

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2023

Real-Time On-Site OpenGL-Based Object Speed Measuring Using Constant Sequential Image

Aiming Deng
University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>



Part of the [Computer Engineering Commons](#), and the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Deng, Aiming, "Real-Time On-Site OpenGL-Based Object Speed Measuring Using Constant Sequential Image" (2023). *Electronic Theses and Dissertations*. 9093.
<https://scholar.uwindsor.ca/etd/9093>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

**Real-time On-site OpenGL-based Object Speed Measuring Using Constant
Sequential Image**

by

Aiming Deng

A Thesis

Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

© 2023 Aiming Deng

**Real-time On-site OpenGL-based Object Speed Measuring Using Constant
Sequential Image**

by

Aiming Deng

APPROVED BY:

B. Boufama
School of Computer Science

M. Mirhassani
Departmental of Electrical and Computer Engineering

R. Muscedere, Advisor
Departmental of Electrical and Computer Engineering

May 12, 2023

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

This thesis presents a method that can detect moving objects and measure their speed of movement, using a constant rate series of sequential images, such as video recordings. It uses the industry standard non-vendor specific OpenGL ES so can be implemented on any platform with OpenGL ES support. It can run on low-end embedded system as it uses simple and basic foundations based on a few assumptions to lowering the overall implementation complexity in OpenGL ES. It also does not require any special peripheral devices, so existing infrastructure can be used with minimal modification, which will further lower the cost of this system.

The sequential images are streamed from an IO device via the CPU into the GPU where a custom shader is used to detect changing pixels between frames to find potential moving objects. The GPU shader continues by measuring the pixel displacement of each object, and then maps this into a practical distance. These results are then sent back to the CPU for future processing.

The algorithm was tested on two real world traffic videos (720p video at 10 FPS) and it successfully extracted the speed data of road vehicles in view on a low-end embedded system (Raspberry Pi 4).

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor Dr. Roberto Muscedere for all of his advice that helped me during the developing of this algorithm and the editing this thesis. He provided numerous suggestions on development, Linux programming, and testing. He was very kind and patient while editing this thesis and provided a great number of valuable suggestions and comments.

I also would like to show my appreciation to my friends. They also gave me valuable suggestions during the development of this algorithm and helped me to collect the experimental data.

Finally, I would like to thank the whole Internet community and those people who developed the open and free software and APIs used in this code. Without the open and free software and APIs, it would have been far more difficult for me to develop this algorithm and implement it. The online community was also extremely helpful when I encounter software related problems.

TABLE OF CONTENTS

DECLARATION OF ORIGINALITY	iii
ABSTRACT.....	iv
ACKNOWLEDGEMENTS	v
LIST OF TABLES	x
LIST OF FIGURES	xi
CHAPTER 1 Introduction.....	1
1.1 Objective and Motivation.....	1
1.2 Problem Definition, Methodology and Challenges.....	1
1.3 Contributions	2
1.4 Outline of Thesis	2
CHAPTER 2 Literature Review	4
2.1 V2I.....	4
2.2 Object Detection and Tracking	5
2.3 Stereo Vision	7
2.4 Using Motion Detection at a Specific Point.....	11
2.5 Using Model Size Comparison	14
2.6 Tracking License Plate	15
2.7 Summary	16
CHAPTER 3 Proposed Algorithm.....	17
3.1 Terminology	17
3.2 Algorithm Overview	18
3.3 Region of Interest (ROI)	20
3.4 Moving Object Detection	21
3.4.1 Moving Blob Detection	21
3.4.2 Grouping Blobs to Detect an Object.....	22
3.5 Object Tracking.....	24
3.5.1 Closest Object Tracking and Sampling Rate	25

3.5.2 Merging and Separating Objects	26
3.5.3 Rectification Tracking	27
3.5.4 Object Edge Detection.....	31
3.5.5 Mitigating Tracking Overshoot	32
3.6 Speed Measurement	33
3.7 Summary	36
CHAPTER 4 Implementation Platform Comparison and Constraints	37
4.1 Comparison of Different Implementation Platforms	37
4.1.1 Using a CPU	37
4.1.2 Using Custom Hardware with PLDs (Programmable Logic Devices)	38
4.1.3 Using a GPU	39
4.1.4 Choice of Platform.....	42
4.2 Comparison of Different GPU APIs	42
4.2.1 Vendor Specific GPU APIs	42
4.2.2 OpenCL	43
4.2.3 OpenGL	43
4.2.4 OpenGL ES.....	44
4.2.5 Vulkan	44
4.2.6 Choice of API	44
4.3 Optimizations for OpenGL ES GPUs	44
4.3.1 Hazards	45
4.3.2 Data storage size.....	47
4.3.3 Asynchronized Operation	47
4.3.4 Task Management.....	48
4.3.5 Double Buffering.....	49
4.4 Implementation Details	50
4.4.1 Task Scheduling	50
4.4.2 Individual Stages	52
4.4.3 Pre-defined Roadmap	57
CHAPTER 5 Experimental Results	60

5.1 Intermediate Results	61
5.1.1 Moving Blob Detection	61
5.1.2 Group Blobs to Objects	62
5.1.3 Center Lower Edge Detection	62
5.1.4 Object Tracking and Speed Measurement.....	64
5.1.5 Grouping Results for All the Objects	65
5.1.6 Display on Screen	66
5.2 Accuracy.....	67
5.3 Results	71
5.4 Benchmarking Performance	81
5.4.1 Scene Information.....	82
5.4.2 Overall Performance Benchmark	83
5.4.3 Stage Performance Benchmark	83
CHAPTER 6 Conclusion and Future Work.....	89
6.1 Conclusions	89
6.2 Future Work	89
6.2.1 Automatically Roadmap Generation	89
6.2.2 Spherical Roadmap with Gyroscope	89
6.2.3 Secondary System on the Next Stage	90
6.2.4 Using Vulkan API	90
6.3 Other Considerations	90
REFERENCE.....	92
APPENDICES	95
Appendix A Pre-process.....	95
Appendix A.1 Video decoder	95
Appendix A.2 Roadmap Generator – Key Point Based	96
Appendix A.3 Roadmap Generator – Trigonometry Based	108
Appendix B Main program	119
Appendix B.1.1 common.h.....	119
Appendix B.1.2 common.c	121

Appendix B.1.3a gl.h	122
Appendix B.1.3b gl.c	134
Appendix B.1.4a roadmap.h	152
Appendix B.1.4b roadmap.c	153
Appendix B.1.5a speedometer.h	156
Appendix B.1.5b speedometer.c	157
Appendix B.1.6a th_reader.h	159
Appendix B.1.6b th_reader.c	160
Appendix B.1.7a th_output.h	164
Appendix B.1.7b th_output.c	165
Appendix B.1.8 main.c	166
Appendix C Shader programs	186
Appendix C.1 roadmapCheck.glsl	186
Appendix C.2 project.glsl	187
Appendix C.3 blurFilter.glsl	188
Appendix C.4 edgeFilter.glsl	189
Appendix C.5 changingSensor.fs.glsl	190
Appendix C.6 objectFix.fs.glsl	190
Appendix C.7 edgeRefine.fs.glsl	192
Appendix C.8 measure.fs.glsl	193
Appendix C.9 sample.fs.glsl	196
Appendix C.10 display.glsl	198
Appendix C.11 final.glsl	198
Appendix D Experimental Results	199
Appendix D.1 Performance Benchmarks – Process Time	199
Appendix D.2 Performance Benchmarks – Stage Time	199
VITA AUCTORIS	201

LIST OF TABLES

Table 1 Result by Frame	79
Table 2 Result - Algorithm Speed by Frame vs Satellite Speed	81
Table 3 Performance Benchmark – Scene Information	82
Table 4 Performance Benchmark – Process Time	83
Table 5 Performance Benchmark - Stage Time	85

LIST OF FIGURES

Figure 1 V2X Overview [3].....	4
Figure 2 Stereo Vision Concept [7]	7
Figure 3 Stereo Vision Mismatch	8
Figure 4 Rolling Shutter [8].....	9
Figure 5 Camera Scan Delay	9
Figure 6 Inductive loop [9, p. 44]	11
Figure 7 Motion Detection at Specific Point [10].....	12
Figure 8 Intrusion Line Method Issue.....	13
Figure 9 3D Model Based Speed Measurement [11].....	14
Figure 10 Tracking License Plate [12].....	15
Figure 11 Focus Region	20
Figure 12 Moving Object Detection	21
Figure 13 Moving Object Detection Grouping	23
Figure 14 Object Tracking Basic Example	24
Figure 15 Object Tracking Minimal Rate.....	25
Figure 16 Object Tracking on Road Lanes	28
Figure 17 Blob Tracking Rectified View.....	30
Figure 18 Object Edge Projection Line	31
Figure 19 Object Tracking Ideal Case Without Tracking Overshoot.....	32
Figure 20 Object Tracking Non-Ideal Case with Tracking Overshoot.....	32
Figure 21 Blob Tracking Overshoot Fix.....	33
Figure 22 Screen to Road Domain Trigonometry Distance.....	34
Figure 23 Screen to Road Domain Trigonometry Angle.....	34
Figure 24 Roadmap Rendering	35
Figure 25 VideoCore 4 Architecture Overview [13, p. 13]	40
Figure 26 VideoCore 4 QPU [13, p. 17]	41
Figure 27 GPU Control Hazard with Branch.....	45
Figure 28 GPU Control Hazard without Branch.....	46
Figure 29 OpenGL Asynchronized Operation.....	48
Figure 30 Unoptimized Program CPU-GPU Memory Transfer	48
Figure 31 Optimized Program CPU-GPU Memory Data	49
Figure 32 Unoptimized Algorithm Workflow	50
Figure 33 Optimized Algorithm Workflow	51
Figure 34 Task Scheduling Overview.....	52
Figure 35 Memory Management Reader Buffer 1.....	52
Figure 36 Memory Management Reader Buffer 2.....	53
Figure 37 Conventional Upload [23, p. 394]	53
Figure 38 PBO Upload [23, p. 394].....	54
Figure 39 Unoptimized GPU Data Upload.....	54

Figure 40 Render Buffer	55
Figure 41 Processing Data Flow	55
Figure 42 Window System Framebuffer.....	56
Figure 43 Algorithm Pipeline	57
Figure 44 Trigonometry Error on Uneven surface	57
Figure 45 Experimental Result - Raw Stage – QEW.....	60
Figure 46 Experimental Result - Raw Stage – Hwy3	60
Figure 47 Experimental Result - Moving Detection Stage - QEW	61
Figure 48 Experimental Result - Moving Detection Stage – Hwy3	61
Figure 49 Experimental Result - Moving Detection Fixed Stage - QEW	62
Figure 50 Experimental Result - moving Detection Fixed Stage – Hwy3	62
Figure 51 Experimental Result - Edge Detection – QEW	63
Figure 52 Experimental Result - Edge Detection – Hwy3	63
Figure 53 Experimental Result - Measurement Stage - QEW	64
Figure 54 Experimental Result - Measurement Stage – Hwy3.....	65
Figure 55 Experimental Result - Display Stage – QEW.....	66
Figure 56 Experimental Result - Display Stage – Hwy3.....	67
Figure 57 Accuracy Benchmark – Between frame 330 and 329	68
Figure 58 Accuracy Benchmark – Between frame 329 and 328	68
Figure 59 Accuracy Benchmark – Between frame 330 and 327	69
Figure 60 Accuracy Benchmark – Between frame 327 and 324	69
Figure 61 Accuracy Benchmark - Speed Measurement between frame 330 and 327	70
Figure 62 Result.....	71
Figure 63 Result on Satellite Photo	80
Figure 64 Performance Benchmark – Hwy3 Stage Time vs Objects	87
Figure 65 Performance Benchmark — QEW Stage Time vs Objects.....	88

CHAPTER 1

Introduction

1.1 Objective and Motivation

Excessive speeding is one of the major causes of traffic incidents globally. When a vehicle is travelling at higher speed, it will take a longer distance for the vehicle to stop; upon collision, the involved vehicle suffers a larger impact than travelling at lower speeds. Statistics shows, speeding-related traffic accidents accounts for 25.3% of fatal collisions in Canada in 2019 [1]. In America, there were 11258 speeding-related traffic accidents in 2020, which accounts for 29% of total traffic fatalities of that year [2].

