix

```
rcvCountNonZero: function [
        arr     [image! vector!]        ; -- image or matrix
]
```
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!]        ; -- image or matrix
        /argb                           ; -- includes alpha channel
]
```
Defined in /libs/math/rcvStats.red

rcvSumMat

Returns matrix sum as a float value

```
rcvSumMat: routine [
        mat     [vector!]       ; -- matrix
]
```
Defined in /libs/math/rcvStats.red

rcvMeanImg

Returns mean value of image as an integer value

```
rcvMeanImg: routine [
        src             [image!]        ; -- Red image
        return:         [integer!]
]
```
Defined in /libs/math/rcvStats.red

rcvMeanMat

Returns matrix mean as float value

```
rcvMeanMat: routine [
        mat     [vector!]               ; -- matrix
]
```
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!]        ; -- image or matrix
        /argb                           ; -- includes alpha channel
]
```
Defined in /libs/math/rcvStats.red

RcvStdImg

Returns standard deviation value of image as an integer

```
rcvStdImg: routine [
        src             [image!]        ; -- Red image
        return:         [integer!]
]
```
Defined in /libs/math/rcvStats.red

rcvStdMat

Returns standard deviation value of matrix as a float

```
rcvStdMat: routine [
        mat             [vector!]       ; -- matrix
        return:         [float!]
```

]
Defined in /libs/math/rcvStats.red

rcvSTD

Returns standard deviation value of image or matrix as tuple

```
rcvSTD: function [
        arr     [image! vector!]        ; -- image or matrix
        /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!]        ; -- image or matrix
        /argb                           ; -- includes alpha channel
]
```
Defined in /libs/math/rcvStats.red

rcvProdMat

Return matrix product as a float value

```
rcvProdMat: function [
        mat     [vector!]               ; -- matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMinValue

Returns minimal value of image or matrix as tuple

```
rcvMinValue: function [
        arr     [image! vector!]        ; -- image or matrix
]
```
Defined in /libs/math/rcvStats.red

rcvMaxValue

Returns maximum value of image or matrix as tuple

```
rcvMaxValue: function [
        arr     [image! vector!]        ; -- image or matrix
]
```
Defined in /libs/math/rcvStats.red

rcvMaxMat

Return maximum of matrix as a number

```
rcvMaxMat: function [
	mat	[vector!]		; -- matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMinMat

Return minimum of matrix as a number

```
rcvMinMat: function [
	mat	[vector!]		; -- matrix
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMinLocImg

Finds global minimum location in image

```
rcvMinLocImg: routine [
	src	[image!]		; -- Red image
	return: [pair!]
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMinLocMat

Finds global minimum location in matrix

```
rcvMinLocMat: routine [
	mat		[vector!]	; -- matrix
	matSize		[pair!]		; -- matrix size
	return:		[pair!]
]
```
Defined in /libs/matrix/rcvMatrix.red

rcvMinLoc

Finds global minimum location in array and returns as pair

```
rcvMinLoc: function [
	arr	[image! vector!]	; -- image or matrix
	arrSize [pair!]			; -- image or matrix size
]
```
Defined in /libs/math/rcvStats.red

rcvMaxLocImg

Finds global maximun location in image

```
rcvMaxLocImg: routine [
        src1    [image!]            ; -- Red image
        return: [pair!]
]
```

Defined in /libs/math/rcvStats.red

rcvMaxLocMat

Finds global maximum location in matrix

```
rcvMaxLocMat: routine [
        mat         [vector!]       ; -- matrix
        matSize     [pair!]         ; -- matrix size
        return:     [pair!]
]
```

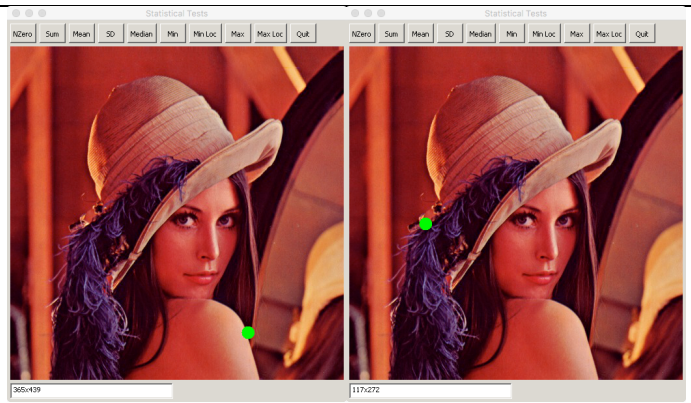
Defined in /libs/math/rcvStats.red

rcvMaxLoc

Finds global maximum location in array and returns as pair

```
rcvMaxLoc: function [
        arr     [image! vector!]    ; -- image or matrix
        arrSize [pair!]             ; -- image or matrix size
]
```

arr: image or vector
arrSize: array size as pair
Defined in /libs/math/rcvStats.red



Min Location Max Location

rcvSortImagebyX

Sorts image columns

```
rcvSortImagebyX: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        b               [vector!]       ; -- temporary block for sorting image
        flag            [logic!]        ; -- if true then reverse sorting
]
```
Defined in /libs/math/rcvStats.red

rcvSortImagebyY

Sorts image lines

```
rcvSortImagebyY: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        b               [vector!]       ; -- temporary block for sorting image
        flag            [logic!]        ; -- if true then reverse sorting
]
```
Defined in /libs/math/rcvStats.red

rcvSortImage

Ascending image sorting

```
rcvSortImage: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
]
```
Defined in /libs/math/rcvStats.red

rcvXSortImage

Image sorting by line

```
rcvXSortImage: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        flag            [logic!]        ; -- if true then reverse sorting
]
```
Defined in /libs/math/rcvStats.red

rcvYSortImage

Image sorting by column

```
rcvXSortImage: function [
src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        b               [vector!]       ; -- temporary block for sorting image
```

```
        flag            [logic!]        ; -- if true then reverse sorting
]
```
Defined in /libs/math/rcvStats.red

Histogram functions

rcvHistoImg

Calculates image histogram by channel

```
rcvHistoImg: routine [
        src             [image!]        ; -- source image
        op              [integer!]      ; -- channel
        return:         [vector!]
]
```

op:
1 Red Channel
2 Green Channel
3 Blue Channel
4 grayscale

Defined in /libs/math/rcvHistogram.red

rcvHistoMat

Calculate matrix histogram (integer or float matrices)

```
rcvHistoMat: routine [
        mat             [vector!]       ; -- matrix
        return:         [vector!]
]
```

Defined in /libs/math/rcvHistogram.red

rcvSumHistoMat

Calculates the cumulative sum of histogram

```
rcvSumHistoMat: routine [
        histo   [vector!]       ; -- matrix
        return: [vector!]
]
```

This is the cumulative-density function for the pixel value n

Defined in /libs/math/rcvHistogram.red

rcvHistogram

Calculates array histogram

```
rcvHistogram: routine [
        arr             [image! vector!]        ; -- image or matrix
        return:         [vector!]               ; -- selected channel
```

```
        /red /green /blue                   ; -- refinement
]
/red: histogram for red channel
/green: histogram for green channel
/blue: histogram for blue channel
```
Defined in /libs/math/rcvHistogram.red

rcvRGBHistogram

Calculates array histogram according to the number of bins

```
rcvRGBHistogram: routine [
        img     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        array   [block!]        ; -- for RGB bins
]destination image can be used to render the effect of the histogram filter
```
Defined in /libs/math/rcvHistogram.red

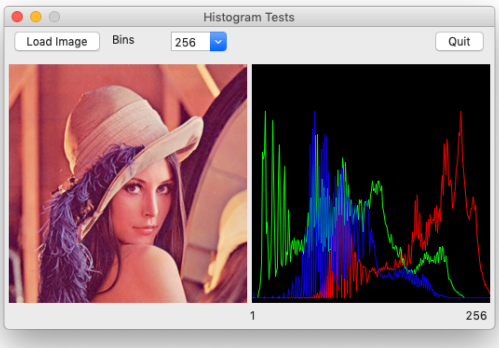
rcvMeanShift

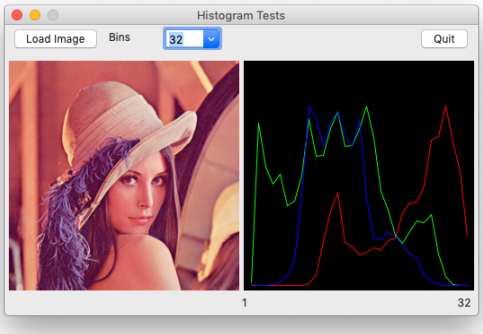
Mean Shift filter on image

```
rcvMeanShift: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        array           [block!]        ; -- block of vectors (typically rgb histogram)
        colorBW         [float!]        ; -- color bandwidth
        converg         [float!]        ; -- mean convergence factor
        op              [logic!]        ; -- rgb value and 255 if true
]
```
Defined in /libs/math/rcvHistogram.red

IMPORTANT: these functions use array type as 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
```



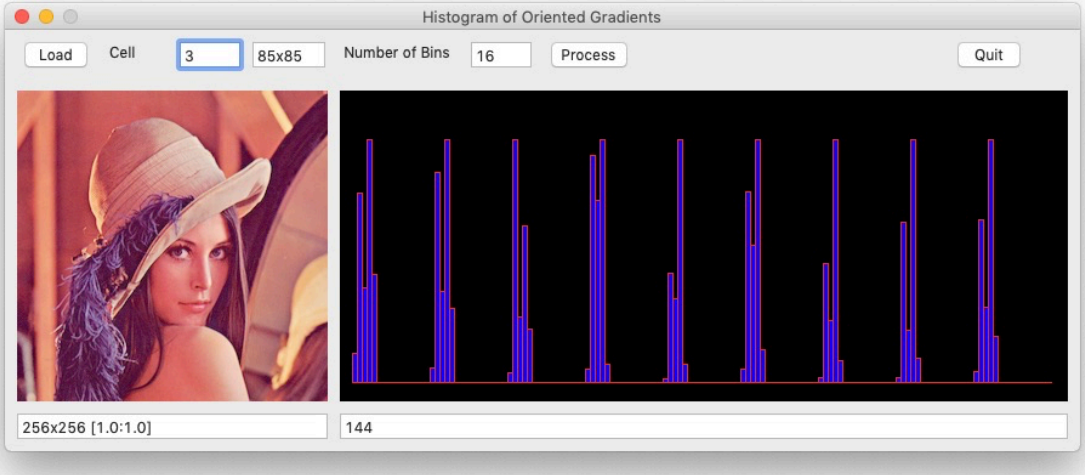


rcvHOG

Histograms of Oriented Gradients

```
rcvHOG: routine [
    src         [image!]        ; input image
    matGx       [vector!]       ; matrix for X gradients
    matGy       [vector!]       ; matrix for Y gradients
    nBins       [integer!]      ; number of bins for the histogram
    nDivs       [integer!]      ; dividor for cell number and cell size computation
    return:     [vector!]       ; Histogram of oriented gradients
]
```

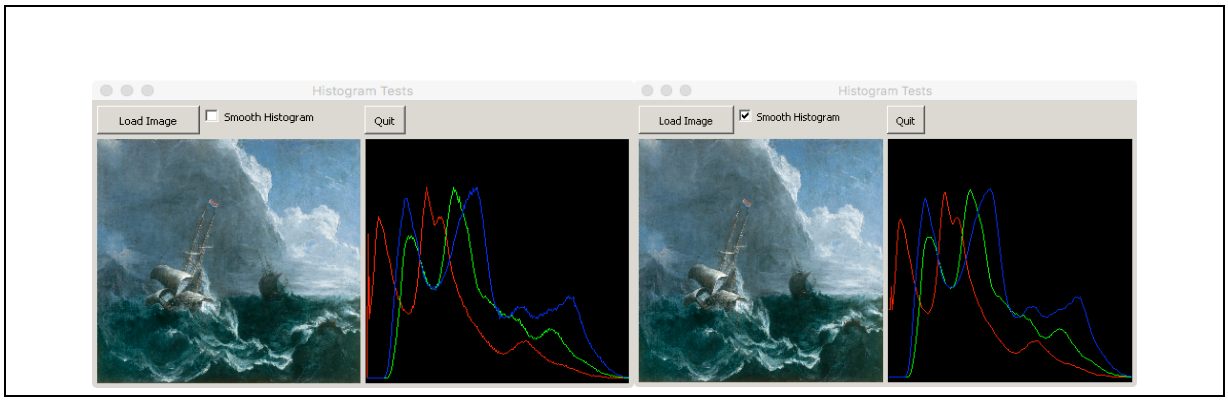
Defined in /libs/math/rcvHistogram.red



rcvSmoothHistogram

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

```
rcvSmoothHistogram: function [
	arr	[vector!]	; -- input histogram as vector!
]
```
Defined in /libs/math/rcvHistogram.red



rcvRangeImage

Gives range value in image as a tuple

```
rcvRangeImage: function [
	source		[image!]	; -- source image
]
```
Defined in /libs/math/rcvStats.red

Central and spatial moments

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

rcvGetMatSpatialMoment

Returns the spatial moment of the matrix as float

```
rcvGetMatSpatialMoment: routine [
        mat             [vector!]       ; -- matrix
        matSize         [pair!]         ; -- matrix size
        p               [float!]        ; -- the order of the moment
        q               [float!]        ; -- the repetition of the moment
]
```

Defined in /libs/math/rcvMoments.red

rcvGetMatCentralMoment

Returns the central moment of the matrix as float

```
rcvGetMatCentralMoment: routine [
        mat             [vector!]       ; -- matrix
        matSize         [pair!]         ; -- matrix size
        p               [float!]        ; -- the order of the moment
        q               [float!]        ; -- the repetition of the moment
]
```

Defined in /libs/math/rcvMoments.red

rcvGetNormalizedCentralMoment

Return the scale invariant moment of the image as float

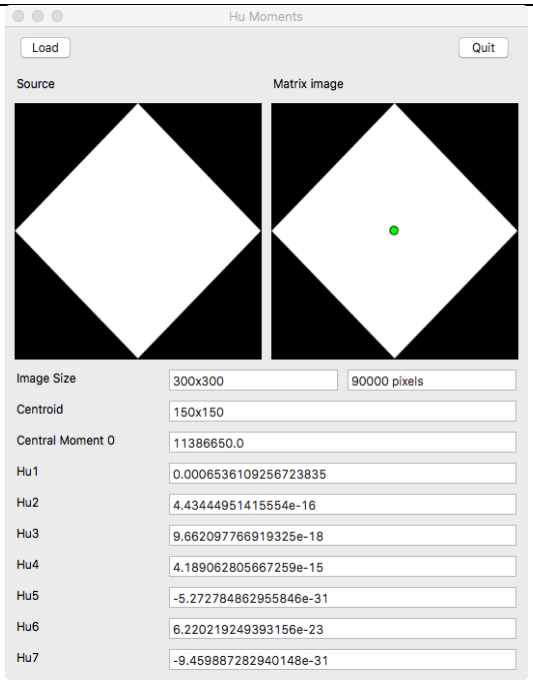
```
rcvGetNormalizedCentralMoment: function [
        mat             [vector!]       ; -- matrix
        matSize         [pair!]         ; -- matrix size
        p               [float!]        ; -- the order of the moment
        q               [float!]        ; -- the repetition of the moment
]
```

Defined in /libs/math/rcvMoments.red

rcvGetMatHuMoments

Returns Hu moments of the image as block

```
rcvGetMatHuMoments: function [
        mat             [vector!]       ; -- matrix
        matSize         [pair!]         ; -- matrix size
]
```
Defined in /libs/math/rcvMoments.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.

Integral Image

rcvIntegralImg

Direct integral image

