MONASH University

FIT3162 Computer Science Project 2
Semester 1, 2020
Monash University

FINAL REPORT

On GPU Acceleration of Line-integral Convolution using OpenCL

Delivered by
Team 01

Bryan Mayson
Kelvin Butler
Anh Nguyen

WORD COUNT: 5800

TABLE OF CONTENTS

I. INTRODUCTION

In present days, image processing plays an indispensable role in cartographic representations whose readability and effectiveness are greatly enhanced by said technology. This consequently leads to the necessity of image filters as they are directly related to image processing. There have been many methods proposed since the introduction of image filtering, such as Median filter, Gaussian filter, or Frequency filter, etc, each of which has its own pros and cons. During the course of two semesters, we have attempted to utilize Graphics Processing Units (GPU's) with Line Integral Convolution (LIC) to further accelerate the computation of rasters filters in map making. More details to the reason why we chose LIC algorithm will be specified later in this report. Provided with Eduard by Monash FIT and ETH Zurich, whose main function is to create shaded relief images from elevation maps, the team's ultimate objective is to optimize the computational process of Gaussian filter through the usage of parallelism and OpenCL. By integrating between C++ and Java environments, as well as utilizing kernels, i.e. small functions, we can achieve said acceleration as we develop parallelized codes on OpenCL, a standard for cross-platform and parallel programming, widely commercialised across large technological industries such as Intel, Nvidia and AMD. We believe that such improvement in filtering speed will be immensely beneficial to the field of cartography.

This report will briefly talk about the project's background, which has been introduced in detail in the proposal, revise our previous literature review, and summarize all the essential points that support our project. Moving on will be our software development methodology as we describe the theoretical usage of OpenCL and how we utilize the algorithm for an ideal partition scheme. We will also highlight some key points in our project management methodology, how we adapted and revised the initial planning, and how certain events and situations have affected our project. Lastly, we discuss the overall result of this project, including the final deliverables, how we met with initial requirements, our work's limitations, how we can improve and apply this project for future use.

II. BACKGROUND

A. Literature Review - A Revision from Project Proposal

1. Line Integral Convolution

As previously mentioned within our proposal line integral convolution is a form of image convolution which allows for the visualisation of the direction of vectors within a vector field. It is commonly used to enable its users to observe fluid motions such wind movements in the tornado or even the wave patterns of the ocean.The principle of the process is to generate streamlines and convulate the corresponding vectors onto the streamlines to obtain corresponding value of the filtered image (Qin et al, 2019). The range of the stream line would also vary depending on the extent of the desired filtered effect through the convolution process.

The convolution process of line integral convolution begins with first identifying the location of the pixel within the 2D vector field to be convoluted to form the desired filtered output. It then follows on by obtaining the streamline obtainable from the data of the vector field which involves this location.The streamline would be the data values of each pixel within the range of the specified

streamline length, with the same directional vector of the one selected. This simply implies that if the streamline length were to be stated to have a value of 10, the selected point within the 2D vector field would then explore 10 pixels upwards and downwards of pixels which contain the same vector direction. These values would then be applied to a normal distribution and proceed to be compiled. Once the values are compiled, the mean result of these values would represent the filtered output of the pixel of the image located by the specified position of the 2D vector field. The diagram to observe how the streamline could be visualized as, can be seen as below.

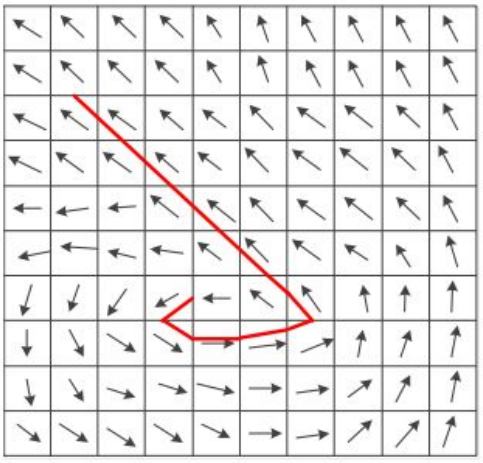


Fig 1.1 Diagram which highlights the vectors with the same direction with in vector field

2. *Elevation Maps*

Elevation maps represent a geographical section of the planet which contains values such as longitude and latitude and would act as the input file of software product.The information of these maps are stored within an ASC file otherwise known as the Action Script Communication Files which are files written in scripts to be used for a specific application or program (ReviverSoft, n.d).

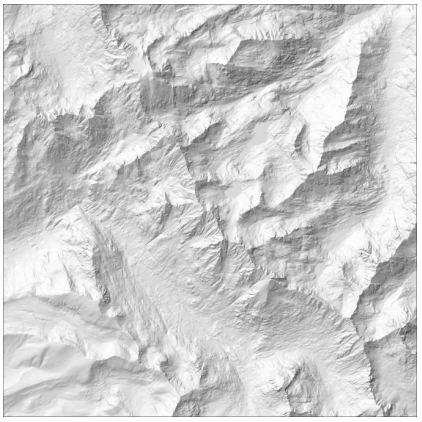


Fig 1.2 Sample Input of a 1500x1500 Elevation Map

3. *OpenCL*

OpenCL is a standard programming interface developed by the Kronos Group which is used to enable developers to accelerate their task parallel or data parallel computations easily within a heterogeneous computing environment (Stone,Gohara & Shi,2010). The interface also handles the

supported programming languages and application programming interfaces (API) executable by the computing devices within the computer. The computer devices may either be the central processing unit of the device (CPU) or even the graphical processing units (GPU) of the device.

