Rebol/OpenCV

This is an initial implementation of the OpenCV extension for Rebol 3, maintained by Oldes. This is an amelioration of the versions of OpenCV for Rebol 2, Rebol 3 and Red, developed by Ldci, which were limited to C-API functions. This new version allows to use the latest version of OpenCV (currently 4.10.0) and gives access to C++ functions. The idea is to develop something like opencv-python (https://pypi.org/project/opencv-python/)

Requirements

Before playing with Rebol-OpenCV, you need to install some stuff. First, you must install Rebol 3 developed and maintained by Oldes. You can get it here:
https://github.com/Siskin-framework/Rebol. The purpose of this Rebol fork is to push the original Carl's source to be usable at least like Rebol 2 but keep the source code clean and project easy to build.
Then you need to install the Siskin-framework builder also developed by Oldes. This is a fabulous Rebol3 based command line utility useful for compilation of various projects (originally designed for building the Rebol project). The code is here: https://github.com/Siskin-framework/Builder. With this tool you can compile Rebol 3 for different OS and different processors. We are using Apple ARM M1 processor and thus generated code is 64-bit.
Now, you need Rebol-OpenCV which can be found here:
https://github.com/Oldes/Rebol-OpenCV.

Installing OpenCV

The code is tested on macOS[1]. To install OpenCV two ways are possible. You can use homebrew (https://brew.sh) which is a great package manager for macOS and Linux. Once Hombrew is installed, open a terminal, and type the command "brew install opencv".

You can also install OpenCV from scratch from GitHub and with cmake:

```
git clone https://github.com/opencv/opencv
cd opencv
mkdir build
cd build
cmake ../
make -j8
```

[1] But Oldes has also successfully tested OpenCV extension with Windows

When building on macOS, the setup expects, that OpenCV includes, and libraries are accessible. So, it is recommended to use something like:

```
ln -s /opt/homebrew/Cellar/opencv/4.10.0/include/opencv4/opencv2 /usr/local/include/opencv2
ln -s /opt/homebrew/Cellar/opencv/4.10.0/lib /usr/local/lib/opencv.
```

Building OpenCV extension

Really easy. Copy Siskin executable to the Rebol-OpenCV-master directory and with Siskin compile the Rebol-OpenCV.nest file. Then copy the /build/ libopencv-macos-arm64.dylib to the Rebol-OpenCV-master directory and rename it as opencv.rebx.

All examples expect, that OpenCV extension is imported using one of these methods: 1. Using direct path to the file: cv: import %path/to/opencv.rebx 2. Using extension in the default location: cv: import 'opencv. Extensions and modules are searched in directories from system/options/module-paths (by default just current directory), and in this case your code must be placed in the directory where opencv.rebx file is present. The best way is to copy the opencv.rebx file to /Users/your_username/.rebol/modules/ (default location)

Reading and displaying images

Obviously, the first idea is to read and display an image. That's what the *imread* and imshow functions are for[2].

```
#!/usr/local/bin/r3
Rebol [
]
cv: import 'opencv
with cv [
        img1: imread/image %../images/mandrill.jpg
        print ? img1                                    ;--as rebol image
        img2: imread %../images/mandrill.jpg
        print ? img2                                    ;--as cvMat
        img3: imread/with %../images/mandrill.jpg 0
        print ? img3                                    ;--as cvMat
        imshow/name img1 "Rebol Image"
        imshow/name img2 "Matrix"
        imshow/name img3 "Grayscale Matrix"
        moveWindow "Rebol Image" 0x0
        moveWindow "Matrix" 260x0
        moveWindow "Grayscale Matrix" 520x0
        waitKey 0
        destroyAllWindows
]
```

[2] Instead of writing full paths to cv commands, like: cv/imread, cv/imshow, it is possible to bind the code to the cv context using with cv [...].