One of the common strategies to keep roads safe is to ticket any driver who exceeds the posted speed limit. To do so, the speed of the vehicle must be measured. This can be performed manually, for example, by a police officer using a hand-held RADAR speed gun. Another method is to use an air patroller which can measure the time of a specific vehicle travelling across two reference points on the road to calculate its speed. However, these methods are slow and generally inefficient. It can even be dangerous for the police officers to stand on or near the road to measure the speed of vehicles. Using automated devices instead of manual power can reduce manpower cost, hence, increasing the efficiency and lower the overall cost.

A common issue all RADAR speed guns have is that they are active devices, meaning they need to send radio signal to the measured object to measure the speed of that object. An unlawful driver can install a RADAR receiver to avoid being detected by RADAR by only reducing speed when a signal is detected.

To reduce speeding more effectively, a passive speed measuring system is required. This device should not emit any signal that can be detected; hence, drivers will not be able to determine when their speed is being measured. Since excessive speeding would lead to being ticketed and possible license suspended and/or vehicle impound, drivers would be more likely to drive under the speed limit. Hopefully doing so will result in safer roads.

1.2 Problem Definition, Methodology and Challenges

This thesis will focus on a method to implement a passive speeding measuring device. More specifically, a video surveillance camera is used, which captures images from the roadway, but will not emit any detectable signals to devices on the road. To make the system more cost effective, an existing road surveillance camera can be used as the video signal input device.

Generally, this problem can be solved using a standard computer to process the video images provided by a camera to extract the speed information of the passing vehicles. There are multiple algorithm image and video processing algorithms that can be used for this propose; however, these algorithms suffer from the following issues:

- Some algorithms, especially for those which identify vehicles at the first stage (e.g. OpenCV), require a large amount of computational power for object identification. Low-end or embedded processors cannot satisfy this computation power requirement; therefore, on-site high-end, or desktop-level processors are required, or the raw images must be sent to central server farm for processing.
- Installing on-site high-end or desktop-level generic processors will increase the overall cost of the system. Furthermore, the risk of theft of these systems will increase as they are essentially home PCs or servers.
- Sending the raw image data to a server farm requires high bandwidth, which will increase the cost of networking infrastructure, especially for remote regions such as rural communities.
- Some algorithms are written for, or optimized based on vendor-specific hardware, such as CUDA. These algorithms cannot run or suffer large performance penalties when using generic processors.

1.3 Contributions

This thesis proposes a simple algorithm and implementation that can be used to measure object speed in constant rate sequential images. Compared with other works, the main contributions of this thesis are:

- Using industry standard non-vendor specific OpenGL ES 3.1 core profile compatible platforms, meaning this algorithm can be implemented on any system that supports OpenGL ES 3.1.
- Using simple and basic algorithms that can be implemented using low-end embedded microprocessors to process the images on site, eliminating the need of high-end or desktop-level systems, or high bandwidth for raw image transferring.
- Using existing surveillance video infrastructure as input devices instead of specially designed video input devices.

1.4 Outline of Thesis

The thesis is organized as follows: Chapter 2 discusses some of the existing video-based speed measuring algorithms; Chapter 3 introduces the proposed algorithm which targets low-cost hardware; Chapter 4 discusses the difference between CPU, GPU, and

programable logic devices along with a range of design constraints, and performance considerations; Chapter 5 shows the experimental results; Finally, chapter 6 presents the conclusions and future works of this thesis.

CHAPTER 2

Literature Review

2.1 V2I

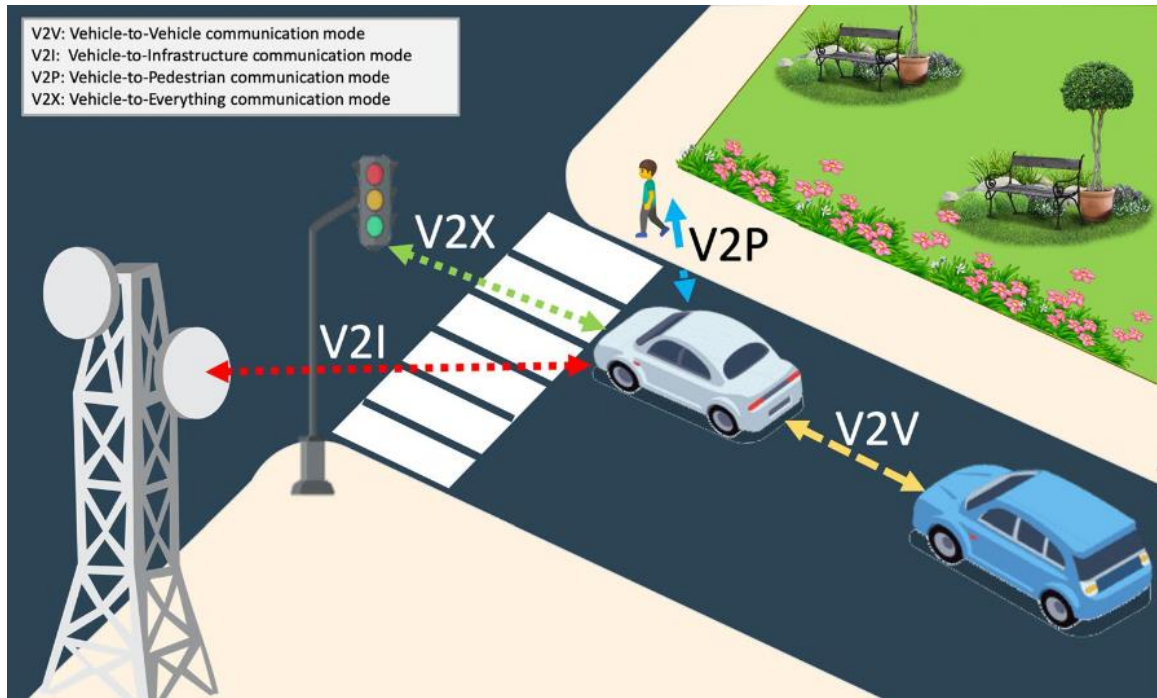


Figure 1 V2X Overview [3]

V2X technology is a relatively new technology; it includes V2V, V2I and V2P. V2X proposes to connect road vehicles and other things into a communication network to improve road safety. Figure 1 shows the concepts of V2X. In this figure, vehicles are connected to other vehicles through a V2V communication network; vehicles are connected to devices on pedestrians through a V2P communication network; vehicles are connected to road infrastructures through a V2I communication network.

Elements (vehicles, road infrastructures, devices on pedestrians) in this network will exchange data, so they are aware of the status of each other. For example, devices on pedestrians can send position information to vehicles to protect pedestrians from being hit. A vehicle in front of another vehicle can send acceleration information to the back one to avoid tailgating when the front vehicle brakes. Road infrastructures can send road information to vehicles such as signal lights and speed limits to guide vehicles.

The U.S. Department of Transportation proposed a Federal Motor Vehicle Safety Standard (FMVSS) which requires vehicles to share position and motion data (including

speed) in the network [4, p. 60]. In addition, road infrastructures can provide vehicles speed warning to help remind drivers to slow down [4, p. 98]. Although this proposal states personal identification data such as VIN number should not be shared in the network [4, p. 59], it does not explicitly state that the system should not be used for over-speed detection. The system is well suited for it since this information can be relayed to a secondary system that records the violators' speed and vehicle's license plate.

However, this implementation has a few drawbacks:

- This implementation requires large scale infrastructure such as beacons for receiving speed data from road vehicles. This requires very large investments in implementation and maintenance on the infrastructure [4, p. 21], making it economically not favorable. This implementation may be feasible for cities with high population densities (such as the Greater Toronto Area), but for most other areas of Canada (such as Northern Ontario), this implementation is not financially practical.
- This implementation requires road vehicles to have transponder devices installed to relay information to the roadside infrastructure. It is possible to install this transponder for new road vehicles at the factory as OEM systems; however, for existing vehicles, it is not practical to request the customers to install the transponder, nor to recall these vehicles to install the transponder. Furthermore, because the later-installed transponder must be connected in the vehicle's CAN network, it exposes the vehicle into an untrusted public network if not properly configured, which poses a security concern.
- The transponder installed on vehicle may be compromised by a driver [4, p. 204] if they want to avoid been ticketed when speeding. It is also politically not practical to introduce regulations to prevent drivers from modifying their own vehicles.

2.2 Object Detection and Tracking

A computer vision system may be installed on roadside infrastructures to measure vehicle speeds. Compared to the V2I method, this method does not rely on the potential unreliable information provided by each vehicle.

An example of object tracking is given on OpenCV's website [5] which shows it can be used to track multiple objects in a scene. This method can be divided into two steps:

1. Object identification: Since there may be multiple objects of interest in the same scene, a feature vector is provided for each of the objects to identify them. For road vehicle speed measurement, this method should detect all road vehicles in

the scene and provide this feature vector for each of them, such as type, color, size, etc. Other objects on the road such as road signs, signal lights, pedestrians, and noise should be filtered out.

2. Object tracking: This step tracks a specific object in two frames by matching a specific object in one frame to another object in another frame with highest similarity of the feature vector. This step can also be used to find the displacement of the same object in two frames. The speed of the object can then be derived by dividing this displacement by the inter-frame time interval.

There are multiple off-the-shelf algorithms which can be used for object identification and object tracking in OpenCV, respectively. Different algorithms yield different performance and accuracy for different situations. A comparison of the success rate and processing time among several algorithms in OpenCV are given in [6]. For example, the MIL tracker has a higher success rate than the MediaFlow tracker for high-speed movement but requires higher processing time. The MIL tracker also has higher accuracy than the BOOST tracker when the interested object has a similar color as the background, but lower accuracy if the resolution of the input image is low. In some cases, the result of these algorithms may not be predictable. For example, the success rate of the MIL tracker varies from 24% to 76% when portions of the interested object leaves the view. It may be challenging to choose the best algorithm for vehicle speed measurements on urban and country road environments respectively, due to different traffic speed and density.

The advantage of using object identification and tracking is that this technology has become popular in recent years. The algorithms and software frameworks are constantly evolving; furthermore, these off-the-shelf solutions can be used which minimize development costs. Other than that, there are large number of samples available online which can be used to train the model.

Conversely, this method has drawbacks:

- There are many objects identify algorithms available, however, none of them are perfect. Sometimes they are not able to identify an object of interest, or they give false positive identification on a non-interested object (unpredictable and low success rate [6]). Furthermore, deep learning-based detection and identification algorithms require a tremendous amount of computation power, which are not favorable for low-end embedded systems.
- This method requires memory to store the feature vector of identified objects. When an object exits the scene, its feature vector should be deleted; when an object temporally gets hidden (for example, blocked by another object), its feature vector should be kept. Therefore, this method requires an advanced memory management strategy to decide when to delete feature vector data, so it will not lose tracking temporally hidden object, but also remain small memory footprint.

- Most importantly, when using a GPU, it is not allowed to allocate memory on GPU side. Memory must be allocated by CPU first, then passed to the GPU.
- GPUs run multiple threads (workers) in parallel, sharing memory between GPU workers requiring exclusive synchronization between them.
- When tracking objects, the system needs to match all objects in one frame to other objects in another frame with the highest similarity of the feature vector; this requires a complexity of $O(n^2)$. When the road is busy with large amount of road vehicles, the number of comparing increases dramatically. For low-end embedded systems, the amount of load of this method may be too heavy.