```
rcvIntegralImg: routine [
    src         [image!]        ; -- source image
    dst         [image!]        ; -- image for summed area table
    dst2        [image!]        ; -- image for square summed area table
]
```

Defined in /libs/imgproc/rcvIntegral.red

rcvIntegralMat

Direct integral image on matrix

```
rcvIntegralMat: routine [
        src     [vector!]       ; -- integer matrice
        dst1    [vector!]       ; -- integer matrice for summed area table
        dst2    [vector!]       ; -- integer matrice for square summed area table
        mSize   [pair!]         ; -- integer matrice size as a pair
]
```

Defined in /libs/imgproc/rcvIntegral.red

rcvProcessIntegralImage

Gets boxes in integral image

```
rcvProcessIntegralImage: function [
        src             [image!]        ; -- source image
        w               [integer!]      ; -- image width
        h               [integer!]      ; -- image height
        boxW            [integer!]      ; -- box width
        boxH            [integer!]      ; -- box height
        thresh          [integer!]      ; -- thresholding value
        points          [block!]        ; -- results
]
```

Defined in /libs/imgproc/rcvIntegral.red

rcvProcessIntegralMat

Gets boxes in integral matrix

```
rcvProcessIntegralMat: routine [
        mat             [vector!]       ; -- source matrix
        w               [integer!]      ; -- matrix width
        h               [integer!]      ; -- matrix height
        boxW            [integer!]      ; -- box width
        boxH            [integer!]      ; -- box height
        thresh          [integer!]      ; -- thresholding value
        points          [block!]        ; -- results
]
```

Defined in /libs/imgproc/rcvIntegral.red

rcvIntegral

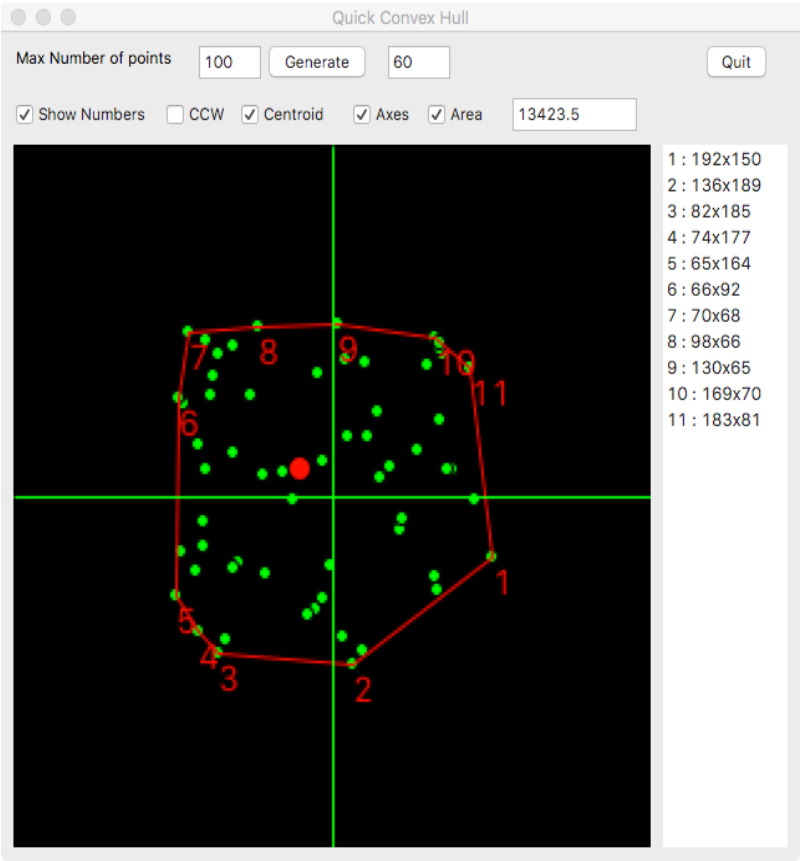
Calculates integral images

```
rcvIntegral: function [
	src		[image! vector!]		; -- image or matrix
	sum		[image! vector!]		; -- image or integer matrice for summed area table
	sqsum	[image! vector!]		; -- image or integer matrice for square summed area
	mSize	[pair!]					; -- image or integer matrice size as a pair
]
```

Defined in /libs/imgproc/rcvIntegral.red

Convex Hull

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. See RedCV_samples documentation for the detail.

rcvCross

Vectors cross product: 3 points are a counter-clockwise turn if rcvCross > 0, clockwise if rcvCross < 0, and collinear if rcvCross = 0 because rcvCross is a determinant that gives the signed area of the triangle formed by A, B and C

```
rcvCross: routine [
        A               [pair!]
```

```
        B               [pair!]
        C               [pair!]
        return:         [integer!]
]
```
Defined in /libs/math/rcvQuickHull.red

cvPointDistance

Square of the distance of point C to the segment defined by points AB

```
cvPointDistance: routine [
        A               [pair!]
        B               [pair!]
        C               [pair!]
        return:         [integer!]
]
```
Defined in /libs/math/rcvQuickHull.red

rcvFindExtrema

Finds minimal and maximal coordinates in block

```
rcvFindExtrema: function [
        points [block!]
]
```
Defined in /libs/math/rcvQuickHull.red

rcvSeparateSets

Separates left and right set

```
rcvSeparateSets: function [
        ptsBlock [block!]
]
```
Defined in /libs/math/rcvQuickHull.red

rcvHullSet

Recursive function for determing set

```
rcvHullSet: function [
        A       [pair!]
        B       [pair!]
        aSet    [block!]
        hull    [block!]
]
```
Defined in /libs/math/rcvQuickHull.red

rcvQuickHull

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

```
rcvQuickHull: function [
        points          [block!]        ; -- Input 2D point set as block of pairs
        /cw/ccw                         ; -- refinement
]
```

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/rcvQuickHull.red

rcvContourArea

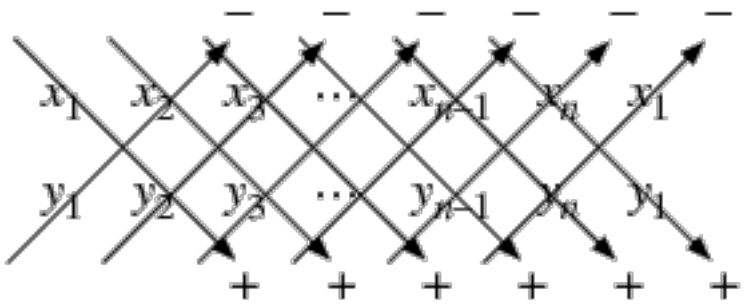
Calculates and returns the area of polygon generated by rcvQuickHull function as float

```
rcvContourArea: function [
        hull    [block!]; -- list of coordinates (as pair) generated by the rcvQuickHull function
        /signed
]
```

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

rcvFlipHV

Left Right, Up down or both directions flip

```
rcvFlipHV: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        op              [integer!]      ; -- flip direction
]
op 1: Left/right 2 Up/down 3 Both
```
Defined in /libs/imgproc/rcvImgProc.red

rcvFlip

Returns Left/Right, Up/Down or both directions image flip

```
rcvFlip: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        /horizontal /vertical /both     ; -- refinement for direction
]
```
Defined in /libs/imgproc/rcvImgProc.red

source Left/Right Up/Down Both

rcvResizeImage

Resizes image

```
rcvResizeImage: routine [
        src     [image!]        ; -- source image
        iSize   [pair!]         ; -- new size as a pair
]
```
Defined in /libs/core/rcvCore.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
```

rcvPyrDown

Performs downsampling step of Gaussian pyramid decomposition

```
rcvPyrDown: function [
        src     [image!]        ; -- source image
]
```

Source image size is divided by 2 and 5x5 Gaussian blurring effect is applied on result image.
Defined in /libs/imgproc/rcvImgProc.red

rcvPyrUp

Performs up-sampling step of Gaussian pyramid decomposition

```
rcvPyrDown: function [
        src     [image!]        ; -- source image
]
```

Source image size is multiplied by 2 and 5x5 Gaussian blurring effect is applied on result image.
Defined in /libs/imgproc/rcvImgProc.red

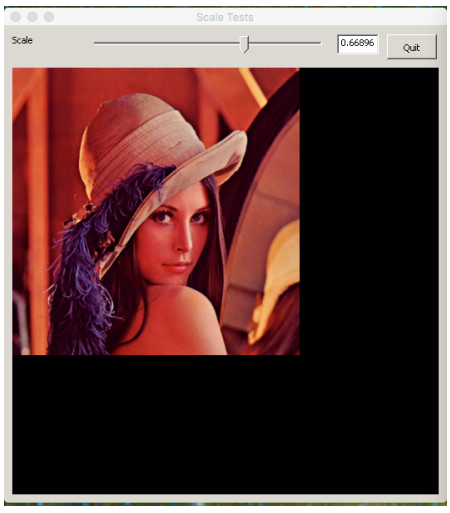
rcvScaleImage

Sets the scale factors: Returns a Draw block

```
rcvScaleImage: function [
        factor  [float!]        ; -- 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]
...
```



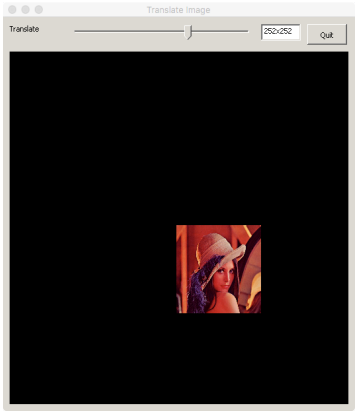
rcvTranslateImage

Sets the origin for drawing commands: Returns a Draw block

```
rcvTranslateImage: function [
        scaleValue          [float!]        ; -- float value to reduce or increase image size
        translateValue      [pair!]         ; -- 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]
...
```



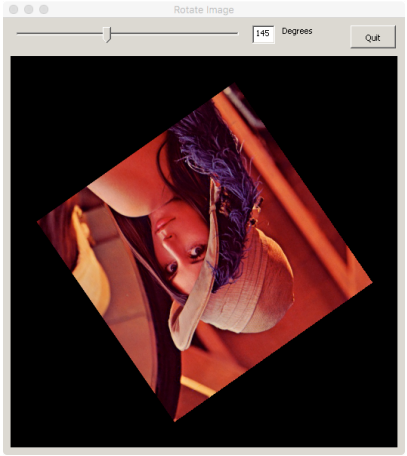
rcvRotateImage

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

```
rcvRotateImage: function [
        scaleValue          [float!]        ; -- float value to reduce or increase image size
        translateValue [    pair!]          ; -- pair to translate image in X and Y direction
        angle               [float!]        ; -- rotation of image in degrees
        center              [pair!]         ; -- center of 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]
...
```



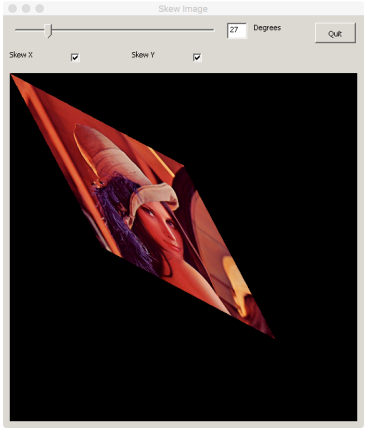
rcvSkewImage

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

```
rcvSkewImage: function [
        scaleValue          [float!]        ; -- float value to reduce or increase image size
        translateValue      [pair!]         ; -- pair to translate image in X and Y direction
        x                   [number!]       ; -- skew along the x-axis in degrees
        y                   [number!]       ; -- skew along the y-axis in degrees
]
```

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]
```



rcvClipImage

Returns a Draw block for image clipping

```
rcvClipImage: function [
        translateValue [pair!]      ; -- pair to translate image in X and Y direction
        start          [pair!]      ; -- up/left coordinate for clipping
        end            [pair!]      ; -- down/right coordinate for clipping
        img            [image!]     ; -- source image
]
```

rcvCropImage

Crop source image to destination image

```
rcvClipImage: function [
        src     [image!]            ;-- source image
        dst     [image!]            ;-- destination image
        origin  [pair!]]            ;-- origin is source image for cropping
]
```

rcvCropImage is similar to rcvClipImage but does'nt require Draw for rendering. Destimation image size determines the size of cropping source image.

rcvEffect

General routine for generating various effects on image

```
rcvEffect: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        param1          [float!]        ; -- angle parameter
        op              [integer!]      ; -- ee code for op values
]
```

Defined in /libs/imgproc/rcvImageProc.red

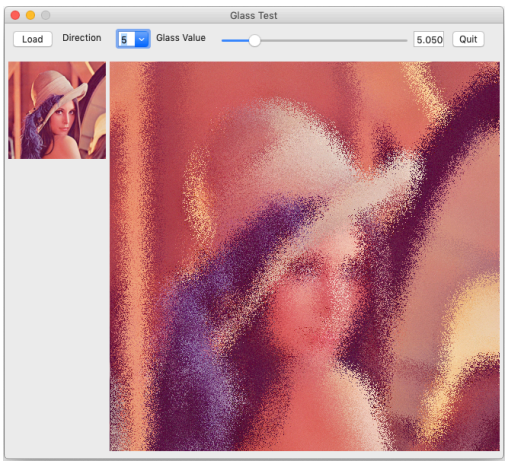
rcvGlass

Glass effect on image

```
rcvGlass: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        v       [float!]        ; -- random value
        op      [integer!]      ; -- effect direction
]
```

op: 1 horizontal, 2 vertical, 3 both direction, 4 and 5 oblique

Defined in /libs/imgproc/rcvImageProc.red

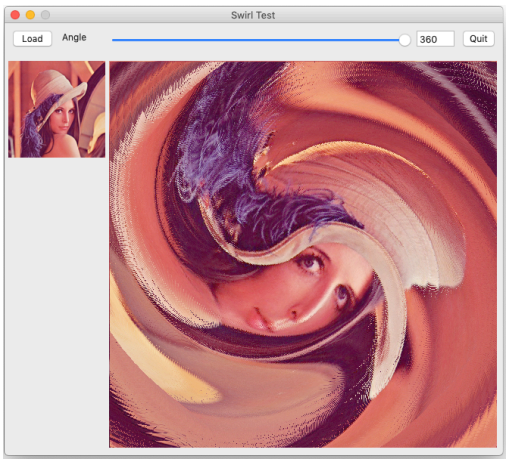


rcvSwirl

Swirl effect on image

```
rcvSwirl: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        theta [float!]   ; -- angle in radian]
```

Defined in /libs/imgproc/rcvImageProc.red



rcvWave

General routine for wave effects

```
rcvWave: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        param1          [float!]        ; -- α factor
        param2          [float!]        ; -- β factor
         op             [integer!]      ; -- wave effects
]
```

Defined in /libs/imgproc/rcvImageProc.red

rcvWaveH

Horizontal wave effect on image

```
rcvWaveH: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        alpha   [float!]        ; -- α factor
        beta    [float!] ; -- β factor
]
```

α and β are used to strengthen the e□ect

Defined in /libs/imgproc/rcvImageProc.red

rcvWaveV

Vertical wave effect on image

