Image Processing With CUDA

Final report

Submitted for the MEng in

Computer Science for Games Programming

April 2024

by

Dawid Kisielewski

Word count: XXXXX

Abstract

Image processing is a field with a wide use of GPU acceleration. GPU’s parallel nature makes them perfect for doing computation on images as each thread can represent one pixel of the image and most algorithms don’t need to manipulate shared data. Exploring the differences between image processing algorithms running sequentially on the CPU and parallel on the GPU allows us to develop an understanding of the characteristics of both GPU programming as well as GPU architecture.

**Explain gpu, do not use don’t. More in depth explanation of your project, explain exactually what you are doing.**

Acknowledgements

Contents

[Abstract i](#_Toc164243418)

[Acknowledgements ii](#_Toc164243419)

[1 Introduction 3](#_Toc164243420)

[1.1 Background to the project 3](#_Toc164243421)

[1.2 Aims and objectives 3](#_Toc164243422)

[1.2.1 Objective 1 – Research Image Processing Algorithms 3](#_Toc164243423)

[1.2.2 Objective 2 – Creating CPU implementations 3](#_Toc164243424)

[1.2.3 Objective 3 – Research GPU Architecture 3](#_Toc164243425)

[1.2.4 Objective 4 – Creating GPU Implementation 3](#_Toc164243426)

[1.2.5 Objective 5 – Compare CPU and GPU Performance 3](#_Toc164243427)

[1.3 Research question 4](#_Toc164243428)

[2 Literature review 5](#_Toc164243429)

[2.1 Image Processing 5](#_Toc164243430)

[2.1.1 Edge Detection 6](#_Toc164243431)

[2.1.2 Image Segmentation 8](#_Toc164243432)

[2.1.3 Noise Reduction 8](#_Toc164243433)

[2.2 GPU 8](#_Toc164243434)

[2.2.1 GPU Architecture 8](#_Toc164243435)

[2.2.2 GPU Compute 8](#_Toc164243436)

[2.3 Similar Studies 9](#_Toc164243437)

[3 Requirements 10](#_Toc164243438)

[3.1 Product requirements 10](#_Toc164243439)

[3.2 Functional requirements 10](#_Toc164243440)

[3.2.1 Interfaces 10](#_Toc164243441)

[3.2.2 Functional Capabilities 10](#_Toc164243442)

[3.2.3 Performance Levels 10](#_Toc164243443)

[3.2.4 Data Structures/Elements 10](#_Toc164243444)

[3.2.5 Safety 10](#_Toc164243445)

[3.2.6 Reliability 10](#_Toc164243446)

[3.2.7 Security/Privacy 10](#_Toc164243447)

[3.2.8 Quality 10](#_Toc164243448)

[3.2.9 Constraints and Limitations 10](#_Toc164243449)

[3.2.10 Performance requirements 10](#_Toc164243450)

[3.3 Design constraints 10](#_Toc164243451)

[4 Design 11](#_Toc164243452)

[4.1 Software design 11](#_Toc164243453)

[4.2 Hardware design 11](#_Toc164243454)

[4.3 Experimental design 11](#_Toc164243455)

[5 Implementation and testing 12](#_Toc164243456)

[5.1 Implementation 12](#_Toc164243457)

[5.2 Testing 12](#_Toc164243458)

[6 Evaluation and discussion of results 13](#_Toc164243459)

[7 Conclusion 14](#_Toc164243460)

[References 15](#_Toc164243461)

# Introduction

## Background to the project

Image processing is the process of computing an image to enhance it or modify the image to ease the extraction of data from it. These processes can vary and accomplish different things, anything from removing grain and noise from images, segmenting the image into different sections to detecting and highlighting edges in the image as well as many other tasks.

Graphics Processing Units (GPUs) were originally created to accelerate graphics pipelines such as DirectX and Vulkan using hardware. Over the years they have gained thousands of more cores making them excellent at computing many tasks other than graphics calculations. Image processing algorithms are one of these tasks that have benefited from using the GPUs parallel nature to decrease the time taken for the algorithms to complete.

The are a few different APIs that allow you to write programs for the GPU. A few examples are CUDA, OpenCL, compute shaders in graphics APIs and many others. CUDA is a closed source API made by Nvidia for Nvidia GPUs, which means that programs created using CUDA can only execute on Nvidia GPUs. Recently there has been a translation layer created called ZLUDA that allow non-Nvidia GPUs to run the programs.