2.3 Stereo Vision

Stereo vision uses two camera views to construct a 3D model of a scene; it is mainly used to measure the distance between the target object and the camera. By measuring the change of distance in a specific period, the speed of that object can be calculated. Like object detection and tracking methods, this method only relies on roadside infrastructures; however, it does not require the large computation resources like the object identification and tracking methods.

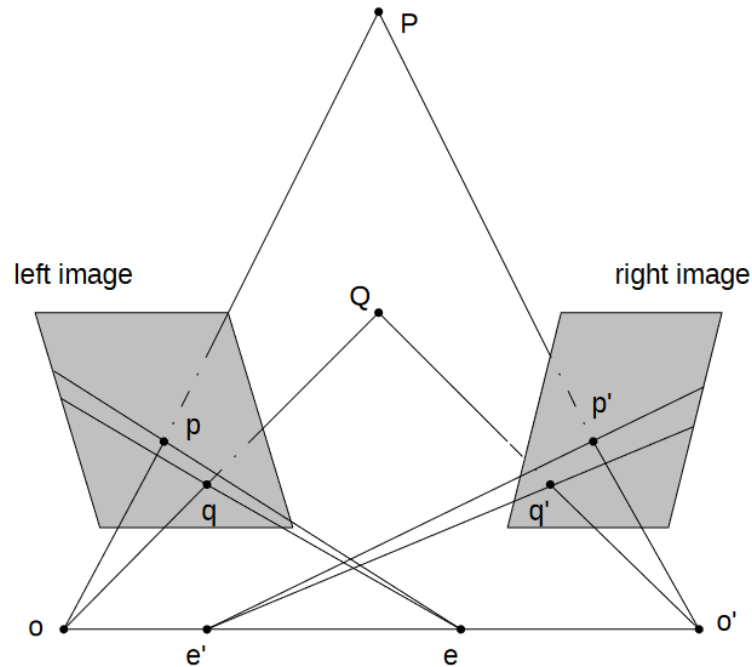


Figure 2 Stereo Vision Concept [7]

Figure 2 shows the concept of a stereo vision system. In this figure, P and Q are two objects in the scene, p , q and p' , q' are the projected points on left and right screen, respectively. Because of the difference of object location in the scene, they cast different positions on the left and right screen. It is clear to say that, for an object X , the further the distance between object and the camera, the smaller the angle $\angle X$. When the object X is at infinite far distance, the angle $\angle X$ will be 0° , whereas when the object X is at 0 distance, the angle $\angle X$ will be 180° . At the screens' point of view, the further object will be located more on the left-side of left screen and right-side of the right screen; closer objects will be located on the right-side of the left screen and left-side of the right screen.

After proper calibration, the stereo vision method shows very good accuracy when measure the distance between the camera and the object of interest, however, this system has a few drawbacks as well:

- This system requires two cameras, which increases the cost of the system.
- For a road-side infrastructure, the cameras are in an extremely hazardous environment. The cameras require protection from rain, dust, and other debris. Since the two cameras need to be synchronized as the distance between cameras (baseline) and the direction of each camera are directly affecting the distance calculation. When any of the cameras are hit by debris, there is potential for the camera to move thus changing its direction or the baseline length; this requires the system to be re-calibrated manually which is not ideal for road infrastructure as maintenance is costly. To solve this issue, a special calibration object may be placed near the road which the system will use to periodically check calibration and perform self-calibration if necessary.

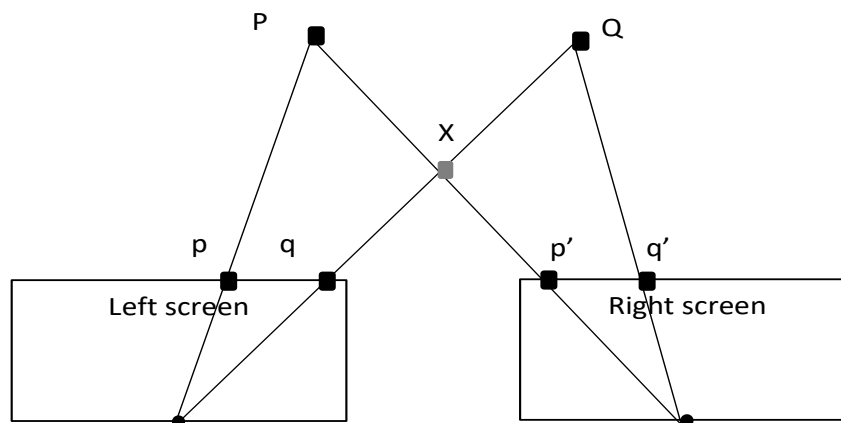


Figure 3 Stereo Vision Mismatch

- The algorithm must match a specific point in the left screen and its associated point in the right screen. When there are multiple objects of interest, the system

may not be able to correctly match these points; if the match is incorrect, the result will be incorrect. An example is shown in Figure 3. There are two objects P and Q in this scene. p and q are the projection of these two objects in left screen; p' and q' are the projection of these two objects in right screen. If the system mistakenly matches q in left screen and p' in right screen, a false object X will be perceived in the model.



Figure 4 Rolling Shutter [8]

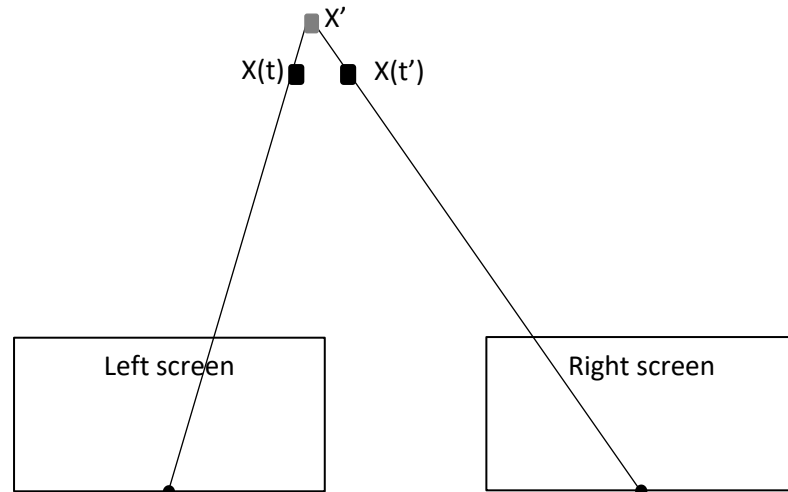


Figure 5 Camera Scan Delay

- When a camera is taking snapshot of a scene, the camera may use either a global shutter (record all pixel of the scene in one instant) or rolling shutter (scanning the pixels in the scene in series). An example of the result of rolling shutter is shown in Figure 4. Here, the image sensor is not capturing all the pixels in one instance; instead, the image sensor captures the pixels in series, from top to bottom. The effect is especially noticeable when recording fast-moving objects using a slower camera. Furthermore, when the system sends the shutter command to both cameras, each camera's response may be slightly delayed compared to the other. In stereo vision system, rolling shutter and the difference of response times will cause error in the measurement result. An example is shown in Figure 5, the left screen and right screen record a horizontally moving object X at different time ($X(t)$ for left screen and $X(t')$ for right screen). When calculating the location of object X , the result will be incorrect for location X' .
- Ultimately, this system relies on the resolution and quality of the input images; when the images resolution are low, the system cannot accurately measure the location of the objects of interest; when the images are blur, the system cannot provide correct measurements. Therefore, the system will provide the wrong speed reading for that object. This issue is especially critical when the object is located at a further distance or in bad weather.

2.4 Using Motion Detection at a Specific Point

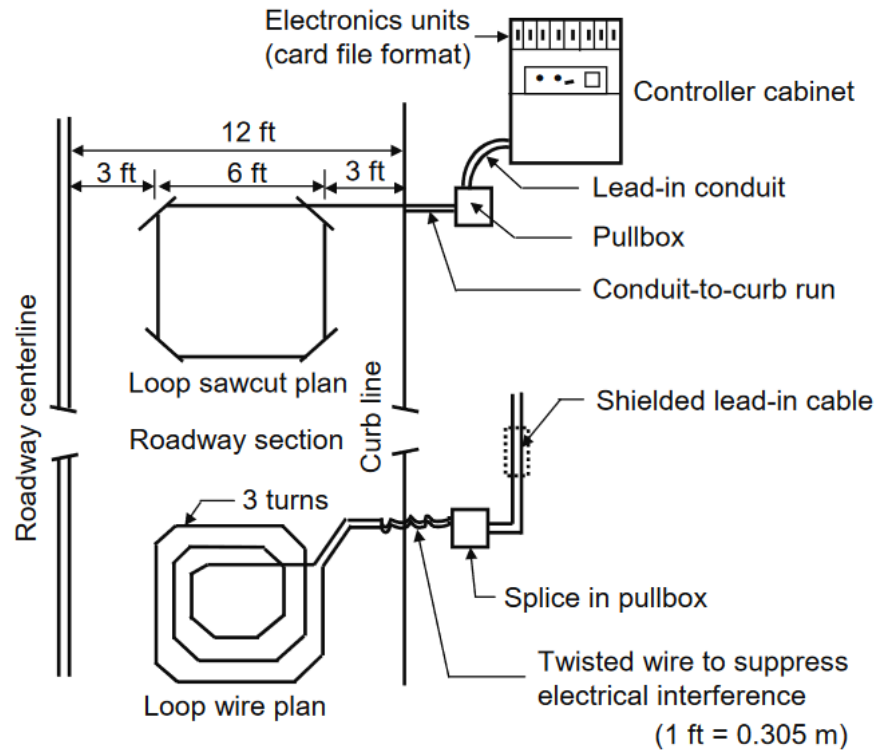


Figure 6 Inductive loop [9, p. 44]

When a road vehicle passes through an inductive loop, it causes distortion in the magnetic field inside the loop, which then generates a current in the coil wire [9, p. 44]. This can be used to detect when a road vehicle reaches a specific point on the road. The speed of the vehicle can be measured by examining the time it takes to travel across the loop, or the time it takes to move from one loop to another.

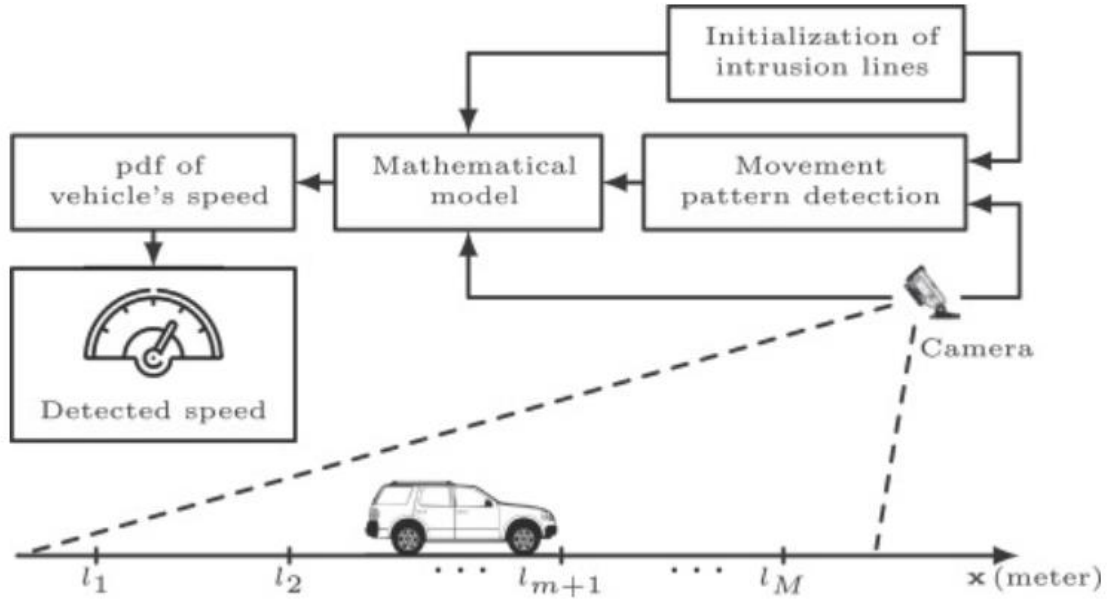


Figure 7 Motion Detection at Specific Point [10]

Instead of using physical loops installed in the pavement, [10] uses a camera to measure the timestamp of a specific vehicle passing through a set of virtual intrusion lines in a scene to provide the speed measurement, as shown in Figure 7. The advantages of this method include:

- Compared to the physical inductive loop (which must be installed in the pavement), a camera mounted over the road is more accessible for maintenance and is in general more cost effective [9, p. 37].
- A camera is a passive device and cannot be detected by signal detectors.
- A physical inductive loop may emit magnetic fields which may cause interference for other devices, including life supporting devices of the driver and passengers.
- Compared to other advanced video-based algorithms, this algorithm only requires object detection, which is less error prone and consumes less computation power, making it favorable for low-end embedded systems.

