

CS3221 Project 1

NAME: GOH KA HIAN

Student ID: A0164383B

|  |
| --- |
| 1. Tabulation of results |

There are two machine used for testing,

|  |  |  |
| --- | --- | --- |
|  | **Machine A** | **Machine B** |
| **CPU** | Intel® Core™ i5-4200U CPU | Intel® Core™ i5-4570 CPU |
| **GPU** | Nvidia GeForce GT 740M | Nvidia GeForce GTX 780 |

Figure 1.1 – shows the CPU and GPU used by machine A and B

From, Figure 1.1, we can see that Machine B has a better CPU and GPU specification than Machine A.

For this project, all testing was done using window 10 with chrome web browser.

A pipelined sequence of filtering functions has been implemented:

GrayScale -> Sobel Edge detection -> Blurring -> Sharpening -> Emboss -> Blinking -> Animation

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **None** | **GrayScale** | **Sobel** | **Blur** | **Sharpen** | **Emboss** | **Blink** | **Animation** |
| **Intel® Core™ i5-4200U CPU** | 60 | 4.5 | 4 | 4.2 | 4.4 | 4.6 | 1.8 | 2.2 |
| **Nvidia GeForce GT 740M** | 60 | 40 | 35 | 35 | 38 | 35 | 40 | 40 |
| **Intel® Core™ i5-4570 CPU** | 60 | 6.3 | 6 | 6.2 | 6.2 | 6.2 | 2.5 | 3 |
| **Nvidia GeForce GTX 780** | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |

Figure 1.2 – shows the AVG FPS when a single filtering function is enabled for each CPU and GPU

We can note that there are little differences on the impact on AVG FPS for each filtering function, one thing they have in common is all the filtering function only uses one kernel function provided by GPU.js.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Number of Filter and Effect:** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **Intel® Core™ i5-4200U CPU** | 4.5 | 3 | 2.1 | 1.7 | 1.4 | 1.2 | 0.4 |
| **Nvidia GeForce GT 740M** | 40 | 24 | 15.8 | 11.5 | 9.5 | 8 | 7.1 |
| **Intel® Core™ i5-4570 CPU** | 6.4 | 4 | 3 | 2.4 | 2 | 1.7 | 0.6 |
| **Nvidia GeForce GTX 780** | 60 | 43 | 29 | 22 | 18.2 | 15.2 | 13.1 |

Figure 1.3 – shows the AVG FPS for each CPU and GPU based on number of filtering functions enabled.

One simple observation is that when there is more filtering function enabled, we can observe that the workload increases both for GPU and CPU.

//talk about how scaling

|  |
| --- |
| 1. Separable Convolution |

Most of the filter are implemented using a 3x3 kernel direct convolution for this project. Direct convolution requires M\*N multiplications for a MxN matrix, and hence an attempt has been made to lessen the numbers of multiplications by using separable convolution which only requires M+N multiplications.

An M\*N matrix is separable if it can become a product of a 1 row and a 1 column matrices.

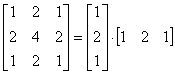


Figure 2.1 - Shows a 3x3 sobel kernel separated in to a 3x1 and 1x3 matrix

Using direct convolution:

http://www.songho.ca/dsp/convolution/files/conv2d_eq23.gif

Figure 2.2 – Direct Convolution for a single input

From figure 2.2, it is observed that there are 9 multiplications per input in total using direct convolution.

Using Separable convolution:

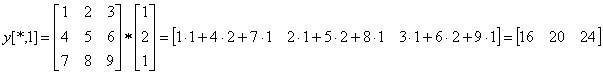


Figure 2.3 – Separable Convolution step 1 (vertical 1D convolution) for 3 input

Firstly, by performing a vertical convolution. Note that though figure 2.3 shows 9multiplication, it is actually 9multiplications for 3input, 3 for each input. A disadvantage of separable convolution is that we require an auxiliary storage for the first convolution.

http://www.songho.ca/dsp/convolution/files/conv2d_eq25.gif

Figure 2.4 – Separable Convolution step 2 (horizontal 1D convolution) for single input

Secondly, performing a horizontal convolution, which requires 3 multiplications. In total, 3+3=6 multiplications for a single input for a 3x3 matrix convolution.

For this project, a direct convolution kernel function has been broken down to 2 kernel function, one for performing 1D vertical convolution, another for 1D horizontal convolution. Both of which resulted in the same graphical outcome with different level of performance.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **3x3 Direct Convolution** | **3x3 Separable Convolution** | **5x5 Direct Convolution** |
| **No. of kernel function used** | 1 | 2 | 1 |
| **Total No. of multiplication**  **600x800 image** | 4320000 | 2880000 | 7200000 |
| **Intel® Core™ i5-4200U CPU,**  **AVG FPS** | 4.2 | 3 | 3.7 |
| **Nvidia GeForce GT 740M,**  **AVG FPS** | 35 | 25 | 3.8 |
| **Intel® Core™ i5-4570 CPU,**  **AVG FPS** | 6.2 | 4.4 | 5.5 |
| **Nvidia GeForce GTX 780,**  **AVG FPS** | 60 | 44 | 5.5 |

Figure 2.5 – tabulation of result from the project implementation with blur filter implemented differently

The result has shown a decreased performance for separable convolution as shown from fig 2.5 despite having 1.5times less multiplication, this is most likely due to the separable convolution requiring 2 kernel function instead of 1 kernel function. As it was not clear how gpu.js was implemented fully, it was hard to analyze how the kernel function works. However, it is clear that utilizing more kernel function results in more overhead, possibly due to the switching of processors.

A 5x5 direct convolution for blurring filter has also been implemented. A 5x5 direct convolution takes into account of more neighboring input, which could be more suited in cases which the filtering function requires higher level features. However, it requires more computation time, 15multiplication per input for a 5x5 kernel.

|  |
| --- |
| 3. Discussion |

This project was implemented mainly using JavaScript does not support multi-threading, gpu.js kernel function was what made it possible to allow parallel execution of thread. However, kernel function is expensive, as it was clear that an implemented separable convolution that made use of multiple kernel function has incurred higher performance cost than a direct convolution that uses a single kernel function despite having less multiplication operations.

//any other limitation. machine?