ADVACAM
Imaging the Unseen

Beam hardening correction: Bath example

Online version:
https://wiki.advacam.cz/wiki/Beam_hardening_correction:_Bath_example

Contents

Introduction 3
Code 3

Introduction

This example using the config file named config.txt to configure steps of the process. File contents is like this:

```
o, First: line type: o - comment, r - referrence plate, m - measuring
o, Ref. cfg. line: r, thickness [mm], acqTime [s], num of expositions, output files: d data
/ p picture / b both / n none, name
r, 0, 0.005, 200, d, dir_beam
r, 1, 0.006, 200, d, ref_1
r, 2, 0.010, 200, d, ref_2
r, 3, 0.015, 200, d, ref_3
o, Measuring cfg. line: m, acqTime [s], num of expositions, bad px setting: d device / b BH
system, output files: d data / p picture / b both / n none, name
m, 0.005, 200, d, b, test1
m, 0.005, 200, d, b, test2
m, 0.01, 200, b, b, test3
m, 0.02, 200, b, b, test4
```

The example program read the line, display it in the console, will ask if it should do it and wait for user response. If yes, program acquire required number of frames, view simple histogram in the console, do the processing and save images, if reqired.

This example is using many auxiliary functions. Complete code is in the example project named MiniPixTpx3-Thickness-bath. To get usable acquisition times do a experimental exposures in the Pixet program, or get them using a second sample project named MiniPixTpx3-Thickness-auto that has auto-tuning and image preview in the console.