The programming language used to allow for processes to be computed by OpenCL is similar in nature to the C programming environment.The term used to describe a function compiled within the OpenCL environment is known as a "kernel".The programs within the OpenCL environment is also designed to be compiled at run time such that it would be able to be applicable to various host devices.

Currently OpenCL is considered to be one of the most common parallelism frameworks used within the technological industry due to its applicability amongst the various pieces of the hardware found within the computers of today. The Kronos group has developed multiple open source support libraries which allows the OpenCL environment to run on well known GPUs such as Intel Nvidia and AMD Radeon.

4. Bernstein's Condition for Parallelism

Bernstein's Condition for Parallelism consists of three main requisites to act as a guideline in determining whether a linear program is able to be implemented in parallel (Bernstein, 1996). The equations of these conditions stated by Bernstein are as follows,

$$I_0 \cap O_1 = \Phi \sim anti\ dependency$$
$$I_1 \cap O_0 = \Phi \sim flow\ dependency$$
$$I_0 \cap O_1 = \Phi \sim output\ dependency$$

where I and O represent the inputs and outputs of their respective processes respectively. If the linear program is able to fulfill these conditions, then the program is deemed to be parallelable.

Through careful observation and analysis of the line integral convolution process, we were able to identify that the convolution process of each pixel within the image is not is not dependent on the result of the convolution of the previous and following pixels vice versa. As such, no forms of dependency can be observed with the overall process which allows us to conclude that applying linear integral convolution is parallelizable as it has fulfilled Bernstein's Conditions for Parallelism.

B. Summary of essentials

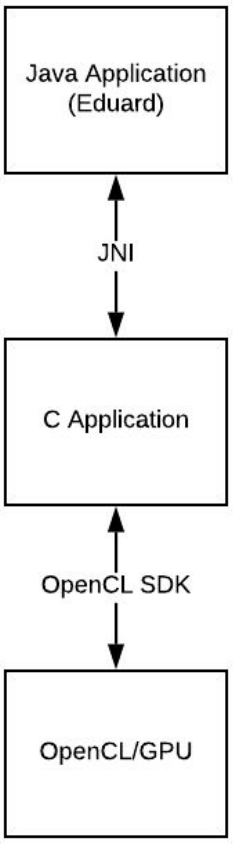
The process of line of integral convolution on an elevation map is proven to be parallelizable as its computational process fulfills Bernstein's Condition for Parallelism. With OpenCL being one of the most common parallelism frameworks used within the technological industry, the remaining contents within this report would explore how we have used the OpenCL framework to achieve the acceleration of applying a line integral convolution filter onto an elevation map, through parallelization.

III. METHODOLOGY

A. Usage of Toolchain

As stated in the project proposal, the solution was developed using OpenCL to provide the accelerated algorithm and the Java Native Interface (JNI) to integrate it with the existing codebase. OpenCL code is written in C or C++ which means that there must be an interface of some kind in order to be used by the java code.

JNI was picked as the framework to build this interface with because it is the standard way of running external or "native" libraries from java and it comes packaged with the java software development kit. The C OpenCL libraries themselves are also an interface that manages scripts known as kernel scripts to run on a parallel device in a computer such as a GPU or multi-core CPU.



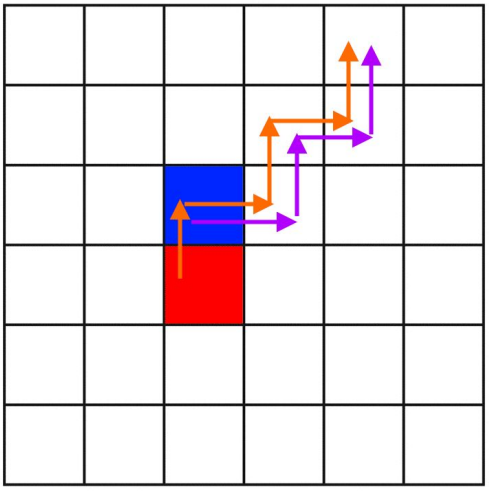
Building the toolchain up from base components, rather than using a prebuilt library for communicating with OpenCL allowed for a far greater level of control over the data as it passed though the various parts of the program.

For C code to be run with JNI special headers needed to be generated and included in the file so that the JVM could recognise the functions and handle the data. Java is able to generate this header automatically from method signatured defined as "native" within the Java source code and once generated, can be used for any methods with the same name and signature.

B. Data Handling

Within OpenCL there are two main structures used to store large amounts of data: image and buffer. Images are designed to hold multiple dimensions of data, with multiple values at each point. Buffers on the other hand, are designed to store one dimensional data and essentially function like a standard list, with slightly lower levels of performance than an image object.