```
rcvWaveV: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        alpha   [float!]        ; -- α factor
        beta    [float!] ; -- β factor
]
```

α and β are used to strengthen the e□ect

Defined in /libs/imgproc/rcvImageProc.red

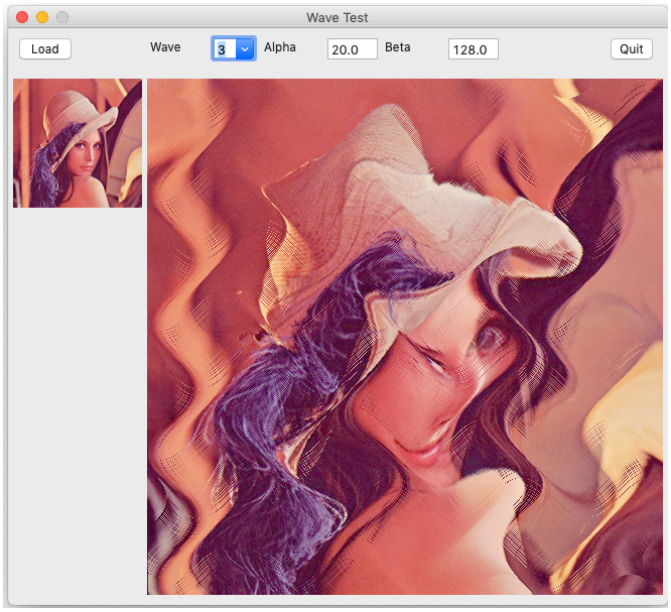
rcvWaveHV

Vertical and horizontal wave effect on image

```
rcvWaveHV: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        alpha   [float!]        ; -- α factor
        beta    [float!] ; -- β factor
]
```

α and β can be used to strengthen the e□ect

Defined in /libs/imgproc/rcvImageProc.red



Distances Functions

General distance functions

rcvDegree2xy

Returns XY coordinates from angle in degree and distance between 2 points

```
rcvDegree2xy: function [
        rho     [number!]     ; -- distance between 2 points
        theta [number!]       ; -- angle in degrees
]
```

Defined in /libs/math/rcvDistance.red

rcvRadian2xy

Returns XY coordinates from angle in radian and distance between 2 points

```
rcvDegree2xy: function [
        rho     [number!]     ; -- distance between 2 points
        theta [number!]       ; -- angle in degrees
]
```

Defined in /libs/math/rcvDistance.red

rcvGetEuclidianDistance

Returns Euclidian distance between 2 points as float

```
rcvGetEuclidianDistance: function [
        a       [pair!]       ; -- first point as pair
        b       [pair!]       ; -- second point as pair
]
```

Defined in /libs/math/rcvDistance.red

rcvGetEuclidian2Distance

Returns Squared Euclidian distance between 2 points as float

```
rcvGetEuclidian2Distance: function [
        a       [pair!]       ; -- first point as pair
        b       [pair!]       ; -- second point as pair
]
```

Defined in /libs/math/rcvDistance.red

rcvGetManhattanDistance

Returns Manhattan distance between 2 points as float

```
rcvGetManhattanDistance: function [
        a       [pair!]       ; -- first point as pair
        b       [pair!]       ; -- second point as pair
]
```

Defined in /libs/math/rcvDistance.red

rcvGetChessboardDistance

Returns Chessboard distance between 2 points as float

```
rcvGetChessboardDistance: function [
        a       [pair!]         ; -- first point as pair
        b       [pair!]         ; -- second point as pair
]
```
Defined in /libs/math/rcvDistance.red

rcvGetChebyshevDistance

Returns Chebyshev distance between 2 points as float

```
rcvGetChebyshevDistance: function [
        a       [pair!]         ; -- first point as pair
        b       [pair!]         ; -- second point as pair
]
```
Defined in /libs/math/rcvDistance.red

rcvGetMinkowskiDistance

Returns Minkowski distance between 2 points as float

```
rcvGetMinkowskiDistance: function [
        a       [pair!]         ; -- first point as pair
        b       [pair!]         ; -- second point as pair
        p       [float!]        ; -- 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!]         ; -- first point as pair
        b       [pair!]         ; -- 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!]         ; -- first point as pair
        b       [pair!]         ; -- second point as pair
]
```
Defined in /libs/math/rcvDistance.red

rcvDistance2Color

Returns tuple value modified by distance

```
rcvDistance2Color: routine[
        dist    [float!]        ; -- distance between points
        t       [tuple!]        ; -- color as tuple
]
```
Defined in /libs/math/rcvDistance.red

rcvGetAngle

Returns angle in degrees from 2 points coordinates as float

```
rcvGetAngle: function [
        p       [pair!]         ; -- first point as pair
        cg      [pair!]         ; -- 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!]         ; -- point as 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!]        ; -- block of distance (rho) normalized values
]
```
Defined in /libs/math/rcvDistance.red

Distance Mapping

rcvVoronoiDiagram

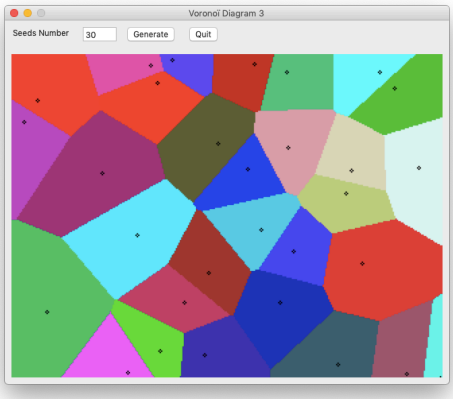
Creates Voronoï diagram

```
rcvVoronoiDiagram: routine [
        peaks           [block!]        ; -- block of coordinates as pair
        peaksC          [block!]        ; -- block of color as tuple
        img             [image!]        ; -- image for rendering
        param1          [logic!]        ; -- for seeds visualization
        param2          [integer!]      ; -- kind of distance used for rendering
        param3          [float!]        ; -- p value for Minkowski
]
```

parm2: 1: Euclidian 2: Manhattan 3: Minkowski 4: Chebyshev

Defined in /libs/math/rcvDistance.red

param3: only for Minkowski distance with 3.0 as default value



rcvDistanceDiagram

Creates Distance diagram (based on Boleslav Březovský's sample)

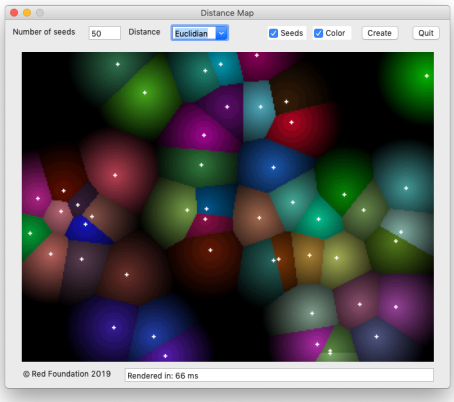
```
rcvDistanceDiagram: routine [
        peaks           [block!]        ; -- block of coordinates as pair
        peaksC          [block!]        ; -- block of color as tuple
        img             [image!]        ; -- image for rendering
        param1          [logic!]        ; -- for seeds visualization
```

```
        param2          [integer!]      ; -- kind of distance used for rendering
        param3          [float!]        ; -- p value for Minkowski
]
```

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

Defined in /libs/math/rcvDistance.red

param3: only for Minkowski distance with 3.0 as default value



kMean Algorithm

rcvKNearest

Distance or index of the closest cluster center

```
rcvKNearest: routine [
        pt              [vector!]       ; -- coordinate data
        centroid        [block!]        ; -- centroid
        op              [integer!]      ; -- op is used for distance or index computation
        return:         [float!]
]
```

Defined in /libs/math/rcvCluster.red

rcvKMInitData

Creates block data of centroid array

```
rcvKMInitData: function [
        count           [integer!]      ; -- number of elements of the array
]
```

Defined in /libs/math/rcvCluster.red

rcvKMGenCentroid

Generates centroids initial values

```
rcvKMGenCentroid: routine [
        array   [block!]        ; -- block generated by rcvKMInitData
]
```

Defined in /libs/math/rcvCluster.red

rcvKMInit

k-means first initialization

```
rcvKMInit: routine [
        points          [block!]        ; -- data block
        centroid        [block!]        ; -- centroid block
        tmpblk          [block!]        ; -- temporary block used for sum computation
]
```

Defined in /libs/math/rcvCluster.red

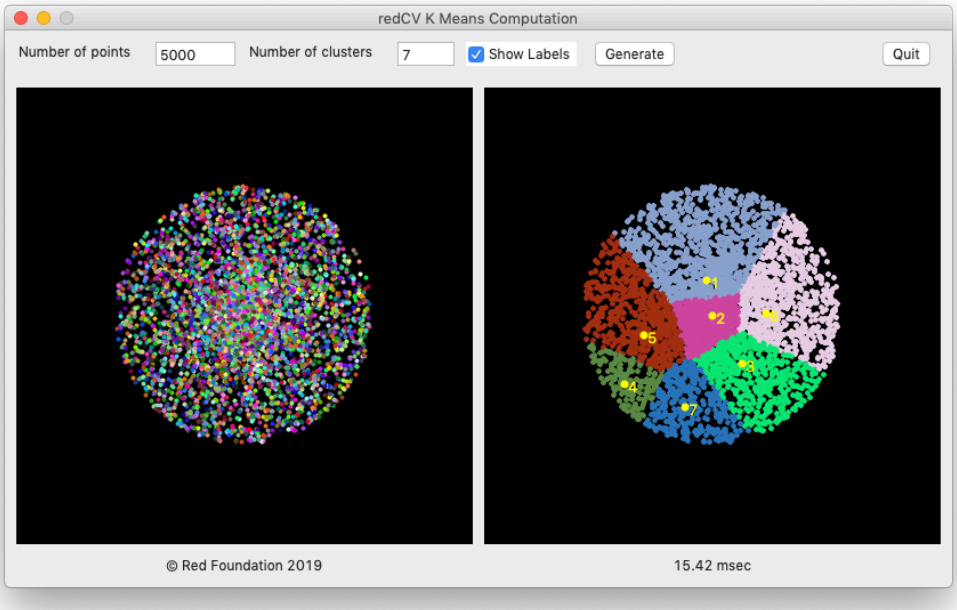
rcvKMCompute

Lloyd K-means clustering with convergence

```
rcvKMCompute: function [
        points          [block!]        ; -- data block
        centroid        [block!]        ; -- centroid block
]
```

Defined in /libs/math/rcvCluster.red

Important: **k-means functions require redCV 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.



Flow and Gradient

rcvMakeGradient

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

```
rcvMakeGradient: routine [
        src     [vector!]       ; -- integer matrix
        dst     [vector!]       ; -- integer matrix
        mSize   [pair!]         ; -- matrix size as a pair
]
```
Defined in /libs/math/rcvChamfer.red

rcvMakeBinaryGradient

Makes a binary [0 1] matrix for contour detection

```
rcvMakeBinaryGradient: routine [
        src             [vector!]       ; -- integer matrix
        mat             [vector!]       ; -- integer matrix
        maxG            [integer!]      ; -- max gradient value
        threshold       [integer!]      ; -- integer value for binary thresholding
]
```
Defined in /libs/math/rcvChamfer.red

rcvFlowMat

Returns the distance map to binarized gradient as float

```
rcvFlowMat: routine [
        input           [vector!]       ; -- input float matrix
        output          [vector!]       ; -- output float matrix
        scale           [float!]        ; -- scale 0..255
]
returns max distance
```

Defined in /libs/math/rcvChamfer.red

rcvnormalizeFlow

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

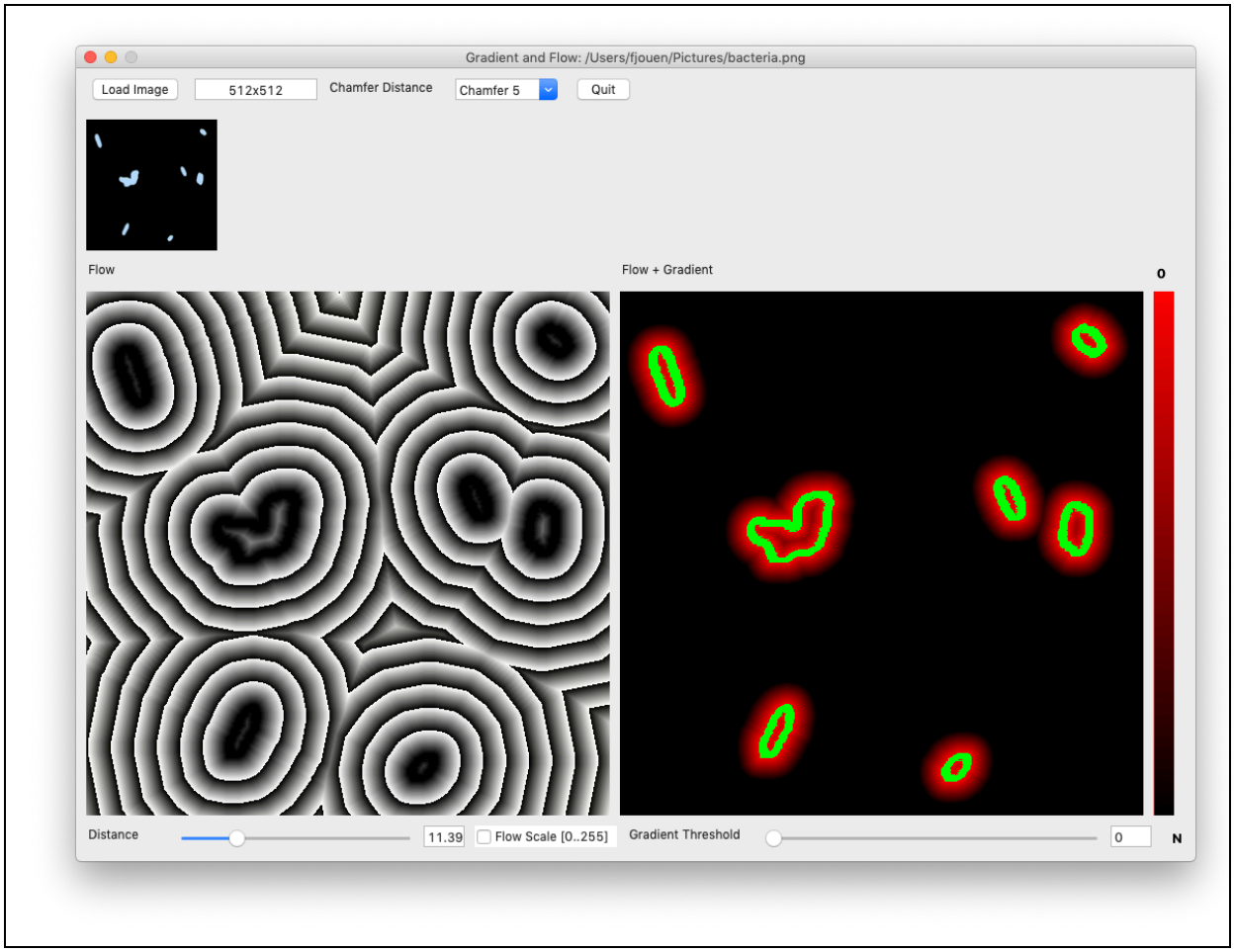
```
rcvnormalizeFlow: routine [
        input   [vector!]       ; -- input integer matrix
        factor  [float!]        ; -- value used for normalization such as max gradient value
```

]
Defined in /libs/math/rcvChamfer.red

rcvGradient&Flow

Creates an image including flow and gradient values

```
rcvGradient&Flow: routine [
        input1  [vector!]       ; -- flow integer matrix
        input2 [vector!]        ; -- gradient integer matrix
        dst     [image!]        ; -- Red image for mixing flow and gradient
]
```
Defined in /libs/math/rcvChamfer.red



Chamfer Distance

rcvChamferDistance

Selects a pre-defined chamfer kernel

```
rcvChamferDistance: function [
```

```
        chamferMask  [block!]          ; -- Kernel
]
```
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/rcvChamfer.red

rcvChamferCreateOutput

Creates a distance map (float!). Returns vector

```
rcvChamferCreateOutput: function [
        mSize   [pair!]             ; -- matrix size as pair
]
```
Defined in /libs/math/rcvChamfer.red

rcvChamferInitMap

Initializes distance map