Code

```
/**
 * Copyright (C) 2021 ADVACAM
 * @author     Pavel Hudecek <pavel.hudecek@advacam.com>
 *
 * Example of thickness measuring with beam hardening compensation feature
 *  B - Bath version
 */

/* Used API functions:

pxcGetLastError
pxcInitialize
```

```
pxcGetDevicesCount
pxcGetDeviceName
pxcGetDeviceChipID

pxcSetTimepix3Mode

pxcMeasureSingleFrameTpx3
pxcGetMeasuredFrameTpx3
pxcMeasureMultipleFrames
pxcMeasureMultipleFramesWithCallback

pxcRegisterAcqEvent
pxcMeasureContinuousTest
pxcAbortMeasurement

pxcAddBHMask
pxcBHMaskCount
pxcRemoveBHMask
pxcApplyBHCorrection
pxcGetDeviceBadPixelMatrix
pxcGetBHBadPixelMatrix
pxcInterpolateBadPixels
pxcGetDeviceAndBHBadPixelMatrix

requires: pxcapi.h, pxcore.dll, minipix.dll, pixet.ini, link with pxcore.lib
*/

#include "pxcapi.h"
#include <cstring>
#include <iostream>
#include <conio.h>
#include <thread>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <sstream>
//#include <vector>
using namespace std;

// wait for user choice from 2 to 3 keys (key3=0 or key4=0 -> not used)
// key1/key2/key3 is lower case, keyboard input is independent
==================================
char choiceKey(char key1, char * text1, char key2, char * text2, char key3=0, char *
```

```
text3="", char key4=0, char * text4="") {
    char ch;

    while(1) {
        ch = _getch();
        if (ch<'a' && ch>='A' && ch<='Z') ch += 'z'-'Z'; // convert to lower case
        if (ch==key1) {
            if (text1[0]!=0) printf("%s\n", text1);
            break;
        } else if (ch==key2) {
            if (text2[0]!=0) printf("%s\n", text2);
            break;
        } else if (key3!=0) if (ch==key3) {
            if (text3[0]!=0) printf("%s\n", text3);
            break;
        } else if (key4!=0) if (ch==key4) {
            if (text4[0]!=0) printf("%s\n", text4);
            break;
        }
    }
    return ch;
}

#define ERRMSG_BUFF_SIZE    512
#define ENTER_ON            true
#define ENTER_OFF           false

// primary use to show function name, return code, last error message and optional enter
void printErrors(const char* fName, int rc, bool enter) { //
==================================
    char errorMsg[ERRMSG_BUFF_SIZE];
    pxcGetLastError(errorMsg, ERRMSG_BUFF_SIZE);
    if (errorMsg[0]>0) {
        printf("%s %d err: %s", fName, rc, errorMsg);
    } else {
        printf("%s %d err: ---", fName, rc);
    }
    if (enter) printf("\n");
}

#define ABORT_only  true
#define ABORT_ask   false

// if rc!=0 printError and return true or ask the user for quit (return true for quit)
```

```
bool errorCheck(char * fName, int rc, bool abortOnly) { //
===================================
    if (rc!=0) {
        printErrors(fName, rc, ENTER_ON);
        printf("Press A to abort or C to continue:\n");
        if (abortOnly==ABORT_ask) {
            if (choiceKey('a', "Choice: Abort", 'c', "Choice: Continue")=='c') return
false;
        }
        return true;
    }
    return false;
}

#define CHT_Si      0
#define CHT_CdTe    1
char chipType = CHT_Si;

const unsigned CHIP_xRes = 256;
const unsigned CHIP_yRes = 256;
const unsigned CHIP_pixels = CHIP_xRes * CHIP_yRes;

unsigned fileIdx=0;           // index of saved file
double saveTmpD[CHIP_pixels]; // temp for convert in saveTo.. overload

// Save image to file. Fname add info: exposition, min, max, limit (frames*1023), chip
// Data auto-ranged to min-max
// Filename is automatically incremented, but only in the session. Old files can be
overwritten.
void saveToBMP(double *frame, char *info, double expos, unsigned limit=0) { //
============================
    const unsigned HEAD_end = 54;
    static unsigned char fTmp[HEAD_end + CHIP_pixels * 3] = { // image file temp
        'B', 'M', 54, 0, 3, 0, 0, 0, 0, 0, 54, 0, 0, 0, 40, 0,
        0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 24, 0, 0, 0,
        0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,    0, 0, 0, 0, 0, 0
    };  // header of 24b BMP file, size 256x256
    // 2-6 file size, 10-14 data offset, 14-17 DIB head size, 18-21 width, 22-25 height,
28/29 bits per px, 38-41 x pix/m, 42-45 y pix/m
    char fName[150];
    char chip[]="CdTe";
    FILE *imageFile;
    unsigned n;
```

```
    printf("Saving... ");

    double min=4000000000.0, max=0.0;
    for (n=0; n<CHIP_pixels; n++) {
        if (frame[n]<min) min = frame[n];