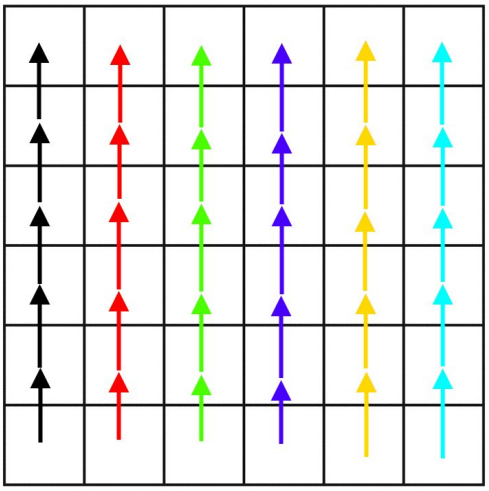
Preliminary experimentation with the technology showed that even poorly optimised code run in parallel could be many times faster than code run on a single thread. This led the team to focus on the implementation of OpenCL code to reduce the amount of memory used, as even small maps could take up nearly 100MB of memory space. Buffer became the choice of data structure because until recent versions of CopenCL image objects could not be both read from and written two, so using them in the solution would have required at least twice the amount of memory for one map to be allocated to process that map.Secondarily, JNI does not support passing two dimensional arrays which means that the data had to be flattened into a single list anyway and converting it back would have increased the overhead in processing time.



C. Partitioning Scheme

Initially there were two main candidates for the LIC algorithm. The first was to calculate the value for each point individually by following the upward and downward slope from the point and calculating the weighted average. While this version is far easier to implement it does redundantly evaluate the gradient at each point multiple times as the points downhole from a point will calculate those same gradients when they are looking uphill.

The second option was to keep an array containing points that were directly uphill/downhill from each other. While this option reduces the number of redundant gradient calculations it requires a large amount of management to make sure that every point is calculated as hex of the streamlines move through the map.



The final implementation used the first method as time constraints made it unlikely that the second could be well implemented and it was unclear if the reduction in redundant gradient calculations would offset the additional overhead of managing the stream lines.

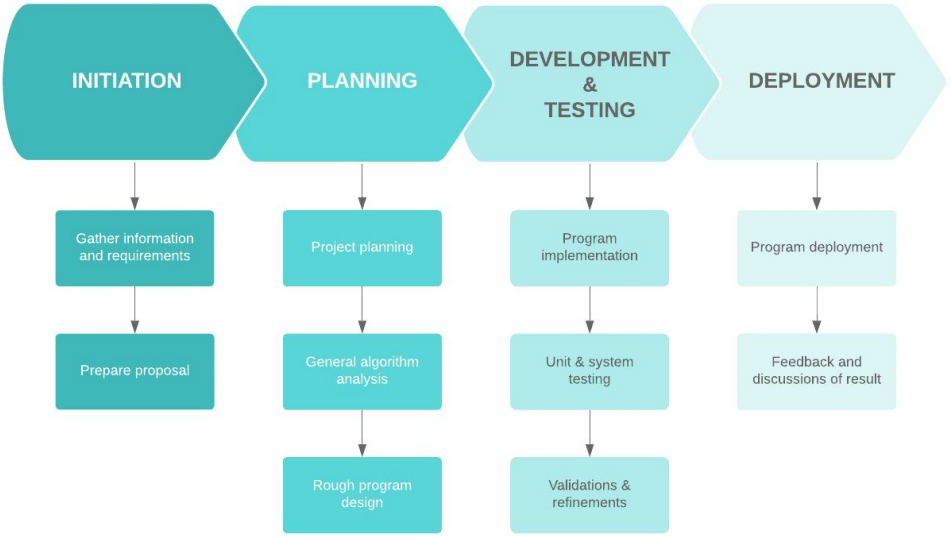
Using the first option the team was able to produce results that were similar to the output of the original serial code with some artifacts.

IV. PROJECT MANAGEMENT

As discussed above, the project's main objective is to optimize the usage of image filters for the benefit of cartography. We spent our last semester planning and doing many research to make sure we are equipped with enough knowledge before going into the implementation phase this semester. We have also prepared a Gantt Chart in the previous proposal, indicating certain phases and milestones for this year. However, there have been many events that consequently affected our project's execution, which will be justified more in the following sections.

A. Project Management Approach

The team's initial approach was solely waterfall methodology for this project. The decision was not made lightly and we were totally aware of certain risks, which we intended to avoid at all costs. With a non-changing requirement and a fixed timeline, the waterfall methodology was totally possible to execute on this project.



Nonetheless, COVID19 outbreak was beyond our expectations. As the working environment transitioned from university, i.e. face-to-face, into home, i.e remote contact, there were many issues arisen amid the initial period of the development phase. More details are discussed in our Team Management Report. The event had definitely sparked concerns between team members, and therefore resulting in several discussions as to whether we should adapt and change our original approach.

Waterfall method was still utilized because we still needed to complete one stage before moving on to another. For instance, we would not be able to develop the software without thoroughly researching and understanding what line integral convolution is exactly, how the integration between C++ and Java works, as well as how kernels communicate in OpenCL. The team also wanted to simplify the actions in each stage because all members will focus on one aspect at the time, thereby expecting to complete a stage much faster. Moreover, as this project is considered small-scale, with an objective that will not be changed under any circumstances, it was logical to adopt the waterfall methodology.

However, as the team revised and readjusted, we decided to incorporate the agile methodology, or Kanban method to be specific, in our approach as well to ensure minimum delay to our project's schedule. It was mostly applied during our development and testing phase. By adopting the Kanban method, the team knew which tasks were to be prioritized, whether a certain task was being too committed, or one person was committing to too many tasks at once during this phase. As we suffered a major delay in our schedule, agile methodology was really helpful and considerably sped up our progress to keep up with initial planning while maintaining our product quality. Furthermore, since we adopted a hybrid approach, our team's task allocation was incredibly efficient and kept redundancy or conflicts to bare minimum, which boosted the project's performance, in spite of many unanticipated difficulties.