## Aims and objectives

### Objective 1 – Research Image Processing Algorithms

To research different image processing techniques and algorithms. This objective will allow me to research commonly used image processing algorithms as well as how they are implemented. By knowing the available algorithms, I will be able to select appropriate algorithms to implement for the purposes of studying the performance differences between the CPU and GPU.

### Objective 2 – Creating CPU implementations

To implement the algorithms chosen in the first objective on the CPU. In order to compare performance differences between the CPU and GPU it requires the algorithms to exist on both CPU and GPU. This objective aims to create the selected algorithms on the CPU to be ready for comparison to the GPU implementations.

### Objective 3 – Research GPU Architecture

To research GPU architectures and how computation is executed on it. By researching GPU architectures, more specifically the architectures of the GPUs which are going to be used to run the image processing techniques, this will help develop an understanding of how to write performant code on the GPU and how GPU programming is different from CPU programming.

### Objective 4 – Creating GPU Implementation

To implement the algorithms on the GPU, which have already been implemented on the CPU. This objective will implement the algorithms chosen in the first objective on the GPU. To ensure that the GPU implementation is as performant as we can get it, we will be using Nsight compute.

### Objective 5 – Compare CPU and GPU Performance

To compare performance metrics produced by running the algorithms with different sized images and parameters, we will be able to create graphs to analyse the differences between the performances. Using the knowledge gained from researching GPU architecture and comparing the results gained to the results of preexisting studies, this will allow us to come to a clearer conclusion.

## Research question

*“How does the performance of image processing Algorithms differ between running on the CPU and GPU?”*

The aim of this project its to analyse the differences between image processing algorithms on CPUs and GPUs. This will involve aliasing the differences in time to run, any variation in output result as well as the actual time it takes to develop and implement the algorithms.

# Literature review

## Image Processing

Image processing is the field of running computation on an image resulting in the image being modified to ease the process of extracting data from the image. There is a wide range of modifications that can occur to the image such as removing graininess and other noise, segmenting the image into different sections, detecting, and enhancing edges and many other modifications (Niblack, W. 1986)

Image processing is used in a wide range of applications. The most common application of image processing is noise reduction. All sensory equipment will have some sort of noise that will make it harder to use the data gathered, which also applies to digital image sensors. Images produced by these sensors will have some noise meaning noise reduction/suppression algorithms are used to remove some of the noise while reserving the actual image. (Alvi, F. 2023)

Another commonly used technique is edge detection which is the process of finding edges in an image. Usually the result of the algorithm is a black and white image where all the edges are shaded white and the rest of the image is black. Some algorithms display a gradient depending on how strong the edge issuch as the Sobel operator (see section 2.1.1.1). A smaller set of algorithms display the direction of the edge using two colours. Edge detection can be used in many applications such as image stylisation, parts of computer vision pipeline and many others. (Encyclopaedia of Mathematics. 2011)

Image segmentation is another example of an image processing technique which is a vast field that encompasses a lot of different other techniques. The general purpose of segmentation is to separate different parts of an image from one another. An example of this could be separating different structures in an organ or to separate different parts of an image according to colour.

The techniques we will be focusing on will be edge detection, image segmentation and noise reduction. We will be using the image in figure 1 to display the results of each image processing algorithm.

Fig 1: An image widely used test image processing algorithms. The image is of model Lenna Forsén

Figure 1: An image widely used to test image processing algorithms. The image is of model Lena Forsén (https://imageprocessingplace.com/root\_files\_V3/image\_databases.htm)

### Edge Detection

There is a variety of different edge detecting algorithms. We will be focusing on the classical approach for image processing as modern techniques involve neural networks and machine learning techniques that already leverage the power of the GPU. There are a handful of algorithms we can study which mainly use the method of convolution to produce a result. These algorithms are:

* Sobel
* Canny
* Roberts
* Prewitt

#### Sobel

The Sobel operator is one of the most common edge detection algorithms used, that runs on grayscale images using two separate convolutions. One convolution finds the horizontal edges and the other find the vertical edges. Both convolutions are 9x9 and are rotations of one another.

is the convolution of the kernel for the vertical edges and the is the convolution of the kernel for horizontal edges. These kernels are run for every pixel resulting with two separate images each containing separate edge directions. The magnitude is then found using the two directions calculated previously using the Pythagorean theorem followed by normalizing it. The resulting value is the edge intensity and can be set to the output pixel.