        if (frame[n]>max) max = frame[n];
    }
    for (n=0; n<CHIP_pixels; n++) { // convert the data -----------------
        fTmp[HEAD_end+n*3+0] = (unsigned char)((frame[n]-min) * 255.0 / (max-min));
        fTmp[HEAD_end+n*3+1] = (unsigned char)((frame[n]-min) * 255.0 / (max-min));
        fTmp[HEAD_end+n*3+2] = (unsigned char)((frame[n]-min) * 255.0 / (max-min));
    }

    if (++fileIdx>999) fileIdx=0;
    if (chipType==CHT_Si) strcpy_s(chip, 5, "Si");
    sprintf_s(fName, 150, "img-%03u-%s exp_%1.2e_s min_%1.2e max_%1.2e lim_%u %s.bmp",
fileIdx, info, expos, min, max, limit, chip);

    if (fopen_s(&imageFile, fName, "wb")!=0) {
        printf("ERROR: Cannot open image file");
        return;
    }
    for (n = 0; n < HEAD_end + CHIP_pixels*3; n++) {
        fprintf(imageFile, "%c", fTmp[n]);
    }
    fclose(imageFile);
    printf("Saved file: %s\n", fName);
}

void saveToBMP(unsigned *frame, char *info, double expos, unsigned limit=0) { //
=========================
    for (unsigned n=0; n<CHIP_pixels; n++) {
        saveTmpD[n] = (double)frame[n];
    }
    saveToBMP(saveTmpD, info, expos, limit);
}

void saveToTXT(double *frame, char *info, double expos, unsigned limit=0) { //
===========================
    char fName[150];
    char chip[]="CdTe";
    FILE *imageFile;
    unsigned n;
```

```
    printf("Saving... ");

    double min=4000000000.0, max=0.0;
    for (n=0; n<CHIP_pixels; n++) {
        if (frame[n]<min) min = frame[n];
        if (frame[n]>max) max = frame[n];
    }

   if (++fileIdx>999) fileIdx=0;
    if (chipType==CHT_Si) strcpy_s(chip, 5, "Si");
    sprintf_s(fName, 150, "img-%03u-%s exp_%1.2e_s min_%1.2e max_%1.2e lim_%u %s.txt",
fileIdx, info, expos, min, max, limit, chip);

    if (fopen_s(&imageFile, fName, "wt")!=0) {
        printf("ERROR: Cannot open image file");
        return;
    }
    for (n = 0; n < CHIP_pixels; n++) {
        fprintf(imageFile, "%f", frame[n]);
        if (n < CHIP_pixels-1 && !(n % 256)) fprintf(imageFile, "\n");
        else fprintf(imageFile, ", ");
    }
    fclose(imageFile);
    printf("Saved file: %s\n", fName);
}

void saveToTXT(unsigned *frame, char *info, double expos, unsigned limit=0) { //
============================
    for (unsigned n=0; n<CHIP_pixels; n++) {
        saveTmpD[n] = (double)frame[n];
    }
    saveToTXT(saveTmpD, info, expos, limit);
}

// show 1 line histogram of single frame
void showHistogram(unsigned *data) { //
============================================================
    unsigned hist[100+5];
    unsigned n, max, hDiv, avg;

    for (n=0; n<100; n++) hist[n]=0;

    for (n=0, max=0; n<CHIP_pixels; n++) if (data[n]>max) max=data[n];
    hDiv = max/100;
```

```
    for (n=0; n<CHIP_pixels; n++) hist[data[n]/hDiv]++;

    for (n=0, max=0, avg=0; n<100; n++) {
        if (hist[n]>max) max=hist[n];
        avg += hist[n];
    }
    avg /= 100;

    printf("Histogram: 0|");
    for (n=0; n<100; n++) {
        if (hist[n]>=avg*10) {
            printf("#");
        } else if (hist[n]>=avg*2) {
            printf("*");
        } else if (hist[n]>=avg/5) {
            putchar('0' + (hist[n] * 5) / avg);
        } else if (hist[n]>=avg/10) {
            printf(".");
        } else {
            printf("_");
        }
    }
    printf("|max\n");
}

// integrate multiple acqs to "data" array
int multiAcq(double frameTime, unsigned frameCount, unsigned *data) { //
================================
    static unsigned short frameTotEvent[CHIP_pixels];
    static double tmp[CHIP_pixels];
    int rc;          // return codes
    unsigned fn, pn; // frame number, pixel number
    unsigned size = CHIP_pixels;   // buffer/chip pixel count, input/output
pxcMeasureSingleFrameTpx3

    cout << "multiAcq time=" << frameTime << " cnt=" << frameCount << ": ";
    for (pn=0; pn<CHIP_pixels; pn++) data[pn]=0;

    for (fn=0; fn<frameCount; fn++) {
        cout << ".";
        rc = pxcMeasureSingleFrameTpx3(0, frameTime, tmp, frameTotEvent, &size,
PXC_TRG_NO);
        if (rc!=0) return rc;
        for (pn=0; pn<CHIP_pixels; pn++) data[pn] += (unsigned)frameTotEvent[pn];
```

```
    }
    cout << endl;
    showHistogram(data);
    return 0;
}

inline string trim(string& str) { //
================================================================================
    str.erase(0, str.find_first_not_of(' '));       //prefixing spaces
