**相机基础参数设置**

#include "pch.h"

#include<stdio.h>

#include<opencv.hpp>

#include <iostream>

using namespace cv;

#include<opencv2/opencv.hpp>

using namespace cv;

int main()

{

char c = 0;

VideoCapture capture(1);

//设置摄像头参数 不要随意修改

capture.set(CV\_CAP\_PROP\_FRAME\_WIDTH, 960);//宽度

capture.set(CV\_CAP\_PROP\_FRAME\_HEIGHT, 840);//高度

capture.set(CV\_CAP\_PROP\_FPS, 100);//帧数

//capture.set(CV\_CAP\_PROP\_BRIGHTNESS, 120);//亮度 1

//capture.set(CV\_CAP\_PROP\_CONTRAST,23);//对比度 40

//capture.set(CV\_CAP\_PROP\_SATURATION, 32);//饱和度 50

//capture.set(CV\_CAP\_PROP\_HUE, 3);//色调 50

//capture.set(CV\_CAP\_PROP\_EXPOSURE, 50);//曝光 50

//

//打印摄像头参数

printf("width = %.2f\n", capture.get(CV\_CAP\_PROP\_FRAME\_WIDTH));

printf("height = %.2f\n", capture.get(CV\_CAP\_PROP\_FRAME\_HEIGHT));

printf("fbs = %.2f\n", capture.get(CV\_CAP\_PROP\_FPS));

printf("brightness = %.2f\n", capture.get(CV\_CAP\_PROP\_BRIGHTNESS));

printf("contrast = %.2f\n", capture.get(CV\_CAP\_PROP\_CONTRAST));

printf("saturation = %.2f\n", capture.get(CV\_CAP\_PROP\_SATURATION));

printf("hue = %.2f\n", capture.get(CV\_CAP\_PROP\_HUE));

printf("exposure = %.2f\n", capture.get(CV\_CAP\_PROP\_EXPOSURE));

while (1)

{

Mat frame;

capture >> frame;

namedWindow("video");

imshow("video", frame);

c = cvWaitKey(1);

if (c == 27)//Esc键退出

{

break;

}

}

return 0;

}

**获取相机标定用的图片**

// 3DMCAM\_2.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include "pch.h"

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <time.h>

#include <windows.h>

using namespace cv;

// ÐÞ¸ÄÕâÀï

int width = 1280;

int height = 480;

int main()

{

clock\_t start, end;

namedWindow("1", WINDOW\_AUTOSIZE);

namedWindow("2", WINDOW\_AUTOSIZE);

VideoCapture cap(1);

cap.set(CV\_CAP\_PROP\_FRAME\_WIDTH, width);

cap.set(CV\_CAP\_PROP\_FRAME\_HEIGHT, height);

printf("width = %.2f\n", cap.get(CV\_CAP\_PROP\_FRAME\_WIDTH));

printf("height = %.2f\n", cap.get(CV\_CAP\_PROP\_FRAME\_HEIGHT));

Sleep(100);

start = clock();

width = cap.get(CV\_CAP\_PROP\_FRAME\_WIDTH);

height = cap.get(CV\_CAP\_PROP\_FRAME\_HEIGHT);

Mat frame;

int cap\_no = 1;

//for (int i = 8; i > 0; i--)

//{

// printf("%d\n", i);

// waitKey(1000);

//}

while (1)

{

cap >> frame;

//Sleep(100);

Rect leftRect(0, 0, width >> 1, height);

Rect rightRect(width >> 1, 0, width >> 1, height);

Mat leftFrame = Mat(frame, leftRect);

Mat rightFrame = Mat(frame, rightRect);

imshow("1", leftFrame);

imshow("2", rightFrame);

end = clock();

if ((end - start) / CLOCKS\_PER\_SEC > 5 \* cap\_no)

{

if (cap\_no < 10)

{

imwrite("left0" + std::to\_string(cap\_no) + ".jpg", leftFrame);

imwrite("right0" + std::to\_string(cap\_no) + ".jpg", rightFrame);

}

else

{

imwrite("left" + std::to\_string(cap\_no) + ".jpg", leftFrame);

imwrite("right" + std::to\_string(cap\_no) + ".jpg", rightFrame);

}

printf("cap\_no=%d\n", cap\_no);

cap\_no++;

}

if (cap\_no > 17) break;

if (waitKey(100) >= 0) break;

}

system("pause");

return 0;

}

**相机标定生成内外参数**

// 3DCAM.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include "pch.h"

/\* This is sample from the OpenCV book. The copyright notice is below \*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* License:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Oct. 3, 2008

Right to use this code in any way you want without warranty, support or any guarantee of it working.