This method also has drawbacks. As noted by the authors, its accuracy is limited by the framerate of the camera as it counts the number of frames between the object hitting intrusion lines to calculate speed. Furthermore, that thesis only focuses on one-lane environments.

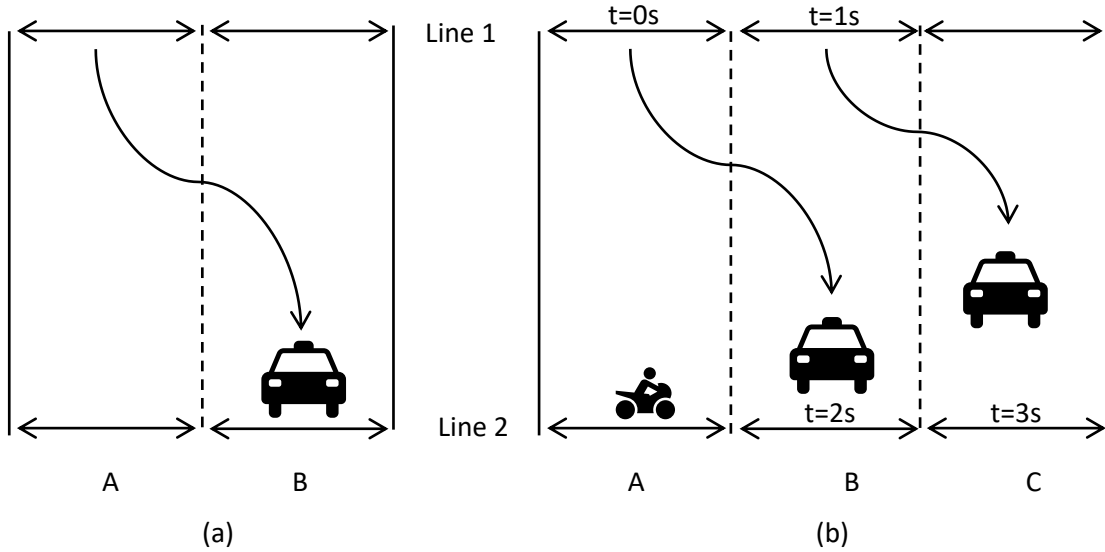


Figure 8 Intrusion Line Method Issue

Figure 8 (a) shows the issue with a two-lane road and a vehicle changing lanes between two intrusion lines. The result is that this vehicle will have a starting timestamp on *lane A* at *line 1* but an ending timestamp on *lane B* at *line 2*; the system is unable to calculate the speed of either object on either lane.

Figure 8 (b) shows another realistic situation; assume the time to travel the distance between the two intrusion lines is 2 seconds. A vehicle hit *line 1* on *lane A* at 0 seconds but hit *line 2* on *lane B* at 2 seconds as it changed lanes to avoid a slow-moving motorcycle. Another vehicle hit *line 1* on *lane B* at 1 seconds but hit *line 2* on *lane C* at 3 second to avoid the first vehicle. Although both vehicles travel under the speed limit (which uses 2 seconds to travels the distance), but from the system's perspective, the vehicle on *lane B* only uses 1 second, which generates a false speeding situation.

2.5 Using Model Size Comparison

Model size comparison [11] measures the size of the vehicle instead of the distance between the vehicle and the camera.

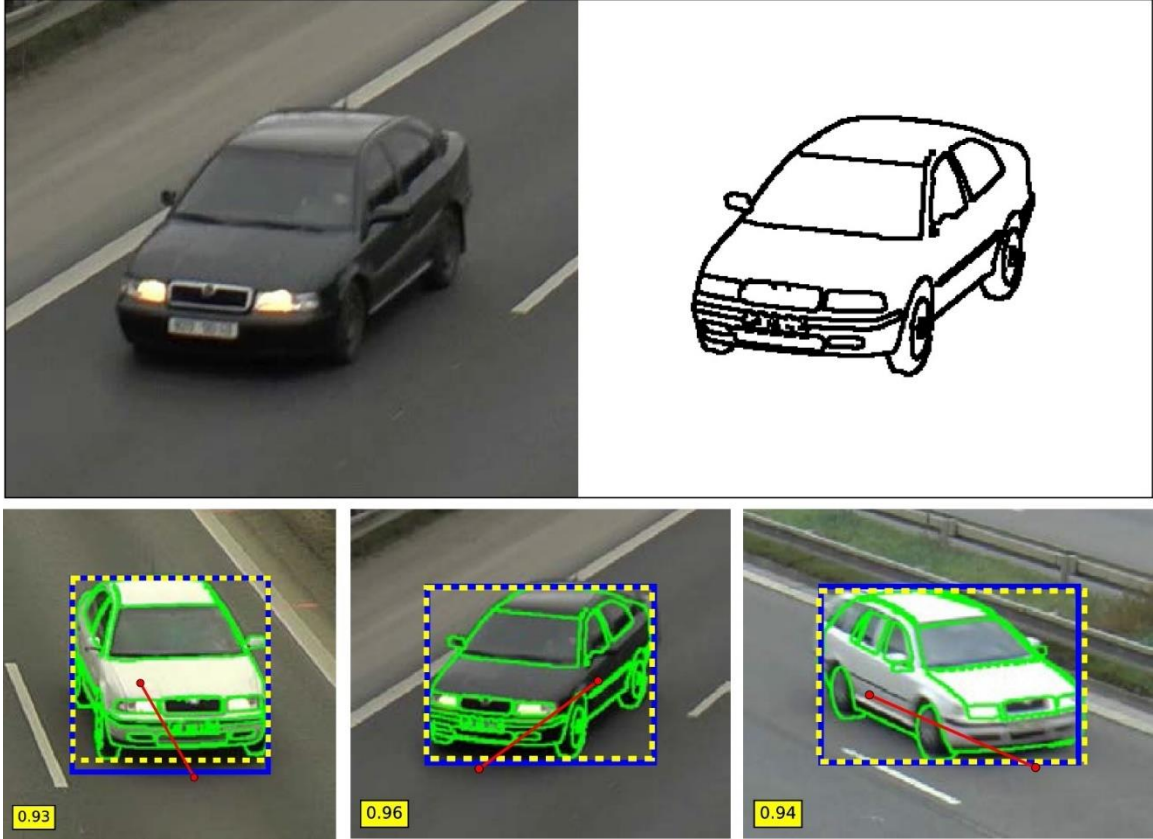


Figure 9 3D Model Based Speed Measurement [11]

In perspective views, a closer object looks larger, and a farther object looks smaller. Therefore, on a camera, a closer vehicle occupies more pixels and a farther vehicle occupies less pixels. Figure 9 shows the concept of this method. When a vehicle is identified, it is compared with a model library to identify the actual model of that vehicle. Then, the size of the identified vehicle on camera is compared with the model in library to find the ratio of size between them. Next, this ratio is passed into an algorithm to determine the distance between the vehicle and the camera.

Compared to the stereo vision method, this method only requires one camera, hence it does not require camera calibration as well as synchronization between the two cameras. Furthermore, most cameras are built with same common aspect ratio and field of view (or can be easily set), making it an off-the-shelf unit.

However, this method has drawbacks when identify the model of the vehicle which can end up with miscalculated distance:

- It requires a library with a large set of vehicle models, which needs to be updated often as models change from year to year from all the vendors.
- This library cannot cover all specialized mobile equipment (e.g., cranes, lift trucks, or any customized variants). Furthermore, the identification process may be tricked when seeing a vehicle carrier.
- Some vehicles may also share similar shapes but may be different sizes. For example, a Ford F-150 pickup truck and Ford ranger pickup truck has similar shape, but the size difference is considerable.

2.6 Tracking License Plate

One issue associated with road vehicle detection and tracking is the ability to properly detect the road vehicle and provide a correct bounding box for tracking. If the system fails to detect the road vehicle or incorrectly detects the road vehicle, the system will not continue the process to measure the speed or simply provide inaccurate results.



Figure 10 Tracking License Plate [12]

Tracking the license plate [12] (see Figure 10) may simplify this process as the plate is always a simple group of characters. An optical character recognition algorithm can be used to replace the object identification algorithm in this step. In most case, license plate only contains 26 alphabetical characters and 10 numeric characters, making the identifying license plate much easier than identifying road vehicles itself. Furthermore, the position of the licence plate can be directly used, without making bounding box of the entire vehicle and then find center point to determine the location of that vehicle.

There are currently multiple digital systems that are capable of reliably reading license plate already in use, including ALPR (Automated License Plate Reader).

However, using license plates has drawbacks:

- In Ontario, license plates are required on both the front and rear of most road vehicles; however, vehicle in other provinces such as Alberta only require license plate on the rear.
- Some aged license plates are corroded making it difficult to read properly.
- Snow, dirt, and other debris may impede the reading as well.
- Decoration text on the body of the vehicle may be mistakenly used by this system (i.e. phone numbers, address, etc.)
- Leisure vehicles like snow mobiles have their license plate on the side.

2.7 Summary

This chapter has briefly covered some of the existing methods used in determining a vehicle's speed. Each have their own advantages and disadvantages in terms of implementation complexity. Ultimately, a fusion of algorithms that utilizes the lowest cost hardware will be desired.

CHAPTER 3

Proposed Algorithm

3.1 Terminology

There are two domains used in this thesis to describe the principle of the proposed algorithm: **Screen-domain** and **Road-domain**. Generally speaking, a **computer** “thinks” in the **screen-domain**, it reads and draws pixels in the screen-domain, while a **human** looking at the pixels on a screen would also realize objects in the **Road-domain**. A better terminology is “world-domain”, but for this algorithm’s purpose, the main focus are objects on road, so the term “road-domain” will be used.

In the screen-domain, there are three types of coordinate systems:

- The first one is **absolute** pixel system, where it will always refer the top-left corner of the screen to (x=0 px, y=0 px), and refer the bottom-right corner to (x=width px, y=height px), where **width** and **height** are the size of the video frame in the unit of pixels.
- The second one is **normalized device coordinate, or NDC**, which is used in the OpenGL vertex shader to address any point in the framebuffer. It will always refer the bottom-left corner to (-1.0, -1.0), and refer the top-right corner to (1.0, 1.0). This system is not used when describing this algorithm.
- The third one is **normalized texture coordinate, or NTC**, which is used in OpenGL to address any point in a texture object. It will always refer to the top-left corner as (0.0, 0.0), and the bottom-right corner as (1.0, 1.0).

In the road-domain, the origin point (anchor location of the camera pole) will always be (x=0 m, y=0 m). It will always refer forward to +y axis and backward to -y axis; and always refer left-hand side to -x axis and right-hand side to +x axis.

When referring to the coordinate in this document, “(x, y)” will be used, where the first element is the x-axis coordinate, and the second element is the y-axis coordinate. This document uses two ways to differentiate road-domain and screen-domain coordinates:

- (rx, ry) refers to the road-domain coordinate while (sx, sy) refers to the screen-domain coordinate. In this notation, “r” means the road-domain; “s” means the screen-domain. This notation is used when there is no actual coordinate value.
- (3.5 m, 7.3 m) refers to the road-domain coordinate, while (98 px, 128 px) refers to the screen-domain coordinate. In this notation, “m” (meters) is the road-domain

unit, commonly used to describe the geographic location or distance; “px” (pixel) is a screen-domain unit, commonly used to describe the pixel on the screen. This notation is used when there is an actual coordinate value. If there is no unit, like (0.5, 0.8), it means the screen-domain in NTC.

