# Opencv\_CUDA开发环境配置

## 1.1 下载地址

Nividia显卡驱动下载：https:/www.nvidia.cn/Download/index.aspx?lang=cn

Cuda下载：<https://developer.nvidia.com/cuda-downloads>

CUDNN下载：https://developer.nvidia.com/cudnn-download-survey

Opencv下载：<https://opencv.org/>

opencv拓展包下载：https://github.com/opencv/opencv\_contrib/tags

Cmake：<https://cmake.org>

## 1.2 编译（以opencv4.1.0\_cuda10.1为例）

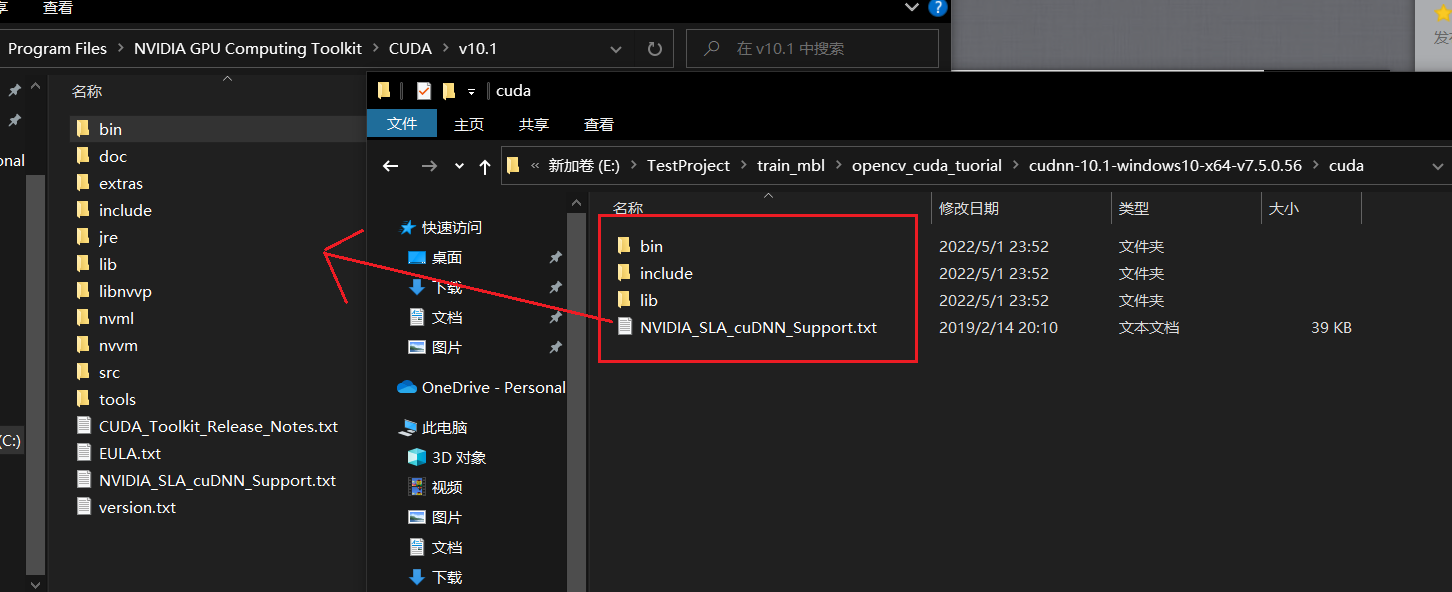
### 1.2.1 安装显卡驱动

在官网下载自己显卡所对应的驱动



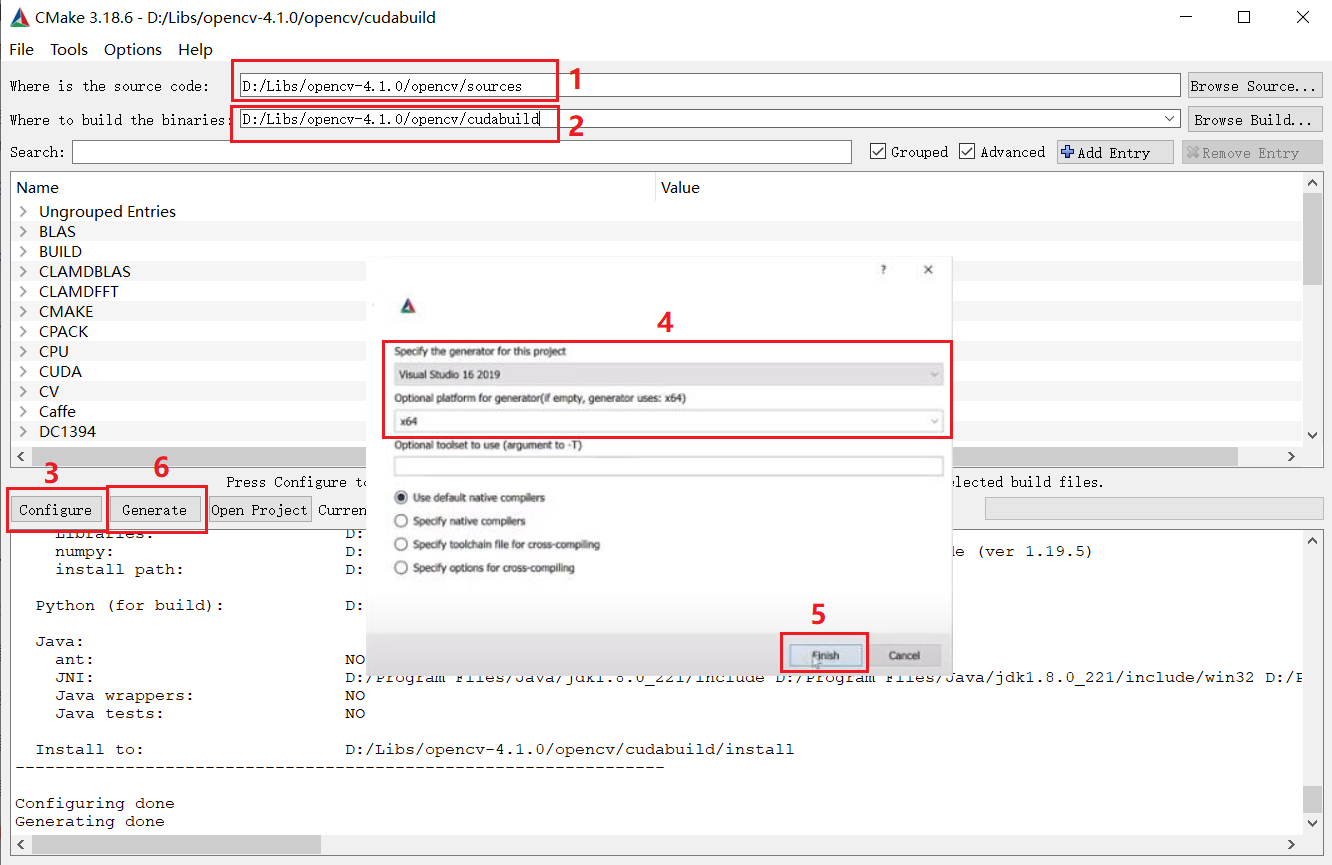
### 1.2.2 安装CUDA10.1与CUDNN

下载安装完CUDA10.1后，将CUDNN解压，将CUDNN的根目录复制到CUDA10.1的安装目录中



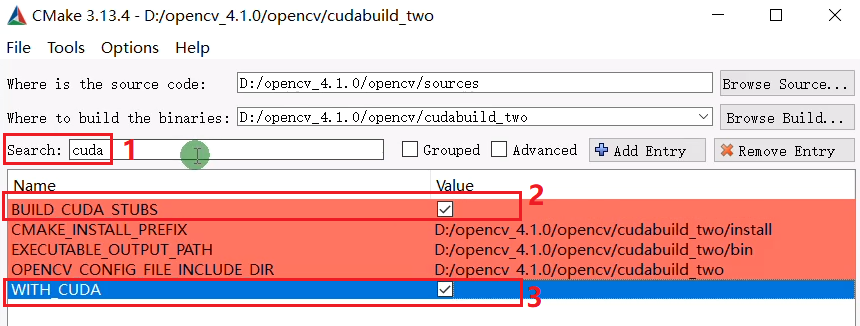
### 1.2.3 cmake编译opencv\_cuda

（1）打Cmake，选择源代码目录（source code）和编译输出目录（build the binaries），选择对应的Visual Studio版本，进行配置和生成，完成opencv源码编译。

