Tommi Järvenpää
2553157

Time used: ~21.5h

Assignment 4 Report

In this assignment, I developed a C++ implementation for calculating Zero-mean Normalized Cross Correlation (ZNCC) between two images, obtaining the Cross Correlation between them, and finally performing Occlusion Filling. The report's folder structure is shown in Figure 1. More detailed information on the functions discussed in this report can be seen in the extensive commenting on the code files. The codes were created on Windows 10 with Visual Studio 2019. Sources for different code snippets are displayed in the code files and at the end of this report. The following sections describe each part of this assignment

```
|
|->Assignment4_Report.pdf
|->lodepng.cpp & .h
|->main.cpp
|->ImageFunctions.cpp & .h
|->im0_grayscaled.png
|->im1_grayscaled.png
|->im0_zncc.png & normalized
|->im1_zncc.png & normalized
|->cross_check.png & normalized
|->occlusion_fill.png & normalized
```

Figure 1: Folder structure

Image Loading & Saving

The functions **ReadImage** and **WriteImage** are implemented in *ImageFunctions.cpp*. These functions utilize LodePNG[1] to read and write the image files of this assignment. [2] and [3] were used as sources for using LodePNG with C++ to load and save images. Both functions output profiling info, i.e. text and their execution time[4]. Image reading takes approx. 1.3 seconds, and writing 1.8 seconds

Resizing Images

The function **ResizeImage** is implemented in *ImageFunctions.cpp*. It reduces a given RGBA image's size by four. This is done by dropping pixels, so that only every 4th pixel is saved. The function outputs its execution time and the new image dimentions. Takes approx. 0.05 seonds.

Image Grayscaling

GrayScaleImage is implemented in *ImageFunctions.cpp*. This function takes a RGBA image and removes its colors by summing a pixels weighted red, green, and blue values. The weight were found from [5]. Takes approx. 0.22 seconds.

Calculating the Zero-mean Normalized Cross Correlation

CalcZNCC is implemented in *ImageFunctions.cpp*. The functions contents is based on the mathematical formula and the pseudocode provided in this assignment's documentation. The function calculates a ZNCC value for each pixel based on the given left and right image. When executed in *main.cpp*, this function is called twice. First with **im0.png** set as the left image and **im1.png** as the right image, and a second time with **im1.png** as rigth and **im0.png** as left. This allows us to get the two images required by **CrossCheck**. The values for the window and disparity values, in this function, were selected by trying different combinations. They are: Window y-axis = 20, window x-axis = 15, window size = 300, and maximum disparity = 55. While, e.g., increasing the window size would make the results better, the window's size had to be limited so execution times would stay bareable. When calculating ZNCC with **im1.png** as the left image, the maximum disparity was set to 0, and minimum to -55. This function outputs its execution time and text. Takes approx. 470 seconds., i.e. about 8 minutes, so a very long time.

Image Cross Checking

CrossCheck is implemented in *ImageFunctions.cpp.* This function takes the two images created with **CalcZNCC** and creates a new image. The left and right image are compared pixel-by-pixel by calculating the absolute value of left pixel minus right pixel. If the absolute value exceeds a threshold, zero is assigned to the corresponding pixel in the new image. If the threshold was not exceeded, the pixel value from the right image is used. 3 was used as the threshold. Starting from 8 was succested in the documentation, but 3 produced better looking results in this implementation. This function outputs its execution time and text. Takes approx. 0.02 seconds.

Occlusion Filling

OcclusionFill is implemented in *ImageFunctions.cpp.* It takes the image created by **CrossCheck** and replaces the value of pixels which were previously assigned to zero. New values are found by spreading across the pixel's neighborhood until a non-zero value is reached. This will then serve as the pixel's new value. In the case that no new value is found, the program stops. The searched neighborhood's size is 150x150. This makes sure that a non-zero value is always found in the case of our test images, but also keeps the neighborhood small enough for this task. Takes approx. 7.7 seconds.

Additional Work: Output Image Normalization

The images created by the functions in this assignment produce images that are very dark looking. By normalizing their pixel values, they can be made much better looking. Normalization was done with **NormalizeImage**, implemented in *ImageFunctions.cpp*. It uses the C++ functions *max_element()* and *min_element()* to first find the images maximum and minimum pixel value. A loop then goes through each pixel and uses the formula *255*(val-min) / (max-min)* [6] to calculate a new value in the range [0,255]. Takes approx. 0.19 seconds