```
rcvChamferInitMap: function [
        input           [vector!]       ; -- a binary [0/1] matrix
        output          [vector!]       ; -- a float matrix
]
```
Defined in /libs/math/rcvChamfer.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!] ; -- float matrix created by rcvChamferInitMap function
        chamfer         [block!]  ; -- selected pre-defined kernel used for distance computation
        mSize           [pair!]   ; -- matrix size
]
```
Defined in /libs/math/rcvChamfer.red

rcvChamferNormalize

Normalizes distance map

```
rcvChamferNormalize: routine [
        output          [vector!]       ; -- output matrix
        value           [integer!]      ; -- normalization value
]
```

Defined in /libs/math/rcvChamfer.red

Image enhancement

rcvMakeTranscodageTable

Creates a transcoding 256 vector for affine enhancement

```
rcvMakeTranscodageTable: function [
        n       [percent!]      ; -- percent of values to exclude
]
```
Defined in /libs/math/rcvHistogram.red

This function is used by rcvContrastAffine method. See below.

rcvEqualizeContrast

Enhances matrix contrast with affine transform

```
rcvEqualizeContrast: routine [
        mat     [vector!]       ; -- source matrix
        table   [vector!]       ; -- transcoding table
]
```
Defined in /libs/math/rcvHistogram.red

rcvContrastAffine

Enhances image contrast with affine function

```
rcvContrastAffine: function [
        image   [vector!]       ; -- 8-bit matrice
        n       [percent!]      ; -- percent of values to exclude
]
```
Defined in /libs/math/rcvHistogram.red

rcvEqualizeHistoMat

Histogram equalization for float or integer matrices

```
rcvEqualizeHistoMat: routine [
        mat             [vector!]       ; -- integer or float matrix
        sumHisto        [vector!]       ; -- 32-bit matrix
        constant        [float!]        ; -- for thresholding
]
```
Defined in /libs/math/rcvHistogram.red

rcvHistogramEqualization

This function performs histogram equalization on the input image array

```
rcvHistogramEqualization: function [
	image		[vector!]	; -- 32-bit matrix
	gLevels		[integer!]	; -- number of gray levels in the new image
]
```
Defined in /libs/math/rcvHistogram.red

Thresholding

rcvFilterBW

General B&W Filter routine

```
rcvFilterBW: routine [
        src1            [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        thresh          [integer!]      ; -- minimal thresholding value
        maxValue        [integer!]      ; -- maximal thresholding value
        op              [integer!]      ; -- used for creating various binary thresholding
]
```

op:
1 binary
2 binary Inverted
3 truncate
4 to Zero
5 to Zero Inverted

Defined in /libs/core/rcvCore.red

rcv2BWFilter

Binarization of RGB image according to threshold value

```
rcv2BWFilter: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        thresh [integer!]       ; -- 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!]        ; -- source image
        dst             [image!]        ; -- destination image
        thresh          [integer!]      ; -- threshold integer value
        mValue          [integer!]      ; -- maximal integer value
        /binary /binaryInv /trunc /toZero /toZeroInv
]
```

refinements are used for thresholding type

binary:dst(x,y) = mValue, if src(x,y)>threshold, 0, otherwise

binaryInv: 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

Defined in /libs/core/rcvCore.red

rcvInRange

Extracts sub array from image according to lower and upper rgb values

```
rcvInRange: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destinatioin image
        lower   [tuple!]        ; -- lower tuple
        upper   [tuple!]        ; -- lower tuple
        op      [integer!]      ; -- if op = 0 image is binarized else colors are extracted
]
```

Defined in /libs/core/rcvCore.red

cvInRangeMat

Extracts sub array from matrix according to lower and upper values

```
rcvInRangeMat: routine [
        src     [vector!]       ; -- source matrix
        dst     [vector!]       ; -- destination matrix
        lower   [integer!]      ; -- lower tuple
        upper   [integer!]      ; -- lower tuple
        op      [integer!]      ; -- if op = 0 image is binarized else colors 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) kernels. 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!]         ; -- uneven size for kernel (e.g 3x3)
        sigma   [float!]        ; -- required variance (e.g. 1.0)
]
```

Defined in /libs/imgproc/rcvImgProc.red

rcvMakeGaussian2

Creates a gaussian kernel

```
rcvMakeGaussian2: function [
        kSize   [pair!]         ; -- size as pair for kernel
        sigma   [float!]        ; -- variance
]
```

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 variance of the distribution.

rcvGaussianFilter

Fast Gaussian 2D filter

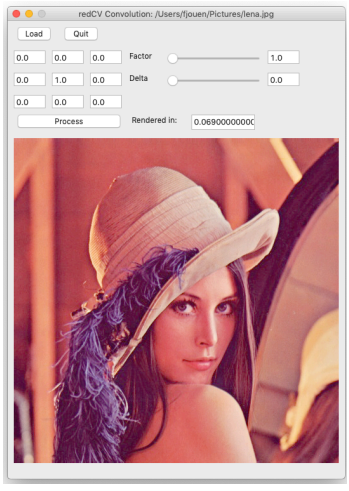
```
rcvGaussianFilter: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        kSize   [pair!]         ; -- kernel size
        sigma   [float!]        ; -- variance
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvConvolve

Convolves an image with the kernel

```
rcvConvolve: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        kernel  [block!]        ; -- kernel matrix as block
        factor  [float!]        ; -- multiplier factor as float
        delta    [float!]       ; -- 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: routine [
        src     [vector!]       ; -- source matrix
```

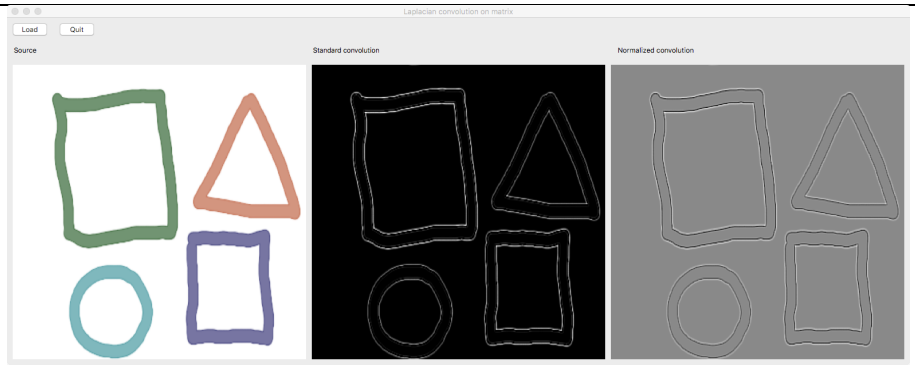
```
        dst     [vector!]       ; -- destination matrix
        mSize   [pair!]         ; -- source matrix size
        kernel  [block!]        ; -- kernel matrix as block
        factor  [float!]        ; -- multiplier factor as float
        delta   [float!]        ; -- bias for image brightness
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvConvolveNormalizedMat

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

```
rcvConvolveNormalizedMat: routine [
        src     [vector!]       ; -- source matrix
        dst     [vector!]       ; -- destination matrix
        mSize   [pair!]         ; -- source matrix size
        kernel  [block!]        ; -- kernel matrix as block
        factor  [float!]        ; -- multiplier factor as float
        delta   [float!]        ; -- bias for image brightness
]
```
Defined in /libs/imgproc/rcvImgProc.re

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: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        channel         [integer!]      ; -- image channel to process (RGB
        kernel          [block!]        ; -- kernel matrix as block
```

```
        factor          [float!]        ; -- multiplier factor as float
        delta           [float!]        ; -- bias for image brightness
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvFilter2D

Basic convolution filter

```
rcvFilter2D: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        kernel  [block!]        ; -- kernel matrix as block
        factor  [float!]        ; -- multiplier factor as float
        delta    [float!]       ; -- 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: routine [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        kernel  [block!]        ; -- 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

rcvDoGFilter

Difference of Gaussian

```
rcvDoGFilter: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        kSize   [pair!]         ; -- kernel size
        sig1    [float!]        ; -- variance 1
        sig2    [float!]        ; -- variance 2
        factor  [float!]        ; -- normalization
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvKuwahara

Kuwahara non-linear smoothing filter

```
rcvKuwahara: routine  [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        kSize   [pair!]         ; -- kernel size
]
```

Defined in /libs/imgproc/rcvImgProc.red

The Kuwahara filter combines noise reduction (blurring) with edge preservation.

rcvNLFilter

Non linear conservative filter for images

```
rcvNLFilter: function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
        kSize   [pair!]         ; -- kernel size
]
```

Defined in /libs/imgproc/rcvImgProc.red

rcvBinomialFilter

Binomial filter

```
rcvBinomialFilter: function [
        src     [image! vector!]        ; -- source image or matrix
        dst     [image! vector!]        ; -- destination image or matrix
        iSize   [pair!]                 ; -- source size as a pair
        f       [float!]                ; -- float value for filter thresholding
]
```

Defined in /libs/imgproc/rcvImgProc.red

rcvLowPass

This filter produces a simple average of the 9 nearest neighbors of each pixel in the image

```
rcvLowPass: function [
        src     [image! vector!]        ; -- source image or matrix
        dst     [image! vector!]        ; -- destination image or matrix
        iSize   [pair!]                 ; -- source size as a pair
        f       [float!]                ; -- float value for filter thresholding
]
```

Defined in /libs/imgproc/rcvImgProc.red

cvBinomialLowPass

Weights are formed from the coefficients of the binomial series

```
cvBinomialLowPass: function [
```

```
        src     [image! vector!]        ; -- source image or matrix
        dst     [image! vector!]        ; -- destination image or matrix
        iSize   [pair!]                 ; -- source size as a pair
        f       [float!]                ; -- float value for filter thresholding
]
```
Defined in /libs/imgproc/rcvImgProc.red

Uniform Weight Convolutions: Blurring is typical of low pass filters

rcvSharpen

Image or matrix sharpening

```
rcvSharpen: function [
        src     [image! vector!]        ; -- source image or matrix
        dst     [image! vector!]        ; -- destination image or matrix
        iSize   [pair!]                 ; -- source size as a pair
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvHighPass

This filter produces a simple average of the 9 nearest neighbors of each pixel in the image

```
rcvHighPass: function [
        src     [image! vector!]        ; -- source image or matrix
        dst     [image! vector!]        ; -- destination image or matrix
        iSize   [pair!]                 ; -- source size as a pair
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvHighPass2

This filter removes low pass values from original image by substraction

```
rcvHighPass2: function [
        src     [image! vector!]        ; -- source image or matrix
        dst     [image! vector!]        ; -- destination image or matrix
        iSize   [pair!]                 ; -- source size as a pair
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvBinomialHighPass

Non-Uniform (Binomial) Weight Convolution

```
rcvBinomialHighPass: function [
        src     [image! vector!]        ; -- source image or matrix
        dst     [image! vector!]        ; -- destination image or matrix
        iSize   [pair!]                 ; -- source size as a pair
```

]

Defined in /libs/imgproc/rcvImgProc.red

High Pass Filters show the edges in the image

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 pre-defined filters which are fast and easy to use. There are two components in derivative filters. The x derivative extracts vertical edges and the y derivative extracts horizontal edges.

Edges routines

rcvMagnitude

Gets Gradient G= Sqrt Gx^2 +Gy^2

```
rcvMagnitude: routine [
        srcX    [image!]        ; -- first image with X components
        srcY    [image!]        ; -- second image with Y components
        dst     [image!]        ; -- result image
]
```

Defined in /libs/imgproc/rcvImgProc.red

rcvDirection

atan Gradient y / Gradient x -> angle in degrees

```
RcvDirection: routine [
        srcX    [image!]        ; -- first image with X components
        srcY    [image!]        ; -- second image with Y components
        dst     [image!]        ; -- result image
]
```

Defined in /libs/imgproc/rcvImgProc.red

rcvProduct

Gradient x*Gradient y product

```
rcvProduct: routine [
        srcX    [image!]        ; -- first image with X components
        srcY    [image!]        ; -- second image with Y components
        dst     [image!]        ; -- result image
]
```

Defined in /libs/imgproc/rcvImgProc.red

First derivative filters

rcvSobelMat

Fast Sobel on Matrix

```
rcvSobelMat: routine [
    src         [vector!]       ; -- source matrix
    dst         [vector!]       ; -- destination matrix
    mSize       [pair!]         ; -- source matrix size
]
```

Defined in /libs/imgproc/rcvImgProc.red

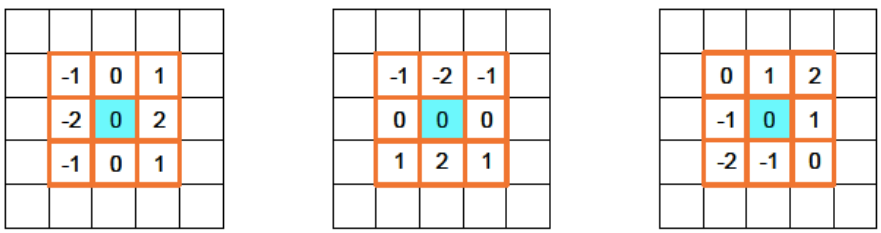
rcvSobel

Direct Sobel edges detection for image or matrix

```
rcvSobel: function [
        src             [image! vector!]        ; -- source image or matrix
        dst             [image! vector!]        ; -- destination matrix or image
        iSize           [pair!]                 ; -- source matrix size
        direction       [integer!]              ; -- direction
        op              [integer!]              ; -- kernel inversion
]
```

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
```

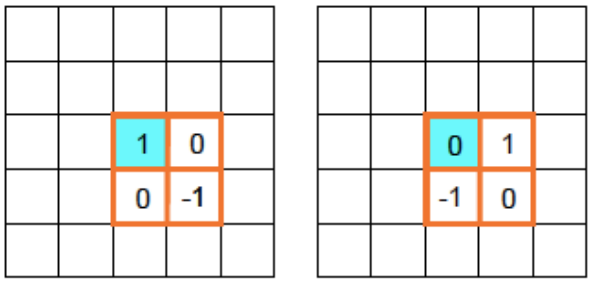
rcvRoberts

Robert's cross edges detection for image or matrix

```
rcvRoberts: function [
        src             [image! vector!]        ; -- source image or vector
        dst             [image! vector!]        ; -- destination image or vector
        iSize           [pair!]                 ; -- source size
        direction       [integer!]              ; -- direction
]
```

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



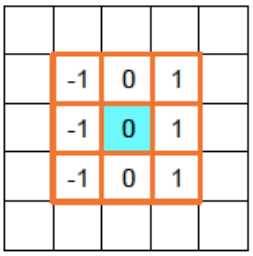
rcvPrewitt

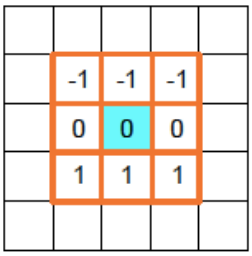
Computes an approximation of the gradient magnitude of the input image

```
rcvPrewitt: function [
        src             [image! vector!]        ; -- source image or vector
        dst             [image! vector!]        ; -- destination image or vector
        iSize           [pair!]                 ; -- source size
        direction       [integer!]              ; -- direction
        op              [integer!]              ; -- kernel inversion
]
```

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

```
rcvMDIF: function [
        src             [image! vector!]        ; -- source image or matrix
        dst             [image! vector!]        ; -- destination image or matrix
        iSize           [pair!]                 ; -- source size as pair
        direction       [integer!]              ; -- gradient direction
]
```

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)
Defined in /libs/imgproc/rcvImgProc.red

rcvGradientMasks

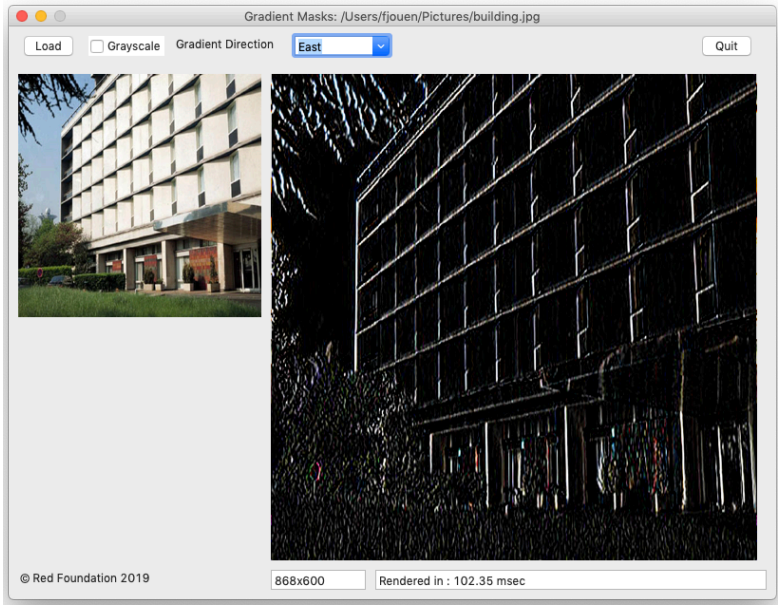
Fast gradient mask filter

```
rcvGradientMasks: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        direction       [integer!]      ; -- gradient direction (8)
]
```

Defined in /libs/imgproc/rcvImgProc.red

Very efficient use of rcvGradientMasks Filter for line detection.