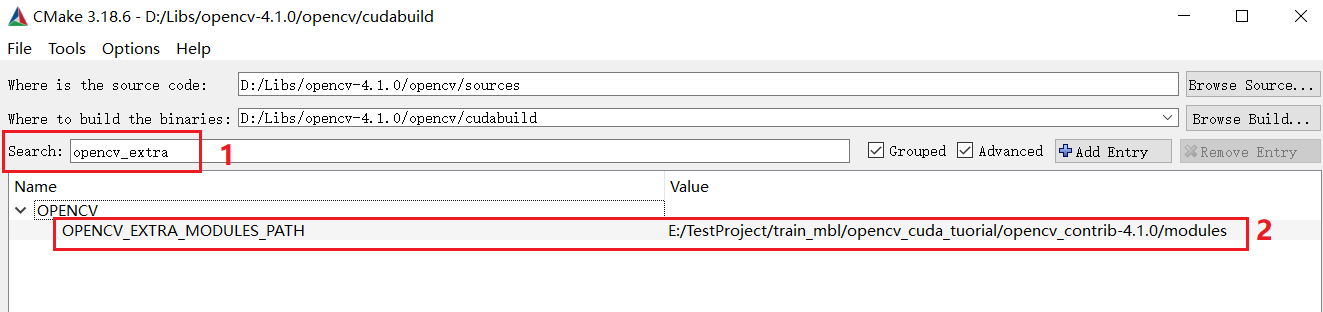


（2）选择opencv\_cuda扩展包选项，再编译

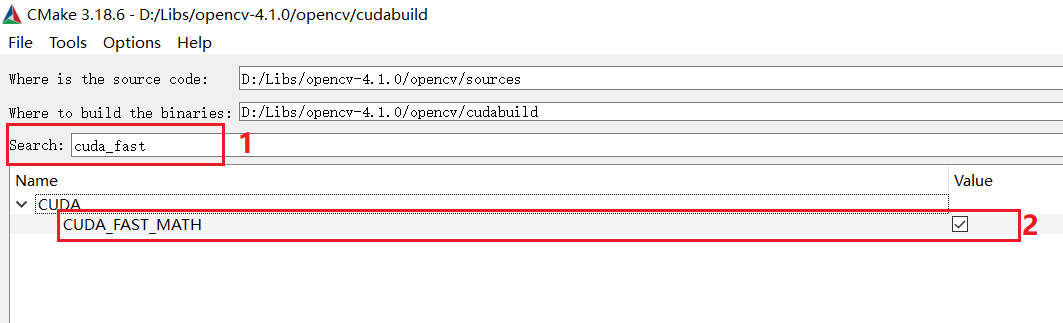
搜索并勾选BUILD\_CUDA\_STUBS和WITH\_CUDA。



OPENCV\_EXTRA\_MODULES\_PATH选择下载的opencv扩展包里modules的路径，点击配置（Configure）



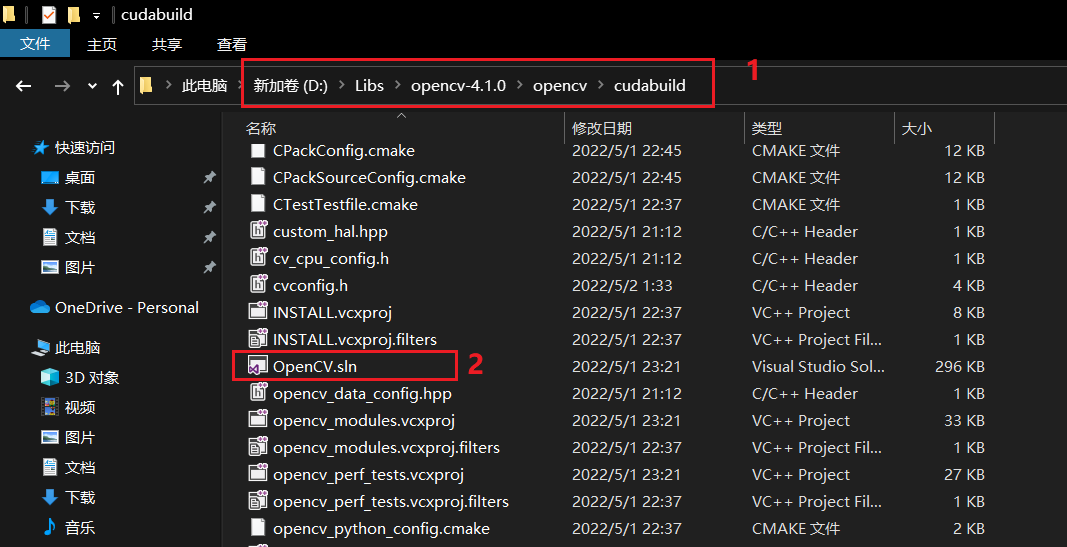
搜索并勾选cuda\_fast\_math，并点击生成（Generate）



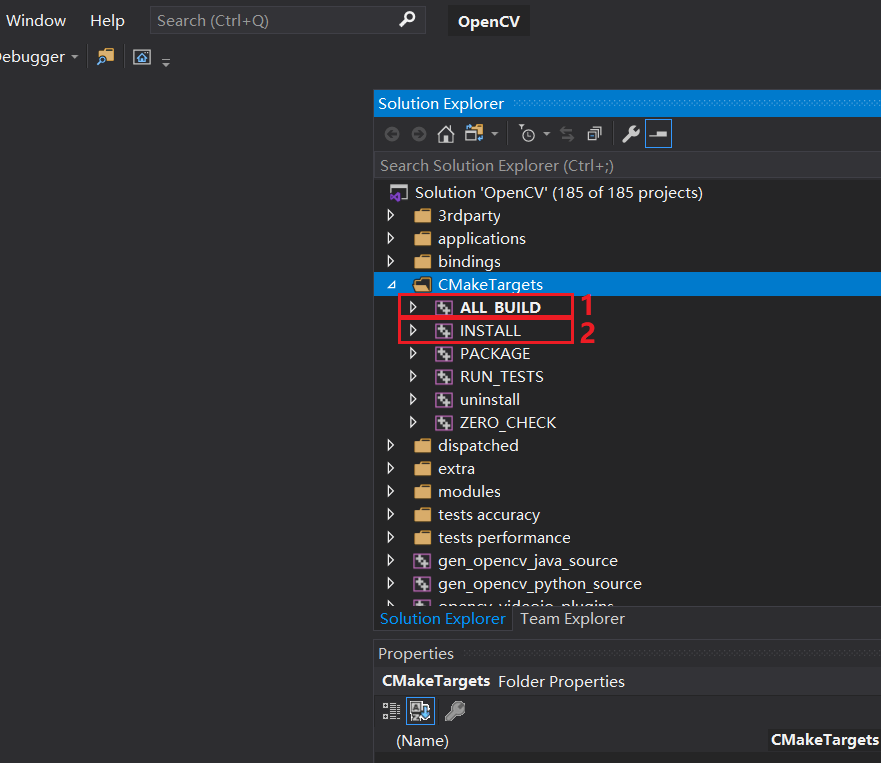