B. Project Resources, Execution, and Planning

In general, the team still executed according to the original schedule of the project phase, as proposed in the project proposal. However, the team knew, in advance, that the proposed schedule was subject to change because for example, each member would have different unit deadlines to adhere to. It was inevitable that our planned timeline would be slightly modified to correspond to members' availability. And as mentioned in the previous section, we did alter our original approach to accommodate this initial planning. The development phase, in particular, was delayed by a few weeks due to personnel issues. Besides, unfamiliarity with C++ language affected the execution, and prolonged the development phase. There were also some issues projected along the implementation regarding machine setup and preparation. These factors all contributed to some minor delays in our timeline, yet we still managed to keep as close to the original planning as possible. Nevertheless, our plan implementation-wise did not change much. Despite some time constraint, the team successfully implemented the partition scheme that we introduced and explained in our project proposal.

Regarding the project technical implementation's resources, the algorithms used for image processing are fairly "costly," and therefore require specific hardware that could handle fast processing. They must also integrate with CPUs and GPUs, and have graphics cards that support OpenCL SDK, since it is the key software of this whole project. Java JDK must also be installed. Our product was implemented on a quad-core, i5 4690k computer system with Radeon R7 370 graphics card. For local development, a Java IDE and compiler such as Eclipse and IntelliJ was used. For online collaboration, we used Monash FIT Gitlab and Visual Studio to push, pull and commit changes as necessary. This is also our version control method of this project. On another hand, regarding resources for the project management process, we utilized Trello - our Kanban swimlanes, as we adopted agile methodology. Our main platform for remote meetings was Discord server. We also used LucidChart, Google Docs and Sheets to organize our progress and reports.

The team conducted meetings once a week or every two week depending on each meeting's objectives and members' availability. During more critical periods, such as Interim Presentation, we carried out extra meetings to prepare for the checkpoints. This allowed for last-minute changes and modifications if necessary. We also had our own code checkpoints to ensure implementation was going according to plan. We also occasionally went to consultation to report on our current progress to the unit supervisors. At the end of the development phase, we kept sending outputs to our Discord server to constantly check the program's efficiency after every major change. In spite of many hardships, the team was always open to discussion and ready to take action whenever any member had an issue. As a matter of fact, we did not have a trouble-free execution phase, but it was as smooth and harmonious as it could be, considering the problems outlined in the Team Management Report, thanks to great communication among the team. Concerns were quickly resolved, and members always consulted each other to ensure unanimity in the final result.

C. Risk Management

We have created another risk register to closely manage the risks of our project this semester. This register was updated every two weeks to strictly monitor the risks we were facing in real time. We organized the final risk register based on each risk's level of impact on the project. Similarly to last semester's register, we also assessed each risk's probability, impact, consequences, response strategy and implementation plan, and their owners. There were, however, many unexpected risks

occurred in the course of this semester that we did not consider during our planning phase, which required immediate actions from the team. The most impactful risks stemmed from both personnel and technological issues, but overall, poor time management was our most critical risk.

The first risk that we anticipated during our planning was members' lack of motivation. Going forward in the beginning of this semester, the team was supposed to have a strategy to respond to this risk, yet unable to take action in time. In fact, we did not account for a pandemic outbreak to happen during this period, which entailed social distancing, quarantine, and the transition of on-campus to off-campus. Despite means of communication such as Facebook, Discord, etc. the first few weeks were extremely difficult for the team to actually focus on the project. In other words, members quickly lost motivation and could not keep track of the project timeline. The issue was later on addressed, because the team finally familiarized ourselves with the whole study-from-home situation. According to our risk assessment in the previous semester, this is a medium probability, high impact risk, which can be damaging to the whole project. As we faced it this semester, it was indeed detrimental to our project, because we immediately suffered a two-week loss of time. Since we could not avoid the risk, the team accepted the impact and dealt with its consequences instead. The first response was to gather the team, and instantly modify our original schedule accordingly. We also altered our initial management approach as aforementioned, and encouraged each other to keep up the spirit and reinforced our mentality. Another related risk was inconsistency in members' personal schedules. Because we had foreseen risk last semester, mitigation was properly applied. We did not have much trouble conducting important meetings for checkpoints, or delegating tasks to members, also thanks to members' high sense of responsibility and self-organization. Due to our prompt response to these risks, the project was able to progress as intended.

Another critical risk we faced, which was not included in last year's register, was that the technical knowledge for the actual program implementation was more overwhelming than we all predicted. Not only OpenCL, but we also had to look into Java Native Interface (JNI), and revised our knowledge of Java. It took us weeks to finally grasp the essential points for the project, and apply them to our implementation scheme. Our development phase consequently suffered from delays and a bottleneck of time approximately in the crucial middle period of this semester. The team once again acknowledged the risk, readjusted some middle checkpoints, but still ensured we would have time to deliver a good program. Following this risk, we also encountered some other minor technical issues. The environment for our program requires a complex number of steps and preparations for it to run as designed, so it took us a while to figure them out. We were also unsure of our outputs at around week 9, and could not work out where it went wrong in the implementation.