1: North 2: Northeast 3: East 4: Southeast 5: South 6: Southwest 7: West 8: Northwest



rcvKirsch

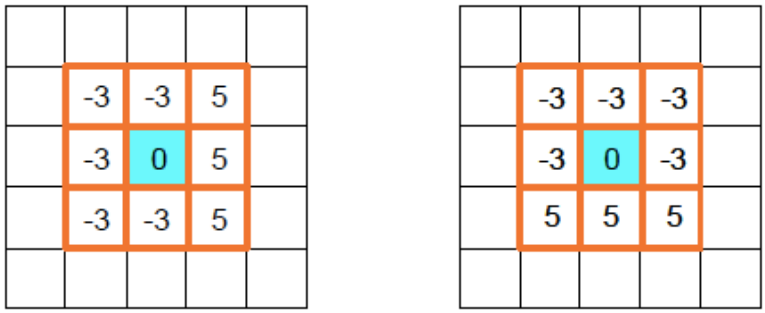
Computes an approximation of the gradient magnitude of the input image

```
rcvKirsch: function [
        src             [image! vector!]        ; -- source image or matrix
        dst             [image! vector!]        ; -- destination image or matrix
```

```
        iSize           [pair!]         ; -- source size as pair
        direction       [integer!]      ; -- gradient direction
        op              [integer!]      ; -- kernel order
]
```

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] or for kernel combination [3,4]
Defined in /libs/imgproc/rcvImgProc.red

Used Hx and Hy kernels



rcvNeumann

Computes the discrete gradient by forward finite differences

```
rcvNeumann: routine [
    src         [image!]        ; -- source image
    dst1        [image!]        ; -- image for derivative along the x axis
    dst2        [image!]        ; -- image derivative along the y axis
    op          [integer!]      ; -- forward or backward computation
]
```

op :
1: Computes the discrete gradient by forward finite differences
2: Computes the divergence by backward finite differences
Defined in /libs/imgproc/rcvImgProc.red

rcvGradNeumann

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

```
rcvGradNeumann: function [
   src          [image!]          ; -- source image
   dst1         [image!]          ; -- image for derivative along the x axis
   dst2         [image!]          ; -- image derivative along the y axis
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvDivNeumann

Computes the divergence by backward finite differences

```
rcvDivNeumann: function [
   src          [image!]          ; -- source image
   dst1         [image!]          ; -- image for derivative along the x axis
   dst2         [image!]          ; -- image derivative along the y axis
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvRobinson:

Robinson Filter

```
rcvRobinson:  function [
        src     [image!]        ; -- source image
        dst     [image!]        ; -- destination image
]
```
Defined in /libs/imgproc/rcvImgProc.red

These edge detection filters are also called compass masks since they are defined by taking a single mask and rotating it to the eight major compass orientations: North, Northwest, West, Southwest, South, Southeast, East, and Northeast

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: $(\frac{\partial^2 I(x,y)}{\partial y^2})$

Edge points can be detected by finding the zero-crossings of the second derivative.

rcvDerivative2

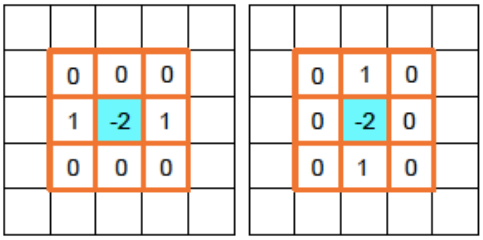
A fast approximation of the second derivative of an image

```
rcvDerivative2: function [
```

```
        src             [image! vector!]    ; -- source image or matrix
        dst             [image! vector!]    ; -- destination image or matrix
        iSize           [pair!]             ; -- source size as pair
        factor          [float!]            ; -- multiplier factor for convolution
        direction       [integer!]          ; -- gradient direction
]
```

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



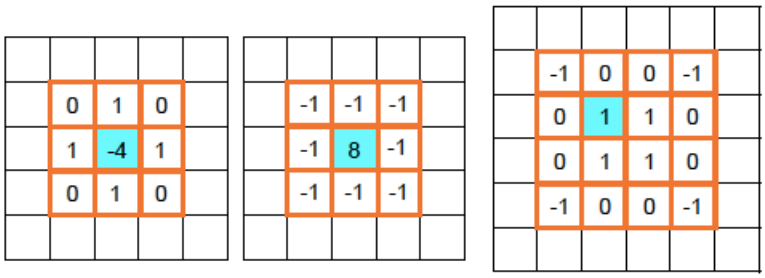
rcvLaplacian

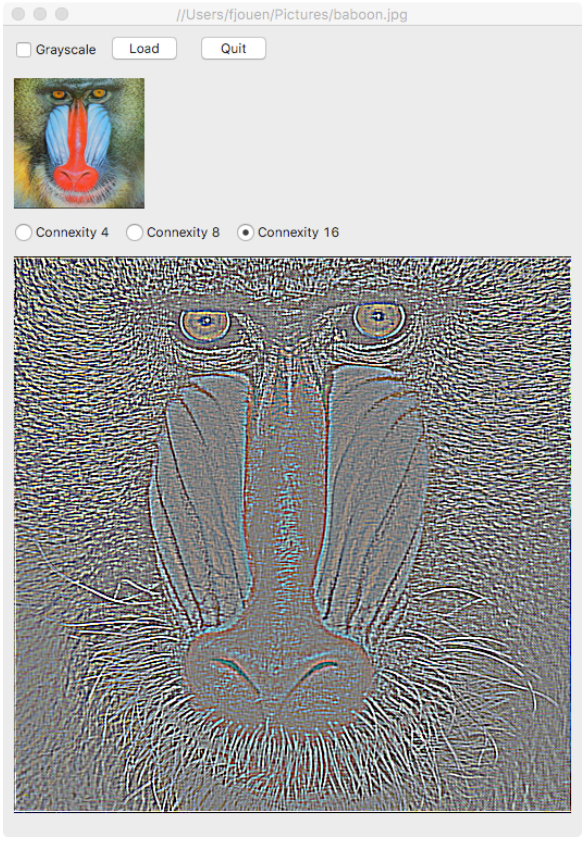
Computes the Laplacian of an image or matrix. The Laplacian is an approximation of the second derivative of an image

```
rcvLaplacian: function [
        src             [image! vector!]    ; -- source image or matrix
        dst             [image! vector!]    ; -- destination image or matrix
        iSize           [pair!]             ; -- source size as pair
        connexity       [integer!]          ; -- neighbor pixels (4, 8 16)
]
```

Defined in /libs/imgproc/rcvImgProc.red

Used kernels for 4, 8 or 16 connexity





rcvDiscreteLaplacian

Discrete Laplacian Filter

```
rcvDiscreteLaplacian: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
]
```

Defined in /libs/imgproc/rcvImgProc.red

rcvLaplacianOfRobinson

Laplacian of Robinson Filter

```
rcvLaplacianOfRobinson: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvLaplacianOfGaussian

Laplacian of Gaussian

```
rcvLaplacianOfGaussian: function [
        src             [image! vector!]        ; -- source image or matrix
        dst             [image! vector!]        ; -- destination image or matrix
        iSize           [pair!]                 ; -- source size as pair
        op              [integer!]              ; -- for 2 different kernels
]
```
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: routine [
        srcX    [image!]        ; -- X Sobel derivative image
        srcY    [image!]        ; -- Y Sobel derivative image
        mat     [vector!]       ; -- result gradient (float matrix)
]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvEdgesDirection

Angles in degrees with atan2 function

```
rcvEdgesDirection: routine [
        srcX    [image!]        ; -- X Sobel derivative image
        srcY    [image!]        ; -- Y Sobel derivative image
        matA    [vector!]       ; -- result angle float matrix
        ]
```
Defined in /libs/imgproc/rcvImgProc.red

rcvEdgesSuppress

Non-maximum suppression

```
rcvEdgesDirection: routine [
        matA    [vector!]       ; -- angle float matrix
        matG    [vector!]       ; -- gradient float matrix
        matS    [vector!]       ; -- result float matrix
        mSize   [pair!]         ; -- matrices size
]
```

Defined in /libs/imgproc/rcvImgProc.red

rcvDoubleThresh

Double thresholding

```
rcvDoubleThresh: routine [
        gradS           [vector!]       ; -- non-maximum suppression matrix
        doubleT         [vector!]       ; -- result integer matrix
        lowT            [integer!]      ; -- low threshold value
        highT           [integer!]      ; -- high threshold value
        lowV            [integer!]      ; -- low value associated to low threshold
        highV           [integer!]      ; -- high value associated to high threshold
]
```

Defined in /libs/imgproc/rcvImgProc.red

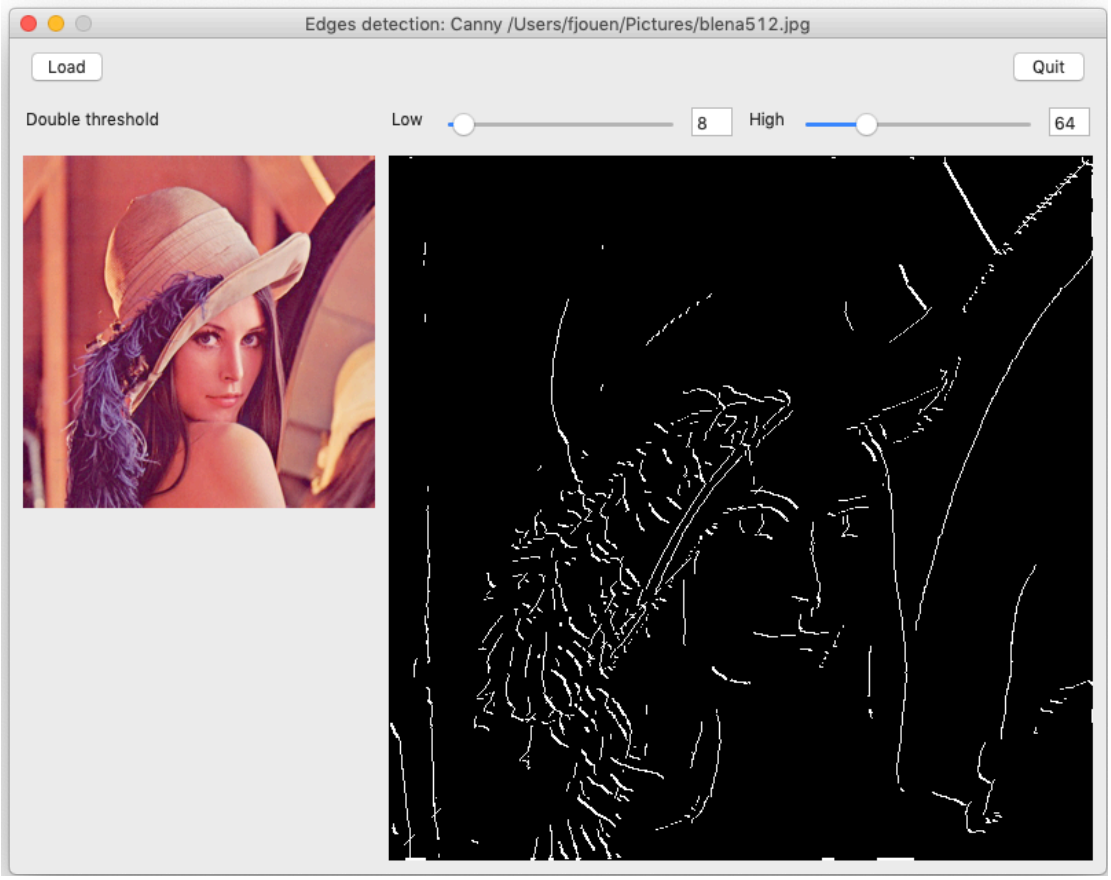
```
if v < lowT [_setIntValue as integer! mDTValue 0 unit2]
f all [v >= lowT v <= highT] [_setIntValue as integer! mDTValue weak unit2]
if v >= highT [_setIntValue as integer! mDTValue strong unit2]
```

rcvHysteresis

non-maximum suppression to thin out the edges

```
rcvHysteresis: routine [
        doubleT         [vector!]       ; -- integer matrix generated by rcvDoubleThresh
        edges           [vector!]       ; -- result integer matrix
        iSize           [pair!]         ; -- matrices size
        weak            [integer!]      ; -- low value associated to low threshold
        strong          [integer!]      ; -- high value associated to high threshold
]
```

Defined in /libs/imgproc/rcvImgProc.red


Edges detection: Canny /Users/fjouen/Pictures/blena512.jpg
Load
Quit
Double threshold
Low
8
High
64

Lines and points 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!]    ; -- source image width
        h [integer!]    ; -- source image height
]
```

Defined in /libs/imgproc/rcvHough.red

rcvGetAccumulatorSize

Gets Hough space accumulator size as pair

```
rcvGetAccumulatorSize: function [
        acc [vector!]   ; -- matrix
]
```

Defined in /libs/imgproc/ rcvHough.red

rcvHoughTransform

Makes Hough transform

```
rcvHoughTransform: routine [
        mat             [vector!]       ; -- image as matrix
        accu            [vector!]       ; --Hough accumulator
        w               [integer!]      ; -- matrix width
        h               [integer!]      ; -- matrix height
        treshold        [integer!]      ; -- thresholding value (e.g. 128)
]
```

Defined in /libs/imgproc/ rcvHough.red

rcvGetHoughLines

Gets lines in the accumulator according to threshold

```
rcvGetHoughLines: routine [
        accu            [vector!]       ; --Hough accumulator
        img             [image!]        ; -- Red image
        threshold       [integer!]      ; -- thresholding value
        lines           [block!]        ; -- results in a block
]
```