中间没有报错，正确提示Configuring done和Generating done后，CMake部分编译完成。

### 1.2.4 使用Visual Studio编译opencv\_cuda

(1)找到CMake的编译输出目录，打开Opencv.sln



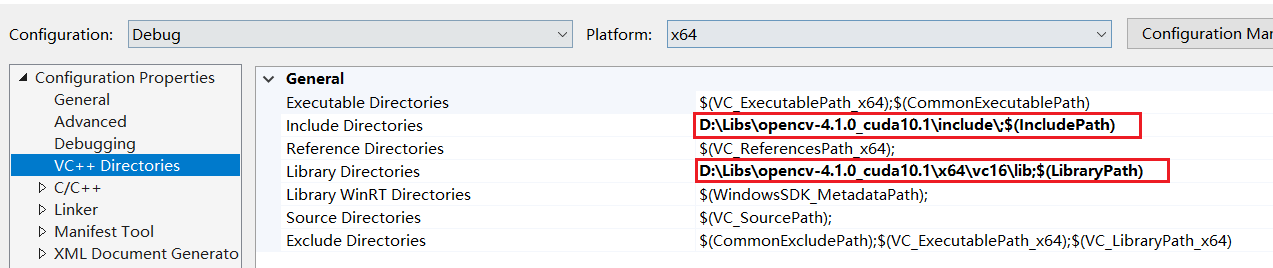
（2）分别对CMakeTargets下的ALL\_BUILD和INSTALL进行编译



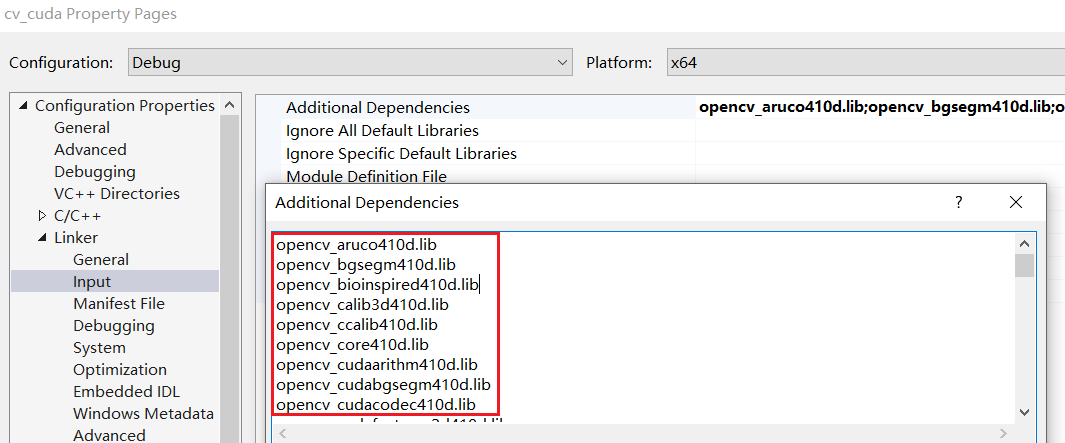
（3）完成编译，检查编译输出目录下的install里是否正确生成opencv\_cuda相关的库。可删除install文件夹外其它的所有文件。