BOOK: It would be nice if you cited it:

Learning OpenCV: Computer Vision with the OpenCV Library

by Gary Bradski and Adrian Kaehler

Published by O'Reilly Media, October 3, 2008

AVAILABLE AT:

http://www.amazon.com/Learning-OpenCV-Computer-Vision-Library/dp/0596516134

Or: http://oreilly.com/catalog/9780596516130/

ISBN-10: 0596516134 or: ISBN-13: 978-0596516130

OPENCV WEBSITES:

Homepage: http://opencv.org

Online docs: http://docs.opencv.org

Q&A forum: http://answers.opencv.org

Issue tracker: http://code.opencv.org

GitHub: https://github.com/opencv/opencv/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#include "opencv2/calib3d.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/highgui.hpp"

#include "opencv2/imgproc.hpp"

#include <vector>

#include <string>

#include <algorithm>

#include <iostream>

#include <iterator>

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

using namespace cv;

using namespace std;

static int print\_help()

{

cout <<

" Given a list of chessboard images, the number of corners (nx, ny)\n"

" on the chessboards, and a flag: useCalibrated for \n"

" calibrated (0) or\n"

" uncalibrated \n"

" (1: use cvStereoCalibrate(), 2: compute fundamental\n"

" matrix separately) stereo. \n"

" Calibrate the cameras and display the\n"

" rectified results along with the computed disparity images. \n" << endl;

cout << "Usage:\n ./stereo\_calib -w=<board\_width default=9> -h=<board\_height default=6> -s=<square\_size default=1.0> <image list XML/YML file default=../data/stereo\_calib.xml>\n" << endl;

return 0;

}

static void

StereoCalib(const vector<string>& imagelist, Size boardSize, float squareSize, bool displayCorners = false, bool useCalibrated = true, bool showRectified = true)

{

if (imagelist.size() % 2 != 0)

{

cout << "Error: the image list contains odd (non-even) number of elements\n";

return;

}

const int maxScale = 2;

// ARRAY AND VECTOR STORAGE:

vector<vector<Point2f> > imagePoints[2];

vector<vector<Point3f> > objectPoints;

Size imageSize;

int i, j, k, nimages = (int)imagelist.size() / 2;

imagePoints[0].resize(nimages);

imagePoints[1].resize(nimages);

vector<string> goodImageList;

for (i = j = 0; i < nimages; i++)

{

for (k = 0; k < 2; k++)

{

const string& filename = imagelist[i \* 2 + k];

Mat img = imread(filename, 0);

if (img.empty())

break;

if (imageSize == Size())

imageSize = img.size();

else if (img.size() != imageSize)

{

cout << "The image " << filename << " has the size different from the first image size. Skipping the pair\n";

break;

}

bool found = false;

vector<Point2f>& corners = imagePoints[k][j];

for (int scale = 1; scale <= maxScale; scale++)

{

Mat timg;

if (scale == 1)

timg = img;

else

resize(img, timg, Size(), scale, scale, INTER\_LINEAR\_EXACT);

found = findChessboardCorners(timg, boardSize, corners,

CALIB\_CB\_ADAPTIVE\_THRESH | CALIB\_CB\_NORMALIZE\_IMAGE);

if (found)

{

if (scale > 1)

{

Mat cornersMat(corners);

cornersMat \*= 1. / scale;

}

break;

}

}

if (displayCorners)

{

cout << filename << endl;

Mat cimg, cimg1;

cvtColor(img, cimg, COLOR\_GRAY2BGR);

drawChessboardCorners(cimg, boardSize, corners, found);

double sf = 640. / MAX(img.rows, img.cols);

resize(cimg, cimg1, Size(), sf, sf, INTER\_LINEAR\_EXACT);

imshow("corners", cimg1);

char c = (char)waitKey(500);

if (c == 27 || c == 'q' || c == 'Q') //Allow ESC to quit

exit(-1);

}

else

putchar('.');

if (!found)

break;

cornerSubPix(img, corners, Size(11, 11), Size(-1, -1),

TermCriteria(TermCriteria::COUNT + TermCriteria::EPS,

30, 0.01));

}

if (k == 2)

{

goodImageList.push\_back(imagelist[i \* 2]);

goodImageList.push\_back(imagelist[i \* 2 + 1]);

j++;

}

}

cout << j << " pairs have been successfully detected.\n";

nimages = j;

if (nimages < 2)

{

cout << "Error: too little pairs to run the calibration\n";

return;

}

imagePoints[0].resize(nimages);

imagePoints[1].resize(nimages);

objectPoints.resize(nimages);

for (i = 0; i < nimages; i++)

{

for (j = 0; j < boardSize.height; j++)

for (k = 0; k < boardSize.width; k++)

objectPoints[i].push\_back(Point3f(k\*squareSize, j\*squareSize, 0));

}

cout << "Running stereo calibration ...\n";

Mat cameraMatrix[2], distCoeffs[2];

cameraMatrix[0] = initCameraMatrix2D(objectPoints, imagePoints[0], imageSize, 0);

cameraMatrix[1] = initCameraMatrix2D(objectPoints, imagePoints[1], imageSize, 0);