Object: Object refers to everything on the road-domain. An object can be a car, a truck, a pedestrian, a sign near the road, or even a deer jumping across the road.

Blob: Computers lack the ability to realize objects. In the computer’s point of view, everything is blobs made from one or multiple pixels.

Location: Location is used to describe where an object is. This terminology is used in the *road-domain*. Sometimes “geographic location” or “road-domain location” is used to exclusively state this terminology for the road-domain. An example is location (x=3.5 m, y=7.2 m), note the unit is in meters (m).

Position: Position is used to describe where a pixel is. This is used in the *screen-domain*. Sometimes “screen position” is used to exclusively state this terminology for the screen-domain. Examples are position (x=30 px, y=27 px) and (x=0.3, y=0.95), note the unit is in pixels (px) when using absolute pixel system or there is no unit when using NTC.

3.2 Algorithm Overview

This chapter describes the fundamental principle of the proposed algorithm. When describing this algorithm, some considerations of the design and optimization based on chosen platform are introduced. Further details of the design consideration and platform optimization phase are discussed in the next character.

The proposed algorithm can be divided into two steps:

1. Detect a moving object in the scene.
2. Measure the distance traveled by the detected moving objects.

It is important to state that this algorithm only detects moving objects; there is no object identification and hence no object tracking is performed. Avoiding object identification eliminates issues associated with false identification and tracking. This simplifies the proposed algorithm which makes it favorable for implementation on embedded systems. However, simplifying the algorithm poses some hazards:

- Since there is no object identification, the algorithm will measure the speed of all moving objects in the scene, including pedestrians, bicycles, etc.

- The algorithm will measure the instant speed of each object by finding the closest object in the previous frame.

To remedy this, a few assumptions are made:

- That road vehicles travel faster than other road objects, such as pedestrians, bicycles, etc. In most cases, road vehicles will travel at speeds greater than 50 km/h in urban areas and 80km/h in rural areas, whereas pedestrians and bicycles travel at speeds lower than 20km/h. Any object traveling slower than this threshold will be ignored.
- Those objects meet a particular minimum size to be detected. Pedestrians and bicycles are much smaller than road vehicles and will therefore be ignored.
- That all objects travel in a relatively straight direction parallel to the lane; sudden direction changes are not dealt with.
- The speed of the edge pixels of an object to determine its speed. Identifying an object in an image is a complex algorithm, but identifying the edge pixel of an object in an image is a relatively simpler task.
- The edge detection of an object is always successful and the detected edge travels with the object at the same speed.
- Speed is measured by analysing the displacement of an object from one frame to another object in another frame with the closest distance.
- The vehicle only moves a short amount between the frame times.

This algorithm is designed and optimized for execution on a GPU using OpenGL or the OpenGL ES API. It may be possible to execute this algorithm on a CPU using a software emulation layer; but it is likely the performance will be significantly worse than using a native GPU.

Due to the limitations of GPUs and OpenGL, special considerations are required. The most important are:

- Shader programs running on a GPU cannot allocate memory. This is intentional to prevent race conditions between the GPU threads (workers).
 - o To access a buffer, the buffer space must be allocated by the CPU first, then supply it to the GPU for further use.
 - o To read external data, the data must be read into a buffer on the CPU side. This buffer is then uploaded to the GPU side for further reading.
 - o To write data to external space, a buffer must be created on the CPU side. This buffer is then supplied to the GPU side for writing. After the GPU is finished writing, this buffer is downloaded to the CPU side for external access by the CPU.

- The GPU runs multiple workers at the same time. Writing data to shared resources requires exclusive synchronization between the workers to avoid race conditions. The exclusive synchronization between workers is costly and is therefore best to avoid operations that require synchronization.
- The CPU and GPU have their own local data space. Accessing local data is fast and requires no synchronization; accessing shared data or transferring data are slow and require synchronization. Again, it is best to avoid accessing non-local data. Furthermore, data should be transmitted between the GPU and the CPU using asynchronous memory accesses.

3.3 Region of Interest (ROI)

In most cases, the road only occupies a portion of the scene provided by a camera, so most of the pixels do not need to be processed. Therefore, a region of interest is defined to enclose only the pixels that need to be processed. Doing so saves computational power by reducing the number of workload pixels and eliminates the “noise” outside of the road area. For example, if there is a high-speed railway alongside a 50 km/h road, the region of interest can be used to exclude the pixels representing the railway. Without using a region of interest, every train passing by will trigger the speeding alarm.



Figure 11 Focus Region

Figure 11 shows an example of region of interest where the road takes less than 1/3 of the entire image. By ignoring the area outside the red polygon, the workload is dramatically

reduced. Internally, this polygon is constructed into an OpenGL vertex array object (VAO) and passes them to the GPU during the shader stage, where only the pixels inside are processed.

3.4 Moving Object Detection

3.4.1 Moving Blob Detection

Prior to any image analysis, a blur filter is applied to remove any high-frequency noise (i.e. generated on the camera CCD) and noise introduced by image compression and decompression.

To begin detecting moving objects, this algorithm compares the current and previous images as those pixels are likely to be changed.

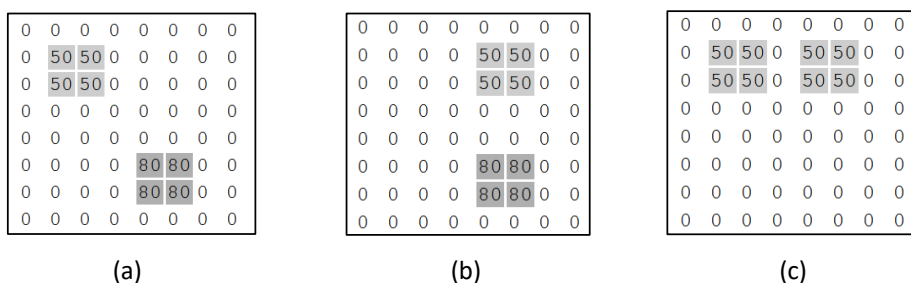


Figure 12 Moving Object Detection

An example of two objects with their color values are shown in Figure 12. From image (a) to image (b), the upper object moves in the right direction for three pixels, and the lower object does not move. Blobs in image (c) can be highlighted by calculating the absolute difference of image (a) from image (b).

When a pixel in the screen-domain involves a moving object, the change of the RGB value from image to image is generally large. Conversely, when the object is not moving, this change may still be non-zero RGB value normally due to ambient light conditions and/or sensor noise. In the practical implementation, a threshold on the RGB channels is used to determine whether this change is indicative of a moving object. This can be illustrated by the following code:

```

#define TH 30 //threshold

struct RGB { uint8_t r, g, b; };

struct RGB pf[height][width]; //previous frame
struct RGB cf[height][width]; //current frame

bool movingObject[height][width];
foreach (x, y in FocusRegion) {
    movingObject[y][x]
        = abs(pf[y][x].r - cf[y][x].r) > TH
        || abs(pf[y][x].g - cf[y][x].g) > TH
        || abs(pf[y][x].b - cf[y][x].b) > TH;
}

```

This algorithm describes a simple absolute subtraction of each pixel in the focus region. The computation itself is typical of hardware saturation operations typically found in GPUs, and therefore is generally implemented quite efficiently.

When a group of pixels are not changing above the pre-set threshold, the algorithm will not detect this as a moving object. One example of a fixed object is parked vehicle, however, for this application, a fixed object has no speed; hence, it can be ignored.

It is important to state, this algorithm is not directly detecting moving objects; in fact, it is detecting the result of object moving by using the change in pixels' color value. As the example in Figure 12 shows, the result in (c) does not highlight the moving object but the pixel has changed color. Despite that, the changing of color value is always travelling with the moving object. Therefore, measuring the travel speed of the changing of color value is equivalent to the speed of the object itself.

When the color of the measured vehicle is close to the road, such as mat grey, this algorithm will not work because it cannot be clearly seen. In this case, infrared cameras may be used to sense the infrared generated by tire and engine of road vehicles.

3.4.2 Grouping Blobs to Detect an Object

The proposed algorithm assumes that most road vehicles have a uniform-colored surface. When the moving distance of a specific vehicle between two frames is less than its length, the moving object will only detect fragments of this vehicle.

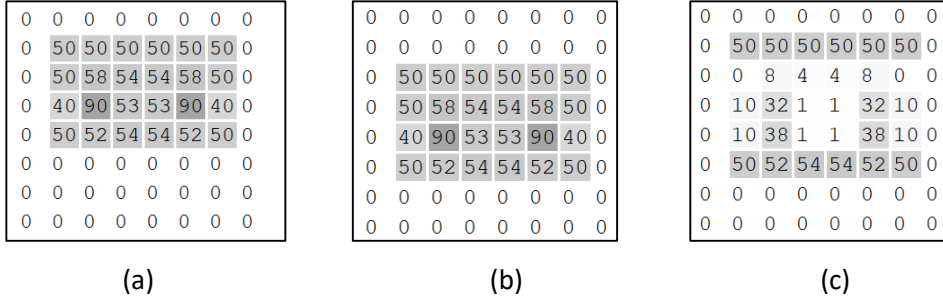


Figure 13 Moving Object Detection Grouping

Figure 13 shows an example of this issue using an object and its color value; (a), (b) and (c) are representing the scene in previous frame, in next frame, and the difference between two frames respectively. As the object has only moved one pixel down (which is less than its height of 4 pixels), and the pixels' color values in the object are similar, this results in two blobs being detected.

To compensate for the issue of having multiple blobs per object, the proposed algorithm groups these blobs to restore the whole object. A simple method is used assuming all the blobs are less than one-car length away to each other. If the distance between the blobs is greater than the size of the vehicle, it is assumed they are not from the same vehicle. This concept is shown in the following code:

```

/* Get width and height of vehicle based on perspective view position */
/* This information is saved in roadmap, see later section for detail */
void getVehicleSize(int pos[static 2], int size[static 2]) {
    size[0] = ...;
    size[1] = ...;
}

bool blob[height][width];

/* For each pixel in the scene */
for (int y = 0; y < height; y++) for (int x = 0; x < width; x++) {
    if (!blob[y][x]) {
        int size[2]; getVehicleSize({x,y}, size);

        /* Search blob in all direction within 1-car size */
        bool leftSearch = 0, rightSearch = 0,
             upSearch = 0, downSearch = 0;
        for (int i = x - 1; i > x - size[0]; i--) if (blob[y][i]) {
            leftSearch = 1; break;
        }
        for (int i = x + 1; i < x + size[0]; i++) if (blob[y][i]) {
            rightSearch = 1; break;
        }
        for (int i = y - 1; i > y - size[1]; i--) if (blob[i][x]) {
            upSearch = 1; break;
        }
        for (int i = y + 1; i < y + size[1]; i++) if (blob[i][x]) {
            downSearch = 1; break;
        }
        if ( leftSearch && rightSearch && upSearch && downSearch )
            blob[y][x] = 1;
    }
}

```

3.5 Object Tracking

The previous step generates several objects in the scene, each representing a moving object. The next step is to track these objects to get their displacement during a period.



Figure 14 Object Tracking Basic Example

Figure 14 shows a simple example scenario where two objects are traveling in the left direction. In this example, location P' and Q' represent objects in the first frame; location P and Q represent the objects in the second frame.

The challenge here is for the algorithm to track the correct object (i.e. it should correctly track P' to P and Q' to Q respectively).

3.5.1 Closest Object Tracking and Sampling Rate

The proposed algorithm assumes that the position of an object in the scene will not suddenly change in a relatively short period of time. If this is the case, the easiest way to track a specific object is to find the next closest object in the following frame. In the above example, the proposed algorithm will search for the closest objects to P and Q when being tracked and will find P' and Q' respectively.