    str.erase(str.find_last_not_of(' ')+1);         //surfixing spaces
    return str;
}

int measureThicknessBH() { //
================================================================================
    int rc;                          // return codes
    unsigned frameTmp[CHIP_pixels]; // measured data
    double frameOut[CHIP_pixels];   // BH corrected output data
    unsigned char badPixels[CHIP_pixels];// combined "bad" pixels from
pxcGetDeviceAndBHBadPixelMatrix
    double frameTime;                // frame acquisition time
    double refThick;                 // thickness of ref. plate
    char fileSet;                    // files setting: d - data, p - picture, b - both, n -
none
    char bpSet;                      // bad px setting: c - chip, b - BH system
    string name;                     // line name/comment and used in output filenames
    unsigned size = CHIP_pixels; // buffer/chip pixel count, input/output
pxcMeasureSingleFrameTpx3
    string line, valStr;             // for config lines decoding
    fstream cfgFile("config.txt", ios::in);
    unsigned n, lineCnt;
    char fileOutName[20], fileOutName2[20], fileOutName3[20]; // output filenames

    rc = pxcSetTimepix3Mode(0, PXC_TPX3_OPM_EVENT_ITOT); // set mode on device 0
    if (errorCheck("pxcSetTimepix3Mode", rc, ABORT_ask)) return -1;
    lineCnt=1;
   while (getline(cfgFile, line, '\n')) {
            cout << "L " << lineCnt++ << " > " << line << endl;
        stringstream linestream(line);
        getline(linestream, valStr, ',');
        switch (valStr.at(0)) { // ---------------------------------------------------
            case 'o': // comment
                break;
            case 'r': // reference
```

```
    case 'm': // measuring
        cout << "- Prepare this step and pres Y to do this, or press N to No: ";
        if (choiceKey('y', "Yes", 'n', "No")=='n') continue;
}

switch (valStr.at(0)) { // ------------------------------------------------------
    case 'o': // comment
        break;
    case 'r': // reference ----------------------------------------------
        getline(linestream, valStr, ','); // ref thick
        refThick = stof(valStr);
        getline(linestream, valStr, ','); // acq time
        frameTime = stof(valStr);
        getline(linestream, valStr, ','); // num of expositions
        n = stoi(valStr);
        multiAcq(frameTime, n, frameTmp); // measure
        getline(linestream, valStr, ','); // file saving settings
        fileSet = trim(valStr).at(0);
        //cout << "fs >" << valStr << "<" << endl;
        getline(linestream, name, ',');   // name of the step and files
        name = trim(name);
        if (fileSet!='n') {
            cout << "save " << name << " set=" << fileSet << endl;
            strcpy_s(fileOutName, 20, name.c_str());
        }
        if (fileSet=='d') {         // data file
            saveToTXT(frameTmp, fileOutName, frameTime*(double)n, 1023*n);
        } else if (fileSet=='p') {  // picture file
            saveToBMP(frameTmp, fileOutName, frameTime*(double)n, 1023*n);
        } else if (fileSet=='b') {  // both files
            saveToBMP(frameTmp, fileOutName, frameTime*(double)n, 1023*n);
            saveToTXT(frameTmp, fileOutName, frameTime*(double)n, 1023*n);
        }
        rc = pxcAddBHMask(frameTmp, CHIP_pixels, frameTime*(double)n, refThick);
        if (errorCheck("pxcAddBHMask", rc, ABORT_ask)) return -1;
        break;
    case 'm': // measuring -----------------------------------------------
        getline(linestream, valStr, ','); // acq time
        frameTime = stof(valStr);
        getline(linestream, valStr, ','); // num of expositions
        n = stoi(valStr);
        multiAcq(frameTime, n, frameTmp); // measure
        getline(linestream, valStr, ','); // bad px settings
        bpSet = trim(valStr).at(0);
```


```
                getline(linestream, valStr, ','); // file saving settings
                fileSet = trim(valStr).at(0);
                getline(linestream, name, ',');   // name of the step and files
                name = trim(name);
                if (fileSet!='n') {
                    cout << "save " << name << " set=" << fileSet << endl;
                    strcpy_s(fileOutName,  20, (name + "-raw").c_str());
                    strcpy_s(fileOutName2, 20, (name + "-bhc").c_str());