Mat R, T, E, F;

double rms = stereoCalibrate(objectPoints, imagePoints[0], imagePoints[1],

cameraMatrix[0], distCoeffs[0],

cameraMatrix[1], distCoeffs[1],

imageSize, R, T, E, F,

CALIB\_FIX\_ASPECT\_RATIO +

CALIB\_ZERO\_TANGENT\_DIST +

CALIB\_USE\_INTRINSIC\_GUESS +

CALIB\_SAME\_FOCAL\_LENGTH +

CALIB\_RATIONAL\_MODEL +

CALIB\_FIX\_K3 + CALIB\_FIX\_K4 + CALIB\_FIX\_K5,

TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 100, 1e-5));

cout << "done with RMS error=" << rms << endl;

// CALIBRATION QUALITY CHECK

// because the output fundamental matrix implicitly

// includes all the output information,

// we can check the quality of calibration using the

// epipolar geometry constraint: m2^t\*F\*m1=0

double err = 0;

int npoints = 0;

vector<Vec3f> lines[2];

for (i = 0; i < nimages; i++)

{

int npt = (int)imagePoints[0][i].size();

Mat imgpt[2];

for (k = 0; k < 2; k++)

{

imgpt[k] = Mat(imagePoints[k][i]);

undistortPoints(imgpt[k], imgpt[k], cameraMatrix[k], distCoeffs[k], Mat(), cameraMatrix[k]);

computeCorrespondEpilines(imgpt[k], k + 1, F, lines[k]);

}

for (j = 0; j < npt; j++)

{

double errij = fabs(imagePoints[0][i][j].x\*lines[1][j][0] +

imagePoints[0][i][j].y\*lines[1][j][1] + lines[1][j][2]) +

fabs(imagePoints[1][i][j].x\*lines[0][j][0] +

imagePoints[1][i][j].y\*lines[0][j][1] + lines[0][j][2]);

err += errij;

}

npoints += npt;

}

cout << "average epipolar err = " << err / npoints << endl;

// save intrinsic parameters

FileStorage fs("intrinsics.yml", FileStorage::WRITE);

if (fs.isOpened())

{

fs << "M1" << cameraMatrix[0] << "D1" << distCoeffs[0] <<

"M2" << cameraMatrix[1] << "D2" << distCoeffs[1];

fs.release();

}

else

cout << "Error: can not save the intrinsic parameters\n";

Mat R1, R2, P1, P2, Q;

Rect validRoi[2];

stereoRectify(cameraMatrix[0], distCoeffs[0],

cameraMatrix[1], distCoeffs[1],

imageSize, R, T, R1, R2, P1, P2, Q,

CALIB\_ZERO\_DISPARITY, 1, imageSize, &validRoi[0], &validRoi[1]);

fs.open("extrinsics.yml", FileStorage::WRITE);

if (fs.isOpened())

{

fs << "R" << R << "T" << T << "R1" << R1 << "R2" << R2 << "P1" << P1 << "P2" << P2 << "Q" << Q;

fs.release();

}

else

cout << "Error: can not save the extrinsic parameters\n";

// OpenCV can handle left-right

// or up-down camera arrangements

bool isVerticalStereo = fabs(P2.at<double>(1, 3)) > fabs(P2.at<double>(0, 3));

// COMPUTE AND DISPLAY RECTIFICATION

if (!showRectified)

return;

Mat rmap[2][2];

// IF BY CALIBRATED (BOUGUET'S METHOD)

if (useCalibrated)

{

// we already computed everything

}

// OR ELSE HARTLEY'S METHOD

else

// use intrinsic parameters of each camera, but

// compute the rectification transformation directly

// from the fundamental matrix

{

vector<Point2f> allimgpt[2];

for (k = 0; k < 2; k++)

{

for (i = 0; i < nimages; i++)

std::copy(imagePoints[k][i].begin(), imagePoints[k][i].end(), back\_inserter(allimgpt[k]));

}

F = findFundamentalMat(Mat(allimgpt[0]), Mat(allimgpt[1]), FM\_8POINT, 0, 0);

Mat H1, H2;

stereoRectifyUncalibrated(Mat(allimgpt[0]), Mat(allimgpt[1]), F, imageSize, H1, H2, 3);

R1 = cameraMatrix[0].inv()\*H1\*cameraMatrix[0];

R2 = cameraMatrix[1].inv()\*H2\*cameraMatrix[1];

P1 = cameraMatrix[0];

P2 = cameraMatrix[1];

}

//Precompute maps for cv::remap()

initUndistortRectifyMap(cameraMatrix[0], distCoeffs[0], R1, P1, imageSize, CV\_16SC2, rmap[0][0], rmap[0][1]);

initUndistortRectifyMap(cameraMatrix[1], distCoeffs[1], R2, P2, imageSize, CV\_16SC2, rmap[1][0], rmap[1][1]);

Mat canvas;

double sf;

int w, h;

if (!isVerticalStereo)

{

sf = 600. / MAX(imageSize.width, imageSize.height);

w = cvRound(imageSize.width\*sf);

h = cvRound(imageSize.height\*sf);

canvas.create(h, w \* 2, CV\_8UC3);