A person wearing a hat

Description automatically generated

Figure 2: The original image (Left) and the result of the Sobel (Right) (Generated using my implementation)

As seen from figure 2 the output result shows a gradient across an edge, where the brighter the pixel appears the stronger the edge is in that location.

A useful side effect of calculating the vertical and horizontal edges separately means that the angle of the edge can be calculated and displayed in colour. There are quite a few applications where this function is used, such as artistic uses of the difference of gaussians (Winnemoller H. 2012).

#### Canny

The Canny edge detector gets its name from John F. Canny who created it. The edge detector starts of by preforming the same steps as a Sobel i.e. it uses the two kernels to find horizontal and vertical edges, then it finds the direction of the edge as well as the gradient (the edge). After performing these steps it continues on to perform two additional steps that result in a cleaner and more usable result.

After the Sobel operator, it finds the centre of each edge changing the image from an image with a gradient over the edges, to an image where all the edges are a single pixel wide. This is possible due to the ability of the Sobel operator to calculate the direction of the edge meaning that we can look at the pixel values across the edge and find the strongest value across the gradient. The pixel with the greatest value will be set as the centre of the edge and the rest of the pixels across the edge will be set to black. The last step in the process is to find the edges that we are sure are actual edges in the image and not a slight change in the colour of an object or noise. To do this we perform hysteresis thresholding which requires two additional parameters to be entered to complete. These parameters are a lower bound and an upper bound. The first step in the threshold is to discard all the pixels under the lower bound entered. Next, it marks all the pixels at or above the upper bound. Finaly, it travels along the edges of each strong edge, as found from the previous step, and marks all the pixels in between the bounds as strong edges and all the pixels that are not connected to any strong edges are discarded. This process of steps ensures that only the strongest and most prominent edges in the image is displayed as well as the edges exact position.

A person wearing a hat

Description automatically generated

Figure 3: The original image (Left) and the result of the Canny edge detector (Right) (Generated using MATLAB)

As seen from Figure 3 the result of the Canny edge detector produces a more precise result compared to the Sobel operator shown in Figure 2. This is not true for most applications as each time a new image is used the lower and upper bound of the hysteresis threshold needs to be adjusted for the result to be usable.

#### Roberts

The Roberts edge detector (also known as the Roberts cross) is a convolution filter similar to the Sobel operator where it also uses two kernels and has the ability to find the direction of the edge. Unlike the Sobel operator, the Roberts does not find horizontal and vertical edges, instead it finds edges going diagonally.

The Robers cross uses two 2x2 kernels. Here we have that calculates the intensity of the edge in one diagonal and that does the same but in the opposite diagonal. After this step the magnitude is calculated using the Pythagorean theorem with the resulting image being similar to the result of a Sobel operator, except it’s a lot dimmer and it doesn’t capture as many edges as the Sobel does. It is also more liable to noise.

#### Prewitt

The Prewitt operator works exactly the same way as the Sobel operator. The only difference between the two operators is the kernels they use. The Prewitt operator use the following kernels:

As you can see the only difference between the Prewitt and the Sobel operators’ kernels are the waits from the closest neighbour across the edge. This means that the resulting image is extremely close to the Sobel result.

#### Edge Detector Conclusion

All edge detectors on display in this section are well documented and have uses in real life but for the purpose of this study only one will be implemented. The edge detector which is going to be implemented will be the Sobel operator. This is due to a few reasons, notably the Sobel operator is the most commonly used algorithm other than the Canny edge detector. Compared with the Canny edge detector, the Sobel operator is much easier to implement. With the time constraint this has been a large factor in this decision.

### Image Segmentation

“Image segmentation is the process of dividing an image into meaningful regions” (Niblack W. 1986). These sections depend on the algorithm used and desired result. For example, an image of an organ could be separated into different structures inside of it using colours. It could also be used to separate different objects from within an image. Segmentation algorithms use a lot of different techniques such as machine learning and neural networks others use typical logical processing techniques.

There are two types of image segmentation, semantic and instance. Semantic segmentation is the simplest as it only looks at individual pixels and determine if those specific pixels should be grouped together. Instance segmentation looks at the overall picture and finds the boundary of an object.

The segmentation techniques discussed will be:

* Thresholding
* K-Means
* Neural network models

#### Thresholding

Thresholding is one of the simplest and most commonly used image segmentation techniques. This technique always returns a binary image (an image containing only 2 colours), usually black and white. The simplest version of this algorithm takes a threshold parameter which is checked against every pixel in the image. If the pixel is below the threshold its value is set to back, otherwise the value of the pixel is set to white. This is the simplest form of image segmentation as it only requires one if statement to be run on every pixel value.