                    strcpy_s(fileOutName3, 20, (name + "-bpc").c_str());
                }
                if (fileSet=='d') {         // data file raw
                    saveToTXT(frameTmp, fileOutName, frameTime*(double)n, 1023*n);
                } else if (fileSet=='p') {  // picture file raw
                    saveToBMP(frameTmp, fileOutName, frameTime*(double)n, 1023*n);
                } else if (fileSet=='b') {  // both files raw
                    saveToBMP(frameTmp, fileOutName, frameTime*(double)n, 1023*n);
                    saveToTXT(frameTmp, fileOutName, frameTime*(double)n, 1023*n);
                }

                rc = pxcApplyBHCorrection(frameTmp, CHIP_pixels, frameTime*(double)n,
frameOut);
                if (errorCheck("pxcApplyBHCorrection", rc, ABORT_ask)) return -1;

                if (fileSet=='d') {         // data file BH corrected
                    saveToTXT(frameOut, fileOutName2, frameTime*(double)n, 1023*n);
                } else if (fileSet=='p') {  // picture file BH corrected
                    saveToBMP(frameOut, fileOutName2, frameTime*(double)n, 1023*n);
                } else if (fileSet=='b') {  // both files BH corrected
                    saveToBMP(frameOut, fileOutName2, frameTime*(double)n, 1023*n);
                    saveToTXT(frameOut, fileOutName2, frameTime*(double)n, 1023*n);
                }

                if (bpSet=='d') { // bad px setting: d - device, b - device + BH system
                    rc = pxcGetDeviceBadPixelMatrix(0, badPixels, CHIP_pixels);
                    if (errorCheck("pxcGetDeviceBadPixelMatrix", rc, ABORT_ask)) return -1;
                } else {
                    rc = pxcGetDeviceAndBHBadPixelMatrix(0, badPixels, CHIP_pixels);
                    if (errorCheck("pxcGetDeviceAndBHBadPixelMatrix", rc, ABORT_ask))
return -1;
                }
                //for (n=0; n<CHIP_pixels; n++) badPixels[n] = (unsigned char)frameTmp[n];
                rc = pxcInterpolateBadPixels(badPixels, frameOut, 256, 256);
                if (errorCheck("pxcInterpolateBadPixels", rc, ABORT_ask)) return -1;
```

```
                if (fileSet=='d') {         // data file BH+BP corrected
                    saveToTXT(frameOut, fileOutName3, frameTime*(double)n, 1023*n);
                } else if (fileSet=='p') {  // picture file BH+BP corrected
                    saveToBMP(frameOut, fileOutName3, frameTime*(double)n, 1023*n);
                } else if (fileSet=='b') {  // both files BH+BP corrected
                    saveToBMP(frameOut, fileOutName3, frameTime*(double)n, 1023*n);
                    saveToTXT(frameOut, fileOutName3, frameTime*(double)n, 1023*n);
                }
                break;
            default:
                cout << "Unknown line type: " << valStr << endl;
        }
    }
    return 0;
}

int main (int argc, char const* argv[]) { //
##################################################
    // Initialize Pixet
    int rc = pxcInitialize();
    if (rc) {
        printf("Could not initialize Pixet:\n");
        printErrors("pxcInitialize", rc, ENTER_ON);
        return -1;
    }

    int connectedDevicesCount = pxcGetDevicesCount();
    printf("Connected devices: %d\n", connectedDevicesCount);

    if (connectedDevicesCount == 0) return pxcExit();

    for (unsigned devIdx = 0; (signed)devIdx < connectedDevicesCount; devIdx++){
        char deviceName[256];
        memset(deviceName, 0, 256);
        pxcGetDeviceName(devIdx, deviceName, 256);

        char chipID[256];
        memset(chipID, 0, 256);
        pxcGetDeviceChipID(devIdx, 0, chipID, 256);
        printf("Device %d: Name %s, (first ChipID: %s), ", devIdx, deviceName, chipID);

        double bias;
        rc = pxcGetBias(devIdx, &bias);
        if (bias<0.0) {
```

```
            if (devIdx==0) chipType = CHT_CdTe;
            printf("Chip material detected: CdTe\n");
        } else if (bias==0.0) {
            if (devIdx==0) chipType = CHT_CdTe;
            printf("Chip material not detected!\n");
        } else {
            if (devIdx==0) chipType = CHT_Si;
            printf("Chip material detected: Si\n");
        }
    }
printf("================================================================================
================\n");

    measureThicknessBH();

    printf("Exiting...\n");
    return pxcExit(); // Exit Pixet core
}
```