}

else

{

sf = 300. / MAX(imageSize.width, imageSize.height);

w = cvRound(imageSize.width\*sf);

h = cvRound(imageSize.height\*sf);

canvas.create(h \* 2, w, CV\_8UC3);

}

for (i = 0; i < nimages; i++)

{

for (k = 0; k < 2; k++)

{

Mat img = imread(goodImageList[i \* 2 + k], 0), rimg, cimg;

remap(img, rimg, rmap[k][0], rmap[k][1], INTER\_LINEAR);

cvtColor(rimg, cimg, COLOR\_GRAY2BGR);

Mat canvasPart = !isVerticalStereo ? canvas(Rect(w\*k, 0, w, h)) : canvas(Rect(0, h\*k, w, h));

resize(cimg, canvasPart, canvasPart.size(), 0, 0, INTER\_AREA);

if (useCalibrated)

{

Rect vroi(cvRound(validRoi[k].x\*sf), cvRound(validRoi[k].y\*sf),

cvRound(validRoi[k].width\*sf), cvRound(validRoi[k].height\*sf));

rectangle(canvasPart, vroi, Scalar(0, 0, 255), 3, 8);

}

}

if (!isVerticalStereo)

for (j = 0; j < canvas.rows; j += 16)

line(canvas, Point(0, j), Point(canvas.cols, j), Scalar(0, 255, 0), 1, 8);

else

for (j = 0; j < canvas.cols; j += 16)

line(canvas, Point(j, 0), Point(j, canvas.rows), Scalar(0, 255, 0), 1, 8);

imshow("rectified", canvas);

char c = (char)waitKey();

if (c == 27 || c == 'q' || c == 'Q')

break;

}

}

static bool readStringList(const string& filename, vector<string>& l)

{

l.resize(0);

FileStorage fs(filename, FileStorage::READ);

if (!fs.isOpened())

return false;

FileNode n = fs.getFirstTopLevelNode();

if (n.type() != FileNode::SEQ)

return false;

FileNodeIterator it = n.begin(), it\_end = n.end();

for (; it != it\_end; ++it)

l.push\_back((string)\*it);

return true;

}

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

{

Size boardSize;

string imagelistfn;

bool showRectified;

cv::CommandLineParser parser(argc, argv, "{w|11|}{h|8|}{s|1.0|}{nr||}{help||}{@input|stereo\_calib.xml|}");

if (parser.has("help"))

return print\_help();

showRectified = !parser.has("nr");

imagelistfn = parser.get<string>("@input");

boardSize.width = parser.get<int>("w");

boardSize.height = parser.get<int>("h");

float squareSize = parser.get<float>("s");

if (!parser.check())

{

parser.printErrors();

return 1;

}

vector<string> imagelist;

bool ok = readStringList(imagelistfn, imagelist);

if (!ok || imagelist.empty())

{

cout << "can not open " << imagelistfn << " or the string list is empty" << endl;

return print\_help();

}

StereoCalib(imagelist, boardSize, squareSize, true, true, showRectified);

return 0;

}

**图片校正和视差图获取**

// 3DCAM\_04.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include "pch.h"

/\*

\* stereo\_match.cpp

\* calibration

\*

\* Created by Victor Eruhimov on 1/18/10.

\* Copyright 2010 Argus Corp. All rights reserved.

\*

\*/

#pragma warning(disable : 4996)

#include "opencv2/calib3d/calib3d.hpp"

#include "opencv2/imgproc.hpp"

#include "opencv2/imgcodecs.hpp"

#include "opencv2/highgui.hpp"

#include "opencv2/core/utility.hpp"

#include <stdio.h>

using namespace cv;

static void print\_help()

{

printf("\nDemo stereo matching converting L and R images into disparity and point clouds\n");

printf("\nUsage: stereo\_match <left\_image> <right\_image> [--algorithm=bm|sgbm|hh|sgbm3way] [--blocksize=<block\_size>]\n"

"[--max-disparity=<max\_disparity>] [--scale=scale\_factor>] [-i=<intrinsic\_filename>] [-e=<extrinsic\_filename>]\n"

"[--no-display] [-o=<disparity\_image>] [-p=<point\_cloud\_file>]\n");

}

static void saveXYZ(const char\* filename, const Mat& mat)

{

const double max\_z = 1.0e4;

FILE\* fp = fopen(filename, "wt");

for (int y = 0; y < mat.rows; y++)

{

for (int x = 0; x < mat.cols; x++)

{

Vec3f point = mat.at<Vec3f>(y, x);

if (fabs(point[2] - max\_z) < FLT\_EPSILON || fabs(point[2]) > max\_z) continue;

fprintf(fp, "%f %f %f\n", point[0], point[1], point[2]);

}

}

fclose(fp);

}

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

{

std::string img1\_filename = "";

std::string img2\_filename = "";

std::string intrinsic\_filename = "";

std::string extrinsic\_filename = "";

std::string disparity\_filename = "";

std::string point\_cloud\_filename = "";

enum { STEREO\_BM = 0, STEREO\_SGBM = 1, STEREO\_HH = 2, STEREO\_VAR = 3, STEREO\_3WAY = 4 };

int alg = STEREO\_SGBM;

int SADWindowSize, numberOfDisparities;

bool no\_display;

float scale;