It is also possible to have a variable threshold. This involves this formula:

In this formula is the threshold value, is an arbitrary value (for the purposes of this we will be using 0.18 like show in Wayne Niblack’s book on image processing), is the standard deviation of the pixel and its neighbours, and is the mean value of the pixel and its neighbours.

A collage of white splatters

Description automatically generated

Figure 4: From top left going clock wise, The origrinal image of a cluster of stars, the image with a hight threshold, the image with a low threshold, the image with a varing threshold (Niblack W. 1986)

Figure 4 shows the difference between having a static threshold. The original image shows a cluster of many stars where the stars in the middle of the image merge together and appear brighter than the ones on the edge of the image. With a high threshold most of the stars around the edge of the image are culled and only the stars in the centre remain. With a low threshold all that remains from the stars in the centre is a blob meaning nothing can be comprehended from it. With the varying threshold all the stars on the edges are preserved and most of the stars in the centre of the image are preserved.

#### K-Means

K-means is an unsupervised machine learning clustering algorithm. Machine learning is the process of an application gradually improving on its accuracy over time learning. This being an unsupervised machine learning model means that human input isn’t required to teach the program for it to improve. Clustering is the act of grouping object together in a way where objects in one group have more in common with each other compared to objects in other groups. All this in combination means that k-means clustering is an algorithm that learns improves itself to eventually group objects.

K-means clustering can be used to segment an image. K-means is a semantic segmentation technique as it doesn’t understand/recognise any objects in the image but rather tries to group pixels into k number of colours. This process is also called pixel quantization which is the general process of reducing the number of colours in an image. The result of the k-means cluster is always results in a different result as the machine learning process.

K-means clustering requires 2 parameters, the number of clusters and the number of times the algorithm will iterate. When working with coloured images it is important to think of the image as a 3d graph where each axis represents red, green, and blue (this is reduced to 1d graph if grayscale and 2d if only 2 colours are taken into account) and every pixel is placed somewhere on the graph. There are 2 parts of the k-means algorithm assignment and update. First, we randomly place k number of “means” on the graph. Next, we iterate over the assignment step then the update step the amount of times specified by the parameter. In the assignment step the distance between all the pixels and the means and each pixel is assigned to the mean closes to them. In the update step each groups mean is moved to its centroid (the mean position of the group). By repeating these step the means will move towards large clusters and find pixels that are similarly coloured. At the end of the algorithm the colour of every pixel is set to the colour the group mean it is in. this results in a image with k number of colours.

A person wearing a hat

Description automatically generated

Figure 5: The original image (left), the image quantized with 5 colours using 10 iterations (generated from implementation)

As seen in figure 5, the k-means successfully selects 5 appropriate colours to quantize the image with only 10 iterations. This technique is quite useful to reduce the colours in an image to ever execute the features in an image or to compress the image by reducing the colours presents in the image.

#### Neural Network Models

### Noise Reduction

## GPU

Graphical processing Units (GPUs) are a specialized piece of hardware designed to make the rendering of 3D graphics faster and more streamline. Graphics rendering is well suited for parallel tasks meaning that the GPU is highly parallel. This means that the GPU is efficient for tasks that involve a lot of the same calculation performed multiple times on different sets of data but highly inefficient at complicated sequential logical processes that the CPU preforms.

GPUs can trace back to early 70s home video game console which contain specialised chips (often called Picture Processing Units or Video Interface Chips). These chips were designed to process video to be able to be displayed on screen. At the time home computers would not have been able to display images/videos and would only be able to display characters from the machines code page (a file containing all the characters the computer can display) (Singer G. 2023).

### GPU Architecture

### GPU Compute

Before the 2000s GPUs could not be programmed on, with the only way to use the hardware was with the graphics API that were available. This changed with the advent of programable shaders which are programs written to be run on the GPU to calculate positions of vertices’, the lighting and the colour of every pixel. The first GPUs with programable shaders were the GeForce 3 series of GPUs from Nvidia in 2001. These shaders are written in a graphics language such as High Level Shader Language (HLSL) and OpenGL Shader Language (GLSL). These languages were designed to make graphics programming easy and not to do general tasks like compute. They do not have the same types and data structures that are present in typical programming languages such as C++ and C#.