## 1.3 配置opencv4.1.0\_cuda10.1运行环境

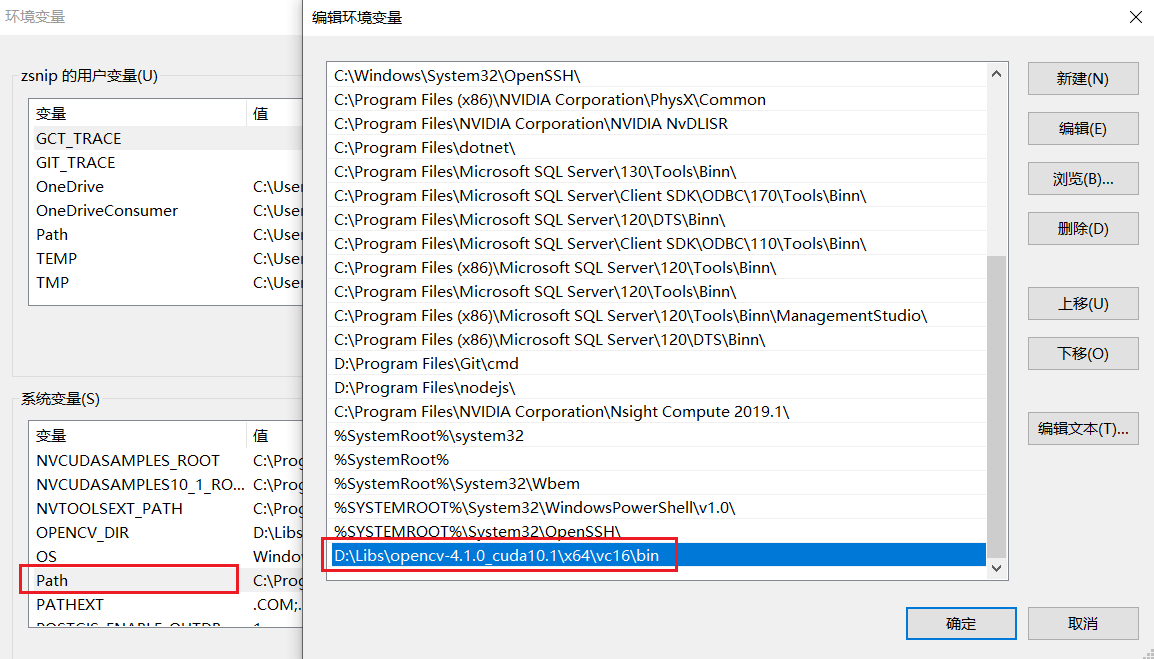
（1）配置Include和Library地址



（2）将lib文件夹里的\*d.lib（Debug的静态库）名全部导入Linker->Input->Additional Dependencies



（3）配置动态库环境变量



（4）运行测试代码（cuda\_tutorial）

#include <iostream>

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

using namespace cv;

using namespace cv::cuda;

using namespace std;

int main()

{

    cuda::printCudaDeviceInfo(cuda::getDevice());

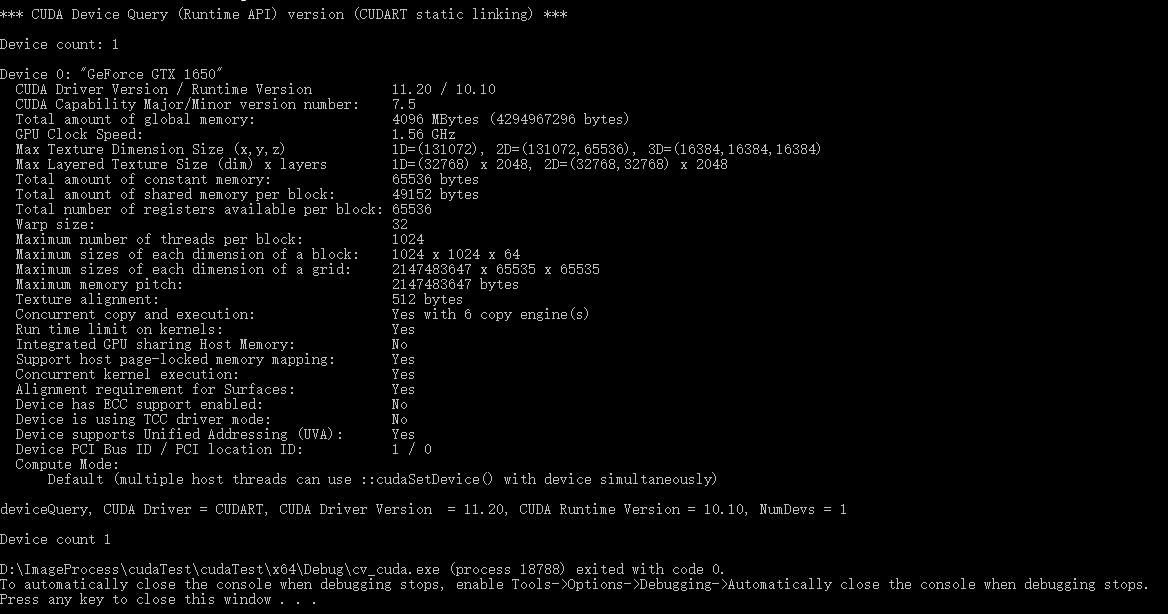
    int count = getCudaEnabledDeviceCount();

    printf("Device count %d \n",count);

    return 0;