Ptr<StereoBM> bm = StereoBM::create(16, 9);

Ptr<StereoSGBM> sgbm = StereoSGBM::create(0, 16, 3);

/\* cv::CommandLineParser parser(argc, argv,

"{@arg1||}{@arg2||}{help h||}{algorithm||}{max-disparity|0|}{blocksize|0|}{no-display||}{scale|1|}{i||}{e||}{o||}{p||}");

\*/

cv::CommandLineParser parser(argc, argv,

"{@arg1|left01.jpg|}{@arg2|right01.jpg|}{help h||}{algorithm|bm|}{max-disparity|80|}{blocksize|7|}{no-display||}{scale|1|}{i|intrinsics.yml|}{e|extrinsics.yml|}{o||}{p||}");

if (parser.has("help"))

{

print\_help();

return 0;

}

img1\_filename = parser.get<std::string>(0);

img2\_filename = parser.get<std::string>(1);

if (parser.has("algorithm"))

{

std::string \_alg = parser.get<std::string>("algorithm");

alg = \_alg == "bm" ? STEREO\_BM :

\_alg == "sgbm" ? STEREO\_SGBM :

\_alg == "hh" ? STEREO\_HH :

\_alg == "var" ? STEREO\_VAR :

\_alg == "sgbm3way" ? STEREO\_3WAY : -1;

}

numberOfDisparities = parser.get<int>("max-disparity");

SADWindowSize = parser.get<int>("blocksize");

scale = parser.get<float>("scale");

no\_display = parser.has("no-display");

if (parser.has("i"))

intrinsic\_filename = parser.get<std::string>("i");

if (parser.has("e"))

extrinsic\_filename = parser.get<std::string>("e");

if (parser.has("o"))

disparity\_filename = parser.get<std::string>("o");

if (parser.has("p"))

point\_cloud\_filename = parser.get<std::string>("p");

if (!parser.check())

{

parser.printErrors();

return 1;

}

if (alg < 0)

{

printf("Command-line parameter error: Unknown stereo algorithm\n\n");

print\_help();

return -1;

}

if (numberOfDisparities < 1 || numberOfDisparities % 16 != 0)

{

printf("Command-line parameter error: The max disparity (--maxdisparity=<...>) must be a positive integer divisible by 16\n");

print\_help();

return -1;

}

if (scale < 0)

{

printf("Command-line parameter error: The scale factor (--scale=<...>) must be a positive floating-point number\n");

return -1;

}

if (SADWindowSize < 1 || SADWindowSize % 2 != 1)

{

printf("Command-line parameter error: The block size (--blocksize=<...>) must be a positive odd number\n");

return -1;

}

if (img1\_filename.empty() || img2\_filename.empty())

{

printf("Command-line parameter error: both left and right images must be specified\n");

return -1;

}

if ((!intrinsic\_filename.empty()) ^ (!extrinsic\_filename.empty()))

{

printf("Command-line parameter error: either both intrinsic and extrinsic parameters must be specified, or none of them (when the stereo pair is already rectified)\n");

return -1;

}

if (extrinsic\_filename.empty() && !point\_cloud\_filename.empty())

{

printf("Command-line parameter error: extrinsic and intrinsic parameters must be specified to compute the point cloud\n");

return -1;

}

int color\_mode = alg == STEREO\_BM ? 0 : -1;

Mat img1 = imread(img1\_filename, color\_mode);

Mat img2 = imread(img2\_filename, color\_mode);

if (img1.empty())

{

printf("Command-line parameter error: could not load the first input image file\n");

return -1;

}

if (img2.empty())

{

printf("Command-line parameter error: could not load the second input image file\n");

return -1;

}

if (scale != 1.f)

{

Mat temp1, temp2;

int method = scale < 1 ? INTER\_AREA : INTER\_CUBIC;

resize(img1, temp1, Size(), scale, scale, method);

img1 = temp1;

resize(img2, temp2, Size(), scale, scale, method);

img2 = temp2;

}

Size img\_size = img1.size();

Rect roi1, roi2;

Mat Q;

if (!intrinsic\_filename.empty())

{

// reading intrinsic parameters

FileStorage fs(intrinsic\_filename, FileStorage::READ);

if (!fs.isOpened())

{

printf("Failed to open file %s\n", intrinsic\_filename.c\_str());

return -1;

}

Mat M1, D1, M2, D2;

fs["M1"] >> M1;

fs["D1"] >> D1;

fs["M2"] >> M2;

fs["D2"] >> D2;

M1 \*= scale;

M2 \*= scale;

fs.open(extrinsic\_filename, FileStorage::READ);

if (!fs.isOpened())

{

printf("Failed to open file %s\n", extrinsic\_filename.c\_str());

return -1;

