## **REDCV FUNCTIONS DOCUMENTATION**

Most of functions are calling Red/System routines for faster image rendering. All redCV routines can be directly called from a red program (not for newbies). For a more convenient access, Red/System routines are "exported" to red functions. All red routines are prefixed with underscore (e.g. \_rcvCopy). Only red functions are documented. Doc string summarizes the function and calling function from red code sample is in *italic*.

```
All includes to redCV libraries are declared in a single file (/libs/redcv.red).

[
#include %core/rcvCore.red ; Basic image creating and processing functions
#include %highgui/rcvHighGui.red ; Fast Highgui functions
#include %matrix/rcvMatrix.red ; Matrices functions
#include %imgproc/rcvImgProc.red ; Color space conversions Convolution and other
#include %math/rcvRandom.red ; Random laws for generating random images
#include %math/rcvStats.red ; Statistical functions for images

]
```

Just deactivate the #include if you don't need functions, but rcvCore is obligatory.

More documentation to come.

Library: redCV/libs/core/ rcvcore.red

Function	Doc String
Basic Image I/O	
rcvCreateImage: function [size [pair!] return: [image!]]	Create empty (black) image
img: rcvCreateImage 640x480	
rcvReleaseImage: routine [src [image!]]	Delete image from memory
rcvRelease img	
rcvLoadImage: function [fileName [file!] return:	Load image from file
[image!] /grayscale]	/grayscale refinement loads a
img: rcvLoadImage %test.jpg	grayscale image
rcvLoadImageB: function [fileName [file!] return:	Load image from file and return
[binary!] /alpha]	image as binary
bin: rcvLoadImageB %test.png (bin=rgb)	
bin: rcvLoadImageB/apha %test.png (bin=argb)	
rcvSaveImage: function [src [image!] fileName [file!]]	Save image to file
rcvSaveImage img %test.jpg	
rcvCloneImage: function [src [image!] return: [image!]]	Return a copy of source image
dst: rcvCloneImage src	
rcvCopyImage : function [src [image!] dst [image!]]	Copy source image to
rcvCopyImage src dst	destination image
rcvRandomImage: function [size [pair!] value [tuple!]	Create a random uniform or
/uniform /alea return: [image!]]	pixel random image
dst: rcvRandomImage/uniform 640x480 red	
rcvZeroImage: function [src [image!]]	All pixels to 0
rcvZeroImage src	
rcvDecodeImage	TBD
rcvDecodeImageM	TBD
cvEncodeImage	TBD

Library: redCV/libs/core/rcvcore.red

Function	Doc String
Image Conversion	9
rcv2Gray: function [ src [image!] dst [image!] /average	Convert RGB image to Grayscale
/luminosity /lightness return: [image!]]	according to refinement
rcv2Gray/average src dst	
rcv2BGRA: function [src [image!] dst [image!]]	Convert RGBA => BGRA
rcv2BGRA src dst	
rcv2RGBA: function [src [image!] dst [image!]]	Convert BGRA => RGBA"
rcv2RGBA src dst	
rcv2BW: function [src [image!] dst [image!]]	Convert RGB image => Black and
rcv2BW src dst	White
rcv2BWFilter: function [src [image!] dst [image!] thresh	Convert RGB image => Black and
[integer!]]	White according to threshold
rcv2BWFilter src dst 64	
rcvSplit: function [src [image!] dst [image!]/red /green	Split source image in RGB
/blue]	separate channels
rcvSplit/blue src dst (->blue channel)	
rcvInvert: function [source [image!] dst [image!]]	Similar to NOT image
rcvInvert src dst	
Math Operators on image	
rcvAdd: function [src1 [image!] src2 [image!] dst	dst: src1 + src2
[image!]]	ust. sici i sicz
rcvAdd image1 image2 destImage	
rcvSub: function [src1 [image!] src2 [image!] dst	dst: src1 - src2
[image!]]	430.3101 3102
rcvSub image1 image2 destImage	
rcvMul: function [src1 [image!] src2 [image!] dst	dst: src1 * src2
[image!]]	
rcvMul image1 image2 destImage	
rcvDiv: function [src1 [image!] src2 [image!] dst	dst: src1 / src2
[image!]]	,
rcvDiv image1 image2 destImage	
rcvMod: function [src1 [image!] src2 [image!] dst	dst: src1 // src2 (modulo)
[image!]]	
rcvMod image1 image2 destImage	
rcvRem: function [src1 [image!] src2 [image!] dst	dst: src1 % src2 (remainder)
[image!]]	
rcvRem image1 image2 destImage	
rcvAbsDiff: function [src1 [image!] src2 [image!] dst	dst: absolute difference src1
[image!]]	src2
rcvAbsDiff image1 image2 destImage	