One more technical risk that left an immense impact on our project, was that we had to reduce our testing phase as we tried to handle all other risks. With the testing phase reduced, there could be bugs that we did not account for, and therefore directly affecting our project's performance. The most effective action we could take at that point was that all members had to adjust our own personal schedule, so we could make up for the lost time. Another mitigated risk was that the computer on which the majority of our project was coded is not compatible with the latest version of OpenCL. We immediately responded to the risk by lengthy discussion, and decided that it would not affect our final deliverable.

Due to unforeseen events and situations, our team has run into many risks that were not planned during our planning phase. Although we tried to promptly respond to each of them in the

most efficient way in our capability, with many unanticipated risks, our project's final outcome could take a plunge in quality.

D. Limitations

Most of our team's limitations derived from personnel and organization. Even though they were not major issues, they did partly affect our overall progress.

First of all, due to unequal levels of skills, it took more time for a member to comprehend the project's implementation than another, which subsequently affected our timeline. Another primary limitation was that because of COVID19 pandemic outbreak, quarantine and social distancing limited ourselves to remote online meetings only. They were not as efficient as face-to-face meetings since it could be hard to focus while one was relaxing at home. Being at home made it harder to organize personal schedules and adhere to deadlines as well. We had unfortunately limited our own potentials to implement a better program due to disorganization while being home. Another issue was that we were not completely connected to the project itself in the beginning, which was also partly the reason why we could not find motivation going into the implementation phase. We had to spend time doing more research to educate ourselves on image processing and filtering to find more inspiration and incentive to carry out this project.

V. OUTCOMES

A. Final results and Product delivery

Over the course of this semester, we have successfully implemented an accelerated version of the Eduard program, which uses line integral convolution algorithm to apply the Gaussian filters, by incorporating parallelism. In the correct environment, the implementation can speed up the filtering 10 times faster. This program was built based on the initial Eduard Java program. The team has added new classes and methods in Java environment, which through JNI headers, communicates with our OpenCL implementation of kernels codes that run the calculation tasks in parallel.

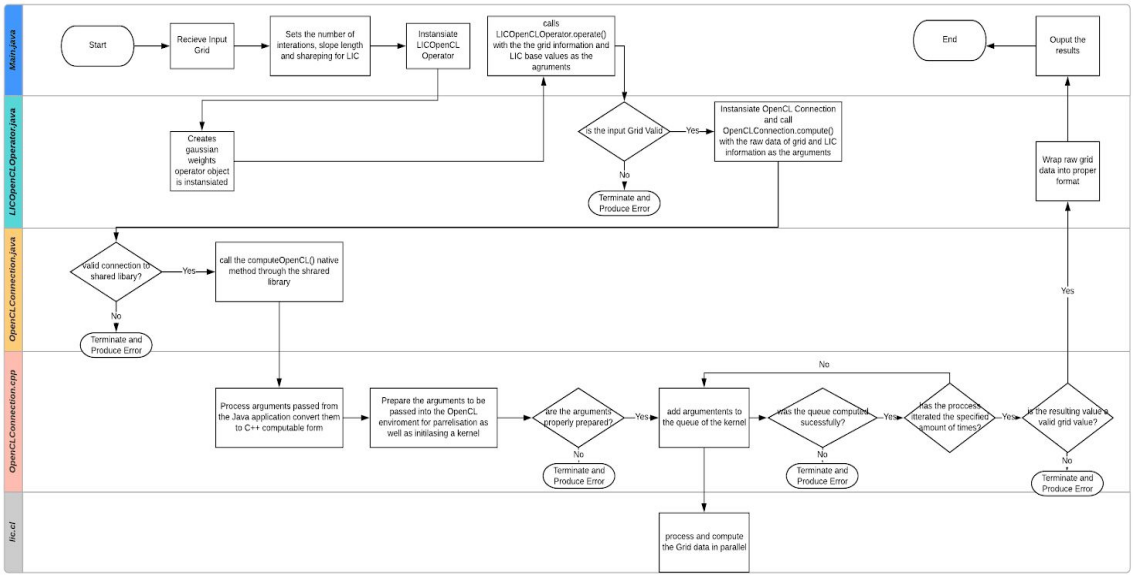


Fig 5.1 Diagram which represents the processed of the software product

B. How requirements are met

Looking back at our project characteristics and requirements stated within our previous design proposal, we are able to observe that most of our in scope requirements are all met by the end of this semester. This can be observed from the fact that our software product is accessible from a Java Virtual Machine and contains the ability to speed up the computational process of raster filters through the GPU while utilizing the line integral convolution filter.

1. Accessible from a Java Virtual Machine

A Java Virtual Machine is an emulation of a computer system which allows a device to compile and process Java applications. Previously the provided linear implementation of the software product was implemented entirely in the Java language and designed to be accessible from a Java Virtual Machine. To ensure that this requirement remained fulfilled within our software product. We decided that the software product would be a modification of the provided linear application with additional features which enables the application to accelerate its computational operations through the GPU.To further ensure this, we have determined that it is essential to that the beginning and end of the software product would alway involve the use of the Java Virtual Machine.

The fulfillment of the requirement can be further observed from the fact that to launch/run the software product, a Java IDE and compiler such as Eclipse or IntelliJ is required.

2. Speeds up the Computational Process of Raster Filters through the GPU

As discussed previously within this document, OpenCL is one of the most common parallelism frameworks used within the technological industry which involves the use of the device's GPU. As such, implementing the computational process of line integral convolution with the aid of OpenCL will surely yield desirable results as found within many existing research involving it. When compared to the provided linear implementation of the program we are able to observe that our

software product , implemented with the aid of OpenCL, was able to accelerate the computational process of the filter by significantly when handling input of large sizes.

