High level stitching API (Stitcher class)

Prev Tutorial: High Dynamic Range Imaging
Next Tutorial: How to Use Background Subtraction Methods

Original author	Jiri Horner
Compatibility	OpenCV >= 3.2

Goal

In this tutorial you will learn how to:

- use the high-level stitching API for stitching provided by
  - **cv::Stitcher**
- learn how to use preconfigured Stitcher configurations to stitch images using different camera models.

Code

C++ Python

This tutorial's code is shown in the lines below. You can download it from here.

Note: The C++ version includes additional options such as image division (–d3) and more detailed error handling, which are not present in the Python example.

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/stitching.hpp"

#include <iostream>

using namespace std;
using namespace cv;

bool divide_images = false;
Stitcher::Mode mode = Stitcher::PANORAMA;
vector<Mat> imgs;
string result_name = "result.jpg";

void printUsage(char** argv);
int parseCmdArgs(int argc, char** argv);

int main(int argc, char* argv[])
{
    int retval = parseCmdArgs(argc, argv);
    if (retval) return EXIT_FAILURE;

    Mat pano;
    Ptr<Stitcher> stitcher = Stitcher::create(mode);
    Stitcher::Status status = stitcher->stitch(imgs, pano);

    if (status != Stitcher::OK)
    {
        cout << "Can't stitch images, error code = " << int(status) << endl;
        return EXIT_FAILURE;
    }

    imwrite(result_name, pano);
    cout << "stitching completed successfully\n" << result_name << " saved!";
    return EXIT_SUCCESS;
}


void printUsage(char** argv)
{
    cout <<
         "Images stitcher.\n\n" << "Usage :\n" << argv[0] <<" [Flags] img1 img2 [...imgN]\n\n"
         "Flags:\n"
         "  --d3\n"
         "      internally creates three chunks of each image to increase stitching success\n"
         "  --mode (panorama|scans)\n"
         "      Determines configuration of stitcher. The default is 'panorama',\n"
         "      mode suitable for creating photo panoramas. Option 'scans' is suitable\n"
         "      for stitching materials under affine transformation, such as scans.\n"
```

```
        "  --output <result_img>\n"
        "      The default is 'result.jpg'.\n\n"
        "Example usage :\n" << argv[0] << " --d3 --mode scans img1.jpg img2.jpg\n";
}


int parseCmdArgs(int argc, char** argv)
{
    if (argc == 1)
    {
        printUsage(argv);
        return EXIT_FAILURE;
    }

    for (int i = 1; i < argc; ++i)
    {
        if (string(argv[i]) == "--help" || string(argv[i]) == "/?")
        {
            printUsage(argv);
            return EXIT_FAILURE;
        }
        else if (string(argv[i]) == "--d3")
        {
            divide_images = true;
        }
        else if (string(argv[i]) == "--output")
        {
            result_name = argv[i + 1];
            i++;
        }
        else if (string(argv[i]) == "--mode")
        {
            if (string(argv[i + 1]) == "panorama")
                mode = Stitcher::PANORAMA;
            else if (string(argv[i + 1]) == "scans")
                mode = Stitcher::SCANS;
            else
            {
                cout << "Bad --mode flag value\n";
                return EXIT_FAILURE;
            }
            i++;
        }
        else
        {
            Mat img = imread(samples::findFile(argv[i]));
            if (img.empty())
            {
                cout << "Can't read image '" << argv[i] << "'\n";
                return EXIT_FAILURE;
            }

            if (divide_images)
            {
                Rect rect(0, 0, img.cols / 2, img.rows);
                imgs.push_back(img(rect).clone());
                rect.x = img.cols / 3;
                imgs.push_back(img(rect).clone());
                rect.x = img.cols / 2;
                imgs.push_back(img(rect).clone());
            }
            else
                imgs.push_back(img);
        }
    }
    return EXIT_SUCCESS;
}
```

Explanation

C++ Python

The most important code part is:

```
    Mat pano;
    Ptr<Stitcher> stitcher = Stitcher::create(mode);
    Stitcher::Status status = stitcher->stitch(imgs, pano);

    if (status != Stitcher::OK)
    {
        cout << "Can't stitch images, error code = " << int(status) << endl;
        return EXIT_FAILURE;
    }
```

A new instance of stitcher is created and the **cv::Stitcher::stitch** will do all the hard work.

cv::Stitcher::create can create stitcher in one of the predefined configurations (argument `mode`). See **cv::Stitcher::Mode** for details. These configurations will setup multiple stitcher properties to operate in one of predefined scenarios. After you create stitcher in one of predefined configurations you can adjust stitching by setting any of the stitcher properties.

If you have cuda device **cv::Stitcher** can be configured to offload certain operations to GPU. If you prefer this configuration set `try_use_gpu` to true. OpenCL acceleration will be used transparently based on global OpenCV settings regardless of this flag.

Stitching might fail for several reasons, you should always check if everything went good and resulting pano is stored in `pano`. See **cv::Stitcher::Status** documentation for possible error codes.

Camera models

There are currently 2 camera models implemented in stitching pipeline.

- *Homography model* expecting perspective transformations between images implemented in **cv::detail::BestOf2NearestMatcher** **cv::detail::HomographyBasedEstimator** **cv::detail::BundleAdjusterReproj** **cv::detail::BundleAdjusterRay**
- *Affine model* expecting affine transformation with 6 DOF or 4 DOF implemented in **cv::detail::AffineBestOf2NearestMatcher** **cv::detail::AffineBasedEstimator** **cv::detail::BundleAdjusterAffine** **cv::detail::BundleAdjusterAffinePartial** **cv::AffineWarper**

Homography model is useful for creating photo panoramas captured by camera, while affine-based model can be used to stitch scans and object captured by specialized devices.

> **Note**
> Certain detailed settings of **cv::Stitcher** might not make sense. Especially you should not mix classes implementing affine model and classes implementing Homography model, as they work with different transformations.

Try it out

If you enabled building samples you can found binary under `build/bin/cpp-example-stitching`. This example is a console application, run it without arguments to see help. `opencv_extra` provides some sample data for testing all available configurations.

to try panorama mode run:

```
./cpp-example-stitching --mode panorama <path to opencv_extra>/testdata/stitching/boat*
```

to try scans mode run (dataset from home-grade scanner):

```
./cpp-example-stitching --mode scans <path to opencv_extra>/testdata/stitching/newspaper*
```

or (dataset from professional book scanner):

