Project Report

Fehime Betul CAVDARLI - 2339372

January 19, 2019

1 Introduction

In this project I try to implement one of the deghosting algorithms selected from the taxonomy that is explained in the [2] (Tursun & Akyuz) paper. The name of this algorithm is [1] *Bitmap Movement Detection: HDR for Dynamic Scenes* and introduced by Pece & Kautz. While implementing this algorithm I use [5] OpenCV (Open Source Computer Vision Library) with C++ and Visual Studio Professional 2015.

2 Setup

In order to run this project, you need to have the OpenCV library installed on your computer. Other tools is defined as Visual Studio. One of the points that should be considered here is that your version of Visual Studio and your OpenCV version must be compatible when downloading your OpenCV version. I use OpenCV 4.0 and it is compatible with the vc14 that means Visual Studio 2015. This OpenCV version also compatible with vc15 that also means Visual Studio 2017.
After downloading the OpenCV library from [5] into your computer, you should extract it to a suitable place on your computer. You should then go to this folder and add the *opencv- build- include* path to the location shown in Figure 2-(a). To enter this place, right click on your project and select properties as shown in Figure 1. Then you should add the *opencv- build- x64- vc14- lib* path to the location shown in Figure 2-(b). Finally, you should add the file that has the extension as *.lib* in this path to the location shown in Figure 2-(c).

3 Algorithm

Exposure fusion and other HDR techniques create well-exposed images from different exposures LDR images. For these techniques it is essential that the scene must be static. If there is any small motion between exposures they produce artefacts called ghosting.
In this paper they propose the *Bitmap Movement detection (BMD)* algorithm. It detects clusters of moving pixels for each input exposure images. By doing this, first, Median Threshold Bitmap algorithm that is introduced by Ward [3] is applied to each exposures to create stack of bitmaps. If the scene is static, they expect that each pixel preserve its bit value across all bitmaps. If the value in a changes they know that there is movement. So in order to detect movement pixels, they sum up all bitmaps. However, this may contain a certain amount of noise that could lead to incorrect movement detection. They refine it using a sequence of morphological dilation and erosion in order to generate the final motion map. After the final motion map is created, then it is converted into 'cluster map' where each identified cluster has a different label. This yield a labelled motion map. Exposure Fusion [4], which is a technique to fuse a bracketed sequence of exposure of LDR images, use this motion map as a guide to fill this cluster pixels in the final image with the best-exposed exposure. As a result each moving cluster contains values from a single exposure.

4 Implementation

I would like to emphasize some important methods when describing the implementation. The methods which are mentioned here but not described in details on this section are described in Table 1.
As the first method I would like to describe the method "PeceKautzMoveMask". This method

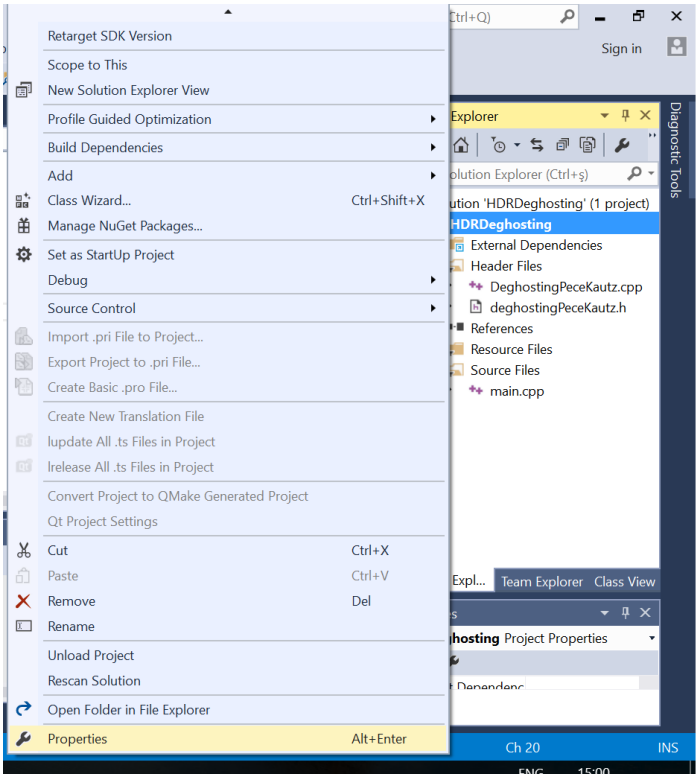
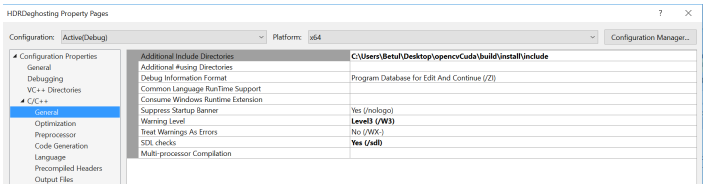
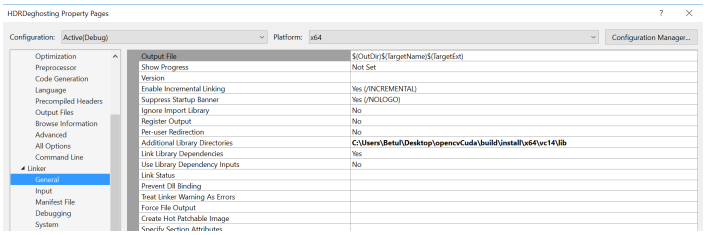