Math operators with scalar (integer !)	
main eperators trial scalar (mesger t)	
rcvAddS: function [src [image!] dst [image!] val	dst: src + integer! value
[integer!]]	
rcvAddS source destination 128	
rcvSubS: function [src [image!] dst [image!] val	dst: src - integer! value
[integer!]]	
rcvSubS source destination 128	
rcvMulS: function [src [image!] dst [image!] val	dst: src * integer! value
[integer!]]	
rcvMubS source destination 2	detuere / integer Lyalue
rcvDivS: function [src [image!] dst [image!] val [integer!]]	dst: src / integer! value
rcvDivS source destination 2	
rcvModS: function [src [image!] dst [image!] val	dst: src // integer! value
[integer!]	(modulo)
rcvModS source destination 4	(ea.a.e)
rcvRemS: function [src [image!] dst [image!] val	dst: src % integer! value
[integer!]]	(remainder)
rcvRemS source destination 2	
rcvPow: function [src [image!] dst [image!] val	dst: src ^integer! value
[integer!]]	
rcvPow source destination 2	
rcvLSH: function [src [image!] dst [image!]val [integer!]]	Left shift image by value
rcvLSH source destination 2	
rcvRSH: function [src [image!] dst [image!] val	Right Shift image by value
[integer!]]	
rcvRSH source destination 2	less as severe rest
rcvSQR: function [src [image!] dst [image!] val [integer!]]	Image square root
rcvSQR source destination 2	
Math operators with scalar (tuple!)	
main operators man scalar (capier,	
rcvAddT: function [src [image!] dst [image!] val [tuple!]]	dst: src + tuple! value
rcvAddT source destination 128.128.128	·
rcvSubT: function [src [image!] dst [image!] val [tuple!]]	dst: src - tuple! value
rcvSubT source destination 32.32.32	
rcvMulT: function [src [image!] dst [image!] val [tuple!]]	dst: src * tuple! value
rcvMulT source destination 2.2.2	
rcvDivT: function [src [image!] dst [image!] val [tuple!]]	dst: src / tuple! value
rcvDivT source destination 2.2.2	
rcvModT: function [src [image!] dst [image!] val	dst: src // tuple! value (modulo)
[tuple!]]	
rcvModT source destination 2.2.2	det. ove 0/ toolel · · · · · ·
rcvRemT: function [src [image!] dst [image!] val [tuple!]]	dst: src % tuple! value (remainder)
rcvRemT source destination 2.2.2	(Temamuer)
revneill source destillation 2.2.2	

