Image file reading and writing

Modules

- C API
- Flags used for image file reading and writing
- iOS glue

Functions

Mat	cv::imdecode (InputArray buf, int flags) Reads an image from a buffer in memory. More...
Mat	cv::imdecode (InputArray buf, int flags, Mat *dst)
bool	cv::imencode (const String &ext, InputArray img, std::vector< uchar > &buf, const std::vector< int > &params=std::vector< int >()) Encodes an image into a memory buffer. More...
Mat	cv::imread (const String &filename, int flags=IMREAD_COLOR) Loads an image from a file. More...
bool	cv::imreadmulti (const String &filename, std::vector< Mat > &mats, int flags=IMREAD_ANYCOLOR) Loads a multi-page image from a file. More...
bool	cv::imwrite (const String &filename, InputArray img, const std::vector< int > &params=std::vector< int >()) Saves an image to a specified file. More...
static bool	cv::imwritemulti (const String &filename, InputArrayOfArrays img, const std::vector< int > &params=std::vector< int >())

Detailed Description

Function Documentation

◆ imdecode() [1/2]

Mat cv::imdecode (InputArray buf,
int flags
)

Python:
cv.imdecode(buf, flags) -> retval

#include <opencv2/imgcodecs.hpp>

Reads an image from a buffer in memory.

The function imdecode reads an image from the specified buffer in the memory. If the buffer is too short or contains invalid data, the function returns an empty matrix (Mat::data==NULL).

See cv::imread for the list of supported formats and flags description.

Note
In the case of color images, the decoded images will have the channels stored in **B G R** order.

Parameters
- **buf** Input array or vector of bytes.
- **flags** The same flags as in cv::imread, see cv::ImreadModes.

◆ imdecode() [2/2]

Mat cv::imdecode (InputArray buf,
int flags,
Mat * dst
)

Python:
cv.imdecode(buf, flags) -> retval

#include <opencv2/imgcodecs.hpp>

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters
- **buf**
- **flags**
- **dst** The optional output placeholder for the decoded matrix. It can save the image reallocations when the function is called repeatedly for images of the same size.

◆ imencode()

bool cv::imencode (const String & ext,
InputArray img,
std::vector< uchar > & buf,
const std::vector< int > & params = std::vector< int >()
)

Python:
cv.imencode(ext, img[, params]) -> retval, buf

#include <opencv2/imgcodecs.hpp>

Encodes an image into a memory buffer.

The function imencode compresses the image and stores it in the memory buffer that is resized to fit the result. See cv::imwrite for the list of supported formats and flags description.

Parameters
- **ext** File extension that defines the output format. Must include a leading period.
- **img** Image to be written.
- **buf** Output buffer resized to fit the compressed image.
- **params** Format-specific parameters. See cv::imwrite and cv::ImwriteFlags.

◆ imread()

Mat cv::imread (const String & filename,
int flags = IMREAD_COLOR
)

Python:
cv.imread(filename[, flags]) -> retval

#include <opencv2/imgcodecs.hpp>

Loads an image from a file.

The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (Mat::data==NULL).

Currently, the following file formats are supported:

- Windows bitmaps - *.bmp, *.dib (always supported)
- JPEG files - *.jpeg, *.jpg, *.jpe (see the *Note* section)
- JPEG 2000 files - *.jp2 (see the *Note* section)
- Portable Network Graphics - *.png (see the *Note* section)
- WebP - *.webp (see the *Note* section)
- Portable image format - *.pbm, *.pgm, *.ppm *.pxm, *.pnm (always supported)
- Sun rasters - *.sr, *.ras (always supported)
- TIFF files - *.tiff, *.tif (see the *Note* section)
- OpenEXR Image files - *.exr (see the *Note* section)
- Radiance HDR - *.hdr, *.pic (always supported)
- Raster and Vector geospatial data supported by GDAL (see the *Note* section)