List of image formats supported by Rebol/OpenCV

Below is the list of supported image formats in Rebol/OpenCV:

- Windows bitmap (bmp)
- Portable image formats (pbm, pgm, ppm)
- Sun raster (sr, ras)
- JPEG (jpeg, jpg, jpe)
- JPEG 2000 (jp2)
- TIFF files (tiff, tif)
- Portable network graphics (png)
- Google webp format (webp)

GIFs files are not directly supported by OpenCV; there is no codec for this (license problem), but we can read gif with Rebol as explained later in this document.

Matrices

Matrices are fundamental when developing computer vision programs with OpenCV. Matrices can be used to store 2D images with 1 up to 4 channels of data. Matrices supported by OpenCV are various by combining depth and number of channels, which determines the type of the matrix.

Depth/Channel	C1	C2	C3	C4	
CV_8U	0	8	16	24	Unsigned 8 bits char [0 255]
CV_8S	1	9	17	25	Signed 8 bits char [-128 127]
CV_16U	2	10	18	26	Unsigned 16 bits short integer [0 65535]
CV_16S	3	11	19	27	Signed 16 bits short integer [-32768~32767]
CV_32S	4	12	20	28	Signed 32 bits integer [-2147483648 2147483647]
CV_32F	5	13	21	29	Float 32 bits float [-1.18*10-38 3.40*10-38]
CV_64F	6	14	22	30	Double precision 64 bits float

The way adopted is very flexible, and you can create matrices with different methods.

```
Matrix: [
        "Initialize new cvMat class"
        spec [pair! handle! image! block!]
]
```

```
#!/usr/local/bin/r3
Rebol [
]
cv: import 'opencv
with cv [
        ;--Properties of matrices are automatically detected by OpenCV.
        mat1: Matrix 320x240                    ;--using pair
        mat2: imread %../images/mask.png        ;--using imread image
        img: load  %../images/mandrill.jpg      ;--Rebol load image
        mat3: Matrix :img                       ;--Image -> Matrix
        ;--using blocks
        mat4: Matrix [320x240 CV_64FC4]         ;--320x240 float matrix with 4 channels
        ;--create a 200x200 8-bit blue image with 3 channels from binary string
        ;--when binary data source is shorter when constructing a matrix,it is copied repeatedly
        mat5: Matrix [200x200 CV_8UC3 #{FF0000 FF0000 FF0000}]
        ;--200x200 8-bit black image (from tuple) with 3-channels.
        mat 6: Matrix [200x200 CV_8UC3 0.0.0]
]
```

Matrices properties

Matrices have 7 properties than can be accessed by the *get-property* function.

When your code receives a matrix or an image from an external library or Rebol code, the most common question you have is what is the data type of the elements of this image or matrix? There are two properties in the *Matrix* class that answer this: MAT_DEPTH and MAT_TYPE.

MAT_DEPTH is the data type of each individual element in the image data. MAT_TYPE combines the data type of the elements along with the number of channels in the image.

```
MAT_SIZE: 1
MAT_TYPE: 2
MAT_DEPTH: 3
MAT_CHANNELS: 4
MAT_BINARY: 5
MAT_IMAGE: 6
MAT_VECTOR: 7
```

```
#!/usr/local/bin/r3
Rebol [
]
cv: import 'opencv
with cv [
        mat: Matrix 100x100
        probe get-property mat MAT_SIZE         ;--mat size
        probe get-property mat MAT_TYPE         ;--mat type id
        probe get-property mat MAT_CHANNELS     ;--number of channels (4 = BGRA)
        probe get-property mat MAT_DEPTH        ;--CV depth (0 = CV_8U)
        probe get-property mat MAT_BINARY       ;--raw binary data
        probe get-property mat MAT_VECTOR       ;--Rebol vector value
        probe get-property mat MAT_IMAGE        ;--Rebol image value
]
```

Oldes has also implemented two fantastic properties to facilitate the communication between Rebol and OpenCV: MAT_VECTOR and MAT_IMAGE. This is an example to illustrate how MAT_VECTOR property can be used for image transformation.