}

正确输出如下结果，则表示环境配置成功



# Opencv\_CUDA编程

## 2.1 基本操作

参见cuda\_001项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

int main(int argc, char\*\* argv) {

    Mat image\_host = imread("../data/1.jpg");

    imshow("input", image\_host);

    //图像灰度化

    GpuMat image;

    GpuMat gray;

    image.upload(image\_host);

    cuda::cvtColor(image, gray, COLOR\_BGR2GRAY);

    Mat gray\_host;

    gray.download(gray\_host);

    imshow("gray", gray\_host);

    waitKey(0);

    return 0;

}

## 2.2 算术与位运算

参见cuda\_002项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

int main(int argc, char\*\* argv) {

    Mat src1\_host = imread("../data/WindowsLogo.jpg");

    Mat src2\_host = imread("../data/LinuxLogo.jpg");

    imshow("input1", src1\_host);

    imshow("input2", src2\_host);

    // GPU 对象

    GpuMat src1, src2, dst;

    src1.upload(src1\_host);

    src2.upload(src2\_host);

    cuda::add(src1, src2, dst);

    // cuda::subtract(src1, src2, dst);

    // cuda::multiply(src1, src2, dst);

    Mat result;

    dst.download(result);

    imshow("result", result);

    // 权重加减

    Mat src\_host = imread("../data/building.jpg");

    imshow("input", src\_host);

    GpuMat src;

    src.upload(src\_host);

    GpuMat blank = GpuMat(src.size(), src.type());

    cuda::addWeighted(src1, 0.5, src2, 0.5, 0, dst);

    cuda::bitwise\_not(dst, dst);

    dst.download(result);

    imshow("weigthed add", result);

    GpuMat hsv, rgb, gray, YCrCb;

    cuda::cvtColor(src, hsv, COLOR\_BGR2HSV);

    cuda::cvtColor(src, rgb, COLOR\_BGR2RGB);

    cuda::cvtColor(src, gray, COLOR\_BGR2GRAY);

    cuda::cvtColor(src, YCrCb, COLOR\_BGR2YCrCb);

    Mat hsv\_host, rgb\_host, gray\_host, YCrCb\_host;

    hsv.download(hsv\_host);

    rgb.download(rgb\_host);

    gray.download(gray\_host);

    YCrCb.download(YCrCb\_host);

    imshow("hsv", hsv\_host);

    imshow("rgb", rgb\_host);

    imshow("gray", gray\_host);

    imshow("YCrCb", YCrCb\_host);

    waitKey(0);

    return 0;

}

## 2.3 图像直方图

参见cuda\_003项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

int main(int argc, char\*\* argv) {

    Mat image\_host = imread("../data/chicky\_512.png");

    imshow("input", image\_host);

    GpuMat image(image\_host);

    vector<GpuMat> mv;

    GpuMat hist, hsv;

    cuda::split(image, mv);

    cuda::calcHist(mv[2], hist);

    Mat hist\_host;

    hist.download(hist\_host);

    for (int i = 0; i < hist\_host.cols; i++) {

        int pv = hist\_host.at<int>(0, i);

        printf("total number : %d, of the pixel value : %d \n", pv, i);

    }

    cuda::cvtColor(image, hsv, COLOR\_BGR2HSV);

    cuda::split(hsv, mv);

    cuda::equalizeHist(mv[2], mv[2]);

    cuda::merge(mv, hsv);

    cuda::cvtColor(hsv, image, COLOR\_HSV2BGR);

    Mat result;

    image.download(result);

    imshow("eq-demo result", result);

    // resize and rotate

    GpuMat dst;

    cuda::resize(image, dst, Size(0, 0), 2, 2, INTER\_CUBIC);

    dst.download(result);

    imshow("resize result", result);

    int cx = image.cols / 2;

    int cy = image.rows / 2;

    Mat M = getRotationMatrix2D(Point(cx, cy), 45, 1.0);

    cuda::warpAffine(image, dst, M, image.size());

    dst.download(result);

    imshow("rotate result", result);

    waitKey(0);

    return 0;

}

## 2.4 卷积操作

参见cuda\_004项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

int main(int argc, char\*\* argv) {

    Mat image\_host = imread("../data/lena.jpg");

    imshow("input", image\_host);

    GpuMat image, d\_result3x3, d\_result5x5, d\_result9x9;

    image.upload(image\_host);

    cuda::cvtColor(image, image, COLOR\_BGR2BGRA);

    // create box filter

    // auto filter\_3x3 = cuda::createBoxFilter(image.type(), image.type(), Size(3, 3), Point(-1, -1));

    // auto filter\_5x5 = cuda::createBoxFilter(image.type(), image.type(), Size(5, 5), Point(-1, -1));

    // auto filter\_9x9 = cuda::createBoxFilter(image.type(), image.type(), Size(9, 9), Point(-1, -1));

    // create gaussian filter

    auto filter\_3x3 = cuda::createGaussianFilter(image.type(), image.type(), Size(5, 5), 5);

    auto filter\_5x5 = cuda::createGaussianFilter(image.type(), image.type(), Size(15, 15), 15);

    auto filter\_9x9 = cuda::createGaussianFilter(image.type(), image.type(), Size(25, 25), 25);

    // apply them

    filter\_3x3->apply(image, d\_result3x3);

    filter\_5x5->apply(image, d\_result5x5);

    filter\_9x9->apply(image, d\_result9x9);

    // image gradient

    auto sobel\_dx = cuda::createSobelFilter(image.type(), image.type(), 1, 0, 3);

    auto sobel\_dy = cuda::createSobelFilter(image.type(), image.type(), 0, 1, 3);

    GpuMat grad\_x, grad\_y, gradxy;

    sobel\_dx->apply(image, grad\_x);

    sobel\_dy->apply(image, grad\_y);

    cuda::addWeighted(grad\_x, 0.5, grad\_y, 0.5, 0, gradxy);

    Mat grad\_host;

    gradxy.download(grad\_host);

    imshow("gradient demo", grad\_host);

    // 边缘提取

    GpuMat gray, edges;

    cuda::cvtColor(image, gray, COLOR\_BGRA2GRAY);

    // auto edge\_detector = cuda::createCannyEdgeDetector(50, 150, 3, true);