```
./cpp-example-stitching --mode scans <path to opencv_extra>/testdata/stitching/budapest*
```

Note

Examples above expects POSIX platform, on windows you have to provide all files names explicitly (e.g. `boat1.jpg boat2.jpg...`) as windows command line does not support * expansion.

Stitching detailed (python opencv >4.0.1)

C++ Python

If you want to study internals of the stitching pipeline or you want to experiment with detailed configuration you can use stitching_detailed source code available in C++ or python

stitching_detailed

stitching_detailed.cpp

stitching_detailed program uses command line to get stitching parameter. Many parameters exists. Above examples shows some command line parameters possible :

boat5.jpg boat2.jpg boat3.jpg boat4.jpg boat1.jpg boat6.jpg –work_megapix 0.6 –features orb –matcher homography –estimator homography –match_conf 0.3 –conf_thresh 0.3 –ba ray –ba_refine_mask xxxxx –save_graph test.txt –wave_correct no –warp fisheye –blend multiband –expos_comp no –seam gc_colorgrad

Pairwise images are matched using an homography –matcher homography and estimator used for transformation estimation too –estimator homography

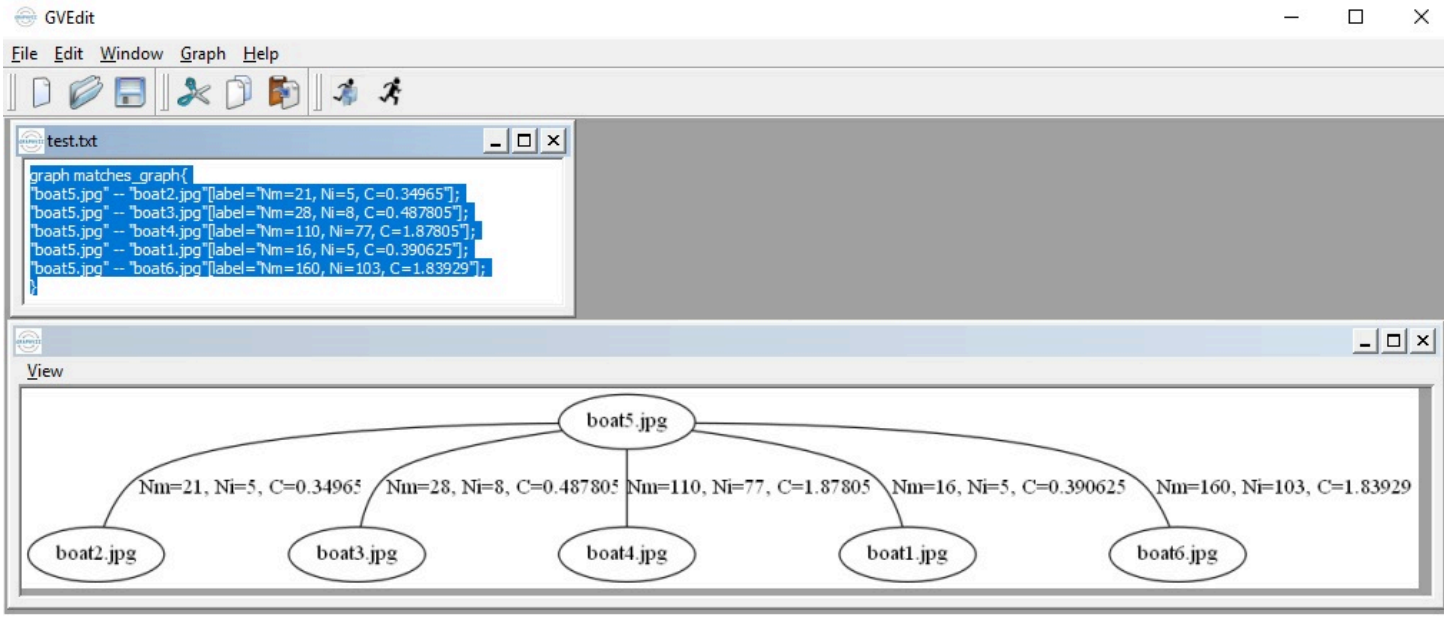
Confidence for feature matching step is 0.3 : –match_conf 0.3. You can decrease this value if you have some difficulties to match images

Threshold for two images are from the same panorama confidence is 0. : –conf_thresh 0.3 You can decrSTXease this value if you have some difficulties to match images

Bundle adjustment cost function is ray –ba ray

Refinement mask for bundle adjustment is xxxxx (–ba_refine_mask xxxxx) where 'x' means refine respective parameter and '_' means don't. Refine one, and has the following format: fx,skew,ppx,aspect,ppy

Save matches graph represented in DOT language to test.txt (–save_graph test.txt) : Labels description: Nm is number of matches, Ni is number of inliers, C is confidence



Perform wave effect correction is no (–wave_correct no)

Warp surface type is fisheye (–warp fisheye)

Blending method is multiband (–blend multiband)

Exposure compensation method is not used (–expos_comp no)

Seam estimation estimator is Minimum graph cut-based seam (–seam gc_colorgrad)

you can use those arguments on command line too :

boat5.jpg boat2.jpg boat3.jpg boat4.jpg boat1.jpg boat6.jpg –work_megapix 0.6 –features orb –matcher homography –estimator homography –match_conf 0.3 –conf_thresh 0.3 –ba ray –ba_refine_mask xxxxx –wave_correct horiz –warp compressedPlaneA2B1 –blend multiband –expos_comp channels_blocks –seam gc_colorgrad

You will get :

For images captured using a scanner or a drone (affine motion) you can use those arguments on command line :

newspaper1.jpg newspaper2.jpg –work_megapix 0.6 –features surf –matcher affine –estimator affine –match_conf 0.3 –conf_thresh 0.3 –ba affine –ba_refine_mask xxxxx –wave_correct no –warp affine

You can find all images in https://github.com/opencv/opencv_extra/tree/4.x/testdata/stitching