Since any object should only travel a limited distance in this short period of time, a distance threshold is applied to limit the searching range. If the tracking object cannot be found, the tracking will abort.

To properly track an object, the frame rate must be greater than double of the number of objects passing a specific point on the road, essentially the Nyquist frequency.

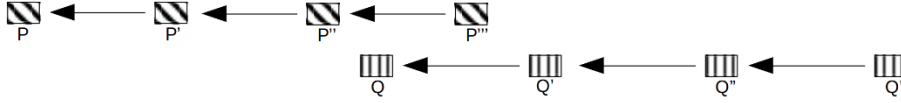


Figure 15 Object Tracking Minimal Rate

Figure 15 shows an example of tracking objects with different frame rates. In this example, P and Q are the objects at current timestamp (t), P' and Q' are objects at previous timestamp ($t-1$), P'' and Q'' are objects at timestamp ($t-2$), and so on. Note the object is travelling in left direction; since the tracking is performed from current frame to previous frame, the tracking is performed in right direction in this figure.

When the sampling period is $1t$, object P travels (from P' to P) less than half the distance between P and Q . In this case, the Nyquist frequency is higher than the signal frequency. Therefore, the tracking of P' to P can be performed correctly since P' is closer to P than Q . When the sampling period increased to $2t$, displacement of object P (from P'' to P) is greater than half the distance between P and Q ; the Nyquist frequency is now lower than the signal frequency. Now P cannot be tracked since P'' is closer to Q than P .

Although a higher sampling frequency helps tracking the correct object, the measurement accuracy is negatively affected as the precision of the screen pixels is limited. If the object is moving slowly enough such that the frame-to-frame pixel differences are small, poor predications will be made of the speed based on these subsequent frames (frame[t] to frame[$t-1$]). In such an event, only the direction of travel of the object should be determined from these frames and not the speed. The object speed can be calculated more accurately by effectively lowering the sampling rate or using a wider spread of frames

(from frame[t]) to frame[t-1-n]). The proposed algorithm does just this by using a two-stage approach, one for the object direction, and one for the speed.

However, there are limitations to this approach as it allows the program to lower than the Nyquist frequency by $\frac{1}{2}$ for the displacement measurements. Further reductions of the sampling rate are found to yield incorrect results. If the sampling period is $3t$ in the above example, the algorithm will mistakenly track Q to P''' . There may be some more intelligence solution to prevent the system from doing this, but that may increase the complexity of this algorithm.

3.5.2 Merging and Separating Objects

Another issue when preform object tracking is when objects merge and separate. To illustrate this issue, assume object P and object Q are travelling in four cases:

- When object P and Q travel separately. In this case, the algorithm will detect both object P and Q . The algorithm performs tracking on each object separately, as discussed in early sections of this thesis.
- When object P and Q merge together. In this case, the algorithm will detect both object A and B before the merge. After the merge, the algorithm will only detect one object, call it object X . When perform tracking, the algorithm will track object X to either object P or object Q , depending on which object is closer. If object P and Q are travelling at different speed, the algorithm will only give the shorter displacement at the merging frame.
- When object P and Q merged. If object P and Q are travelling at the same speed, the algorithm will only detect one object, object X . In this case, this algorithm will consider two objects a united object; therefore, only one displacement measuring will be provided by the algorithm. This displacement can be used to describe the displacement of both objects because they are travelling at the same speed. If object P and Q are travelling at different speed, they will be separated in following frame.
- When object P and Q separate. In this case, the algorithm will detect one object (object X) before the separate. After the separate, the algorithm will detect both object P and Q . When perform tracking, the algorithm will track both object P and Q to X respectively. Displacement measuring given to object P will be the displacement between object P and X , and displacement measuring given to object Q will be the displacement between object Q and X .

3.5.3 Rectification Tracking

The proposed algorithm determines an object speed by tracking its displacement between several sequential frames. Object tracking on a 2D image in general requires considerable computation steps. The following unoptimized code demonstrates this:

```
#define TH 100 //Search distance

bool pf[height][width]; //Previous frame, 1 = moving object detected
bool cf[height][width]; //Current frame, 0 = not detected

/* For simplicity, two-stage search is not shown in this code example */

/* For each object in current frame */
for (int y = 0; y < height; y++) for (int x = 0; x < width; x++) if
(cf[y][x]) {

    /* Find the closest object in previous frame, search from center */
    for (int i = 0; i < TH; i++) for (int j = 0; j < TH; j++) {
        int yu = y - i, yd = y + i; //y up and down search
        int xl = x - j, xr = x + j; //x left and right search
        // Assume yu always >= 0, yd always < height
        // Assume xl always >= 0, xr always < width
        // This is automatically down in GPU shader program
        if ( pf[yu][xl] || pf[yu][xr] || pf[yd][xl] || pf[yd][xr] ) {
            /* Found closest object in previous frame, measure displacement */
        }
    }
}
```

The number of iterations performed for each object is determined by the search distance; For example, if the search distance is 100 pixels in each direction, 40000 iterations will be performed for each pixel of each object. It can be said that the complexity of this algorithm is $O(n^2)$, (big-O notation).



Figure 16 Object Tracking on Road Lanes

For the intended application, searching in each direction is unnecessary as the direction of movement is already known (i.e. the road travel direction). Figure 16 shows the path the road vehicles are likely to travel. This is true for most cases, except:

- Road vehicles may change lane. In this case, the traveling direction is slightly different from road direction. If the object movement less than the width of the object itself, the object should still be found when searching in the road direction. Detail of this situation will be discussed later.
- Road vehicles may perform U-turns. In this case, the vehicle is traveling in a direction that is totally different from the road direction, however, the speed is normally low and therefore can be ignored.

By applying a linear search, the complexity of the algorithm is reduced considerably to $O(n)$. However, the search direction itself is not purely vertical or horizontal. In the above example, the objects will be tracked by using the corresponding linear equations of each direction of travel.

In perspective view, all lanes on a parallel road vanished into one vanishing point on screen point (sx_{van}, sy_{van}) . So, for any road point projected on screen point (sx_{ref}, sy_{ref}) , the line connecting (sx_{ref}, sy_{ref}) and (sx_{van}, sy_{van}) will be the direction of vehicle travel. It is highly likely that this same object will be presented on the line connecting (sx_{ref}, sy_{ref}) and (sx_{van}, sy_{van}) in subsequent frames.

During searching, to find the x-value of a point on a specific line based on a y-value, use:

$$\frac{x - x_{ref}}{y - y_{ref}} = \frac{x - x_{van}}{y - y_{van}} = \frac{x_{van} - x_{ref}}{y_{van} - y_{ref}}$$

Which is:

$$x = \frac{x_{van} - x_{ref}}{y_{van} - y_{ref}} * (y - y_{ref}) + x_{ref} = \frac{x_{van} - x_{ref}}{y_{van} - y_{ref}} * (y - y_{van}) + x_{van}$$

During searching, the algorithm will try multiple iteration with different y-values until the target blob is found. During this process, the reference point(sx_{ref}, sy_{ref}) and vanishing point(sx_{van}, sy_{van}) will never change for the same scene; therefore, a ratio value r can be defined:

$$r = \frac{x_{van} - x_{ref}}{y_{van} - y_{ref}}$$

The equation becomes:

$$x = r * (y - y_{ref}) + x_{ref} = r * (y - y_{van}) + x_{van}$$

This ratio value will be constant for a flat straight lane.

Implement this into code:

```

#define TH 100 //Search distance
#define VAN_X 600 //Define vanish point
#define VAN_Y 200

bool pf[height][width]; //Previous frame, 1 = moving object detected
bool cf[height][width]; //Current frame, 0 = not detected

/* For simplicity, two-stage search is not shown in this code example */

/* For each object in current frame */
for (int y = 0; y < height; y++) for (int x = 0; x < width; x++) if
(cf[y][x]) {

    /* Calculate the slope */
    float r = (VAN_X - x) / (VAN_Y - y);

    /* Find the closest object in previous frame */
    for (int d = 0; d < TH; d++) {
        int yu = y - d, yd = y + d; //y-pos of up and down search
        int xu = r * (yu - VAN_Y) + VAN_X; //x-pos of up and down search
        int xd = r * (yd - VAN_Y) + VAN_X;
        // Assume yu always >= 0, yd always < height
        // Assume xu and xd always >= 0 and < width
        // This is automatically down in GPU shader program
        if ( pf[yu][xu] || pf[yd][xd] ) {
            /* Found closest object in previous frame, measure displacement */
        }
    }
}

```

Although the above algorithm will track an object, its implementation in a GPU may suffer due to heavy mathematical compute load.

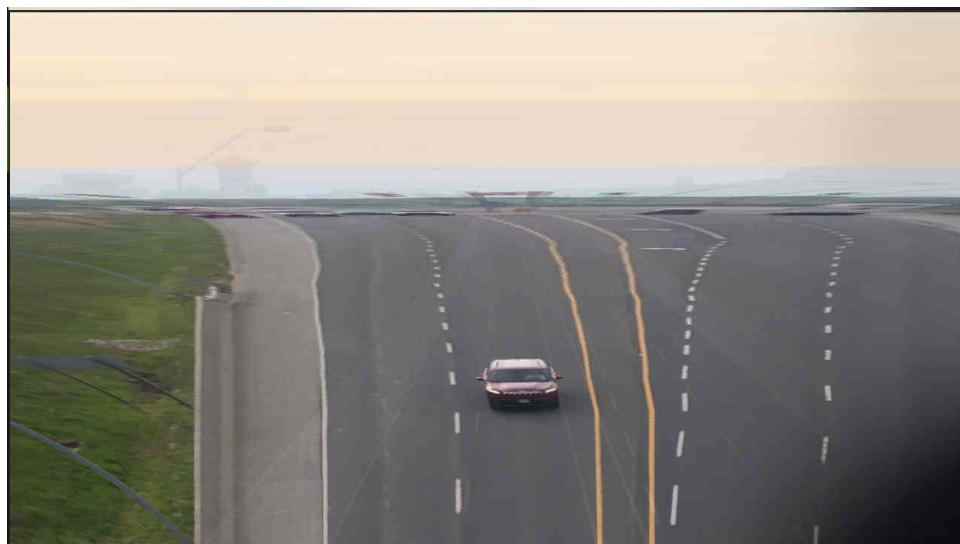


Figure 17 Blob Tracking Rectified View

A better choice is to transform the original scene from perspective view into a rectified view before the y-value iteration, as shown in Figure 17. All travel lanes are now parallel in the vertical direction; therefore, the algorithm only needs to iterate the y-value without changing the x-value. This takes advantage of any vertical pre-fetch. To perform this transformation, a texture object called roadmap can be read to quickly convert the scene between perspective view and rectified view.

3.5.4 Object Edge Detection

When measuring the displacement of an object from one frame to another frame, the proposed algorithm will only perform the measurement on edge pixels instead of all pixels of that object. The rationales for this choice are:

- The edge of the object moves with the object, therefore the speed of pixels on the edge represents the speed of that object. Processing more pixels of that object is redundant and consumes excessive computation power.
- It is simpler to match the pixels on edge from one frame to another frame than pixels inside the object.

To further mitigate measurement variations, the algorithm will measure the displacement on each pixel on the edge, then average the results to determine the speed of each object.