    // edge\_detector->detect(gray, edges);

    auto laplacian\_filter = cuda::createLaplacianFilter(gray.type(), gray.type(), 3, 1.0);

    laplacian\_filter->apply(gray, edges);

    Mat edges\_host;

    edges.download(edges\_host);

    imshow("Canny Edge Demo", edges\_host);

    // 下载数据

    Mat result3, result5, result9;

    d\_result3x3.download(result3);

    d\_result5x5.download(result5);

    d\_result9x9.download(result9);

    // 显示数据

    imshow("filter3x3 result", result3);

    imshow("filter5x5 result", result5);

    imshow("filter9x9 result", result9);

    waitKey(0);

    return 0;

}

## 2.5 高斯双边

参见cuda\_005项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

void cpu\_demo();

int main(int argc, char\*\* argv) {

    //Mat image\_host = imread("../data/lena.jpg");

    //imshow("input", image\_host);

    // cpu\_demo();

    VideoCapture cap;

    cap.open("../data/Megamind.avi");

    Mat frame,  result;

    GpuMat image;

    GpuMat dst;

    while (true) {

        int64 start = getTickCount();

        bool ret = cap.read(frame);

        if (!ret) break;

        image.upload(frame);

        cuda::cvtColor(image, image, COLOR\_BGR2BGRA);

        cuda::bilateralFilter(image, dst, 0, 100, 10);

        // cuda::meanShiftFiltering(image, dst, 7, 50);

        dst.download(result);

        double fps = getTickFrequency() / (getTickCount() - start);

        putText(result, format("FPS: %.2f", fps), Point(50, 50), FONT\_HERSHEY\_SIMPLEX, 1.0, Scalar(0, 0, 255), 2, 8);

        imshow("GPU-demo", result);

        char c = waitKey(1);

        if (c == 27) {

            break;

        }

    }

    waitKey(0);

    return 0;

}

void cpu\_demo() {

    VideoCapture cap;

    cap.open("../data/Megamind.avi");

    Mat frame, result;

    while (true) {

        int64 start = getTickCount();

        cap.read(frame);

        cv::bilateralFilter(frame, result, 0, 100, 14, 4);

        double fps = getTickFrequency() / (getTickCount() - start);

        putText(result, format("FPS: %.2f", fps), Point(50, 50), FONT\_HERSHEY\_SIMPLEX, 1.0, Scalar(0, 0, 255), 2, 8);

        imshow("CPU-demo", result);

        char c = waitKey(1);

        if (c == 27) {

            break;

        }

    }

    waitKey(0);

    return;

}

## 2.6二值形态学操作、颜色对象跟踪

参见cuda\_006项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

void morph\_analysis\_demo();

int main(int argc, char\*\* argv) {

    //morph\_analysis\_demo();

    Mat gray\_host = imread("../data/morph.png", 0);

    imshow("input", gray\_host);

    GpuMat gray, binary, dst;

    gray.upload(gray\_host);

    cuda::threshold(gray, binary, 174, 255, THRESH\_BINARY\_INV);

    Mat se = cv::getStructuringElement(MORPH\_RECT, Size(3, 3));

    auto morph\_filter = cuda::createMorphologyFilter(MORPH\_OPEN, gray.type(), se);

    morph\_filter->apply(binary, dst);

    // cuda::subtract(binary, dst, binary);

    Mat result;

    dst.download(result);

    imshow("binary", result);

    waitKey(0);

    return 0;

}

void morph\_analysis\_demo() {

    VideoCapture cap;

    cap.open("../data/vtest.avi");

    Mat frame\_host, binary;

    GpuMat frame, hsv, mask;

    vector<GpuMat> mv;

    vector<GpuMat> thres(4);

    while (true) {

        int64 start = getTickCount();

        bool ret = cap.read(frame\_host);

        if (!ret) break;

        imshow("frame", frame\_host);

        frame.upload(frame\_host);

        cuda::cvtColor(frame, hsv, COLOR\_BGR2HSV);

        cuda::split(hsv, mv);

        // replace inRange

        cuda::threshold(mv[0], thres[0], 35, 255, THRESH\_BINARY);

        cuda::threshold(mv[0], thres[3], 77, 255, THRESH\_BINARY);

        cuda::threshold(mv[1], thres[1], 43, 255, THRESH\_BINARY);

        cuda::threshold(mv[2], thres[2], 46, 255, THRESH\_BINARY);

        cuda::bitwise\_xor(thres[0], thres[3], thres[0]);

        cuda::bitwise\_and(thres[1], thres[0], mask);

        cuda::bitwise\_and(mask, thres[2], mask);

        cuda::threshold(mask, mask, 66, 255, THRESH\_BINARY);

        Mat se = cv::getStructuringElement(MORPH\_RECT, Size(7, 7));

        auto morph\_filter = cuda::createMorphologyFilter(MORPH\_OPEN, mask.type(), se);

        morph\_filter->apply(mask, mask);

        mask.download(binary);

        imshow("mask", binary);

        // ��ͨ�������

        Mat labels = Mat::zeros(binary.size(), CV\_32S);

        Mat stats, centroids;

        int num\_labels = connectedComponentsWithStats(binary, labels, stats, centroids, 8, 4);

        for (int i = 1; i < num\_labels; i++) {

            int cx = centroids.at<double>(i, 0);

            int cy = centroids.at<double>(i, 1);

            int x = stats.at<int>(i, CC\_STAT\_LEFT);

            int y = stats.at<int>(i, CC\_STAT\_TOP);

            int width = stats.at<int>(i, CC\_STAT\_WIDTH);

            int height = stats.at<int>(i, CC\_STAT\_HEIGHT);

            if (width < 50 || height < 50) {

                continue;

            }

            circle(frame\_host, Point(cx, cy), 2, Scalar(255, 0, 0), 2, 8, 0);

            Rect rect(x, y, width, height);

            rectangle(frame\_host, rect, Scalar(0, 0, 255), 2, 8, 0);