The table and graph below shows the difference in computational speed of the linear and OpenCL implementation of the software when handling input elevation maps of size 9232 x 5973.

Iterations	Serial (ms)	OpenCL (ms)	Speed up factor
2	18951	1323	14.32
4	36620	1591	23.01
6	56601	1856	30.49
8	75991	2209	34.40
10	94348	2534	37.23

Fig 5.2 Table for time taken to produce output for various iterations

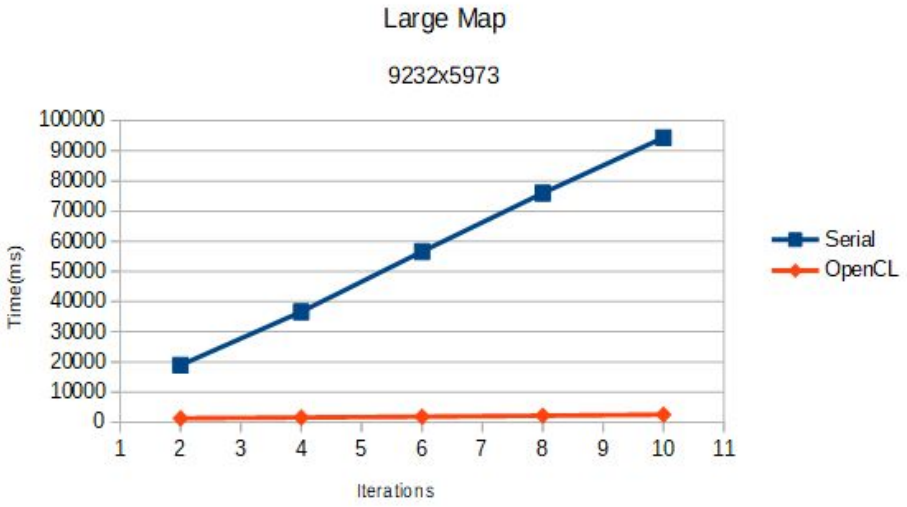


Fig 5.3 Graph for time taken to produce output for various iterations

From the data recorded above, we are able to observe the drastic difference in the computational time of the linear implementation when compared to that of the OpenCL implementation. As such from these results, the requirement of speeding up the raster filter application through the use of the GPU had been fulfilled.

3. *Utilization of Line Integral Convolution*

Within our software product, we ensured that our software product correctly utilizes the line integral convolution filter. Referring back to the things discussed about OpenCL, a “kernel” acts as a function which is compiled within an OpenCL environment. The kernel instructions which would be parallelised within the entire arguments, and as such the kernel used within the software product

specifically follows the steps of calculations for the computation of the line integral convolution of an elevation map.

By taking a closer look at the kernel file used within the software product, it can be observed that each instruction stated within the kernel file follows similar calculation steps of the computational process of line integral convolution.This includes the calculation of obtain the mean value of the pixels with the same vector direction under a normal distribution.As such the software can be observed to be utilising the line integral convolution filter. The resulting output of the software product can also be seen to produce a similar output to the linearly implemented software,which would be further discussed within our results discussion.

C. Justifications of decisions

This section of the report would elaborate on the design choice made to the software product and the reasoning behind its choices

1. Inputs are converted to Primitive Data Types

After careful observation and research on the provided linear program, it was observed that the classes within the program are very dependent on each other .Most of the custom data types/classes used within the linear implementation of the algorithm required the methods of other custom data/types or classes.This existing dependency would mean that for a function within a Java class to run it would require that the other class has to exist. As a result several issues may arise if we wish to pass in the custom data types of the input arguments from the Java application to the C++ application.

As previously mentioned within the literature review, the Java native interface is the tool used to integrate Java applications with the C++ application and it is possible for the JNI to pass custom Java data types. However, it would be difficult within the C++ application to process what the custom data types mean. To avoid this, we have decided that our software product would pass down the input information as well as other information required to compute line integral convolution as primitive data types before handing it to be processed by the C++ application.

2. Following the code structure of the linear implementation

Within the design of the software product's code structure, it can be observed that the newly implemented Java classes follow the same code structure of those of the provided linear implemented classes. The grid input would always be passed into an operator which would then be passed to the class/object which would perform its computation.This design choice was made such that during the development of the software product, we would be able to ensure that each of the new classes would contain similar functionality as to when they were to be coded linearly. Following the code structure of the provided linear program would also benefit the original developer in being able to understand the software product much more easily.

3. Implementation of the filter is not identical to the original

The computation of the line integral convolution filter was also decided to be implemented differently than how it was originally implemented within the provided linear program. This decision

was made due to the fact that identically implementing the process with OpenCL was proven difficult due to several issues.

Firstly, the linear implementation of the line integral convolution filter consisted of numerous dependencies between the current Java class it was located in and several external Java classes. The OpenCL environment is only capable of computing primitive data types in parallel throughout the entire software; as such it would be impossible for computational steps which required custom data types to be parallelised within this environment.

Another issue that arose was that it was difficult to precisely implement the same steps performed within the linear algorithm without completely understanding the nature of how it was implemented linearly. The overwhelming amount of dependencies within the linearly implemented computation function was confusing to follow and as such caused the lack of understanding.If a guide or process flowchart was provided, it would have been possible to follow these steps exactly within the OpenCL implementation of our software.