}

Mat R, T, R1, P1, R2, P2;

fs["R"] >> R;

fs["T"] >> T;

stereoRectify(M1, D1, M2, D2, img\_size, R, T, R1, R2, P1, P2, Q, CALIB\_ZERO\_DISPARITY, -1, img\_size, &roi1, &roi2);

Mat map11, map12, map21, map22;

initUndistortRectifyMap(M1, D1, R1, P1, img\_size, CV\_16SC2, map11, map12);

initUndistortRectifyMap(M2, D2, R2, P2, img\_size, CV\_16SC2, map21, map22);

Mat img1r, img2r;

remap(img1, img1r, map11, map12, INTER\_LINEAR);

remap(img2, img2r, map21, map22, INTER\_LINEAR);

img1 = img1r;

img2 = img2r;

}

numberOfDisparities = numberOfDisparities > 0 ? numberOfDisparities : ((img\_size.width / 8) + 15) & -16;

bm->setROI1(roi1);

bm->setROI2(roi2);

bm->setPreFilterCap(31);

bm->setBlockSize(SADWindowSize > 0 ? SADWindowSize : 9);

bm->setMinDisparity(0);

bm->setNumDisparities(numberOfDisparities);

bm->setTextureThreshold(10);

bm->setUniquenessRatio(15);

bm->setSpeckleWindowSize(100);

bm->setSpeckleRange(32);

bm->setDisp12MaxDiff(1);

sgbm->setPreFilterCap(63);

int sgbmWinSize = SADWindowSize > 0 ? SADWindowSize : 3;

sgbm->setBlockSize(sgbmWinSize);

int cn = img1.channels();

sgbm->setP1(8 \* cn\*sgbmWinSize\*sgbmWinSize);

sgbm->setP2(32 \* cn\*sgbmWinSize\*sgbmWinSize);

sgbm->setMinDisparity(0);

sgbm->setNumDisparities(numberOfDisparities);

sgbm->setUniquenessRatio(10);

sgbm->setSpeckleWindowSize(100);

sgbm->setSpeckleRange(32);

sgbm->setDisp12MaxDiff(1);

if (alg == STEREO\_HH)

sgbm->setMode(StereoSGBM::MODE\_HH);

else if (alg == STEREO\_SGBM)

sgbm->setMode(StereoSGBM::MODE\_SGBM);

else if (alg == STEREO\_3WAY)

sgbm->setMode(StereoSGBM::MODE\_SGBM\_3WAY);

Mat disp, disp8;

//Mat img1p, img2p, dispp;

//copyMakeBorder(img1, img1p, 0, 0, numberOfDisparities, 0, IPL\_BORDER\_REPLICATE);

//copyMakeBorder(img2, img2p, 0, 0, numberOfDisparities, 0, IPL\_BORDER\_REPLICATE);

int64 t = getTickCount();

if (alg == STEREO\_BM)

bm->compute(img1, img2, disp);

else if (alg == STEREO\_SGBM || alg == STEREO\_HH || alg == STEREO\_3WAY)

sgbm->compute(img1, img2, disp);

t = getTickCount() - t;

printf("Time elapsed: %fms\n", t \* 1000 / getTickFrequency());

//disp = dispp.colRange(numberOfDisparities, img1p.cols);

if (alg != STEREO\_VAR)

disp.convertTo(disp8, CV\_8U, 255 / (numberOfDisparities\*16.));

else

disp.convertTo(disp8, CV\_8U);

if (!no\_display)

{

namedWindow("left", 1);

imshow("left", img1);

namedWindow("right", 1);

imshow("right", img2);

namedWindow("disparity", 0);

imshow("disparity", disp8);

printf("press any key to continue...");

fflush(stdout);

waitKey();

printf("\n");

}

if (!disparity\_filename.empty())

imwrite(disparity\_filename, disp8);

if (!point\_cloud\_filename.empty())

{

printf("storing the point cloud...");

fflush(stdout);

Mat xyz;

reprojectImageTo3D(disp, xyz, Q, true);

saveXYZ(point\_cloud\_filename.c\_str(), xyz);

printf("\n");

}

return 0;

}

**SGBM算法获取视差图**

// SGBM\_01.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include "pch.h"

#include <opencv2/opencv.hpp>

#include <highgui.h>

#include <cv.h>

#include <cxcore.h>

#include <iostream>

using namespace std;

using namespace cv;

//void disp2Depth(cv::Mat dispMap, cv::Mat &depthMap, cv::Mat K);

int main()

{

enum { STEREO\_BM = 0, STEREO\_SGBM = 1, STEREO\_HH = 2, STEREO\_VAR = 3, STEREO\_3WAY = 4 };

Mat img1 = imread("left03.jpg");

Mat img2 = imread("right03.jpg");

//IplImage \* img1 = cvLoadImage("left.png", 0);

//IplImage \* img2 = cvLoadImage("right.png", 0);

//cv::StereoSGBM sgbm;

//Ptr <StereoSGBM> sgbm = StereoSGBM::create();