        }

        double fps = getTickFrequency() / (getTickCount() - start);

        putText(frame\_host, format("FPS: %.2f", fps), Point(50, 50), FONT\_HERSHEY\_SIMPLEX, 1.0, Scalar(0, 0, 255), 2, 8);

        imshow("color object tracking", frame\_host);

        //if (fps > 100) {

        //  break;

        //}

        char c = waitKey(1);

        if (c == 27) {

            break;

        }

    }

}

## 2.7特征-角点检测

参见cuda\_007项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

RNG rng(12345);

int main(int argc, char\*\* argv) {

    Mat image\_host = imread("../data/building.jpg");

    imshow("input", image\_host);

    GpuMat src, gray, corners;

    Mat dst;

    src.upload(image\_host);

    cuda::cvtColor(src, gray, COLOR\_BGR2GRAY);

    auto corner\_detector = cuda::createGoodFeaturesToTrackDetector(gray.type(), 1000, 0.01, 15, 3, true);

    corner\_detector->detect(gray, corners);

    corners.download(dst);

    printf("number of corners : %d \n", corners.cols);

    for (int i = 0; i < corners.cols; i++) {

        int r = rng.uniform(0, 255);

        int g = rng.uniform(0, 255);

        int b = rng.uniform(0, 255);

        Point2f pt = dst.at<Point2f>(0, i);

        circle(image\_host, pt, 3, Scalar(b, g, r), 2, 8, 0);

    }

    imshow("corner detect result", image\_host);

    waitKey(0);

    return 0;

}

## 2.8视频背景分析、实时光流分析

参见cuda\_008项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

void optical\_flow\_demo();

void background\_demo();

int main(int argc, char\*\* argv) {

    optical\_flow\_demo();

    waitKey(0);

    return 0;

}

void background\_demo() {

    VideoCapture cap;

    cap.open("../data/vtest.avi");

    auto mog = cuda::createBackgroundSubtractorMOG2();

    Mat frame;

    GpuMat d\_frame, d\_fgmask, d\_bgimg;

    Mat fg\_mask, bgimg, fgimg;

    namedWindow("input", WINDOW\_AUTOSIZE);

    namedWindow("background", WINDOW\_AUTOSIZE);

    namedWindow("mask", WINDOW\_AUTOSIZE);

    Mat se = cv::getStructuringElement(MORPH\_RECT, Size(5, 5));

    while (true) {

        int64 start = getTickCount();

        bool ret = cap.read(frame);

        if (!ret) break;

        // ��������

        d\_frame.upload(frame);

        mog->apply(d\_frame, d\_fgmask);

        mog->getBackgroundImage(d\_bgimg);

        // ��̬ѧ����

        auto morph\_filter = cuda::createMorphologyFilter(MORPH\_OPEN, d\_fgmask.type(), se);

        morph\_filter->apply(d\_fgmask, d\_fgmask);

        // download from GPU Mat

        d\_bgimg.download(bgimg);

        d\_fgmask.download(fg\_mask);

        // ����FPS

        double fps = getTickFrequency() / (getTickCount() - start);

        putText(frame, format("FPS: %.2f", fps), Point(50, 50), FONT\_HERSHEY\_SIMPLEX, 1.0, Scalar(0, 0, 255), 2, 8);

        imshow("input", frame);

        imshow("background", bgimg);

        imshow("mask", fg\_mask);

        char c = waitKey(1);

        if (c == 27) {

            break;

        }

    }

    waitKey(0);

    return;

}

void optical\_flow\_demo() {

    VideoCapture cap;

    cap.open("../data/vtest.avi");

    auto farn = cuda::FarnebackOpticalFlow::create();

    Mat f, pf;

    cap.read(pf);

    GpuMat frame, gray, preFrame, preGray;

    preFrame.upload(pf);

    cuda::cvtColor(preFrame, preGray, COLOR\_BGR2GRAY);

    Mat hsv = Mat::zeros(preFrame.size(), preFrame.type());

    GpuMat flow;

    vector<Mat> mv;

    split(hsv, mv);

    GpuMat gMat, gAng;

    Mat mag = Mat::zeros(hsv.size(), CV\_32FC1);

    Mat ang = Mat::zeros(hsv.size(), CV\_32FC1);

    gMat.upload(mag);

    gAng.upload(ang);

    namedWindow("input", WINDOW\_AUTOSIZE);

    namedWindow("optical flow demo", WINDOW\_AUTOSIZE);

    Mat se = cv::getStructuringElement(MORPH\_RECT, Size(5, 5));

    while (true) {

        int64 start = getTickCount();

        bool ret = cap.read(f);

        if (!ret) break;

        // ��������

        frame.upload(f);

        cuda::cvtColor(frame, gray, COLOR\_BGR2GRAY);

        farn->calc(preGray, gray, flow);

        // ����ת��

        vector<GpuMat> mm;

        cuda::split(flow, mm);

        cuda::cartToPolar(mm[0], mm[1], gMat, gAng);

        cuda::normalize(gMat, gMat, 0, 255, NORM\_MINMAX, CV\_32FC1);

        gMat.download(mag);

        gAng.download(ang);

        // ��ʾ

        ang = ang \* 180 / CV\_PI / 2.0;

        convertScaleAbs(mag, mag);

        convertScaleAbs(ang, ang);

        mv[0] = ang;

        mv[1] = Scalar(255);

        mv[2] = mag;

        merge(mv, hsv);

        Mat bgr;

        cv::cvtColor(hsv, bgr, COLOR\_HSV2BGR);

        // ����FPS

        double fps = getTickFrequency() / (getTickCount() - start);

        putText(f, format("FPS: %.2f", fps), Point(50, 50), FONT\_HERSHEY\_SIMPLEX, 1.0, Scalar(0, 0, 255), 2, 8);

        gray.copyTo(preGray);

        imshow("input", f);

        imshow("optical flow demo", bgr);

        char c = waitKey(1);

        if (c == 27) {

            break;

        }

    }

    waitKey(0);

    return;

}

## 2.9 ORB特征匹配

参见cuda\_009项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

int main(int argc, char\*\* argv) {

    // cpu data

    Mat h\_object\_image = imread("../data/box.png", 0); // with a leather target image