Given that the object travels almost vertically in the rectified view, the left and right edges of the object can be ignored. Only the lower edge of the object is considered due to the following major reasons:

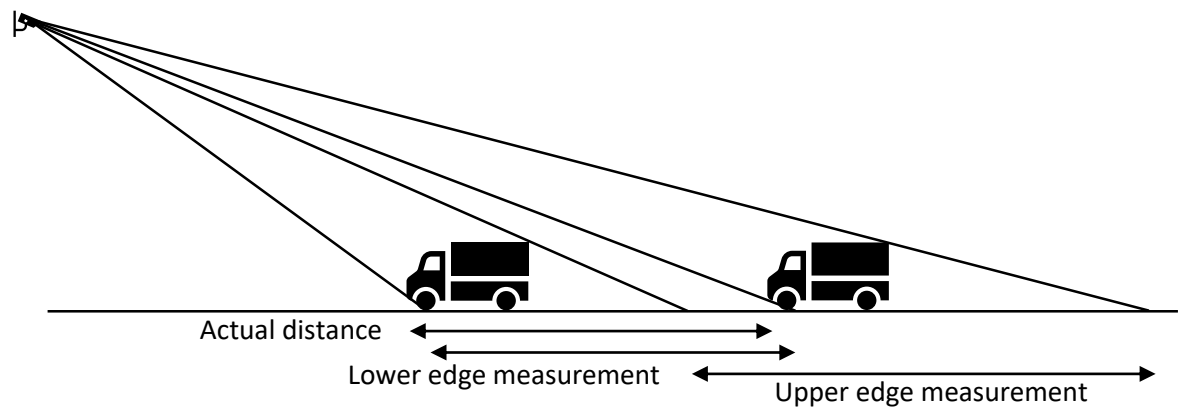


Figure 18 Object Edge Projection Line

- Figure 18 shows the actual displacement of an truck and the projected upper and lower edges. The displacement of the lower edges is closer than the upper edge to the actual displacement.
- Different vehicles have different heights, hence there is no commonly accepted height for the upper edge. For example, the upper edge height of a standard sedan

is about one meter high, but the upper edge height of an 18-wheel semi-truck is about four meters high. Whereas the lower edge of all vehicles is within a foot distance from the road surface.

- Some vehicles have irregular shapes which will poorly define the upper edge of the vehicle, such as top-mounted ladder and bike rack. Whereas the lower edge of all vehicles is generally a flat bumper.

3.5.5 Mitigating Tracking Overshoot

In an ideal case, the proposed algorithm only needs to iterate in the y-axis value to track an object in the rectified view. However, when the object changes lanes, or when the rectification is not perfect, there may be a tracking overshoot issue.

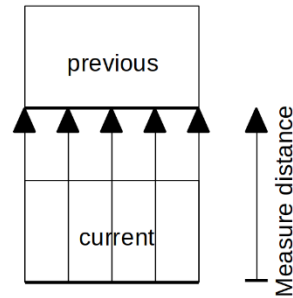


Figure 19 Object Tracking Ideal Case Without Tracking Overshoot

Figure 19 shows an ideal example of an object traveling downwards in a perfect rectified view. When object tracking, the proposed algorithm starts by searching upward from each pixel in the object's lower edge in current frame. Since the objects are traveling absolutely vertical, all the pixel searches will intercept the same object in the previous frame.

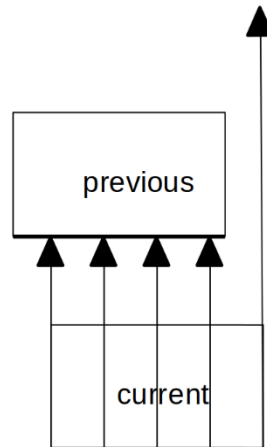


Figure 20 Object Tracking Non-Ideal Case with Tracking Overshoot

Figure 20 shows a non-ideal example during a lane change of road vehicles or some error in rectified view, where the object travels downwards slightly to the right. Like in the previous example, each pixel in this object's lower edge is searched upward. The intercepts occur on the left side, and not the right side, resulting in an overshoot.

If any of these searches exceed the search distance limit, they will be ignored. Whereas if they are intercepted by other objects, it will ultimately provide measurements to the wrong target object which negatively affects the correctness of this algorithm.

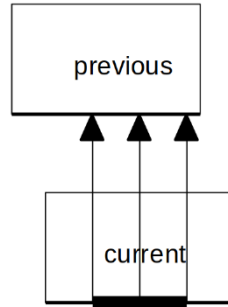


Figure 21 Blob Tracking Overshoot Fix

Figure 21 shows a solution to this overshoot issue where the algorithm searches only the center portion of the object's lower edge. If the horizontal movement is less than the width of the side portion of the edge between two frames, all the searches will be intercepted successfully to avoid overshoot.

The size of the side portion of the lower edge should be carefully selected. A higher value can reduce the error when vehicles change lanes, however if too large, it will reduce the size of the center portion of the lower edge, producing less searches and less reliable results. At some point, there may be no center portion of the lower edge due to the overly large size of the side portions, so no search will be performed in this case. The value should be about 1/3 of the width of the vehicles.

3.6 Speed Measurement

Up to this point, the proposed algorithm can find the start position and end position of an object in a specific time interval, and subsequently find the pixel displacement. However, the position in screen-domain must be converted into a road-domain location to make any practical mean. There are two methods of doing this.

Real-time Trigonometry

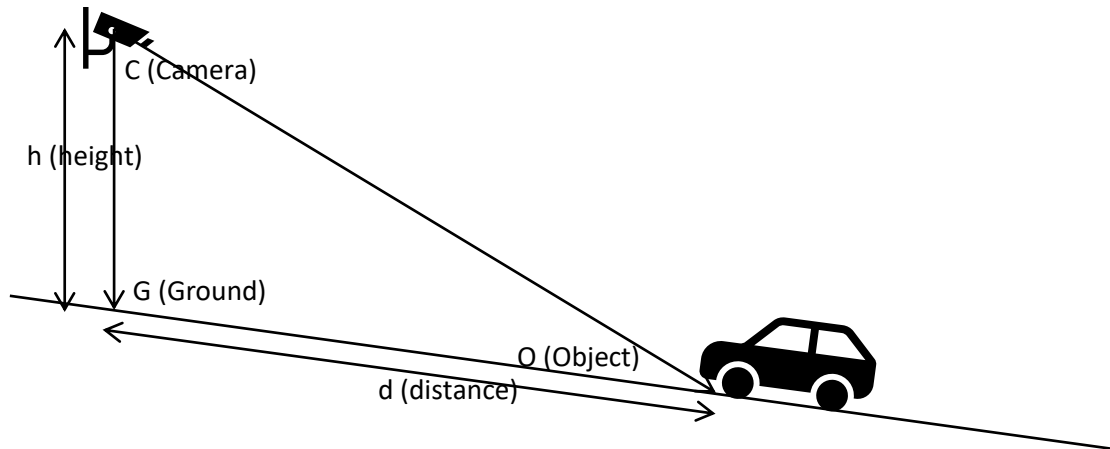


Figure 22 Screen to Road Domain Trigonometry Distance

Figure 22 shows three points, C (Camera), O (Object) and G (ground) as well as measurements d (distance) and h (height). Assuming the road is flat or on a constant incline/decline, d can be determined by using the Sine law:

$$\frac{d}{\sin(\angle OCG)} = \frac{h}{\sin(\angle COG)}$$

Since $\angle CGO$ and h are known at installation time, and the sum of all three angle in a triangle is 180 degrees:

$$d = h * \frac{\sin(\angle OCG)}{\sin(\angle COG)} = h * \frac{\sin(\angle OCG)}{\sin(180 - \angle OCG - \angle CGO)}$$

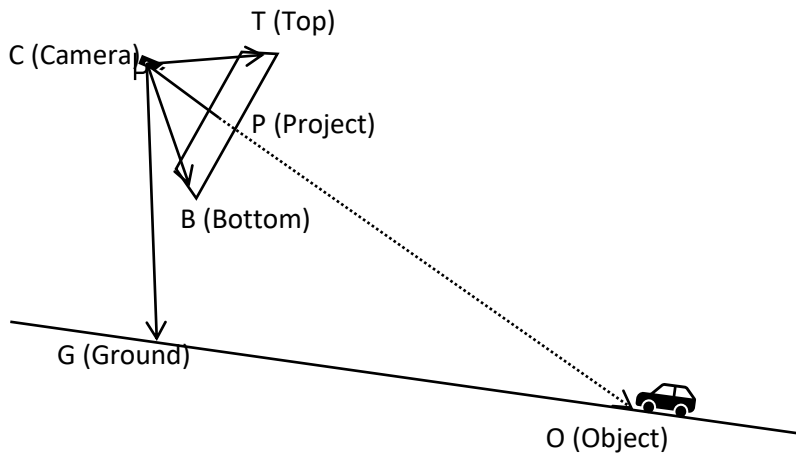


Figure 23 Screen to Road Domain Trigonometry Angle

To determine $\angle OCG$, consider Figure 23. Point C represents the camera; point P represents the projected position of the object on the screen, lines T and B are connecting the camera to the top and bottom boundaries of the screen respectively, line O is the line connecting the camera and the object, and line G is a line connecting camera and the ground.

Using OpenGL NTC where the top of the screen is 0.0 and the bottom of the screen is 1.0, P can be at any position between 0.0 and 1.0. $\angle OCG$ can be described as:

$$\angle OCG = (1.0 - P) * \angle GCT + P * \angle GCB, \quad \text{where } 0.0 \leq P < 1.0$$

Both $\angle GCT$ and $\angle GCB$ are based on the pitch of the camera and the FOV of the camera screen which are known at the installation time.

Combining both equations together gives:

$$d = h * \frac{\sin((1.0 - P) * \angle GCT + P * \angle GCB)}{\sin(180 - ((1.0 - P) * \angle GCT + P * \angle GCB) - \angle CGO)}$$

Pre-Generated Roadmap

A Roadmap can be generated using the above approach, but for each pixel in the orthogonal view. This look-up table can be stored in a texture object in the GPU's memory for quick access.



Figure 24 Roadmap Rendering

Figure 24 shows an example of this roadmap where the road-domain geographic location is divided into grid to better illustrate this concept. To get the road-domain location of an object on the screen, the system uses its screen-domain position to fetch its road-domain location, as described in the following code example:

```
struct RoadInfo {float x, y};

struct RoadInfo road[height][width]; //Road-domain location look-up table

struct RoadInfo getRoadLocation(float posx, float posy) {
    return road[y][x];
}
```

At this point, the algorithm has measured the speed of each pixel on the lower edge of an object. The last step is to group the speed of individual pixels to the speed of an object.

3.7 Summary

This chapter has presented the various stages in the proposed algorithm as well as some of the challenges when creating it. It performs image differentiation for blob detection, blob grouping into objects, object tracking, object displacement, and finally object speed detection. Each stage is designed to use minimal computation resources so that it can be easily implemented in a variety of platforms.

CHAPTER 4

Implementation Platform Comparison and Constraints

4.1 Comparison of Different Implementation Platforms

This section details the advantages and disadvantages of each possible implantation platform.

4.1.1 Using a CPU

Generally, the most conventional way to implement any software type algorithm is to use CPU as it provides the maximum flexibility for a programmer. It is a general-purpose processor designed to handle almost any sequential algorithm. Modern CPUs are highly optimized to operate at extremely high speeds and lower power. The unit cost of CPU is incredibly low which makes it an excellent choice to implement algorithms in the most cost-effective way.

One of the major limitations of a general CPU is that it is capable of handling only one task at a time per core. “Threading”, a hardware/software solution for improving performance, is limited based on the algorithm being implemented as some may gain from this and others may lose. Threading performance is also limited by the features provided by the CPU as some are better than others at this. In general, to increase the throughput of CPU, one way is to use multi-core CPU, another way is to utilize the vector calculation unit with SIMD (Single Instruction Multiple Data) instructions, however some compilers do not do this work automatically.

Multi-core CPUs simply duplicate the circuit of CPU core which can increase the throughput of the CPU but introduces other issues, such as synchronization between CPUs and memory management. Multi-threaded programs suffer from issues such as data racing and locks. Increasing CPU core count will increase the unit cost but will not increase the throughput by “ n ” as not all tasks are easily parallelizable.