rcvAND: function [src1 [image!] src2 [image!] dst [image!]] rcvAND source1 source2 destination rcvOR: function [src1 [image!] src2 [image!] dst [image!]] rcvAND source1 source2 destination rcvXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvXOR source1 source2 destination rcvNAND: function [src1 [image!] src2 [image!] dst [image!]] rcvNAND source1 source2 destination rcvNAND source1 source2 destination rcvNAND source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNAND source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNXOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source blue  dst: src AND tuple! as image	Logical operators on Image	
rcvOR: function [src1 [image!] src2 [image!] dst [image!]] rcvOR source1 source2 destination rcvXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvXOR source1 source2 destination rcvNAND: function [src1 [image!] src2 [image!] dst [image!]] rcvNAND source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNXOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]	[image!]]	dst: src1 AND src2
[image!]] rcvOR source1 source2 destination rcvXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvXOR source1 source2 destination rcvXOR source1 source2 destination rcvNAND: function [src1 [image!] src2 [image!] dst [image!]] rcvNAND source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNXOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNOT: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		
rcvXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvXOR source1 source2 destination rcvNAND: function [src1 [image!] src2 [image!] dst [image!]] rcvNAND source1 source2 destination rcvNAND source1 source2 destination rcvNAND source1 source2 destination rcvNCOR source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNXOR source1 source2 destination rcvNIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNOT: function [src1 [image!] src2 [image!]] rcvNOT source destination rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvXNOS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		dst: src1 OR src2
[image!]] rcvXOR source1 source2 destination rcvNAND: function [src1 [image!] src2 [image!] dst [image!]] rcvNAND source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvNXOR source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]	rcvOR source1 source2 destination	
rcvNAND: function [src1 [image!] src2 [image!] dst [image!]] rcvNAND source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!]] dst [image!]] rcvNXOR source1 source2 destination rcvNXOR source1 source2 destination rcvNIN: function [src1 [image!] src2 [image!]] dst [image!]] rcvMIN: function [src1 [image!] src2 [image!]] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!]] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image  rcvANDS: function [src [image!]] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]	[image!]]	dst: src1 XOR src2
[image!]] rcvNAND source1 source2 destination rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!]] dst [image!]] rcvNXOR source1 source2 destination rcvNIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src1 [image!] src2 [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image  rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		data and NAND and
rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNOR source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNXOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNAX: function [src1 [image!] src2 [image!] dst [image!]] rcvNAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]	[image!]]	dst: src1 NAND src2
[image!]] rcvNOR source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNXOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNAX: function [src1 [image!] src2 [image!]] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] dst: NOT src rcvNOT source destination logical operators and scalar (tuple!) on image  rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvXNDS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		dety are1 NOD are3
rcvNOR source1 source2 destination rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNXOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		ust. SICI NOR SIC2
rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] rcvNXOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]	1	
[image!]] rcvNXOR source1 source2 destination rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!]] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		data area NIVOD rea
rcvNXOR source1 source2 destination  rcvMIN: function [src1 [image!] src2 [image!] dst [image!]]  rcvMIN source1 source2 destination  rcvMAX: function [src1 [image!] src2 [image!] dst [image!]]  rcvMAX source1 source2 destination  rcvNot: function [src [image!] dst [image!]]  rcvNOT source destination  logical operators and scalar (tuple!) on image  rcvANDS: function [src [image!]] dst [image!] value [tuple!] return: [image!]]  rcvANDS source red  rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		dst: src1 NXOR rc2
rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]	1 - 0	
[image!]]  rcvMIN source1 source2 destination  rcvMAX: function [src1 [image!] src2 [image!] dst [image!]]  rcvMAX source1 source2 destination  rcvNot: function [src [image!] dst [image!]]  rcvNOT source destination  logical operators and scalar (tuple!) on image  rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source red  rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		
rcvMIN source1 source2 destination rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] dst: NOT src rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		dst: minimum src1 src2
rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] rcvMAX source1 source2 destination rcvNot: function [src [image!] dst [image!]] dst: NOT src rcvNOT source destination logical operators and scalar (tuple!) on image rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] dst: src OR tuple! as image dst: src XOR tuple! as image dst: src XOR tuple! as image	1 - 0	
[image!]]  rcvMAX source1 source2 destination  rcvNot: function [src [image!] dst [image!]] dst: NOT src  rcvNOT source destination  logical operators and scalar (tuple!) on image  rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source red  rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]		11
rcvNot: function [src [image!] dst [image!]] rcvNOT source destination  logical operators and scalar (tuple!) on image  rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source red  rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  dst: NOT src  dst: NOT src  dst: src AND tuple! as image  dst: src OR tuple! as image  dst: src OR tuple! as image		dst: maximum src1 src2
rcvNOT source destination   logical operators and scalar (tuple!) on image   dst: src AND tuple! as image   tuple!] return: [image!]   rcvANDS source red   rcvORS: function [src [image!] dst [image!] value   tuple!] return: [image!]   rcvANDS source green   rcvXORS: function [src [image!] dst [image!] value   dst: src OR tuple! as image   tuple!] return: [image!]   dst [image!] value   dst: src XOR tuple! as image   tuple!] return: [image!]	rcvMAX source1 source2 destination	
rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source red  rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  dst: src AND tuple! as image  dst: src OR tuple! as image  dst: src XOR tuple! as image		dst: NOT src
[tuple!] return: [image!]]       rcvANDS source red         rcvORS: function [src [image!] dst [image!] value       dst: src OR tuple! as image         [tuple!] return: [image!]]       rcvANDS source green         rcvXORS: function [src [image!] dst [image!] value       dst: src XOR tuple! as image         [tuple!] return: [image!]]	logical operators and scalar (tuple!) on image	
rcvANDS source red  rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  dst: src XOR tuple! as image		dst: src AND tuple! as image
rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  dst: src OR tuple! as image  dst: src XOR tuple! as image	1 - 1 - 1	
[tuple!] return: [image!]]  rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]]  dst: src XOR tuple! as image		Literatura CD Landa Landa
rcvANDS source green  rcvXORS: function [src [image!] dst [image!] value		ast: src OK tuple! as image
rcvXORS: function [src [image!] dst [image!] value dst: src XOR tuple! as image [tuple!] return: [image!]]	1	
[tuple!] return: [image!]]		
		dst: src XOR tuple! as image
	1	

Library: redCV/libs/core/ rcvMatrix.red

Functions: 28

Attention: Matrices with the same type and the same size!