//int numberOfDisparities = ((imgSize.width / 8) + 15) & -16;

cv::Ptr<cv::StereoSGBM> sgbm = cv::StereoSGBM::create(0, 16, 3);

//sgbm->setPreFilterCap(32);

int SADWindowSize = 9;

sgbm->setPreFilterCap (63);

int sgbmWinSize = SADWindowSize > 0 ? SADWindowSize : 3;

int cn = img1.channels();

int numberOfDisparities = 64;

sgbm->setP1(8 \* cn\*sgbmWinSize\*sgbmWinSize);

sgbm->setP2(32 \* cn\*sgbmWinSize\*sgbmWinSize);

sgbm->setMinDisparity (0);

sgbm->setNumDisparities(numberOfDisparities);

sgbm->setUniquenessRatio(10);

sgbm->setSpeckleWindowSize(100);

sgbm->setSpeckleRange(32);

sgbm->setDisp12MaxDiff(1);

int alg = STEREO\_SGBM;

if (alg == STEREO\_HH)

sgbm->setMode(cv::StereoSGBM::MODE\_HH);

else if (alg == STEREO\_SGBM)

sgbm->setMode(cv::StereoSGBM::MODE\_SGBM);

else if (alg == STEREO\_3WAY)

sgbm->setMode(cv::StereoSGBM::MODE\_SGBM\_3WAY);

Mat disp, disp8;

int64 t = getTickCount();

sgbm->compute(img1, img2, disp);

//sgbm((Mat)img1, (Mat)img2, disp);

t = getTickCount() - t;

cout << "Time elapsed:" << t \* 1000 / getTickFrequency() << endl;

disp.convertTo(disp8, CV\_8U, 255 / (numberOfDisparities\*16.));

namedWindow("left", 1);

imshow("left", img1);

namedWindow("right", 1);

imshow("right", img2);

namedWindow("disparity", 1);

imshow("disparity", disp8);

imwrite("sgbm\_disparity.png", disp8);

waitKey();

cvDestroyAllWindows();

return 0;

}

**视差图转为深度图**

#include "pch.h"

#include <opencv2/opencv.hpp>

#include <highgui.h>

#include <cv.h>

#include <cxcore.h>

#include <iostream>

using namespace std;

using namespace cv;

/\*

函数作用：视差图转深度图

输入：

　　dispMap ----视差图，8位单通道，CV\_8UC1

　　K ----内参矩阵，float类型

输出：

　　depthMap ----深度图，16位无符号单通道，CV\_16UC1

\*/

void disp2Depth(cv::Mat dispMap, cv::Mat &depthMap, cv::Mat K)

{

int type = dispMap.type();

float fx = K.at<float>(0, 0);

/\*float fy = K.at<float>(1, 1);

float cx = K.at<float>(0, 2);

float cy = K.at<float>(1, 2);\*/

float baseline = 65; //基线距离65mm

if (type == CV\_8U)

{

/\*const float PI = 3.14159265358;\*/

int height = dispMap.rows;

int width = dispMap.cols;

uchar\* dispData = (uchar\*)dispMap.data;

ushort\* depthData = (ushort\*)depthMap.data;

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

int id = i \* width + j;

if (!dispData[id])

{

//防止0除

//depthData[id] = 255;

continue;

}

else

depthData[id] = fx \* baseline / float(dispData[id]);

}

}

}

else

{

cout << "please confirm dispImg's type!" << endl;

cv::waitKey(0);

}

}

int main()

{

Mat disp8 = imread("left\_disp.png", 0);

Mat depth(disp8.rows, disp8.cols, CV\_16UC1);

Mat K = (Mat\_<float>(3, 3) << 5.8366697641288249e+02, 0., 3.2762020192177897e+02, 0.,

5.3773947573643704e+02, 2.4134709536899661e+02, 0., 0., 1.);

namedWindow("disp", 1);

imshow("disp", disp8);

disp2Depth(disp8, depth, K);

namedWindow("depth", 1);

imshow("depth", depth);

imwrite("depth.png", depth);

waitKey();

cvDestroyAllWindows();

return 0;

}

**视差图深度图填充**

// insertDepth32f.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//sgbm\_disparity.jpg

#include "pch.h"

#include <iostream>

#include <opencv2/opencv.hpp>

using namespace std;

using namespace cv;

void insertDepth32f(cv::Mat& depth, cv::Mat& integralMap)

{

const int width = depth.cols;

const int height = depth.rows;

float \* data = (float \*)depth.data;

//cv::Mat integralMap = cv::Mat::zeros(height, width, CV\_64F);

cv::Mat ptsMap = cv::Mat::zeros(height, width, CV\_32S);

double\* integral = (double\*)integralMap.data;

int\* ptsIntegral = (int\*)ptsMap.data;

memset(integral, 0, sizeof(double) \* width \* height);

memset(ptsIntegral, 0, sizeof(int) \* width \* height);

for (int i = 0; i < height; ++i)

{

int id1 = i \* width;

for (int j = 0; j < width; ++j)