Defined in /libs/imgproc/rcvImgProc.red

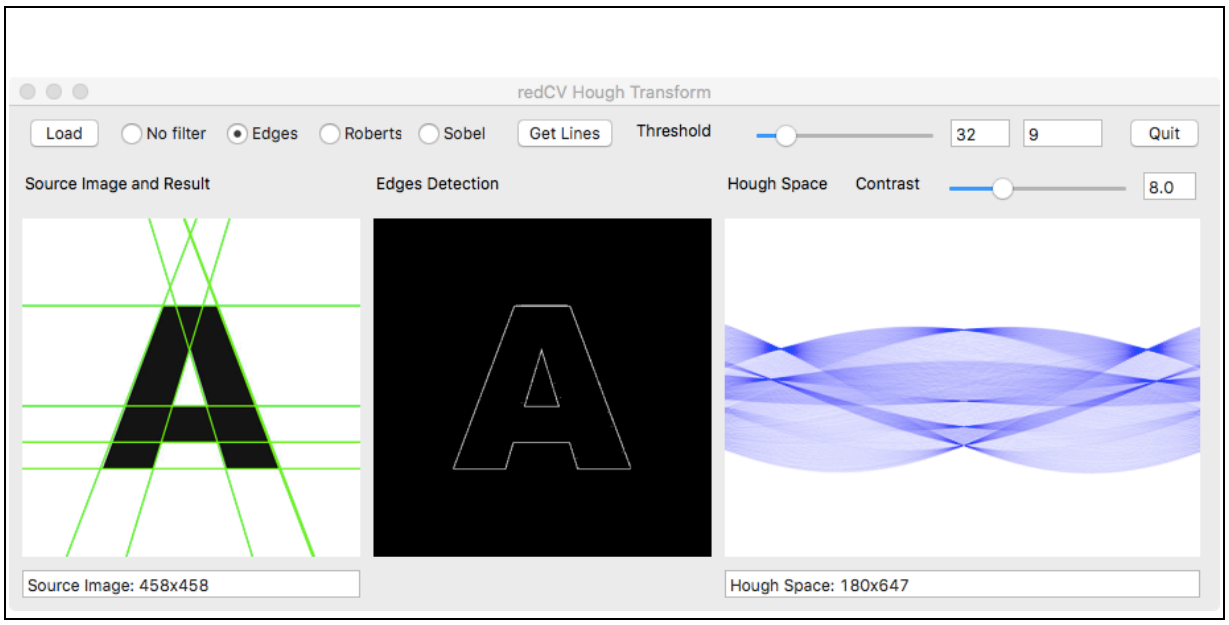
rcvHough2Image

Makes Hough space as red image

```
rcvHough2Image: routine [
        mat             [vector!]       ; -- Hough space matrix
        dst             [image!]        ; -- destination image
        contrast        [float!]        ; -- for image contrast
]
```

Defined in /libs/imgproc/ rcvHough.red

```
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
```



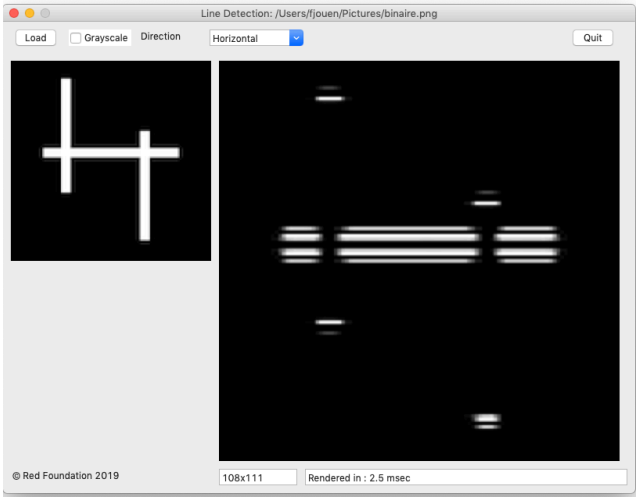
rcvLineDetection

Fast line detection

```
rcvLineDetection: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        direction       [integer!]      ; -- gradient orientation
]
```

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

Defined in /libs/imgproc/rcvImgProc.red



rcvHarris

Harris corner detection

```
rcvHarris: routine [
        srcX    [image!]        ; -- first image with X components
        srcY    [image!]        ; -- second image with Y components
        dst     [image!]        ; -- destination image
        k       [float!]        ; -- constant (0.04 ... 0.15)
        t       [integer!]      ; -- threshold
]
```

Defined in /libs/imgproc/rcvImgProc.red

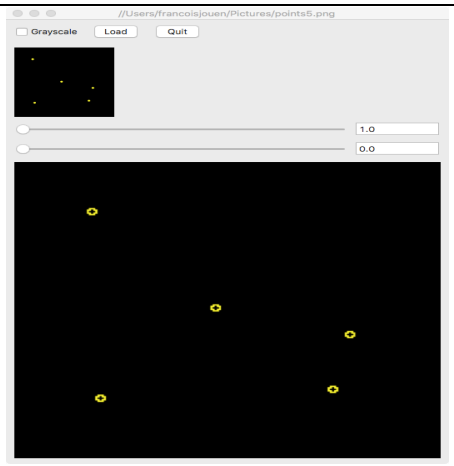
rcvPointDetector

Convolution allowing to find dots in image or matrix

```
rcvPointDetector: function [
```

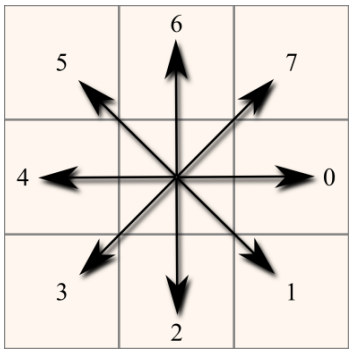
```
        src             [image! vector!]    ; -- source image or matrix
        dst             [image! vector!]    ; -- destination image or matrix
        param1          [float!]            ; -- threshold value
        param2          [float!]            ; -- luminance value
]
```

Defined in /libs/imgproc/rcvImgProc.red



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.

rcvBorderPixel

test if pixel belongs to the shape

```
rcvBorderPixel: routine [
        mat             [vector!]       ; -- source binary matrix
        matSize         [pair!]         ; -- matrix size as pair
        x               [integer!]      ; -- pixel x coordinate
        y               [integer!]      ; -- pixel y coordinate
        value           [integer!]      ; -- pixel value (default 1)
        return:         [logic!]
]
```

Defined in /libs/imgproc/rcvFreeman.red

rcvMatGetBorder

Gets pixels that belong to shape border

```
rcvMatGetBorder: routine [
        mat             [vector!]       ; -- source binary matrix
        matSize         [pair!]         ; -- matrix size as pair value [integer!]
```

```
        value           [integer!]      ; -- pixel value (default 1)
        border          [block!]        ; -- a block to store pixels direction
]
```

Defined in /libs/imgproc/rcvFreeman.red

rcvBorderNeighbors

Gets next contour pixel direction

```
rcvBorderNeighbors: routine [
        mat             [vector!]       ; -- source binary matrix
        matSize         [pair!]         ; -- matrix size as pair
        x               [integer!]      ; -- pixel x coordinate
        y               [integer!]      ; -- pixel y coordinate
        value           [integer!]      ; -- pixel value (default 1)
        return:         [integer!]
]
return value gives the direction
```

Defined in /libs/imgproc/rcvFreeman.red

rcvMatGetChainCode

Gets Freeman Chain code (integer)

```
rcvMatGetChainCode: routine [
        mat             [vector!]       ; -- source binary matrix
        matSize         [pair!]         ; -- matrix size as paircoord [pair!]
        coord           [pair!]         ; -- pixel x and y coordinates as pair
        value           [integer!]      ; -- pixel value (default 1)
]
```

Defined in /libs/matrix/rcvMatrix.red

rcvGetContours

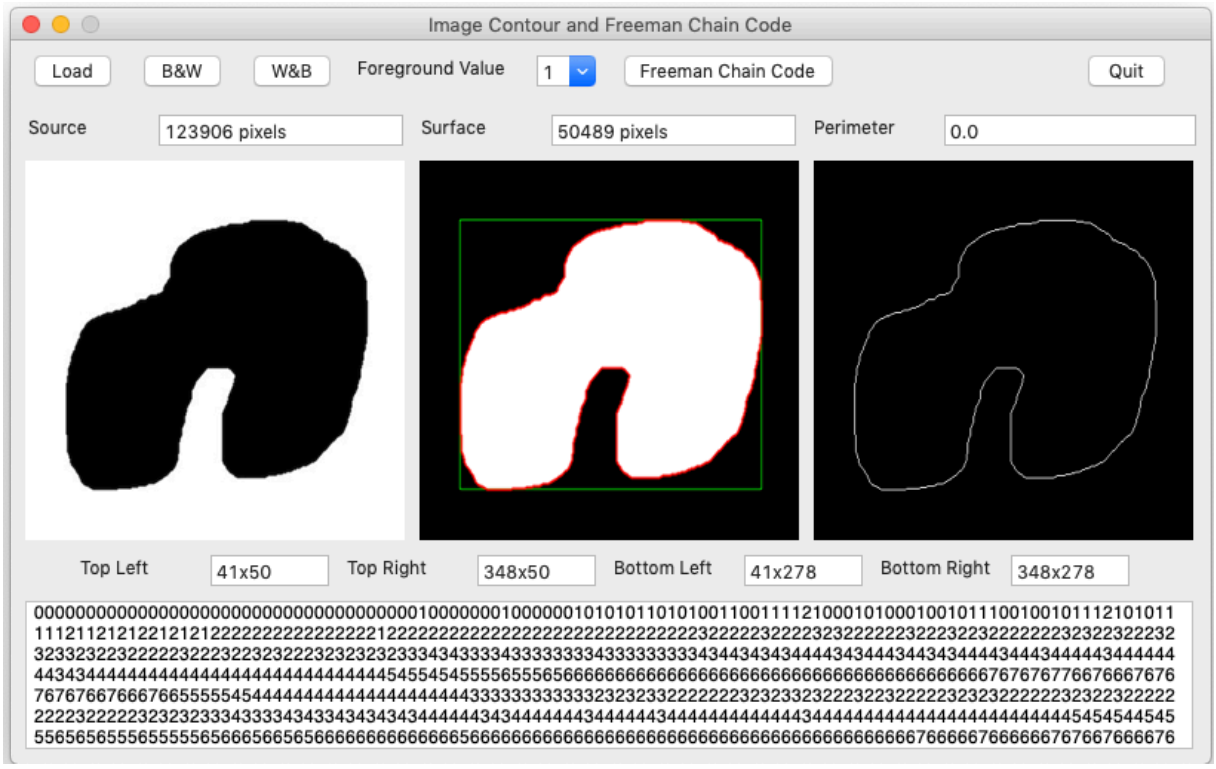
Gets next contour pixel to process

```
rcvGetContours: routine [
        p               [pair!]         ; -- current pixel coordinates
        d               [integer!]      ; -- next pixel direction
        return:         [pair!]
]
```

Defined in /libs/matrix/rcvMatrix.red

These functions use 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

Object detection

redCV supports Haar cascade algorithm for object detection in image. redCV cascades files are based on OpenCV xml files, but file size is reduced by a 4 factor. Face1.txt is a special file with integer values for faster computation. Other cascade files use float values. Titled features are supported. In original Viola and Jones stump-based cascades, each tree has 3 nodes at max. Each node has no leaf and left and right values are used for decision. Features can be tilted. In tree-based improved version (Lienhart and Maydt), nodes can have left and/or right leaves which give an indication where to go for the decision (see redCV_Samples.pdf for details).

rcvCreateArrayPointers

Creates integer pointers that give access to Red arrays by routines

rcvCreateArrayPointers: routine []

For a fast computation we use a lot of arrays (blocks of vector!) and each array is associated to a pointer used as parameter for Red/System routines.

rcvReadTextClassifier

Process classifier file and return number of stages, total number of nodes and original win size

```
rcvReadTextClassifier: func [
        f               [file!]         ;--classifier text file
        nParameters     [integer!]      ;--number of parameters (default 23)
        return:         [block!]        ;--returned values
]
```

All information in classifier file

[Header] Section
how many stages are in the cascaded filter?
the second line of classifierFile is the number of stages
how many filters in each stage?
They are specified in classifier file, starting from third line

Filters are defined in [Nodes] section
First line: window training size
then each stage of the cascaded filter has:
23 parameters per filter
+ 1 threshold parameter per stage
The 23 parameters for each filter are:
1 to 4: coordinates of rectangle 1
5: weight of rectangle 1
6 to 9: coordinates of rectangle 2
10: weight of rectangle 2
11 to 14: coordinates of rectangle 3 (default 0)
15: weight of rectangle 3 (default 0.0)
16: tilted flag
17: threshold of the filter
18: alpha 1 of the filter (node left value)
19: alpha 2 of the filter (node right value)
20: has left node?
21: left node value; 0, 1 or 2
22: has right node?
23: right node value; 0, 1 or 2

rcvCreateHaarCascade

Creates Haar cascade structure

```
rcvCreateHaarCascade: routine [
        nStages         [integer!]      :--Number of stages
        nNodes          [integer!]      ;--Number of filters by stage
        scale           [float!]        ;--default scale factor > 1.0
        wSize           [pair!]         ;--Size of the window used for training
        return:         [integer!]
]
```

returned value is the memory address of the structure

rcvNearestNeighbor

Downsamples an image using nearest neighbor

```
rcvNearestNeighbor: routine [
        src             [image!]
        dst             [image!]
]
```

You can also use rczResizeImage routine.

rcvHaarIntegralImage1

Computes summed-area table and square summed-area table

```
rcvHaarIntegralImage1: routine [
    src                 [image!]        ;-- Source image
    dst1                [vector!]       ;--Sum area table
    dst2                [vector!]       ;--Square sum area table
]
```

Similar to rcvIntegralImg in redCV lib rcvIntegral.red.
Source image is automatically converted to grayscale image and we use fixed-point gray-scale transform, close to openCV transform.

rcvHaarIntegralImage2

Computes summed-area table, square summed-area table and rotated summed-area table

```
rcvHaarIntegralImage2: routine [
"Compute summed-area table, square summed-area table and rotated summed-area table"
    src                 [image!]        ;--Source image
    dst0                [vector!]       ;--For gray scale image
    dst1                [vector!]       ;-- Sum area table
    dst2                [vector!]       ;--Square sum area table
    dst3                [vector!]       ;--Tilted sum area table.
]
```

Similar to rcvIntegralImg in redCV lib rcvIntegral.red.
Source image is automatically converted to grayscale image and we use fixed-point gray-scale transform, close to openCV transform. But we calculate 3 integral images for Lienhart et al. extension using tilted features.

rcvCannyFilter

Canny filtering for faster object detection

```
rcvCannyFilter: routine [
        src             [image!]        ;--Source image
        dst             [image!]        ;--Destination image (Canny filtered)
        gray            [vector!]       ;--Gray scale transform
        lowPass         [vector!]       ;--Gaussian low pass
        canny           [vector!]       ;--Edges detection
]
```

rcvSetImageForCascadeClassifier

Sets images for haar classifier cascade

```
rcvSetImageForCascadeClassifier: routine []
```

This routine loads the four corners of reactangles, but does not do computation based on 4 corners. Rectangles values are used to make scaled rectangles from integral image values. Computation is done next in ScaleImageInvoker routine.

rcvEvalWeakClassifier

Computes each weak classifier value

```
rcvEvalWeakClassifier: routine [
"Compute each weak classifier value"
        variance            [float!]        ;--Normalized variance of the image
        pOffset             [integer!]      ;--coordinates offset
        treeIndex           [integer!]      ;--Node index
        wIndex              [integer!]      ;--Weight index
        rIndex              [integer!]      ;--Rectangle index
        return:             [float!]        ;--value of the classifier
]
```

rcvRunCascadeClassifier

Executes classifier cascade

```
rcvRunCascadeClassifier: routine [
"Execute classifier cascade"
        pt                  [pair!]         ;--Sliding window coordinates
        startStage          [integer!]      ;--First stage number
        threshold           [float!]        ;--Stage threshold
        return:             [integer!]      ;--for decision
]
```

rcvScaleImageInvoker

Searches for objects in image

```
rcvScaleImageInvoker: routine [
        factor              [float!] ;--initial scaling [1.0]
        step                [integer!]      ;--Window sliding  step
        sSize               [pair!]         ;--Integral image size
        p                   [pair!]         ;--Current pixel
        rect                [vector!]       ;--For candidates identified objects
        maxCandidates       [integer!]      ;--Max candidates number
        threshold           [float!]        ;--used by rcvRunCascadeClassifier [1.1]
]
```

