

11/07/2016

REDCV FUNCTIONS DOCUMENTATION

Most of functions are calling Red/System routines implemented in `/libs/core/rcvRoutines.red` 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/rcvRoutines.red           ; All Red/System routines
#include %core/rcvImage.red             ; Image creating functions
#include %core/rcvCore.red              ; Basic image processing 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 `rcvRoutines.red`, `rcvImage.red` and `rcvCore` are obligatory.

More documentation to come.

Library: redCV/libs/core/rcvImage.red

Functions: 12

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

Library: redCV/libs/core/rcvcore.red

Functions: 42

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

Math operators with scalar (integer !)	
rcvAddS: function [src [image!] dst [image!] val [integer!]] <i>rcvAddS source destination 128</i>	dst: src + integer! value
rcvSubS: function [src [image!] dst [image!] val [integer!]] <i>rcvSubS source destination 128</i>	dst: src - integer! value
rcvMulS: function [src [image!] dst [image!] val [integer!]] <i>rcvMulS source destination 2</i>	dst: src * integer! value
rcvDivS: function [src [image!] dst [image!] val [integer!]] <i>rcvDivS source destination 2</i>	dst: src / integer! value
rcvModS: function [src [image!] dst [image!] val [integer!]] <i>rcvModS source destination 4</i>	dst: src // integer! value (modulo)
rcvRemS: function [src [image!] dst [image!] val [integer!]] <i>rcvRemS source destination 2</i>	dst: src % integer! value (remainder)
rcvPow: function [src [image!] dst [image!] val [integer!]] <i>rcvPow source destination 2</i>	dst: src ^integer! value
rcvLSH: function [src [image!] dst [image!] val [integer!]] <i>rcvLSH source destination 2</i>	Left shift image by value
rcvRSH: function [src [image!] dst [image!] val [integer!]] <i>rcvRSH source destination 2</i>	Right Shift image by value
rcvSQR: function [src [image!] dst [image!] val [integer!]] <i>rcvSQR source destination 2</i>	Image square root
Math operators with scalar (tuple !)	
rcvAddT: function [src [image!] dst [image!] val [tuple!]] <i>rcvAddT source destination 128.128.128</i>	dst: src + tuple! value
rcvSubT: function [src [image!] dst [image!] val [tuple!]] <i>rcvSubT source destination 32.32.32</i>	dst: src - tuple! value
rcvMulT: function [src [image!] dst [image!] val [tuple!]] <i>rcvMulT source destination 2.2.2</i>	dst: src * tuple! value
rcvDivT: function [src [image!] dst [image!] val [tuple!]] <i>rcvDivT source destination 2.2.2</i>	dst: src / tuple! value
rcvModT: function [src [image!] dst [image!] val [tuple!]] <i>rcvModT source destination 2.2.2</i>	dst: src // tuple! value (modulo)
rcvRemT: function [src [image!] dst [image!] val [tuple!]] <i>rcvRemT source destination 2.2.2</i>	dst: src % tuple! value (remainder)

Logical operators on Image	
rcvAND: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvAND source1 source2 destination</i>	dst: src1 AND src2
rcvOR: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvOR source1 source2 destination</i>	dst: src1 OR src2
rcvXOR: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvXOR source1 source2 destination</i>	dst: src1 XOR src2
rcvNAND: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvNAND source1 source2 destination</i>	dst: src1 NAND src2
rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvNOR source1 source2 destination</i>	dst: src1 NOR src2
rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvNXOR source1 source2 destination</i>	dst: src1 NXOR rc2
rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvMIN source1 source2 destination</i>	dst: minimum src1 src2
rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvMAX source1 source2 destination</i>	dst: maximum src1 src2
rcvNot: function [src [image!] dst [image!]] <i>rcvNOT source destination</i>	dst: NOT src
logical operators and scalar (tuple!) on image	
rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] <i>rcvANDS source red</i>	dst: src AND tuple! as image
rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] <i>rcvANDS source green</i>	dst: src OR tuple! as image
rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] <i>rcvANDS source blue</i>	dst: src XOR tuple! as image

Library: redCV/libs/imgproc/ rcvImgProc.red

Functions: 8

Function	Doc String
Space Color Conversion	
rcvRGB2XYZ: function [src [image!] dst [image!]] <i>rcvRGB2XYZ src dst</i>	BGR to CIE XYZ color conversion
rcvXYZ2RGB: function [src [image!] dst [image!]] <i>rcvXYZ2RGB src dst</i>	CIE XYZ to RBG color conversion
Image transformation	
rcvFlip: function [src [image!] dst [image!] /horizontal /vertical /both return: [image!]] <i>rcvFlip/horizontal src dst</i> <i>rcvFlip/vertical src dst</i> <i>rcvFlip/both src dst</i>	Left Right, Up down or both directions flip
Image Convolution	
rcvConvolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!]] <i>rcvConvolve src dst noFilter 1.0 0.0</i>	Convolve an image with the kernel
rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!]] <i>rcvFilter2D src dst noFilter 0</i>	Basic convolution Filter
rcvFastFilter2D: function [src [image!] dst [image!] kernel [block!]] <i>rcvFastFilter2D src dst Filter</i>	Faster convolution Filter
Image Filters	
rcvMakeGaussian: function [kSize [pair!] return: [block!]] <i>knl: rcvMakeGaussian 3x3</i>	Create a Gaussian uneven kernel with the following equation $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ <p>Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and σ is the standard deviation of the distribution.</p>
rcvGaussianFilter: function [src [image!] dst [image!] kernel [block!] delta [integer!]] <i>rcvGaussianFilter src dst knl 0</i>	Gaussian 2D Filter

Library: redCV/libs/math/rcvStats.red

Functions: 9

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