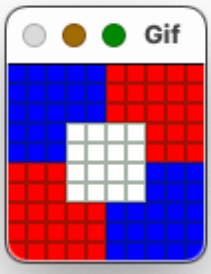
```
#!/usr/local/bin/r3
Rebol [
]
cv: import 'opencv
with cv [
        src: imread %../images/trees.jpg
        ;--using float vector for creating a kernel
         vec: #(float! [
                0.272 0.534 0.131
                0.349 0.686 0.168
                0.393 0.769 0.189]
        )
        ;-- create a 3x3 filter
        sepia-filter: Matrix [CV_32FC1 3x3 :vec]
        ;--apply filter to each pixel
        transform :src :src :sepia-filter
         ;--show result
   I    mshow/name src "Sepia effect"
        waitKey 0
        destroywindow "Sepia effect"
]
```

The result of the filter:

How to read Gif files

GIFs files are not supported by OpenCV; there is no codec for this (license problem), but we can read gif with Rebol and create a matrix from the image size and image/rgb data.

```
#!/usr/local/bin/r3
Rebol [
]
cv: import 'opencv
with cv [
        src: load %../images/sample.gif
        img: Matrix [:src/size CV_8UC3 :src/rgb]        ;--3 channels matrix
        namedWindow win: "Gif"                          ;--create a window
        imshow/name img win                             ;--show the image in the named window
        moveWindow win 260x20                           ;--move the window into some location
        waitKey 0                                       ;--wait for a key
        destroyWindow win
]
```



We can also process animated GIFs file. The trick is to consider the GIF file as a video and to use the *VideoCapture* class.

```
VideoCapture: [
        "Initialize new VideoCapture class"
        src [integer! file! string!]
]
```

This object is useful for accessing cameras and video files.

```
#!/usr/local/bin/r3
Rebol [
]
;--animated gif is considered as a movie
cv: import opencv               ;--import Rebol-OpenCV extension
with cv [
   movie: VideoCapture %../images/dance.gif
   unless movie [quit]
   ;--get some information about the gif file
   print ["Width :" w: get-property :movie CAP_PROP_FRAME_WIDTH]
   print ["Height:" h: get-property :movie CAP_PROP_FRAME_HEIGHT]
   print ["FPS   :" fps: get-property :movie CAP_PROP_FPS]
   print ["Frames:" nbFrames: to integer! get-property :movie CAP_PROP_FRAME_COUNT]
   print "ESC to close animation"
   delay: 1.5 / fps
   count: 0
   frame: read :movie ;--first image
   imshow :frame
   forever [
      read/into :movie :frame ;--no need to create a new matrix for each frame
      setWindowTitle "Image" join "Frame: " count + 1 ;--Redbol languages are one-based
      imshow :frame
      wait delay
      count: count + 1
      ;--when we are at the end of the gif go back to first image
      if count = nbFrames [set-property :movie CAP_PROP_POS_FRAMES 0 count: 0]
      k: pollKey
            if k = 27 [break]
   ]
  waitKey 0
   print "closing.."
   destroyAllWindows
   free :movie ;--we must free the memory: no GC for VideoCapture
   print "done"
```

As for as matrices, we can read *VideoCapture* properties with the *get-property* function. There a lot of properties for this class (see https://docs.opencv.org/4.6.0/d4/d15/group__videoio__flags__base.html#gaeb8dd9c89c10a5c63c139bf7c4f5704d)

But the *VideoCapture* object also allows to set specific properties of the image reader. In this sample we use a counter which is increased when displaying the successive images. When this counter is equal to the number of frames contained in the movie, we set the count to zero and we go back to the first image by setting the CAP_PROP_POS_FRAMES to zero with the *set-property* function. Unlike matrices that are managed by Rebol's garbage collector, the memory must be freed manually. This also allows to stop the camera when it is used with the *VideoCapture* object.