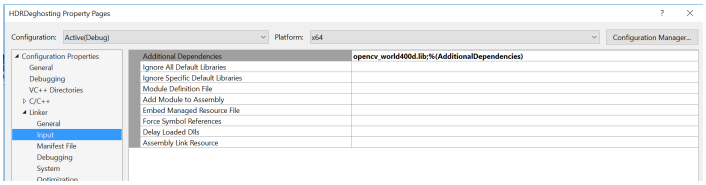
Figure 1: This image shows how you can enter the project properties to add some library paths



(a) Add *opencv- build- include* path to above (Additional Include Directories) location



(b) Add *opencv- build- x64- vc14- lib* path to above(Additional Library Directories) location



(c) Add the file that has the extension as *.lib* in this path to above (Additional Dependencies) location

Figure 2: Show how and where to add library paths

performs the operations mentioned in the *Motion Detection* section of the paper. In this method, firstly Median Threshold Bitmap (MTB) algorithm [Ward] is applied to all input images, and stack of bitmaps are created. Then all these bitmaps are summed to detect the movement pixels, and the resulting image as described in the paper as M*, but in our implementation it is the *moveMask* variable. Some morphological operations are applied for refining to the resulting image, if it contains noise that may cause wrong motion detection. These processes are erosion and dilation. The values for the size of these operations are specified at the end of the paper and I choose these values as default in the this method. You can see the default parameters of this method in our DehostingPeceKautz Class definition. Finally, after the final motion mapping is created, the connectedComponent method, which is one of the existing methods of OpenCV is called, which will ensure that each cluster identified with a different label.

```
cv::Mat DeghostingPeceKautz::PeceKautzMoveMask(int& num, vector<Mat>& imageStack, int iterations ,
                                               int ke_size, int kd_size, float ward_percentile ) {

    int n = imageStack.size();

    cv::Mat moveMask = WardComputeThreshold(imageStack[0], ward_percentile);

    cv::Mat mask;
    for (int i = 1; i < n; i++) {
        mask = WardComputeThreshold(imageStack[i], ward_percentile);
        moveMask = moveMask + mask;
    }

    moveMask.setTo(0, moveMask == n);
    moveMask.setTo(1, moveMask > 0);

    cv::Mat kernel_d = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(kd_size, kd_size));
    cv::Mat kernel_e = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(ke_size, ke_size));

    for (int i = 1; i < n; i++) {
        cv::dilate(moveMask, moveMask, kernel_d, cv::Point(-1, -1), 1);
        cv::erode(moveMask, moveMask, kernel_e, cv::Point(-1, -1), 1);
    }
    cv::Mat moveMask_tmp, moveMaskBinary;
    moveMask.convertTo(moveMaskBinary, CV_8U);
    num = connectedComponents(moveMaskBinary, moveMask_tmp, 4, CV_32S);

    moveMask_tmp.convertTo(moveMask_tmp, CV_32F);
    cv::Mat moveMaskOut = moveMask_tmp.clone();
    cv::multiply(moveMask_tmp, moveMask, moveMaskOut);
    moveMaskOut.setTo(-1, moveMaskOut == 0);
    return moveMaskOut;
}
```

Figure 3: PeceKautzMoveMask method of our DeghostingPeceKautz class

The other part that I want to mention is not the method, but I think it's an important piece of code. In the paper, it is stated that the best-available exposure is used to fill in the corresponding pixels of each individual cluster in the final image. By doing this, it is mentioned that Exposure Fusion's multi-scale blending method is used. In fact, what it means here is to create an image pyramid for each channels of the input images and for the weight map that is created by averaging three quality measures, contrast, saturation and well-exposedness for each exposures. Then these two pyramids are collapsed and the final HDR image is created.

Note that you will see a code part as *baseDetail* in the code above . This is a structure variable created by me. The members of this structure variable are cv :: Mat and vector of cv :: Mat. My purpose in defining this is that the base and detail layers must be kept at the same time in the image pyramid.