Function	Doc String
rcvCreateMat: function [ type [word!] bitSize [integer!]	Create 2-D matrix
mSize [pair!] return: [vector!]]	
mat: rcvCreateMat 'integer! 8 512x512	
rcvReleaseMat: function [mat [vector!]]	Releases Matrix
rcvReleaseMat mat	
rcvCloneMat: function [src [vector!] return: [vector!]]	Returns a copy of source matrix
newmat: rcvCloneMat src	, ,
rcvCopyMat: function [src [vector!] dst [vector!]]	Copy source matrix to
rcvCopyMat src dst	destination matrix
rcvRandomMat: function [mat [vector!] value	Randomizes matrix
[integer!]]	
rcvRandomMat mat 255	
rcvColorMat: function [mat [vector!] value [integer!] ]	Sets matrix color
rcvColorMat mat 128	
rcvImage2Mat: function [src [image!] mat [vector!]]	Red Image to 2-D Matrix
	Converts Red Image to grayscale
rcvImage2Mat src mat	before matrix transformation
	Only 8-bit integer image
rcvMat82Image: function [mat [vector!] dst [image!]]	Matrix to Red image
rcvMat82Image mat dst	8-bit integer matrix
rcvMat16Image: function [mat [vector!] dst [image!]]	Matrix to Red image
rcvMat162Image mat dst	16-bit integer matrix
rcvMat322Image: function [mat [vector!] dst [image!]]	Matrix to Red image
rcvMat322Image mat dst	32-bit integer matrix
rcvConvolveMat: function [src [vector!] dst [vector!]	Fast 2-D matrix convolution
mSize[pair!] kernel [block!] factor [float!] delta [float!]]	For 8-bit integer matrix only
rcvConvolveMat mat1 mat2 img1/size mask 1.0 0.0	
rcvConvertMatScale: function [src [vector!] dst	Converts Matrix Scale
[vector!] srcScale [number!] dstScale [number!] /fast	8<->16<->bit conversions
/normal]	
rcvConvertMatScale/fast mat1 mat2 255 32768	
-	
Math operator for matrix	
rcvAddMat: function [src1 [vector!] src2 [vector!]	Adds two matrices to a new
return: [vector!]]	matrice
m : rcvAddMat m1 m2	
rcvSubMat: function [src1 [vector!] src2 [vector!]	Subtracts two matrices to a new
return: [vector!]]	matrice
m : rcvSubMat m1 m2	

rcvMulMat: function [src1 [vector!] src2 [vector!] return: [vector!] m: rcvMulMat m1 m2	Multiplies two matrices to a new matrice
rcvDivMat: function [src1 [vector!] src2 [vector!] return: [vector!]] m:rcvDivlMat m1 m2	Divides two matrices and returns the result
rcvRemMat: function [src1 [vector!] src2 [vector!] return: [vector!]]  m:rcvRemlMat m1 m2	Remainder src1 by src2
Scalar operations directly modify vector	
rcvAddSMat: function [src [vector!] value [integer!]] rcvAddSMat m1 255	Adds scalar
rcvSubSMat: function [src [vector!] value [integer!]] rcvSubSMat m1 255	Subtracts scalar
rcvMulSMat: function [src [vector!] value [integer!]] rcvSMulSMat m1 255	Multiplies matrix by scalar
rcvDivSMat: function [src [vector!] value [integer!]] rcvSMulSMat m1 16	Divides matrix by scalar
rcvRemSMat: function [src [vector!] value [integer!]] rcvSRemMat m1 16	Remainder
Logical operators	
rcvAndMat: function [src1 [vector!] src2 [vector!] return: [vector!]]  m: rcvAndMat m1 m2	dst: src1 AND src2
rcvOrMat: function [src1 [vector!] src2 [vector!] return: [vector!]  m: rcvOrMat m1 m2	dst: src1 OR src2
rcvXorMat: function [src1 [vector!] src2 [vector!] return: [vector!]] m: rcvXorMat m1 m2	dst: src1 XOR src2
Scalar operations directly modify vector	
rcvAndSMat: function [src [vector!] value [integer!] rcvAndSMat src 127	src AND value
rcvOrSMat: function [src [vector!] value [integer!] rcvOrSMat src 127	src OR value
rcvXorSMat: function [src [vector!] value [integer!]] rcvXorSMat src 127	src XOR value