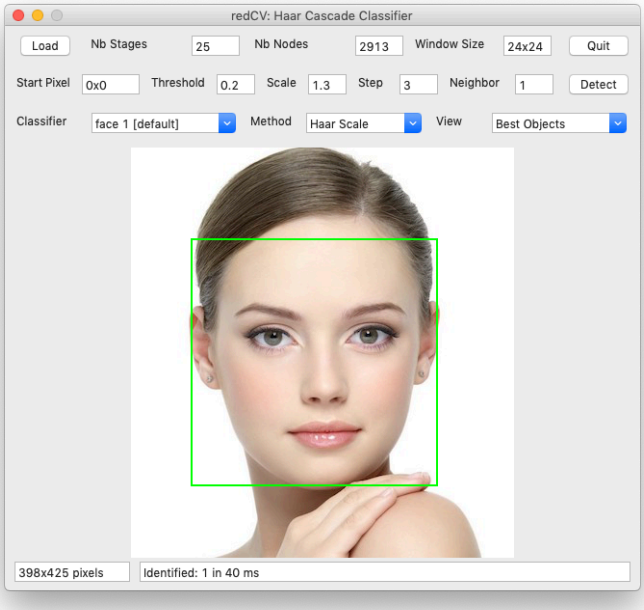
Each stage of the classifier labels the region defined by the current location of the sliding window as either positive or negative. Positive indicates that an object was found and negative indicates no objects were found. If the label is negative, the classification of this region is complete, and the detector slides the window to the next location. If the label is positive, the classifier passes the region to the next stage. The detector reports an object found at the current window location when the final stage classifies the region as positive.

rcvDetectObjects

Makes all job: Process image and find objects

```
rcvDetectObjects: func [
        img             [image!]    ;--red image
        startPos        [pair!]     ;--0x0
        scaleFactor     [float!]    ;--1.2
        step            [integer!]  ;--1
        stageThreshold  [float!]    ;--0.5
        minNeighbors    [integer!]  ;--1
        maxCandidates   [integer!]  ;-- 512
        grouping        [logic!]    ;--true
        method          [integer!]  ;--0 no Canny 1 Canny pruning
        return:         [vector!]   ;--found objects
]
```

This function allows a fine tuning for objects detection with a lot of parameters. Image source can be any supported Red image. RGBA images are automatically converted to gray scale images for faster computation. By default, we process the whole image and the starting position for sliding is 0x0. Scale factor is used for image downsizing and is always > 1.0. Increasing scale factor value accelerates the detection of objects in image. Step parameter can also used for faster detection. By default (1), sliding window is moved pixel by pixel in both X and Y directions. If your image is simple, with only some objects to be detected, you can increase step value for a faster processing. StageThreshold parameter is crucial for a correct detection. This parameter is always > to 0.0. With weak values, you'll get an accurate detection, but with the risk of false-positive detection. With high values, you'll can loss some objects. This parameter depends on the complexity of your image. minNeighbors is used for objects grouping and distance computation between candidates. For avoid time consuming object detection, you can limitate the number of candidates before rectangles grouping. Grouping parameter is used for candidates post-processing. If false, all identified candidates are kept. In other case, candidates are clustered to find the best candidates for object detection. Lastly, the method parameter allows to use a Canny-like preprocessing for faster identification.



rcvPredicate

Rectangles clustering

```
rcvPredicate: func [
        eps             [float!]        ;--distance threshold
        r1              [vector!]       ;--first rectangle
        r2              [vector!]       ;--second one
        return: [logic!]
]
```

rcvPartition

Returns the number of classes

```
rcvPartition: func [
        array   [block!]        ;--array of rectangles as vector!
        labels  [block!]        ;--classes label
        eps      [float!]       ;--threshold value
]
```

rcvGroupRectangles

Groups rectangles by classes

```
rcvGroupRectangles: func [
        array                   [block!]        ;--array of rectangles as vector!
        labels                  [block!]        ;-- for classes labels
        groupThreshold          [integer!]      ;--threshold value for group
        eps                     [float!]        ;--threshold value
]
```

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!]         ; -- kernel size (e.g. 3x3)
        /rectangle /cross/vline/hline           ; -- refinements
]
```

Refinement is used to create a cross-shaped element, a rectangular element, a vertical or an horizontal line element

Defined in /libs/imgproc/rcvMorphology.red

rcvErode

Erodes image by using structuring element

```
rcvErode: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
]
```

kernel: you can also use any customed structuring element

Defined in /libs/imgproc/rcvMorphology.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:

```
dst=erode(src,element):  dst(x,y)=min((x',y') in element)src(x+x',y+y')
```

rcvErodeMat:

Erodes matrice by using structuring element

```
rcvErodeMat: function [
        src             [vector!]       ; -- source matrix
        dst             [vector!]       ; -- destination matrix
        mSize           [pair!]         ; -- matrices size
        kSize           [pair!]         ; -- kernel size as pair
```

```
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
]
```

Defined in /libs/imgproc/rcvMorphology.red

rcvDilate

Dilates image by using structuring element

```
rcvDilate: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
]
```

kernel: you can also use any customed structuring element

Defined in /libs/imgproc/rcvMorphology.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:

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

rcvDilateMat

Dilates matrix by using structuring element

```
rcvDilateMat: function [
        src             [vector!]       ; -- source matrix
        dst             [vector!]       ; -- destination matrix
        mSize           [pair!]         ; -- matrices size
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
]
```

Defined in /libs/imgproc/rcvMorphology.red

rcvOpen

Erodes and Dilates image by using structuring element

```
rcvOpen: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
```

]
kernel: you can also use any customed structuring element

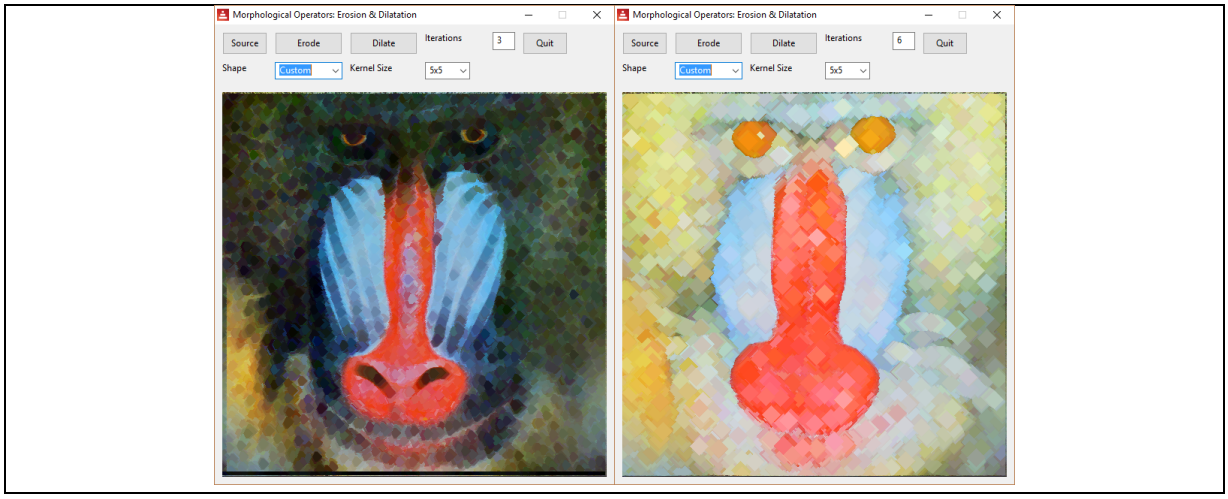
Defined in /libs/imgproc/rcvImgProc.red

rcvClose

Dilates and Erodes image by using structuring element

```
rcvClose: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
]
```
kernel: you can also use any customed structuring element

Defined in /libs/imgproc/rcvMorphology.red



rcvMGradient

Performs advanced morphological transformations using erosion and dilatation as basic operations

```
rcvMGradient: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
        /reverse]                       ; -- reverse order of operations
```
kernel: you can also use any customed structuring element

dst=dilate src – erode src

/reverse: dst=erode src – dilate src

Defined in /libs/imgproc/rcvMorphology.red

rcvTopHat

Performs advanced morphological transformations

```
rcvTopHat: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
]
```

kernel: you can also use any customed structuring element

dst = src – rcvOpen src dst

Defined in /libs/imgproc/rcvMorphology.red

rcvBlackHat

Performs advanced morphological transformations

```
rcvTopHat: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
]
```

kernel: you can also use any customed structuring element
dst = rcvOpen src dst - src
Defined in /libs/imgproc/rcvMorphology.red

rcvMMean

Means image by using structuring element

```
rcvMMean: routine [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
        kernel          [block!]        ; -- block created by cvCreateStructuringElement
]
```

kernel: you can also use any customed structuring element

Defined in /libs/imgproc/rcvMorphology.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: routine [
        src         [image!]    ; -- source image
        dst         [image!]    ; -- destination image
        kSize       [pair!]     ; -- kernel size as pair
        op          [integer!]  ; -- type of mean
]
```

op: parameter for mean computing

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))

Defined in /libs/imgproc/rcvImgProc.red

rcvMedianFiltering

Median Filter routine for images

```
rcvMedianFiltering: routine [
        src         [image!]    ; -- source image
        dst         [image!]    ; -- destination image
        kSize       [pair!]     ; -- kernel size as pair
        kernel      [vector!]   ; -- for storing values
        op          [integer!]  ; -- for creating different median filters
]
```

op is used for creating different median filters as functions
Defined in /libs/imgproc/rcvImgProc.red

rcvMedianFilter

Median Filter for images

```
rcvMedianFilter: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
]
```

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

Defined in /libs/imgproc/rcvImgProc.red

rcvMinFilter

Minimum Filter for images

```
rcvMinFilter: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
]
```

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

Defined in /libs/imgproc/rcvImgProc.red

rcvMaxFilter

Maximum Filter for images

```
rcvMaxFilter: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
]
```

Central pixel value is replaced by the maximum value of neighbors

Defined in /libs/imgproc/rcvImgProc.red

rcvMidPointFilter

Midpoint Filter for images

```
rcvMidPointFilter: function [
        src             [image!]        ; -- source image
        dst             [image!]        ; -- destination image
        kSize           [pair!]         ; -- kernel size as pair
]
```

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

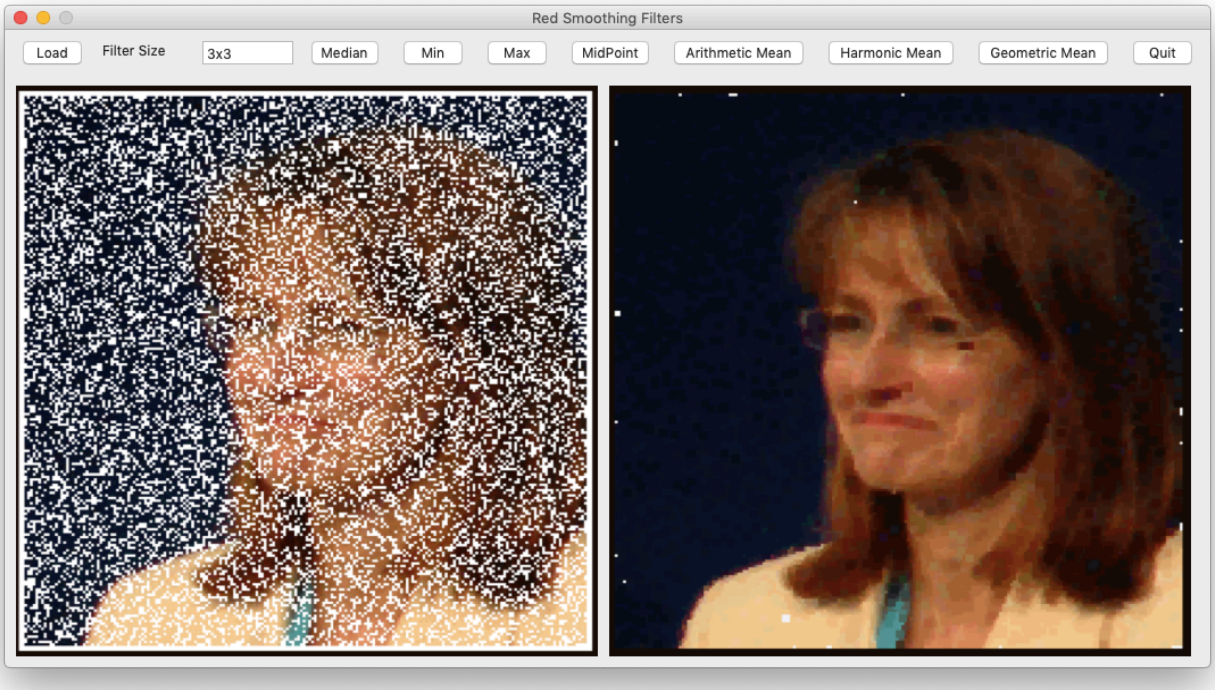
Defined in /libs/imgproc/rcvImgProc.red

rcvMatrixMedianFilter

Median Filter for matrices

```
rcvMatrixMedianFilter: routine [
        src         [vector!]     ; -- source matrix
        dst         [vector!]     ; -- destination matrix
        mSize       [pair!]       ; -- matrices size
        kWidth      [integer!]    ; -- kernel width
        kHeight     [integer!]    ; -- kernel height
        kernel      [vector!]     ; -- kernel for filtering
]
```

Defined in /libs/imgproc/rcvImgProc.red



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

rcvTSStats, rcvTSSDetrend, rcvTSSNormalize and rcvTSMMFiltering
4 internal routines used by the different following functions

rcvTSCopySignal

Makes a copy of original signal

```
rcvTSCopySignal: function [
	signal		[vector!]	; -- 1-D matrix of integer or float values
]
```

Defined in /libs/timeseries/rcvTS.red

rcvTSStatSignal

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

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

Defined in /libs/timeseries/rcvTS.red

rcvTSSDetrendSignal

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

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

Defined in /libs/timeseries/rcvTS.red

rcvTSSNormalizeSignal

Normalize data by replacing each value by a normalized value

```
rcvTSSNormalizeSignal: function [
        signal          [vector!]       ; -- 1-D matrix of integer or float values
        filter          [vector!]       ; -- normalized values (value -mean/sd) in filter matrix
]
```

Defined in /libs/timeseries/rcvTS.red

rcvTSMMFilter

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

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

Defined in /libs/timeseries/rcvTS.red

1-D Savitzky-Golay filters

For faster kernel computations, routines and functions use pre-defined coefficients tables. You'll find these tables in in /libs/timeseries/rcvSGF.red.

rcvSGFiltering

This routine is used to generate the different Savitzky-Golay filters

```
rcvSGFiltering: routine [
        signal          [vector!]       ; -- 1-D matrix of integer or float values
        filter          [vector!]       ; -- to store the result
        kernel          [block!]        ; -- predefined coefficients for faster computation
]
```

Defined in /libs/timeseries/rcvSGF.red

rcvSGFilter

Calculates second order polynomial Savitzky-Golay filter

```
rcvSGFilter: function [
        signal  [vector!]       ; -- 1-D matrix of integer or float values
        filter  [vector!]       ; -- to store the result
        opSG    [integer!]      ; -- allowing cubic, quartic or quintic polynomials filtering
]
```