D. Result discussion

Through the software which we have developed, we were able to produce a resulting filtered elevation when a line integral convolution filter has been applied. However, if we were to compare the results of the provided linear implementation of the software with the current result, a slight difference in the results can be observed.

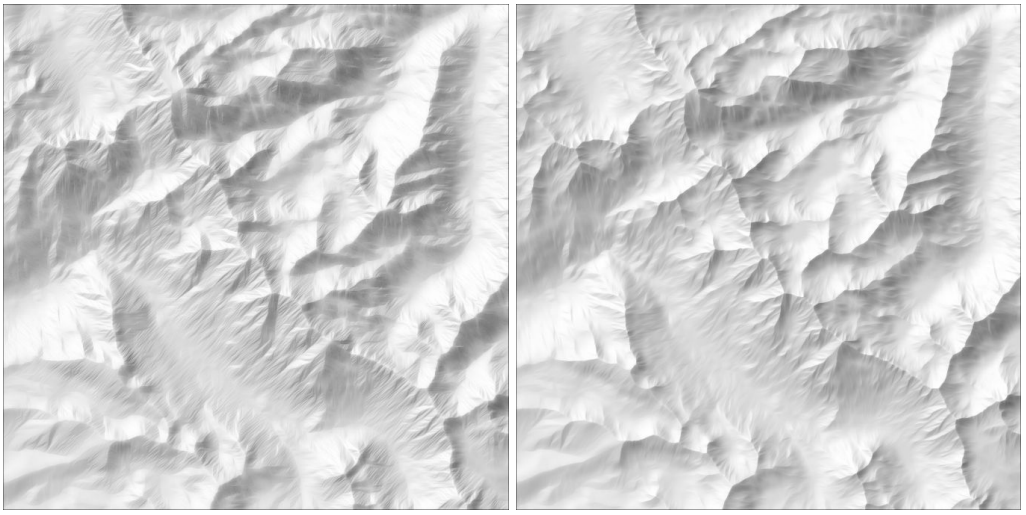


Fig 5.4 Sample Outputs of the OpenCL implementation vs Provided Linear Implementation

A possible reasoning for the slight difference between the outputs shown by the results is due to the sensitive nature of the line integral convolution computation. Line-integral convolution can be seen as a very sensitive algorithm even with a single iteration, the resulting output will vary greatly compared to its input. As such, it might be possible to due the different implementation procedure of the process, the resulting outputs would display a slight difference in the resulting output.

E. Limitations of outcomes

Due to the sensitive nature of the convolution filter process as well as the different ways of implementing it, the resulting output was not identical to that of the output of the linearly

implemented software. However, the resulting output of the OpenCL implementation of the software yields similar results to when it was computed linearly.

When looking into the software used to achieve the acceleration of the raster filter onto elevation maps. It is observed that the current software was made specifically to compute the convolution of an elevation map and is not able to perform any other filtering methods. The modifications which can only be made to the current software would only be modifying the number of iterations the filter would be applied as well as how the method would be applied. If there was a desire to increase the amount of filtering options several changes would have to be made to the entire software which would later on be elaborated in the technical user guide.

The software product is also heavily reliant on the recompilation of the shared library to ensure that the software would function desirely in the current device. This is due to the fact that the shared library establishes the connection between the Java Application and the C++ Application through the use of the Java Native Interface of the Java SDK of the device used to compile the shared library. As such the shared library should always be recompiled for various devices as the Java SDK is located differently for various systems.

Furthermore, there are also issues which arise when it comes to running the software product in certain graphic drivers. The issue involves the OpenCL environment being unable to properly extract the computational information from the kernel of the graphic driver.

F. Improvements and Possible future works

Overall the software product was able to accelerate the computation of raster filters with line integral convolution to a certain extent.However there are several improvements which can be made to software in the near future.

Possible improvements to the software product includes improvements such as improving the reliability of the software product such that it can be applicable for devices of any graphical drive.Another improvement may be in a obtaining a method to allow make it unnecessary for users to recompile the shared library whenever they desire to run it.

For future works, it would be possible that we would seek to improve the reliability of and accessibility of the software product such that it would be accessible throughout all platforms while maintaining the efficiency of the software product.

VI. CONCLUSION

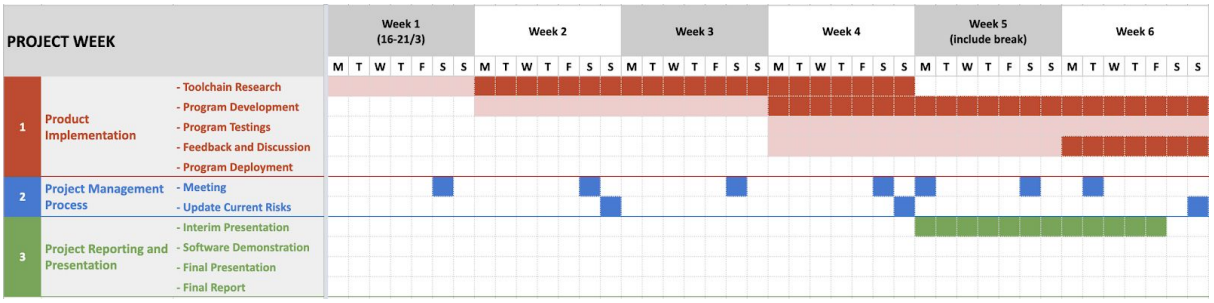
In a nutshell, the project has, to some extent, achieved the initial goal of accelerating the computation of raster filters, with line integral convolution algorithms, using OpenCL technology, in spite of some implementation and management limitations that have been carefully assessed above. During the course of this project from initiation to delivery, we have been able to learn new technologies and knowledge, as well as adapting ourselves to handle difficulties amid development. On a whole, the project is an intriguing one, as it is related to the field of cartography, which we did not have much knowledge of. Based on initial circumstances, the team unanimously agreed that we have successfully overcome the challenges posed and delivered a complete product.