Reading Tiff files

Compressed and uncompressed 8-bit and 16-bit TIFFs images are easy to process. But in some cases, you can get images with specific values. In this case, the function *convertTo* can be used.

```
convertTo: [
        "Converts an array to another data type with optional scaling."
        src   [handle!] "cvMat"
        dst   [handle! none!] "cvMat"
        type  [integer! word!] "desired output matrix type or, rather, the depth since the number of channels are the same as the input has; if rtype is negative, the output matrix will have the same type as the input"
        alpha [number!] "scale factor"
        beta  [number!] "delta added to the scaled values"
]
```

The function converts an array to another data type with optional scaling. The method converts source pixel values to the target data type. OpenCV use a template function *saturate_cast* for accurate conversion from one primitive type to another. This function is applied at the end to avoid possible overflows.

$$m(x,y)=saturate\_cast<rType>(\alpha(*this)(x,y)+\beta)$$

For example, this file is 16-bit grayscale image with only 1 channel not correctly displayed because pixels values are in a 0..4095 range. So, we need to use an alpha scaling factor (256/4096) to correctly visualize the image.

```
Rebol [
]
cv: import 'opencv
with cv [
        mat: imread %../images/IMG_10007.tif
        print ["mat size    :" size: get-property mat MAT_SIZE]
        print ["mat type    :" type: get-property mat MAT_TYPE]
         print ["mat depth   :" depth: get-property mat MAT_DEPTH]
         print ["mat channels:" channels: get-property mat MAT_CHANNELS]
        convertTo :mat :mat CV_8U 0.0625 0 ;-- alpha = 256/4096
        imshow/name mat "16-bit Tiff"
        moveWindow "16-bit Tiff" 10x0
        waitKey 0
        destroyAllWindows
]
```

ConvertTo function is also efficient to display 32-bit TIFFs such as radiographic images.

```
mat size    : 610x600
mat type    : CV_32FC1
mat depth   : 5
mat channels: 1
After Convert
mat size    : 610x600
mat type    : CV_8UC1
mat depth   : 0
mat channels: 1
```

For 32-bit TIFFs file, the *saturate_cast* function makes the job, and the alpha parameter can be fixed to 1.
Sometimes, TIFFs files contain more than 1 image. In this case the *imreadmulti* function is useful. The function *imreadmulti* loads a multi-page image from the specified file into a block of Matrix objects.

```
imreadmulti: [
        src [file! string!] "Loads a multi-page image from a file."
        /image "As Rebol images instead of default cvMat"
        /with "Flag that can take values IMREAD_*"
        flags [integer!] "Default is IMREAD_UNCHANGED"
]
```

```
Rebol [
]
cv: import 'opencv
with cv [
        images: imreadmulti %../images/images.tiff
        n: length? images
        mat: Matrix :images/1
        imshow mat
        onTrackbar: func [val [integer!]][
                setWindowTitle "Image" join "Page: " val
                imshow images/:val
        ]
        ;--create the trackbar in the same window as the source image with the callback
        tb: createTrackbar/with "Value" "Image" n system/contexts/user 'onTrackbar
         setTrackbarMin tb 1                ;--for the first image
         waitKey 0                          ;--waits for any key
        destroyAllWindows
]
```

Since the pages are stored in a Rebol block it is simple to use the path notation to access any image. In this sample, we use trackbar object to read the pages according to the value returned by the trackbar callback.

Code Samples

You'll find a lot of samples organized by categories.

Directory	Description
/colorspace/	Use of *cvtColor* function to change image or matrix color space
/ highgui/	OpenCV highgui tests
/ image_basic/	Reading and displaying images
/ image_conversion/	Use *convertTo* function to convert images and matrices
/ image_filtering/	A lot of filters for images and matrices
/ image_flip/	How to flip an image
/ image_gabor/	Tests with Gabor filter
/ image_gif/	Gif images processing
/ image_misc/	Some tests
/image_morphology/	Morphological operators
/ image_operators/	Logical operators
/ image_QRCode/	Oldes's QRCode creating and reading qrcodes
/ image_resizing/	How to resize an image or a matrix
/ image_roi/	Regions of interest in image
/image_tests/	Random tests on image
/ image_thresholding/	Use of *threshold* function for image thresholding
/ image_tiff/	Tiff images processing
/ matrices/	All about matrices
/video/	Video processing

Thanks to Oldes for sharing this OpenCV extension. Please, enjoy it and have fun.

LDCI September 2024