Note
- The function determines the type of an image by the content, not by the file extension.
- In the case of color images, the decoded images will have the channels stored in **B G R** order.
- When using IMREAD_GRAYSCALE, the codec's internal grayscale conversion will be used, if available. Results may differ to the output of cvtColor()
- On Microsoft Windows* OS and MacOSX*, the codecs shipped with an OpenCV image (libjpeg, libpng, libtiff, and libjasper) are used by default. So, OpenCV can always read JPEGs, PNGs, and TIFFs. On MacOSX, there is also an option to use native MacOSX image readers. But beware that currently these native image loaders give images with different pixel values because of the color management embedded into MacOSX.
- On Linux*, BSD flavors and other Unix-like open-source operating systems, OpenCV looks for codecs supplied with an OS image. Install the relevant packages (do not forget the development files, for example, "libjpeg-dev", in Debian* and Ubuntu*) to get the codec support or turn on the OPENCV_BUILD_3RDPARTY_LIBS flag in CMake.
- In the case you set *WITH_GDAL* flag to true in CMake and IMREAD_LOAD_GDAL to load the image, then the GDAL driver will be used in order to decode the image, supporting the following formats: Raster, Vector.
- If EXIF information is embedded in the image file, the EXIF orientation will be taken into account and thus the image will be rotated accordingly except if the flags IMREAD_IGNORE_ORIENTATION or IMREAD_UNCHANGED are passed.
- By default number of pixels must be less than 2^30. Limit can be set using system variable OPENCV_IO_MAX_IMAGE_PIXELS

Parameters
- **filename** Name of file to be loaded.
- **flags** Flag that can take values of cv::ImreadModes

Examples:
fld_lines.cpp, samples/cpp/connected_components.cpp, samples/cpp/create_mask.cpp, samples/cpp/demhist.cpp, samples/cpp/distrans.cpp, samples/cpp/edge.cpp, samples/cpp/facedetect.cpp, samples/cpp/falsecolor.cpp, samples/cpp/ffilldemo.cpp, samples/cpp/fitellipse.cpp, samples/cpp/grabcut.cpp, samples/cpp/image_alignment.cpp, samples/cpp/lsd_lines.cpp, samples/cpp/pca.cpp, samples/cpp/shape_example.cpp, samples/cpp/squares.cpp, samples/cpp/stitching.cpp, samples/cpp/stitching_detailed.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp, samples/cpp/tutorial_code/HighGUI/AddingImagesTrackbar.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/tutorial_code/ImgProc/Morphology_1.cpp, samples/cpp/tutorial_code/ImgProc/Morphology_2.cpp, samples/cpp/tutorial_code/ImgProc/Pyramids/Pyramids.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, samples/cpp/tutorial_code/ImgTrans/copyMakeBorder_demo.cpp, samples/cpp/tutorial_code/ImgTrans/houghcircles.cpp, samples/cpp/tutorial_code/ImgTrans/houghlines.cpp, samples/cpp/tutorial_code/ImgTrans/Sobel_Demo.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, samples/cpp/tutorial_code/photo/non_photorealistic_rendering/npr_demo.cpp, samples/cpp/tutorial_code/photo/seamless_cloning/cloning_demo.cpp, samples/cpp/warpPerspective_demo.cpp, samples/cpp/watershed.cpp, samples/dnn/colorization.cpp, samples/dnn/openpose.cpp, samples/tapi/hog.cpp, and samples/tapi/squares.cpp.

◆ imreadmulti()

bool cv::imreadmulti (const String & filename,
std::vector< Mat > & mats,
int flags = IMREAD_ANYCOLOR
)

Python:
cv.imreadmulti(filename[, mats[, flags]]) -> retval, mats

#include <opencv2/imgcodecs.hpp>

Loads a multi-page image from a file.

The function imreadmulti loads a multi-page image from the specified file into a vector of Mat objects.

Parameters
- **filename** Name of file to be loaded.
- **mats** A vector of Mat objects holding each page.
- **flags** Flag that can take values of cv::ImreadModes, default with cv::IMREAD_ANYCOLOR.

See also
cv::imread

◆ imwrite()

bool cv::imwrite (const String & filename,
InputArray img,
const std::vector< int > & params = std::vector< int >()
)

Python:
cv.imwrite(filename, img[, params]) -> retval

#include <opencv2/imgcodecs.hpp>

Saves an image to a specified file.