## Library: redCV/libs/imgproc/ rcvImgProc.red

Function  Space Color Conversion  rcvRGB2XY2: function [src [image!] dst [image!]]  rcvRGB2XY2 src dst  rcvXY22RGB: function [src [image!] dst [image!]]  rcvRGB2XY2 src dst  rcvXY22RGB src dst  Image transformation  rcvFlip: function [src [image!] dst [image!]]  rcvFlip: function [src [image!] dst [image!]]  rcvFlip/horizontal /vertical /both return: [image!]]  rcvFlip/horizontal src dst  rcvFlip/both src dst  Image Convolution  rcvConvolve: function [src [image!] dst [image!]]  rcvConvolve: function [src [image!] dst [image!]]  rcvConvolve src dst noFilter 1.0 0.0  rcvFastConvolve: function [src [image!] dst [image!]]  rcvFastConvolve: function [src [image!] dst [image!]]  rcvFastConvolve img1 img2 1 mask 1.0 0.0  rcvFilter2D: function [src [image!] dst [image!]]  rcvFastFilter2D: function [src [image!] dst [image!]]  rcvFastFilter2D: function [src [image!] dst [image!]]  kernel [block!]]  rcvFastFilter2D: function [src [image!] dst [image!]]  rcvFastFilter2D: function [src [image!]] dst [image!]]  kernel [block]]  rcvFastFilter2D: function [src [image!]] dst [image!]  kernel [block]]  rcvFastFilter2D: function [src [image!]] dst [image!]  kernel [block]]  rcvFastFilter2D: function [src [image!]] dst [image!]  kernel [block]]  Faster convolution Filter  Create 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 o is the standard  deviation of the distribution.	Functions: 10	D. C. C.
rcvRGB2XYZ: function [src [image!] dst [image!]]BGR to CIE XYZ color conversionrcvRGB2XYZ src dstCIE XYZ to RBG color conversionrcvXYZ2RGB: function [src [image!] dst [image!]]CIE XYZ to RBG color conversionrcvXYZ2RGB src dstLeft Right, Up down or bothdirections [src [image!] dst [image!]]Left Right, Up down or both/horizontal /vertical /both return: [image!]]Left Right, Up down or both/horizontal /src dstdirections fliprcvFilp/bnizontal src dstrcvFilp/bnizontal src dstrcvFilp/both src dstConvolve an image with the kernel/mage ConvolutionConvolve an image with the kernelrcvConvolve src dst noFilter 1.0 0.0Convolves a 8-bit image with the kernel by channelChannel [integer !] kernel [block!] factor [float!]Convolves a 8-bit image with the kernel by channeldelta [float!]]Convolves img1/channel 1rcvFastConvolve img1 img2 1 mask 1.0 0.0Convolves img1/channel 1rcvFilter2D src dst noFilter 0Basic convolution FilterrcvFastFilter2D src dst noFilter 0Faster convolution FilterImage FiltersFaster convolution FilterrcvMakeGaussian: function [kSize [pair!] return: [block!]]Create a Gaussian uneven kernel with the following equationknl: rcvMakeGaussian 3x3 $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	Function	Doc String
rcvRGB2XYZ src dst       CIE XYZ to RBG color conversion         rcvXYZ2RGB: function [src [image!] dst [image!]]       CIE XYZ to RBG color conversion         rcvXYZ2RGB src dst       Image transformation         rcvFlip: function [src [image!] dst [image!]]       Left Right, Up down or both directions flip         rcvFlip/horizontal /vertical src dst       rcvFlip/horizontal src dst         rcvFlip/both src dst       directions flip         rcvConvolve: function [src [image!] dst [image!]       Convolve an image with the kernel         kernel [block!] factor [float!] delta [float!]]       Convolve an image with the kernel         kernel [block!] factor [float!] delta [float!]]       Convolves a 8-bit image with the kernel by channel         delta [float!]]       Convolves img1/channel 1         rcvFastFiler2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]]       Basic convolution Filter         Basic convolution Filter       Faster convolution Filter         Image Filters       Create a Gaussian uneven kernel with the following equation         knl: rcvMakeGaussian 3x3 $G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+x^2}{2\sigma^2}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and o is the standard	-	
rcvXY22RGB: function [src [image!] dst [image!]] rcvXY22RGB src dst  Image transformation  rcvFlip: function [src [image!] dst [image!] /horizontal /vertical /both return: [image!]] rcvFlip/horizontal src dst rcvFlip/both src dst Image Convolution  rcvConvolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]] rcvFastConvolve function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]] rcvFastConvolve: function [src [image!] dst [image!] Convolves a 8-bit image with the kernel kernel by channel [integer !] kernel [block!] factor [float!] delta [float!]] rcvFastConvolve img1 img2 1 mask 1.0 0.0 rcvFaiter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]] rcvFilter2D src dst noFilter 0 rcvFastFilter2D src dst noFilter 0 rcvFastFilter2D src dst filter Image Filters rcvMakeGaussian: function [kSize [pair!] return: [block!]]  knl: rcvMakeGaussian 3x3 $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	1	BGR to CIE XYZ color conversion
rcvXY22RGB src dstImage transformationrcvFlip: function [src [image!] dst [image!] /horizontal /vertical /both return: [image!]] rcvFlip/horizontal src dst rcvFlip/both src dstLeft Right, Up down or both directions fliprcvFlip/horizontal src dst rcvFlip/both src dstdirections flipImage ConvolutionConvolve an image with the kernelrcvConvolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]] rcvFastConvolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]]Convolves a 8-bit image with the kernel by channeldelta [float!]] rcvFastConvolve img1 img2 1 mask 1.0 0.0Convolves img1/channel 1rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]] rcvFilter2D src dst nofilter 0Faster convolution FilterrcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]]Faster convolution FilterImage Filters rcvMakeGaussian: function [kSize [pair!] return: [block!]]Create a Gaussian uneven kernel with the following equationknl: rcvMakeGaussian 3x3G(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 $\sigma$ is the standard		