Tasks such as image processing running on the GPU have a special name; General Purpose compute on the Graphics Processing Unit (GPGPU). These sorts of programs running on the GPU would have been written using HLSL or GLSL and use their respective graphics API. This changed when Nvidia released Compute Unified Device Architecture (CUDA), which was the first API that allowed programming of the GPU using already existing programming languages. CUDA requires the GPU running the code to have CUDA cores which only Nvidia GPUs contain **CUDA cores are only contained within Nvidia GPUs, which are specifically needed to run the code, meaning all CUDA programs can only run on those specific GPUs**. This means that all CUDA programs can only run on Nvidia GPUs. Soon after the release of CUDA both Microsoft (creators of DirectX) and Khronos Group (the organisation in charge of OpenGL) made similar APIs called DirectCompute and OpenCL respectively (Ghorpade J. 2012).

For this project we had to decide on which API/Language to use. Due to only having access to Nvidia GPUs we have decided to use CUDA. Nvidia GPUS allows us to set up and use CUDA with ease making it easy to set up. Unlike DirectCompute, OpenCL and shader languages, CUDA allows us to have native support allowing the development of the algorithms to be both simpler and efficient without the need of downloading and installing third party SDKs.

Recently there has been a development from a software developer in partnership with AMD on ZLUDA. ZLUDA is a translation layer that allows AMD GPUs to run unmodified CUDA applications at near-native performance (Janik A. 2020).

## Similar Studies

# Requirements

## Product requirements

When completed the program will be able to run image processing algorithms on images using both the CPU and GPU. The application should be able to display and save the images after the algorithm is run. It should also be able to run an automated test and save performance metric to a file that can be used to turn the data into a graph.

## Functional requirements

### Interface

The program will have a command line interface as it is not designed to be used often and by members outside of the research process. The interface should allow the user to select which algorithm they would like to run and any parameters needed for that algorithm. It should also allow the user to select an image to run the algorithm on. In addition to this, the program should also allow the user to run automated benchmarks with customisable parameters to allow targeted tests.

### Algorithms

The program will have the ability to run three different image processing algorithms.

* Sobel Operator
* K-Means Clustering
* Gaussian Blur

These algorithms will be implemented using both C++ and CUDA for the CPU and GPU respectively.

### Benchmark

The program will have the ability to benchmark algorithms and save the results of the benchmark into a file. The benchmark will have parameters that determine the size of the images tested as well as the type of algorithm to be used. The file that contains all the data gathered should be able to be used by graphing software and be turned into a graph.

## Design constraints

Due to time constraints as well as it being out of scope, there will not be an implementation for reading, saving and displaying of images. A library will be used to handle this part of the program.

# Design

## Software design

The main feature of the program will be 2 classes containing image processing algorithms. One class will be called cppImageProcessing and the other will be called cudaImageProcessing. They will both contain public methods of

## Experimental design

# Implementation and testing

## Implementation

## Testing

# Evaluation and discussion of results

# Conclusion

References

Singer, G. (2023). The History of the Modern Graphics Processor. *Techspot*. Available online: <https://www.techspot.com/article/650-history-of-the-gpu/> (accessed 10/04/2024).

Niblack, W. (1986). *An introduction to digital image processing,* Englewood Cliffs, N.J: Prentice-Hall International.

Alvi, F. (2023). Computer Vision and Image Processing: Understanding the Distinction and Interconnection [Blog post]. OpenCV. Dec 13. <https://opencv.org/blog/computer-vision-and-image-processing/> (accessed 13/04/2024).

Encyclopaedia of Mathematics. (2011). Edge Detection. Available Online: <https://encyclopediaofmath.org/index.php?title=Edge_detection&oldid=17883> (accessed 13/04/2024).

Bhardwaj, S., Mittal, A., (2012). A Survey on Various Edge Detection Techniques. *Procedia Technology.* 4. 220-226.

Winnemoller, H. & Kyprianidis, J. E., & Olsen, S., C. (2012). XDoG: An eXtended difference-of-Gaussians compendium including advanced image stylization. *Computer &Graphics. 36(6). 740-753.*

Ghorpade, J. & Parande, J. & Kulkarni, M. & Bawaskar, A. (2012). GPGPU Processing in CUDA Architecture. *Advanced Computing: an International Journal*. 3(1), 105-120.

Janik, A. (2020). ZLUDA. Available online: <https://github.com/vosen/ZLUDA> (accessed 17/04/2024).