The function imwrite saves the image to the specified file. The image format is chosen based on the filename extension (see cv::imread for the list of extensions). In general, only 8-bit single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function, with these exceptions:

- 16-bit unsigned (CV_16U) images can be saved in the case of PNG, JPEG 2000, and TIFF formats
- 32-bit float (CV_32F) images can be saved in TIFF, OpenEXR, and Radiance HDR formats; 3-channel (CV_32FC3) TIFF images will be saved using the LogLuv high dynamic range encoding (4 bytes per pixel)
- PNG images with an alpha channel can be saved using this function. To do this, create 8-bit (or 16-bit) 4-channel image BGRA, where the alpha channel goes last. Fully transparent pixels should have alpha set to 0, fully opaque pixels should have alpha set to 255/65535 (see the code sample below).
- Multiple images (vector of Mat) can be saved in TIFF format (see the code sample below).

If the image format is not supported, the image will be converted to 8-bit unsigned (CV_8U) and saved that way.

If the format, depth or channel order is different, use Mat::convertTo and cv::cvtColor to convert it before saving. Or, use the universal FileStorage I/O functions to save the image to XML or YAML format.

The sample below shows how to create a BGRA image, how to set custom compression parameters and save it to a PNG file. It also demonstrates how to save multiple images in a TIFF file:

```
#include <opencv2/imgcodecs.hpp>

using namespace cv;
using namespace std;

static void paintAlphaMat(Mat &mat)
{
    CV_Assert(mat.channels() == 4);
    for (int i = 0; i < mat.rows; ++i)
    {
        for (int j = 0; j < mat.cols; ++j)
        {
            Vec4b& bgra = mat.at<Vec4b>(i, j);
            bgra[0] = UCHAR_MAX; // Blue
            bgra[1] = saturate_cast<uchar>((float (mat.cols - j)) / ((float)mat.cols) * UCHAR_MAX); // Green
            bgra[2] = saturate_cast<uchar>((float (mat.rows - i)) / ((float)mat.rows) * UCHAR_MAX); // Red
            bgra[3] = saturate_cast<uchar>(0.5 * (bgra[1] + bgra[2])); // Alpha
        }
    }
}

int main()
{
    Mat mat(480, 640, CV_8UC4); // Create a matrix with alpha channel
    paintAlphaMat(mat);

    vector<int> compression_params;
    compression_params.push_back(IMWRITE_PNG_COMPRESSION);
    compression_params.push_back(9);

    bool result = false;
    try
    {
        result = imwrite("alpha.png", mat, compression_params);
    }
    catch (const cv::Exception& ex)
    {
        fprintf(stderr, "Exception converting image to PNG format: %s\n", ex.what());
    }

    if (result)
        printf("Saved PNG file with alpha data.\n");
    else
        printf("ERROR: Can't save PNG file.\n");

    vector<Mat> imgs;
    imgs.push_back(mat);
    imgs.push_back(~mat);
    imgs.push_back(mat(Rect(0, 0, mat.cols / 2, mat.rows / 2)));
    imwrite("test.tiff", imgs);
    printf("Multiple files saved in test.tiff\n");

    return result ? 0 : 1;
}
```

Parameters
- **filename** Name of the file.
- **img** (Mat or vector of Mat) Image or Images to be saved.
- **params** Format-specific parameters encoded as pairs (paramId_1, paramValue_1, paramId_2, paramValue_2, ...) see cv::ImwriteFlags

Examples:
samples/cpp/image_alignment.cpp, samples/cpp/stitching.cpp, samples/cpp/stitching_detailed.cpp, samples/cpp/tutorial_code/photo/seamless_cloning/cloning_demo.cpp, samples/tapi/hog.cpp, and samples/tapi/squares.cpp.

◆ imwritemulti()

static bool cv::imwritemulti (const String & filename,
InputArrayOfArrays img,
const std::vector< int > & params = std::vector<int>()
) inline static

Python:
cv.imwritemulti(filename, img[, params]) -> retval

#include <opencv2/imgcodecs.hpp>

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.