Image transformationLeft Right, Up down or both directions fliprcvFlip: function [src [image!] dst [image!]] rcvFlip/horizontal src dst rcvFlip/both src dstLeft Right, Up down or both directions fliprcvFlip/horizontal src dst rcvFlip/both src dstconvolve function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]] rcvConvolve src dst noFilter 1.0 0.0Convolve an image with the kernelrcvFastConvolve function [src [image!] dst [image!] delta [float!]] rcvFastConvolve img1 img2 1 mask 1.0 0.0Convolves a 8-bit image with the kernel by channelrcvFliter2D: function [src [image!] dst [image!] kernel [block!]]Convolves img1/channel 1rcvFilter2D: function [src [image!] dst [image!] kernel [block!]]Basic convolution FilterrcvFastFilter2D src dst noFilter 0Faster convolution FilterrcvFastFilter2D src dst FilterFaster convolution FilterImage FiltersCreate a Gaussian uneven kernel with the following equationrcvMakeGaussian: function [kSize [pair!] return: [block!]]Create a Gaussian uneven kernel with the following equationknl: rcvMakeGaussian 3x3G(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 $\sigma$ is the standard		CIE XYZ to RBG color conversion
rcvFlip: function [src [image!] dst [image!]	rcvXYZ2RGB src dst	
/horizontal /vertical /both return: [image!]] rcvFlip/horizontal src dst rcvFlip/horizontal src dst rcvFlip/both src dst rcvFlip/both src dst rcvFlip/both src dst revFlip/both src dst roFilter 1.0 0.0	Image transformation	
/horizontal /vertical /both return: [image!]]       directions flip $rcvFlip/horizontal src dst$ $rcvFlip/both src dst$ Image Convolution       Convolve an image with the kernel $kernel [block!]$ factor [float!] delta [float!]] $convolve an image with the kernel$ $kernel [block!]$ factor [float!] delta [float!]] $convolve an image with the kernel$ $kernel [block!]$ factor [float!] $convolve an image with the kernel$ $kernel [block!]$ factor [float!] $convolve an image with the kernel$ $kernel [block!]$ factor [float!] $convolve an image with the kernel         kernel [block!] factor [float!]       convolve an image with the kernel         kernel [block!] factor [float!]       convolve an image with the kernel         kernel [block!] factor [float!]       convolve an image with the kernel         kernel [block!] factor [float!]       convolve an image with the kernel         convolve an image with the kernel       convolve an image with the kernel         convolve an image with the kernel       convolve an image with the kernel         convolve an image with the kernel       convolve an image with the kernel         convolve an image with the kernel       convolve an image with the kernel         convolve an image with the kernel       convolve an image with the kernel         convol$		
rcvFlip/horizontal src dst rcvFlip/vertical src dst rcvFlip/both src dstConvolve an image with the kernel mage Convolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]] rcvConvolve src dst noFilter 1.0 0.0Convolve an image with the kernel convolve an image with the kernel convolve an image with the kernel delta [float!]] rcvFastConvolve: function [src [image!] dst [image!] cvFastConvolve img1 img2 1 mask 1.0 0.0Convolves a 8-bit image with the kernel by channel convolves img1/channel 1rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]] rcvFilter2D: function [src [image!] dst [image!] kernel [block!]]Basic convolution FilterrcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]]Faster convolution Filterlmage FiltersCreate a Gaussian uneven kernel with the following equationknl: rcvMakeGaussian 3x3Create a Gaussian uneven kernel with the following equationknl: rcvMakeGaussian 3x3Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	rcvFlip: function [src [image!] dst [image!]	Left Right, Up down or both
rcvFlip/vertical src dst $rcvFlip/both src dst$ Image ConvolutionConvolve: function [src [image!] dst [image!]kernel [block!] factor [float!] delta [float!]] $rcvFounder src dst noFilter 1.0 0.0$ $rcvFastConvolve src dst noFilter 1.0 0.0$ Convolves a 8-bit image with the kernel by channel [integer !] kernel [block!] factor [float!]delta [float!]] $rcvFastConvolve img1 img2 1 mask 1.0 0.0$ $rcvFastConvolve img1 img2 1 mask 1.0 0.0$ $rcvFilter2D$ : function [src [image!] dst [image!] kernel [block!] delta [integer!]] $rcvFilter2D$ src dst noFilter 0 $rcvFastFilter2D$ src dst noFilter 0Faster convolution Filter $verPastFilter2D$ src dst FilterFaster convolution FilterImage Filters $rcvMakeGaussian$ : function [kSize [pair!] return: [block!]] $rcvPastFilter2D$ src dst Filter $verPastFilter2D$ src dst Filter $rcvMakeGaussian$ : function [kSize [pair!] return: [block!]] $rcvPastFilter2D$ src dst Filter $verPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst Filter $verPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst FilterImage Filters $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst FilterImage Filters $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst FilterImage Filters $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst FilterImage Filters $rcvPastFilter2D$ src dst Filter $rcvPastFilter2D$ src dst Filter	/horizontal /vertical /both return: [image!]]	directions flip
$rcvFlip/both\ src\ dst$ Image Convolution $rcvConvolve:\ function\ [src\ [image!]\ delta\ [float!]]\ rcvConvolve\ src\ dst\ noFilter\ 1.0\ 0.0$ Convolve an image with the kernel $rcvFastConvolve:\ function\ [src\ [image!]\ dst\ [image!]\ Channel\ [integer\ !]\ kernel\ [block!]\ factor\ [float!]\ delta\ [integer\ !]\ rcvFastConvolve\ img1\ img2\ 1\ mask\ 1.0\ 0.0$ Convolves a 8-bit image with the kernel by channel $rcvFastConvolve\ img1\ img2\ 1\ mask\ 1.0\ 0.0$ Convolves img1/channel\ 1 $rcvFilter2D:\ function\ [src\ [image!]\ dst\ [image!]\ kernel\ [block!]\ delta\ [integer!]]\ rcvFilter2D\ src\ dst\ noFilter\ 0$ Basic convolution Filter $rcvFastFilter2D:\ function\ [src\ [image!]\ dst\ [image!]\ kernel\ [block!]]$ Faster convolution Filter $rcvFastFilter2D:\ src\ dst\ Filter$ Faster convolution Filter $rcvMakeGaussian:\ function\ [kSize\ [pair!]\ return:\ [block!]]$ Create a Gaussian uneven kernel with the following equation $rcvMakeGaussian\ 3x3$ $rcvMakeGaussian\ 3x3$ Create a Gaussian uneven kernel with the following equation	rcvFlip/horizontal src dst	