{

int id2 = id1 + j;

if (data[id2] > 1e-3)

{

integral[id2] = data[id2];

ptsIntegral[id2] = 1;

}

}

}

// 积分区间

for (int i = 0; i < height; ++i)

{

int id1 = i \* width;

for (int j = 1; j < width; ++j)

{

int id2 = id1 + j;

integral[id2] += integral[id2 - 1];

ptsIntegral[id2] += ptsIntegral[id2 - 1];

}

}

for (int i = 1; i < height; ++i)

{

int id1 = i \* width;

for (int j = 0; j < width; ++j)

{

int id2 = id1 + j;

integral[id2] += integral[id2 - width];

ptsIntegral[id2] += ptsIntegral[id2 - width];

}

}

int wnd;

double dWnd = 2;

while (dWnd > 1)

{

wnd = int(dWnd);

dWnd /= 2;

for (int i = 0; i < height; ++i)

{

int id1 = i \* width;

for (int j = 0; j < width; ++j)

{

int id2 = id1 + j;

int left = j - wnd - 1;

int right = j + wnd;

int top = i - wnd - 1;

int bot = i + wnd;

left = max(0, left);

right = min(right, width - 1);

top = max(0, top);

bot = min(bot, height - 1);

int dx = right - left;

int dy = (bot - top) \* width;

int idLeftTop = top \* width + left;

int idRightTop = idLeftTop + dx;

int idLeftBot = idLeftTop + dy;

int idRightBot = idLeftBot + dx;

int ptsCnt = ptsIntegral[idRightBot] + ptsIntegral[idLeftTop] - (ptsIntegral[idLeftBot] + ptsIntegral[idRightTop]);

double sumGray = integral[idRightBot] + integral[idLeftTop] - (integral[idLeftBot] + integral[idRightTop]);

if (ptsCnt <= 0)

{

continue;

}

data[id2] = float(sumGray / ptsCnt);

}

}

int s = wnd / 2 \* 2 + 1;

if (s > 201)

{

s = 201;

}

cv::GaussianBlur(depth, depth, cv::Size(s, s), s, s);

}

}

int main()

{

Mat src = imread("depth\_last.png", 0);

Mat depth(src.rows, src.cols, CV\_32F);

src.convertTo(depth, CV\_32F, 1 / 255.0);

namedWindow("MyW", CV\_WINDOW\_AUTOSIZE);

imshow("MyW", depth);

const int width = depth.cols;

const int height = depth.rows;

cv::Mat integralMap = cv::Mat::zeros(height, width, CV\_64F);

insertDepth32f(depth, integralMap);

namedWindow("MyWindow", CV\_WINDOW\_AUTOSIZE);

depth.convertTo(depth, CV\_8U, 255.0);

imshow("MyWindow", depth);

imwrite("insertDep\_last.jpg", depth\*255);

waitKey(0);

return 0;

}

**虚拟视点的合成**

// view\_synthsis\_01.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include "pch.h"

#include <iostream>

#include <string>

#include <opencv.hpp>

using namespace std;

using namespace cv;

void main()

{

string imgPath = "data/source\_images/teddy/";

Mat srcImgL = imread(imgPath + "left\_source.jpg");

Mat dispL = imread(imgPath + "insertDepth\_last.jpg", 0);

dispL = dispL / 4;

int imgHeight = srcImgL.rows;

int imgWidth = srcImgL.cols;

int channels = srcImgL.channels();

Mat dstImgL = Mat::zeros(imgHeight, imgWidth, CV\_8UC3);

uchar\* pImgDataL = (uchar\*)srcImgL.data;

uchar\* pDispDataL = (uchar\*)dispL.data;

uchar\* pDstDataL = (uchar\*)dstImgL.data;

VideoWriter writer("video07.avi", CV\_FOURCC('D', 'I', 'V', 'X'), 30, Size(imgWidth, imgHeight), 1);

int cnt = 0;

int viewCnt = 50;

while (cnt != 4)

{

for (int k = 0; k < viewCnt; k++)

{

dstImgL.setTo(0);

float interp;

if (cnt % 2 == 0) interp = (float)k / viewCnt;

else interp = float(viewCnt - k) / viewCnt;

for (int j = 0; j < imgHeight; j++)

{

for (int i = 0; i < imgWidth; i++)

{

uchar dispL = pDispDataL[j\*imgWidth + i];

float offsetL = dispL \* interp;

int idL = (int)(offsetL + 0.5); //计算视差值

if (idL + i >= imgWidth) continue;

//插值结果

int idxResult = (j\*imgWidth + i)\*channels;

int idx = (j\*imgWidth + i + idL)\*channels;

for (int chan = 0; chan < channels; chan++)

{

pDstDataL[idxResult + chan] = pImgDataL[idx + chan];

}

}

}

namedWindow("show");

//imwrite("show.jpg", dstImgL);

imshow("show", dstImgL);

waitKey(10);

writer << dstImgL;

}

cnt++;

}

writer.release();

}