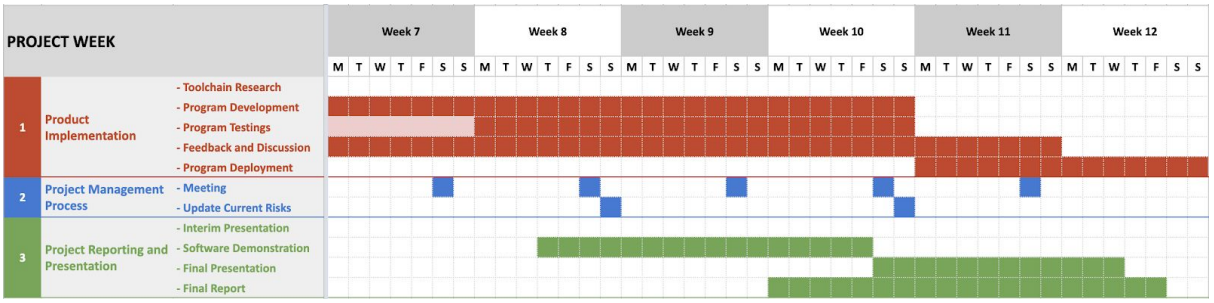
VII. REFERENCES

1. Antao, S., & Sousa, L. (2010). Exploiting SIMD extensions for linear image processing with OpenCL. 2010 IEEE International Conference on Computer Design. doi: 10.1109/iccd.2010.5647672
2. Bernstein AJ (1966) Analysis of programs for parallel processing. IEEE Trans Electron Comput EC-15:757–762
3. Cypher, R., & Sanz, J. L. C. (1994). The Simd model of parallel computation. New York: Springer-Verlag.
4. Hayes Munson, K. A. (2012). How do you know the status of your project?: Project monitoring and controlling. Paper presented at PMI® Global Congress 2012—North America, Vancouver, British Columbia, Canada. Newtown Square, PA: Project Management Institute.
5. Ludwig, Jamie (n.d.). Image Convolution. Portland State University
6. Project Management Institute (2017). A guide to the project management body of knowledge (PMBOK® Guide)— Sixth edition. Newtown Square, PA: Author.
7. Rajkumar, S. (2010). Art of communication in project management. Paper presented at PMI® Research Conference: Defining the Future of Project Management, Washington, DC. Newtown Square, PA: Project Management Institute.
8. ReviverSoft. (n.d.). File Extension Search. Retrieved from https://www.reviversoft.com/file-extensions/asc.
9. Stone, J & Gohara, D & Shi, G. (2010). OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. Computing in science & engineering. 12. 66-72. 10.1109/MCSE.2010.69.
10. Wallmuller E., (2002), Risk management for IT and software projects, Business continuity: IT Risk management for international corporations
11. Oracle. (n.d). Java Native Interface Specification. https://docs.oracle.com/en/java/javase/11/docs/specs/jni/intro.html

VIII. APPENDIX

A. Gantt Charts





B. Risk Register

ID	Rank	Category	Description	Consequence	Probability	Impact	Risk Response Strategy	Response Implementation Plan	Risk Owner
1	*1*	Personnel	Member's personal schedule leads to delay in project schedule.	Project phases are consequentially delayed, and might cause several complications.	High	High	Mitigate	Communicate in advanced to other team members so schedule can be revised in a proper manner.	Project Manager - Bryan
2	1	Personnel	Member's productivity drops due to lack of motivation.	Project is delayed, or final product might not be up to par.	Medium	High	Accept	Motivate among team members occasionally check in on the progress weekly.	Project Manager - Bryan
3	2	Technical	Technical information and knowledge is too complicated.	Delay in schedule to properly equip all members with sufficient knowledge before starting implementation.	Medium	High	Accept	Kick-start on implementation as soon as one member can. Others standby for assistance.	Developer - Kelvin
4	4	Technical	Insufficient time for testing.	Final product is not fully tested before deployment so there is higher risk of bugging.	Low	Medium	Accept	Members adjust personal schedule to make up for lost time.	Quality Assurance - Anh
5	3	Technical	Setup error during compilation.	Take up time, make testing more complicated.	Medium	Low	Mitigate	Research and check documents thoroughly about compilation environment. Also document those errors for later use.	Developer - Kelvin

C. Member's Contribution

Section	Members (% if applicable)
Introduction	Anh
Literature Review	Bryan
Methodology	Kelvin
Project Management	Anh
Outcomes	Bryan
Conclusion	Anh

Other Reports

Reports	Members (% if applicable)
Code Report	Bryan (90%) Kelvin (10%)
Test Report	Kelvin
Team Management Report	Anh