5 Results

My main goal for choosing this deghosting algorithm is that OpenCV does not have any deghosting class. Therefore I want to implement a deghosting algorithm myself in OpenCV. In addition, this algorithm contains simple bitmap operations, so it is already fast however it may be accelerated with OpenCV.

```
vector<baseDetail> tf;
for (int i = 0; i < n; i++) {
        // Laplacian pyramid : image
        vector<baseDetail>  pyrImg = pyrImg3(imageStack[i]);

        //Gaussian pyramid: weight_i
        cv::Mat results;
        divide(weight_move[i], total, results);
        cv::Mat weight_i = RemoveSpecials(results);
        baseDetail  pyrW = pyrGaussGen(weight_i.clone());

        //Multiplication image times weights
        vector<baseDetail> tmpVal = pyrLstS2OP(pyrImg, pyrW);

        if (i == 0) {
                tf = tmpVal;
        }
        else {
                //accumulation
                tf = pyrLst2OP(tf, tmpVal);
        }
}

//Evaluation of Laplacian/Gausian Pyramids
cv::Mat imgOut = Mat::zeros(cv::Size(c, r), CV_32FC3);
vector<Mat>bgr(3);
split(imgOut, bgr);
for (int i = 0; i < col; i++)
{
        bgr[i] = pyrVal(tf[i]);
}
```

Figure 4: Part of the createPeceKautz method of our DeghostingPeceKautz class

Table 1: Brief description of the methods which are not explained on Implementation section

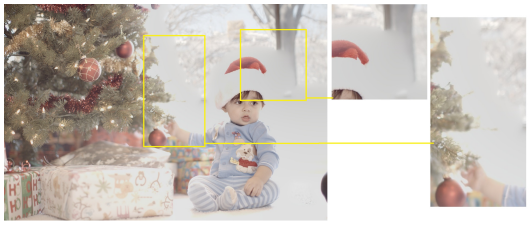
Method	**Definition**
WardComputeThreshold	Create bitmaps from images by applying MTB algorithm
pyrImg3	Create the Laplacian image pyramid for each channels of input images
pyrGaussGen	Create Gaussian image pyramid for weight map
pyrLstS2OP	Multiply weight map by each channels of input images separately
pyrLst2OP	Sum up all the image pyramid of exposures and create one image pyramid that include channels
pyrVal	Sum up all image pyramid layers and create one final image

I try my implementation with different images that I was tried previously in my assignment 2 of this lecture. In that assignment I used this algorithm [6] which have been implemented in MATLAB environment so I compare the algorithm results in terms of the resulting image/colors and running times between MATLAB and OpenCV implementation.
Figure 5 shows that the results of two implementation are quite different from each other. For example, when we look at the first row in this figure, it is seen that in MATLAB implementation, the environment is more suitable and the satured areas are less than the OpenCV implementation, but the ghosting artefacts have been removed in both results. Also in the second row SantaLittleHelper example images, the MATLAB implementation is slightly better than the OpenCV implementation. However, the ghosting artefacts could not be completely removed in both image. In addition, in the last row example images, the OpenCV application succeeded in correcting ghosting effects of moving objects, but weakened the overall color tone of the image, according to the MATLAB implementation. When I look at Table 2 to compare the running times of both implementation, it has been observed that it cannot be accelerated in others except for an image that is the stack ghost image.
Note that the most important thing I need to say here is that I do not need an extra tone-mapping in the HDR images created as a result of the MATLAB implementation, while my OpenCV implementation requires tone-mapping in the resulting HDR image. I use Reinhard tone-mapping algorithm to display in this report but implementation also includes some other tone-mapping results of HDR image. The both MATLAB and OpenCV implementation of this algorithm are run on the computer whose name is HP EliteBook x360 1030 G2. Total memory is 16.00 GB and the processor is Intel (R) Core(TM) i7 - 7600U CPU @2.80 GHz.

(a) Akyuz Image with MATLAB implementation

(b) Akyuz Image with OpenCV implementation

(c) SantaLittleHelper Image with MATLAB implementation



(d) SantaLittleHelper Image with OpenCV implementation

(e) Stack Ghost Image with MATLAB implementation

(f) Stack Ghost Image with OpenCV implementation

Figure 5: The right side(b-d-f) is the results of openCV implementation and the left side (a-c-e) is the results of the MATLAB implementation.

Table 2: Running times in seconds both MATLAB and OpenCV implementation of Pece Kautz deghosting algorithm.

Implementation	**Stack Ghost.**	**SantaLittleHelper**	**Akyuz Image**
MATLAB	5.54	8.64	4.67
OpenCV	3.665	19.0	13.8

References

[1] Pece F., Kautz J.: Bitmap movement detection: HDR for dynamic scenes. In Visual Media Production (CVMP), Conf. on (2010), IEEE, pp. 18. 4, 8, 13

[2] Tursun, O. T., Akyuz, A. O., Erdem, A., & Erdem, E. (2015, May). The state of the art in HDR deghosting: a survey and evaluation. In Computer Graphics Forum (Vol. 34, No. 2, pp. 683-707).

[3] G. Ward, Fast, robust image registration for compositing high-dynamic range photographs from handheld exposures, Journal of Graphics Tools, vol. 8, no. 2, pp. 1730, 2003

[4] T. Mertens, J. Kautz, and F. Van Reeth, Exposure fusion, in Pacic Graphics, 2007, pp. 382390.

[5] `https://opencv.org/opencv-4-0-0.html`

[6] `https://github.com/banterle/HDR_Toolbox`