Image Convolution       Convolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]] rcvConvolve src dst noFilter 1.0 0.0       Convolve an image with the kernel level float!]         rcvFastConvolve: function [src [image!] dst [image!] delta [float!]] rcvFastConvolve img1 img2 1 mask 1.0 0.0       Convolves a 8-bit image with the kernel by channel         cconvolve: function [src [image!] dst [image!] kernel [block!]] delta [integer!]] rcvFilter2D: function [src [image!] dst [image!] kernel [block!]] rcvFilter2D src dst noFilter 0       Basic convolution Filter         rcvFastFilter2D src dst rilter       Faster convolution Filter         Image Filters       Create a Gaussian uneven kernel with the following equation         knl: rcvMakeGaussian 3x3 $G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and σ is the standard	rcvFlip/vertical src dst	
rcvConvolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]] rcvConvolve src dst noFilter 1.0 0.0 rcvFastConvolve: function [src [image!] dst [image!] Convolves a 8-bit image with the kernel left [float!] rcvFastConvolve img1 img2 1 mask 1.0 0.0 rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]] rcvFilter2D src dst noFilter 0 rcvFastFilter2D src dst roFilter 0 rcvFastFilter2D src dst Filter    Image Filters rcvMakeGaussian: function [kSize [pair!] return: [block!]]    knl: rcvMakeGaussian 3x3	rcvFlip/both src dst	
kernel [block!] factor [float!] delta [float!]]   rcvConvolve src dst noFilter 1.0 0.0   rcvFastConvolve: function [src [image!] dst [image!]   Convolves a 8-bit image with the kernel by channel delta [float!]]   rcvFastConvolve img1 img2 1 mask 1.0 0.0   rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]]   rcvFilter2D src dst noFilter 0   rcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]]   rcvFastFilter2D src dst Filter   Image Filters   rcvMakeGaussian: function [kSize [pair!] return: [block!]]   knl: rcvMakeGaussian 3x3 $G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	Image Convolution	
kernel [block!] factor [float!] delta [float!]]   rcvConvolve src dst noFilter 1.0 0.0   rcvFastConvolve: function [src [image!] dst [image!]   Convolves a 8-bit image with the kernel by channel delta [float!]]   rcvFastConvolve img1 img2 1 mask 1.0 0.0   rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]]   rcvFilter2D src dst noFilter 0   rcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]]   rcvFastFilter2D src dst Filter   Image Filters   rcvMakeGaussian: function [kSize [pair!] return: [block!]]   knl: rcvMakeGaussian 3x3 $G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard		
kernel [block!] factor [float!] delta [float!]]   rcvConvolve src dst noFilter 1.0 0.0   rcvFastConvolve: function [src [image!] dst [image!]   Convolves a 8-bit image with the kernel by channel delta [float!]]   rcvFastConvolve img1 img2 1 mask 1.0 0.0   rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]]   rcvFilter2D src dst noFilter 0   rcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]]   rcvFastFilter2D src dst Filter   Image Filters   rcvMakeGaussian: function [kSize [pair!] return: [block!]]   knl: rcvMakeGaussian 3x3 $G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	rcvConvolve: function [src [image!] dst [image!]	Convolve an image with the kernel
rcvFastConvolve: function [src [image!] dst [image!]	kernel [block!] factor [float!] delta [float!]]	
Channel [integer !] kernel [block!] factor [float!] kernel by channel delta [float!]]	rcvConvolve src dst noFilter 1.0 0.0	
Channel [integer !] kernel [block!] factor [float!] kernel by channel delta [float!]]	rcvFastConvolve: function [src [image!] dst [image!]	Convolves a 8-bit image with the
delta [float!]]		
rcvFastConvolve img1 img2 1 mask 1.0 0.0Convolves img1/channel 1rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]]Basic convolution Filterlblock!] delta [integer!]]Faster convolution FilterrcvFastFilter2D src dst noFilter 0Faster convolution FilterrcvFastFilter2D src dst FilterFaster convolution Filterlmage FiltersCreate a Gaussian uneven kernel with the following equationknl: rcvMakeGaussian 3x3 $G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard		,
rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]] rcvFilter2D src dst noFilter 0 rcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]] rcvFastFilter2D src dst Filter		Convolves ima1/channel 1
[block!] delta [integer!]]		
rcvFilter2D src dst noFilter 0rcvFastFilter2D: function [src [image!] dst [image!]Faster convolution Filterkernel [block!]]Faster convolution FilterImage FiltersCreate a Gaussian uneven kernelrcvMakeGaussian: function [kSize [pair!] return:With the following equationknl: rcvMakeGaussian $3x3$ $G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard		
rcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]] rcvFastFilter2D src dst Filter	1	
kernel [block!]]		Faster convolution Filter
Image FiltersCreate a Gaussian uneven kernel with the following equationknl: rcvMakeGaussian $3x3$ $G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard		Tubici com oracion i incer
Image Filters  rcvMakeGaussian: function [kSize [pair!] return: [block!]]  Create 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 $\sigma$ is the standard		
rcvMakeGaussian: function [kSize [pair!] return: [block!]] Create 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 $\sigma$ is the standard		
[block!]] 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 $\sigma$ is the standard		Create a Gaussian uneven kernel
knl: $rcvMakeGaussian\ 3x3$ $G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	•	
Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	[block:]]	with the following equation
Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard		
Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	knl: rcvMakeGaussian 3v3	$G(x,y) = \frac{1}{2\pi^2}e^{-\frac{x^2+y^2}{2\pi^2}}$
horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard	KIII. TEVIVIAKE GAASSIAII SAS	$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-2\sigma^2}$
horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and $\sigma$ is the standard		*
origin, y is the distance along vertical axis measured from the origin and $\boldsymbol{\sigma}$ is the standard		
vertical axis measured from the origin and $\boldsymbol{\sigma}$ is the standard		
origin and $\boldsymbol{\sigma}$ is the standard		
rcvGaussianFilter: function [src [image!] dst [image!]   Gaussian 2D Filter	rcvGaussianFilter: function [src [image!] dst [image!]	Gaussian 2D Filter
kernel [block!] delta [integer!]]		
rcvGaussianFilter src dst knl 0		