    Mat h\_scene\_image = imread("../data/box\_in\_scene.png", 0); // scene image,

    // gpu data

    cuda::GpuMat d\_object\_image;

    cuda::GpuMat d\_scene\_image;

    vector< KeyPoint > h\_keypoints\_scene, h\_keypoints\_object; // CPU key points

    cuda::GpuMat d\_descriptors\_scene, d\_descriptors\_object;   // GPU descriptor

    // Image CPU uploaded to GPU

    d\_object\_image.upload(h\_object\_image);

    d\_scene\_image.upload(h\_scene\_image);

    // 对象检测

    auto orb = cuda::ORB::create();

    // Detect feature points and extract corresponding descriptors

    orb->detectAndCompute(d\_object\_image, cuda::GpuMat(), h\_keypoints\_object, d\_descriptors\_object);

    orb->detectAndCompute(d\_scene\_image, cuda::GpuMat(), h\_keypoints\_scene, d\_descriptors\_scene);

    // Brute Force Violence Matcher

    Ptr< cuda::DescriptorMatcher > matcher = cuda::DescriptorMatcher::createBFMatcher(NORM\_HAMMING);

    vector< vector< DMatch> > d\_matches;

    matcher->knnMatch(d\_descriptors\_object, d\_descriptors\_scene, d\_matches, 2);

    std::cout << "match size:" << d\_matches.size() << endl;

    std::vector< DMatch > good\_matches;

    for (int k = 0; k < std::min(h\_keypoints\_object.size() - 1, d\_matches.size()); k++)

    {

        if ((d\_matches[k][0].distance < 0.9\*(d\_matches[k][1].distance)) &&

            ((int)d\_matches[k].size() <= 2 && (int)d\_matches[k].size()>0))

        {

            good\_matches.push\_back(d\_matches[k][0]);

        }

    }

    std::cout << "size:" << good\_matches.size() << endl;

    // 绘制匹配点对

    Mat h\_image\_result;

    drawMatches(h\_object\_image, h\_keypoints\_object, h\_scene\_image, h\_keypoints\_scene,

        good\_matches, h\_image\_result, Scalar::all(-1), Scalar::all(-1),

        vector<char>(), DrawMatchesFlags::DEFAULT);

    // Find the image pixel 2d coordinates corresponding to the matching point pair

    std::vector<Point2f> object;

    std::vector<Point2f> scene;

    for (int i = 0; i < good\_matches.size(); i++)

    {

        object.push\_back(h\_keypoints\_object[good\_matches[i].queryIdx].pt);

        scene.push\_back(h\_keypoints\_scene[good\_matches[i].trainIdx].pt);

    }

    // 计算单应性矩阵

    Mat Homo = findHomography(object, scene, RANSAC);

    std::vector<Point2f> corners(4); // four corners of the image

    std::vector<Point2f> scene\_corners(4);

    // 透视变换

    corners[0] = Point(0, 0);

    corners[1] = Point(h\_object\_image.cols, 0);

    corners[2] = Point(h\_object\_image.cols, h\_object\_image.rows);

    corners[3] = Point(0, h\_object\_image.rows);

    perspectiveTransform(corners, scene\_corners, Homo);

    // 绘制对象

    line(h\_image\_result, scene\_corners[0] + Point2f(h\_object\_image.cols, 0), scene\_corners[1] + Point2f(h\_object\_image.cols, 0), Scalar(255, 0, 0), 4);

    line(h\_image\_result, scene\_corners[1] + Point2f(h\_object\_image.cols, 0), scene\_corners[2] + Point2f(h\_object\_image.cols, 0), Scalar(255, 0, 0), 4);

    line(h\_image\_result, scene\_corners[2] + Point2f(h\_object\_image.cols, 0), scene\_corners[3] + Point2f(h\_object\_image.cols, 0), Scalar(255, 0, 0), 4);

    line(h\_image\_result, scene\_corners[3] + Point2f(h\_object\_image.cols, 0), scene\_corners[0] + Point2f(h\_object\_image.cols, 0), Scalar(255, 0, 0), 4);

    imshow("Good Matches & Object detection", h\_image\_result);

    waitKey(0);

    return 0;

}

## 2.10 HOG行人检测

参见cuda\_010项目

#include <opencv2/opencv.hpp>

#include <opencv2/cudaimgproc.hpp>

#include <iostream>

using namespace cv;

using namespace cv::cuda;

using namespace std;

int main(int argc, char\*\* argv) {

    VideoCapture cap;

    cap.open("../data/vtest.avi");

    Mat f;

    GpuMat frame, gray;

    namedWindow("input", WINDOW\_AUTOSIZE);

    namedWindow("People Detector Demo", WINDOW\_AUTOSIZE);

    // ���������

    auto hog = cuda::HOG::create();

    hog->setSVMDetector(hog->getDefaultPeopleDetector());

    vector<Rect> objects;

    while (true) {

        int64 start = getTickCount();

        bool ret = cap.read(f);

        if (!ret) break;

        imshow("input", f);

        // HOG detector

        frame.upload(f);

        cuda::cvtColor(frame, gray, COLOR\_BGR2GRAY);

        hog->detectMultiScale(gray, objects);

        // ���Ƽ��

        for (int i = 0; i < objects.size(); i++) {

            rectangle(f, objects[i], Scalar(0, 0, 255), 2, 8, 0);

        }

        // ����FPS

        double fps = getTickFrequency() / (getTickCount() - start);

        putText(f, format("FPS: %.2f", fps), Point(50, 50), FONT\_HERSHEY\_SIMPLEX, 1.0, Scalar(0, 0, 255), 2, 8);

        imshow("People Detector Demo", f);

        char c = cv::waitKey(1);

        if (c == 27) {

            break;

        }

    }

    cv::waitKey(0);

    return 0;

}

# 资源下载目录

## 3.1 opencv4.1.0\_cuda10.1库下载

<https://pan.baidu.com/s/1_0tmYJYCpPDyBxNEPzLirg>

提取码：ppaw

## 3.2 demo下载

<https://github.com/fhqddm/opencv_cuda_demo>或

$ git clone https://github.com/fhqddm/opencv\_cuda\_demo.git