Defined in /libs/timeseries/rcvSGF.red

rcvSGCubicFilter

Calculates second order polynomial Savitzky-Golay filter

```
rcvSGCubicFilter: function [
        signal          [vector!]       ; -- 1-D matrix of integer or float values
        filter          [vector!]       ; -- to store the result
        opSG            [integer!]      ; -- allowing different cubic kernels for filtering
]
```

Defined in /libs/timeseries/rcvSGF.red

rcvSGQuarticFilter

Calculates second order polynomial Savitzky-Golay filter

```
rcvSGQuarticFilter: function [
        signal          [vector!]       ; -- 1-D matrix of integer or float values
        filter          [vector!]       ; -- to store the result
        opSG            [integer!]      ; -- allowing different quartic kernels for filtering
]
```

Defined in /libs/timeseries/rcvSGF.red

rcvSGDerivative1

Calculates derivative Savitzky-Golay filter

```
rcvSGDerivative1: function [
        signal          [vector!]       ; -- 1-D matrix of integer or float values
        filter          [vector!]       ; -- to store the result
        opSG            [integer!]      ; -- allowing different quadratic kernels for filtering
]
```

Defined in /libs/timeseries/rcvSGF.red

1-D Fast Fourier Transform

Thanks to Mel Cepstrum and Toomas Voglaid for their FFT initial code. redCV code is based on http://paulbourke.net/miscellaneous/

rcvFFT

Calculates forward or inverse FFT

```
rcvFFT: routine [
        re      [vector!]       ; -- the real (x) matrix
        im      [vector!]       ; -- the imaginary (y) matrix
        dir     [integer!]      ; -- forward or backward FFT
]
```

This computes an in-place complex-to-complex FFT
re and im are the real (x) and imaginary (y) arrays of 2^m points.
dir: 1 gives forward transform
dir: -1 gives reverse or backward transform

Defined in /libs/timeseries/rcvFFT.red

rcvFFTAmplitude

FFT amplitude. Only float matrices

```
rcvFFTAmplitude: routine [
        re              [vector!]       ; -- the real (x) matrix
        im              [vector!]       ; -- the imaginary (y) matrix
        return:         [vector!]       ; -- calculated amplitude
]
```

Defined in /libs/timeseries/rcvFFT.red

rcvFFTPhase

FFT phase. Only float matrices

```
rcvFFTPhase: routine [
        re              [vector!]       ; -- the real (x) matrix
        im              [vector!]       ; -- the imaginary (y) matrix
        degree          [logic]         ; -- for gradian or degree computation
        return:         [vector!]       ; -- calculated phase
]
```

Defined in /libs/timeseries/rcvFFT.red

rcvFFTFrequency

Returns the FFT sample frequencies and shifts the DC (zero-frequency component) to the center of the spectrum

```
rcvFFTFrequency: routine [
        n               [integer!!]     ; -- window length
        delta           [float!]        ; -- time step (classically inverse of sampling rate)
        return:         [vector!]       ; -- centered spectrum
]
```

Defined in /libs/timeseries/rcvFFT.red

rcvFFTShift

Shifts to the center of the spectrum

```
rcvFFTShift: routine [

        x               [vector!]       ; -- x is a FFT amplitude matrice
        return:         [vector!]
]
```

Defined in /libs/timeseries/rcvFFT.red

rcvFFTFilter

FFT Low or High Pass Filter

```
rcvFFTFilter: routine [
        x           [vector!]       ; -- x is a FFT amplitude matrice
        radius      [float!]        ; radius value
        op          [integer!]      ; low or high pass filter
        return:     [vector!]
]
```

radius is used to select spatial frequency components that fall within or beyond this point

op is used for low or high pass filter selection

2-D Fast Fourier Transform

A full two-dimensional Fourier transform performs a 1-D transform on every scan-line or row of the image, and another 1-D transform on every column of the image, producing a 2-D Fourier transform of the same size as the original image. There is a very elegant explanation of FFT, by Steve Lehar, here: http://cns-alumni.bu.edu/~slehar/

Attention: for faster computation, 2-D FFT routines use arrays defined as a block of vectors!

rcvFFT2D

Perform a 2D FFT inplace given a complex 2D array

```
rcvFFT2D: routine  [
        re      [vector!]       ; -- the real (x) matrix
        im      [vector!]       ; -- the imaginary (y) matrix
        dir     [integer!]      ; -- forward or backward FFT
]
```

The direction dir, 1 for forward, -1 for reverse

Defined in /libs/timeseries/rcvFFT.red

rcvFFT2DShift

Shifts to the center of the spectrum

```
rcvFFT2DShift: routine [
        x           [vector!]       ; -- x is a FFT amplitude matrice
        return:     [block!]
]
```

Defined in /libs/timeseries/rcvFFT.red

rcvTransposeArray

Makes a rotation of array around the center of the spectrum

```
rcvTransposeArray: routine [
        array           [block!]        ; -- array is a shifted FFT amplitude matrice
        return:         [block!]
]
```

Defined in /libs/timeseries/rcvFFT.red

rcvFFTImage

A simple function for image FFT

```
rcvFFTImage: func [
        src             [image!]        ; -- source image
        return:         [image!]        ; -- FFT image
        /forward /backward              ; -- refinement (backward for inverse FFT)
]
```

Defined in /libs/timeseries/rcvFFT.red

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: routine [
        x       [number!]       ; -- integer or float
        y       [number!]       ; -- integer or float
        z       [number!]       ; -- integer or float
]
```

Defined in /libs/timeseries/rcvDTW.red

rcvDTWDistance

Making a 2d matrix to compute distances between all pairs of x and y series

```
rcvDTWDistance: routine [
        x       [block!]        ; -- x 1-D matrix of integer or float values
        y       [block!]        ; -- y 1-D matrix of integer or float values
        dmat    [vector!]       ; -- to store the distances between x and y series
        op      [integer!]      ; -- integer or float matrices
]
```

op: 0 for integer! matrices op:1 for float! matricies

This routine is called by rcvDistances function

Defined in /libs/timeseries/rcvDTW.red

rcvDTWDistances

Making a 2d matrix to compute distances between all pairs of x and y series

```
rcvDTWDistances: function [
        x               [block!]        ; -- x 1-D matrix of integer or float values
        y               [block!]        ; -- y 1-D matrix of integer or float values
        dMatrix         [vector!]       ; -- to store the distances between x and y series
]
```

Defined in /libs/timeseries/rcvDTW.red

rcvDTWRun

Calculate distance and cost matrices

```
rcvDTWRun: routine [
        w       [integer!]      ; -- matrices x size
        h       [integer!]      ; -- matrices y size
        dMat    [vector!]       ; -- matrice for storing distances
        cMat    [vector!]       ; -- matrice for storing costs
]
```

Used by rcvDTWCosts function

Defined in /libs/timeseries/rcvDTW.red

rcvDTWCosts

Making a 2d matrix to compute minimal distance cost

```
rcvDTWCosts: function [
        x       [block!]        ; -- x 1-D block of integer or float values
        y       [block!]        ; -- y 1-D block of integer or float values
        dMat    [vector!]       ; -- matrice for storing distances
        cMat    [vector!]       ; -- matrice for storing costs
]
```

Defined in /libs/timeseries/rcvDTW.red

rcvDTWGetDTW

Returns DTW value

```
rcvDTWGetDTW: function [
        cMat            [vector!]       ; -- minimal cost matrix
        return:         [number!]       ; -- DTW value
]
```

Defined in /libs/timeseries/rcvDTW.red

rcvDTWPath

Finds the path minimizing the distance

```
rcvDTWPath: routine [
        x       [block!]        ; -- x 1-D block of integer or float values
        y       [block!]        ; -- y 1-D block of integer or float values
        cMat    [vector!]       ; -- cost matrix
        xPath   [block!]        ; -- to store optimal distance path
]
```

Used by rcvDTWGetPath function

Defined in /libs/timeseries/rcvDTW.red

rcvDTWGetPath

Finds the path minimizing the distance

```
rcvDTWGetPath: function [
        x       [block!]        ; -- x 1-D block of integer or float values
        y       [block!]        ; -- y 1-D block of integer or float values
        cMat    [vector!]       ; -- cost matrix
        xPath   [block!]        ; -- to store optimal distance path
]
```

Defined in /libs/timeseries/rcvDTW.red

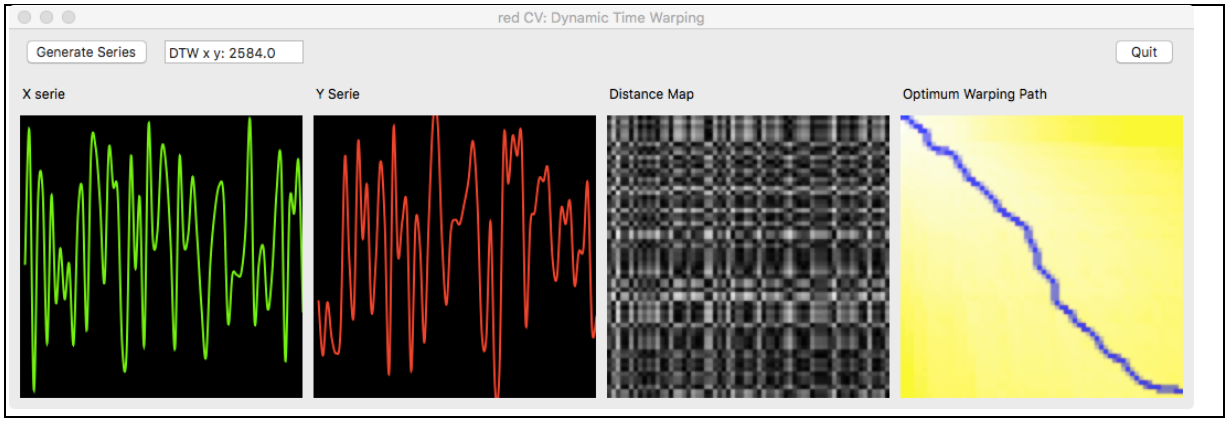
rcvDTWCompute

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

```
rcvDTWCompute: function [
        x       [block!]        ; -- x 1-D block of integer or float values
        y       [block!]        ; -- y 1-D block of integer or float values
        return: [number!]
]
```

Returns DTW value

Defined in /libs/timeseries/rcvDTW.red



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.*

rcvDrawPlot

Draws in window

```
rcvDrawPlot: function [
        window          [face!]
        plot            [block!]
        /clear
]
```

refinement clear can be used to reset the draw block.

rcvNamedWindow

Creates and returns a window

```
rcvNamedWindow: function [
        title           [string!]
]
```

title: windows title as a string
window is returned a face datatype!
Defined in /libs/highgui/rcvHighGui.red

rcvDestroyWindow

Destroys a created window

```
rcvDestroyWindow: function [
        window          [face!]
]
```

window: points to a window created by rcvNamedWindow
Defined in /libs/highgui/rcvHighGui.red

rcvDestroyAllWindows

Destroys all windows

```
rcvDestroyAllWindows: function []
```

Defined in /libs/highgui/rcvHighGui.red

rcvResizeWindow

Sets window size

```
rcvResizeWindow: function [
        window          [face!]
        wSize           [pair!]
]
```

Defined in /libs/highgui/rcvHighGui.red

rcvMoveWindow

Sets window position

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

Defined in /libs/highgui/rcvHighGui.red

rcvShowImage

Shows image in window

```
rcvShowImage: function [
window [face!]
            image     [image!]
]
```

Defined in /libs/highgui/rcvHighGui.red

```
#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[]
Defined in /libs/math/rcvRandom.red

randUnif

Uniform law

```
randUnif: function [
        i [float!]
        j [float!]
]
```
Defined in /libs/math/rcvRandom.red

randExp

Exponential law

randExp: function []
Defined in /libs/math/rcvRandom.red

randExpm

Exponential law with a l degree

```
randExpm: function [
        l [float!]
]
```
l: float value (e.g. 1.0)
Defined in /libs/math/rcvRandom.red

randNorm

Normal law

```
randNorm: function [
```

A [float!]
]
A: float value (e.g. 1.0)
Defined in /libs/math/rcvRandom.red

randLognorm

Lognormal law

randLognorm: function [
A [float!]
B [float!]
z [float!]
]
a: float value
b: float value
z: float value
Defined in /libs/math/rcvRandom.red

randGamma

Gamma law

randGamma: func [
k [integer!]
l [float!] i
]
k: integer value
l: float value
Defined in /libs/math/rcvRandom.red

randDisc

Geometric law in a disc

randDisc: function []
Defined in /libs/math/rcvRandom.red

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
Defined in /libs/math/rcvRandom.red

randChi2

Chi square law

randChi2: function [
d [integer!]
]
v: integer value (e.g. 2)
Defined in /libs/math/rcvRandom.red

randErlang
Erlang law

randErlang: function [
n [integer!]
]
n: integer value (e.g. 2)
Defined in /libs/math/rcvRandom.red

randStudent

Student law

randStudent: function [
n [integer!]
z [float!]
]
n: integer value (e.g. 3)
z: float value (e.g. 1.0)
Defined in /libs/math/rcvRandom.red

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)
Defined in /libs/math/rcvRandom.red

randLaplace

Laplace law

```
randLaplace: function [
        a [float!]
]
```
a: float value (e.g. 1.0)
Defined in /libs/math/rcvRandom.red

randBeta

Beta law

```
randBeta: function [
        a [integer!]
        b [integer!]
]
```
a: integer value (e.g. 1)
b: integer value (e.g. 1)
Defined in /libs/math/rcvRandom.red

randWeibull

Weibull law

```
randWeibull: function [
        a [float!]
        l [float!]
]
```
a: float value (e.g. 1.0)
l: float value (e.g. 1.0)
Defined in /libs/math/rcvRandom.red

randRayleigh

Rayleigh law

```
randRayleigh: function []
```
Defined in /libs/math/rcvRandom.red

Discrete Laws

randBernouilli

Bernouilli law

```
randBernouilli: function [
        p [float!]
]
```

p: float value (e.g. 0.5)

Defined in /libs/math/rcvRandom.red

randBinomial

Binomial law

```
randBinomial: function [
        n [integer!]
        p [float!]
]
```

n: integer value (e.g. 1)

p: float value (e.g. 0.5)

Defined in /libs/math/rcvRandom.red

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)

Defined in /libs/math/rcvRandom.red

randGeo

Geometric law

```
randGeo: func [
        p [float!]
]
```

p: float value (e.g. 0.25)
Defined in /libs/math/rcvRandom.red

randPoisson

Poisson law

```
randPoisson: function [
        l [float!]
]
```

l: float value (e.g. 1.5)
Defined in /libs/math/rcvRandom.red

Misc routines and functions

Defined in /libs/tools/rcvTools.red

Routines are Red/System code. These routines and functions may be used by different redCV modules such as matrix or imgProc.

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!]
]
```

rcvRound

Rounding 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!]
]
```

randf

```
randf: routine [
"returns a decimal value beween 0 and 1"
        m       [float!]
        return: [float!]
]
```

rcvExp

```
rcvExp: routine [
"returns exponential value"
        value   [float!]
        return: [float!]
]
```

rcvLog-2

```
rcvLog-2: routine [
"Return the base-2 logarithm"
        value   [float!]
        return: [float!]
]
```

rcvSquish

```
rcvSquish: routine [
"For image transform"
        x       [float!]
]
```

rcvNSquareRoot

Returns the nth root of Num

```
rcvNSquareRoot: function [
        num     [number!]
```

nroot [number!]
return: [float!]
]

general square root function used by Minkowski Distance

rcvElapsed

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

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