Gaussian Pyramid Decomposition	
rcvResizeImage: function [src [image!] canvas iSize	Resizes image and applies filter for
[pair!]/gaussian return: [pair!]]	Gaussian pyramidal resizing if
	required. Only Gaussian 5x5 kernel
rcvResizeImage/gaussian src canvas iSize	is currently supported. Canvas is a
	base facet.
rcvResizeImage src canvas iSize	If you don't call /Gaussian
	refinement image is just resized.

## Library: redCV/libs/math/rcvStats.red

Function	Doc String
rcvCountNonZero: function [src [image!] return:	Returns number of non zero
[integer!]]	values in image
n: rcvCountNonZero source	
rcvMeanImage: function [src [image!] return: [tuple!]	Returns mean value of image as
/argb]	a tuple rgb or argb
mean: rcvMeanImage source	
rcvSum: function [src [image!] return: [block!] /argb]	Returns sum value of image as a
sum: rcvSum source	block rgb or argb
rcvVarImage: function [src [image!] return: [tuple!]	returns standard deviation
/argb]	value of image as a tuple rgb or
std: rcvVarImage source	argb
rcvMedianImage: function [source [image!] return:	Returns median value of image
[tuple!]]	as tuple argb
median : rcvMedianImage source	
rcvMinImage: function [source [image!] return:	Minimal value in Image as a
[tuple!]]	tuple argb
mini : rcvMinImage source	
rcvMaxImage: function [source [image!] return:	Maximal value in Image as a
[tuple!]]	tuple arg
maxi : rcvMaxImage source	
rcvRangelmage: function [source [image!] return:	Range value in Image as a tuple
[tuple!]]	argb
range: rcvRangeImage source	
rcvSortImage: function [source [image!] dst [image!]]	Ascending image sorting
rcvSortImage source destination	

## Library: redCV/libs/highgui/ rcvHighGui.red

Function	Doc String
rcvNamedWindow: function [name [string!] return:	Creates and shows a window
[window!]]	
s1: rcvNamedWindow "Source"	
rcvDestroyWindow: function [window [face!]]	Destroys a window
rcvDestroyWindow s1	
rcvDestroyAllWindows: function []	Destroys all windows
rcvDestroyAllWindows	
rcvResizeWindow: function [window [face!] wSize	Sets window size
[pair!]]	
rcvResizeWindow s1 512x512	
rcvMoveWindow: function [window [face!] position	Sets window position
[pair!]]	
rcvMoveWindow s1 10x10	
rcvShowImage: function [window [face!] image	Shows image in window
[image!]]	
rcvShowImage s1 image	