Images

Figure 2 illustrates the programs profiling info and output. Figures 3 and 4 elucidate the ZNCC output for **im0.png** and **im1.png**. Figure 5 shows the result obtained after Occlusion Filling. Figure 6 shows how the images look after normalization.

```
Microsoft Visual Studio Debug Console
Reading image im0.png with lodepng
Image im0.png loaded. Took 1553605 microseconds

Reading image im1.png with lodepng
Image im1.png loaded. Took 1359452 microseconds

Donwscaling im0.png by four. New dimetions: width = 735, height = 504
Image downscaled. Took 52970 microseconds

Donwscaling im1.png by four. New dimetions: width = 735, height = 504
Image downscaled. Took 54619 microseconds

Grayscaling im0.png
Image grayscaled. Took 28349 microseconds

Grayscaling im1.png
Image grayscaled. Took 28397 microseconds

Saving image to im0_grey.png
Image im0_grey.png saved. Took 215996 microseconds

Saving image to im1_grey.png
Image im1_grey.png saved. Took 198696 microseconds

Creating a ZNCC disparity map. Left img=im0.png, Right img=im1.png
ZNCC disparity map created. Took 475581529 microseconds

Creating a ZNCC disparity map. Left img=im1.png, Right img=im0.png
ZNCC disparity map created. Took 470101077 microseconds

Saving image to im0_disparity.png
Image im0_disparity.png saved. Took 170770 microseconds

Saving image to im1_disparity.png
Image im1_disparity.png saved. Took 161217 microseconds

Performing Cross Check between im0 and im1
Cross Check done. Took 29043 microseconds

Saving image to cross_check.png
Image cross_check.png saved. Took 159626 microseconds

Performing Occlusion Fill by finding nearest non-zero neighbors
Occllusion Fill done. Took 7705343 microseconds

Saving image to occlusion_fill.png
Image occlusion_fill.png saved. Took 158833 microseconds

Normalizing im0.png disparity map
Image normalized. Saving image to im0_normalized.png
Image im0_normalized.png saved. Took 167789 microseconds

Normalizing im1.png disparity map
Image normalized. Saving image to im1_normalized.png
Image im1_normalized.png saved. Took 169700 microseconds

Normalizing Cross Check
Image normalized. Saving image to cross_check_normalized.png
Image cross_check_normalized.png saved. Took 182889 microseconds

Normalizing Occulsion Fill
Image normalized. Saving image to occlusion_fill_normalized.png
Image occlusion_fill_normalized.png saved. Took 172510 microseconds
```

Figure 2: Program output

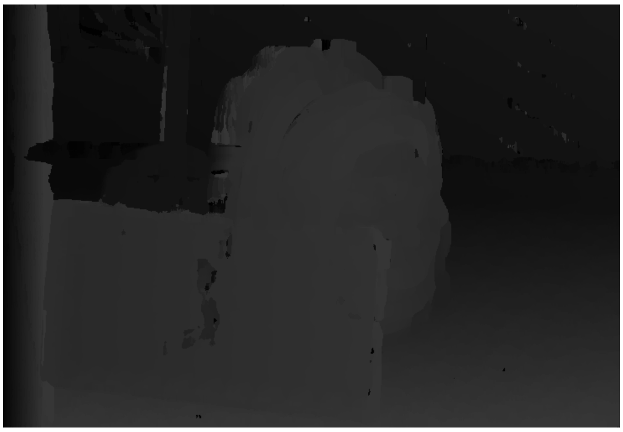

Figure 3: im0.png ZNCC result

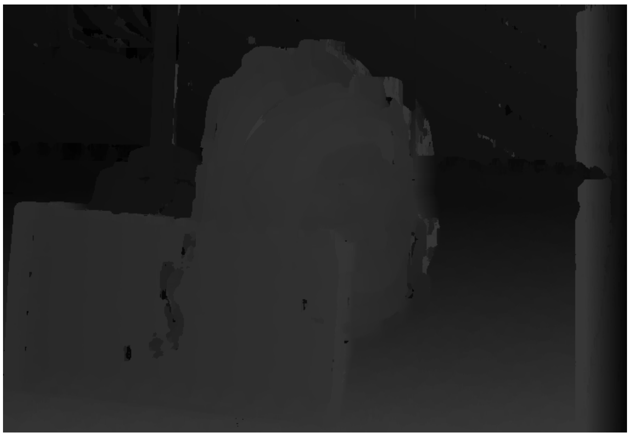

Figure 4: im1.png ZNCC result

Figure 5: Occlusion Fill result

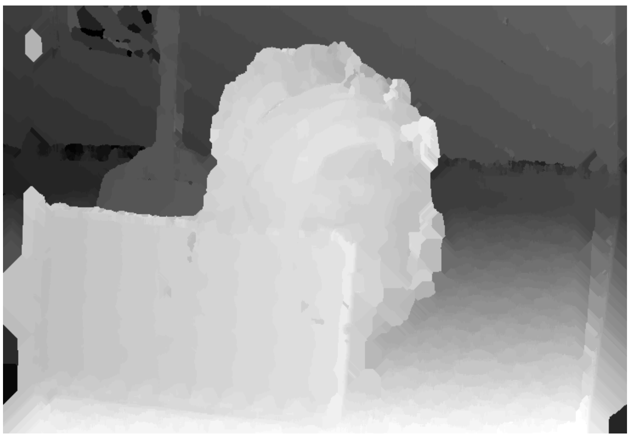

Figure 6: Occlusion Fill result normalized

References

[1] “LodePNG”. URL: https://lodev.org/lodepng/. Accessed: 29.01.2021
[2] “LodePNG Decode Example”. URL: https://raw.githubusercontent.com/lvandeve/lodepng/master/examples/example_decode.cpp. Accessed: 29.01.2021
[3] “LodePNG Encode Example”. URL: https://raw.githubusercontent.com/lvandeve/lodepng/master/examples/example_encode.cpp. Accessed: 29.01.2021
[4] “Acquiring high-resolution time stamps”. URL: https://docs.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps. Accessed: 29.01.2021
[5] “Standard RGB to Grayscale Conversion”. URL: https://stackoverflow.com/questions/17615963/standard-rgb-to-grayscale-conversion. Accessed: 29.01.2021
[6] “How to Normalize Data Between 0 and 100”. URL: https://www.statology.org/normalize-data-between-0-and-100/. Accessed: 14.02.2010