Using vector calculation unit (with SIMD instructions) can increase the throughput of the CPU and has negligible impact on the cost. SIMD (SSE on x86 and NEON on ARM) allows single instructions to work on larger sets of data. On the CPU wafer, the instruction decoder takes a huge amount of area; but the actual data path only consumes a small area. Each core of a modern CPU comes with a either a 64 or 128-bit vector calculation unit.

In the field of imaging processing, a common operation is a kernel filter, which is a dot product of two matrix of size x -by- y . It requires reading the target pixel and its

neighboring pixels, multiplying each pixel by a constant value called kernel (or mask), summing all of them together, and writing the sum back to that pixel. A x -by- y kernel filter can be described using the following formula:

$$P = \sum_{i=0}^{x*y} p_i \cdot m_i$$

Where P is the result of the kernel filter, p_i is the pixel and its neighbors, m_i is the mask applied to each pixel.

A classic example of kernel filter is the Gaussian blur, where the kernel averages the value of the target pixel and its neighbor pixels. This filter is used to remove high-frequency signals such as noise or details of the image. Another example is edge detection, which is used to enhance the high-frequency signals (i.e. the edges of objects in the image).

For example, a 1280-by-720 pixels video generates 921,600 pixels each frame. To apply a 3-by-3 filter on one frame of the video will require the CPU to spend 10 cycles per pixel (9 for a load-and-multiply-accumulate, and 1 for a store). If the CPU is clocked at 1 GHz and can perform memory access in one single clock, a 3-by-3 kernel filter can achieve a maximum processing rate of 108 FPS. For a 5-by-5 kernel, 41 FPS. Neither of these scenarios account for any other overhead that is encountered during the processing, such as system overhead, bus congestion, cache miss, I/O access latency, etc. The actual FPS will be far lower than these values when running in real world situations. Taking advantage of SIMD instructions will certainly improve the computational time, but there is overhead in preparing the information prior to using said instructions.

4.1.2 Using Custom Hardware with PLDs (Programmable Logic Devices)

A hardware designer can define their own logic that fits their application need, instead of using the fixed general-purpose data path in a generic CPU. Custom hardware can achieve significant speed increases as it can be highly parallelized, but at the cost of higher development times and historically higher implementation costs. However, modern advancements in programmable logic devices have made it far more cost effective as PLDs and FPGAs (Field Programmable Gate Arrays) have become larger, faster, and cheaper. Most modern devices include hard-core components such as RAMs, ALUs and in some cases full CPUs.

In general, there are less investments in FPGA technology than CPU technology, making the unit cost of FPGA generally higher. CPUs are generally easy to program whereas FPGA programming requires a higher-level understanding of circuit design which makes

the overall development cost of FPGAs much higher. As such, the FPGA platform is not economically favorable.

4.1.3 Using a GPU

GPU (Graphic Processor Unit) is a vector processor mainly used to accelerate 3D graphics rendering. It removes certain functionality that is commonly seen on generic CPUs such as function calls & stack operations, but provides higher throughput by using multiple data paths, and special application specific units such as video encoders & decoders.

Since the rise of the home video gaming industry, consumers have demanded more and more 3D image processing power, which translated to a tremendous investment into GPU development over the years. Overall, GPUs are a good choice for high performance processing, and relatively low unit costs.

Historically, GPUs used to be for graphic purposes only as they had a fixed pipeline designed for 3D graphics (OpenGL 1.0 era). In recent years, customizable graphic render requirements and new highly parallel computation-intensive technologies such as neural networks, deep learning, and AI (artificial intelligence), have gained increasing popularity. The industry result was GPGPU (General Purpose GPU) which gives graphic application developers the ability to define their own render pipeline, as well as computation application developers the ability to define their own algorithms for non-graphic applications.

For embedded applications, one example of a GPU is the VideoCore GPU found in low-cost ARM processors. In this thesis, the algorithm is implemented on a Raspberry Pi 4; it contains a quad-core ARM A72 CPU with an integrated VideoCore 6 GPU, however, there is no publicly available document for VideoCore 6. A VideoCore 4 [13] will be used as reference here for discussion.

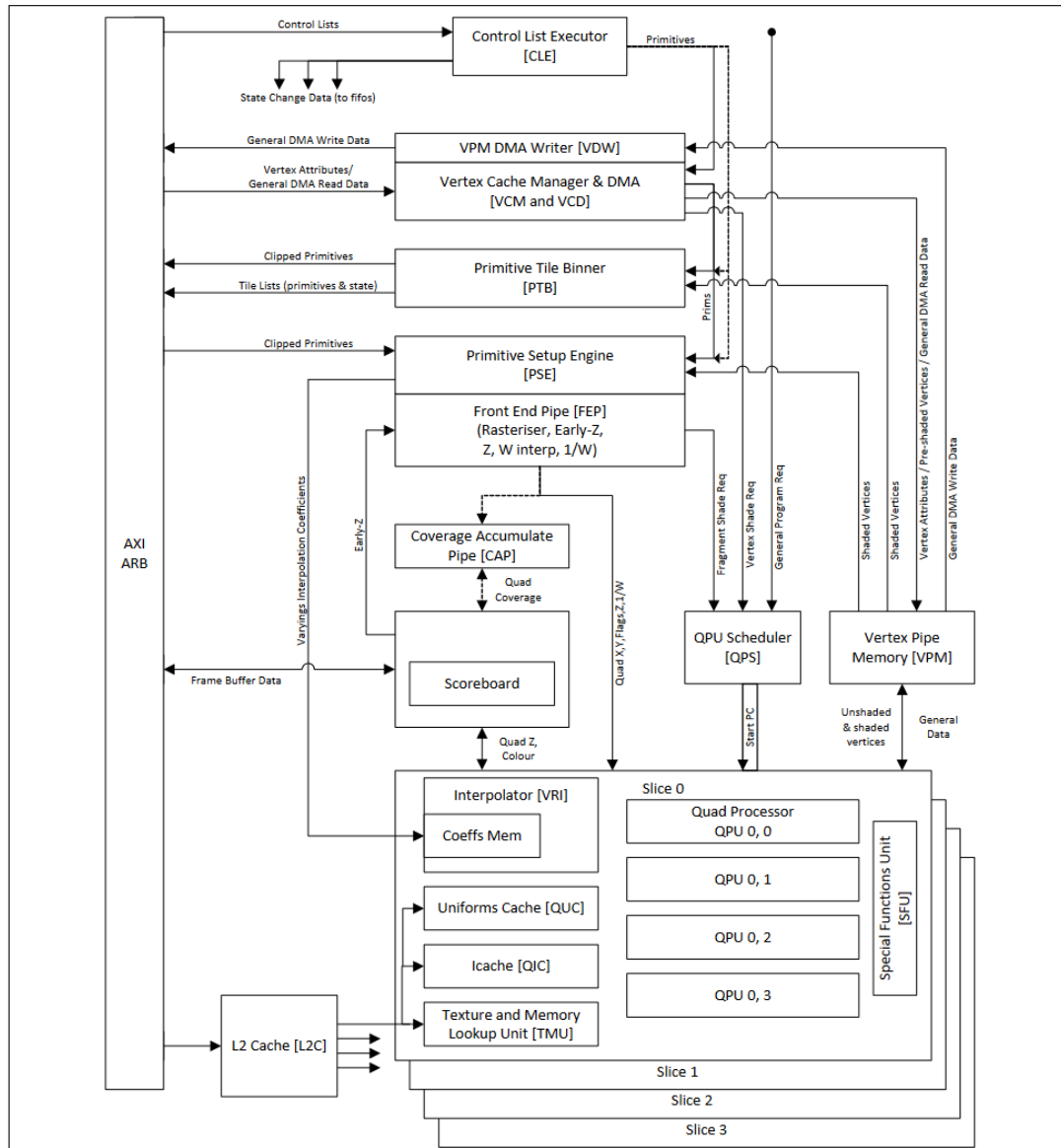


Figure 25 VideoCore 4 Architecture Overview [13, p. 13]

Figure 25 details in the internals of the VideoCore 4 GPU [13, p. 13]. The GPU interacts with the CPU via memory mapping with the AXI bus. There is a DMA controller, scheduler, memory controller and execution controller which feed data to slices (0-3) and control the actual operation of the slices which are responsible for the main computational work.

Each slice includes 4 QPUs (Quad Processor), a SFU (Special Function Unit), and a few other controllers and interfaces. A QPU is a SIMD processor that is used for general

computation such as addition, multiplication, and bitwise operations. The SFU is used for “exotic” operations, such as log, square root, and exponents [13, p. 23]. They also include a uniform cache, instruction cache, one or two TMUs (Texture and Memory Lookup Unit) and a varying interpolation unit.

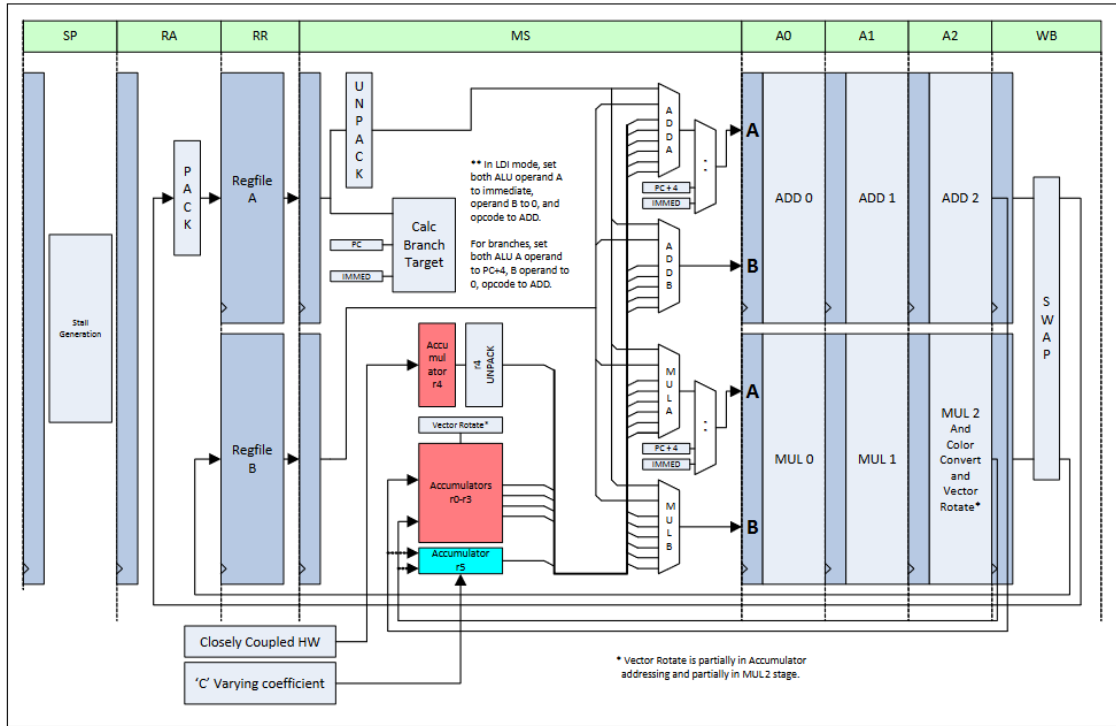


Figure 26 VideoCore 4 QPU [13, p. 17]

Figure 26 shows the details of the QPU pipeline. It is a 64-bit 16-way SIMD processor with two ALUs (one for multiplication and one for addition) [13, p. 16]. Each ALU has access to 32 registers, (64 total registers for holding temporary data). Two parallel ALUs are presented, one is for addition and bitwise operations, another one is for multiplying. This is useful for MAC (multiply-accumulate) and MAD (multiply-add) operations. A stall generator is presented in the pipeline to prevent any memory breach. This GPU can manipulate multiple data elements in parallel, however, there are limitations:

- Some hardware is shared between multiple data paths in the QPU. As such, there will be stalls when there are multiple QPUs needing to access the TMU or SFU [13, p. 25].
- Although a QPU can perform operations on multiple small pieces of data at the same time, the operation performed on all pieces of small data is the same. For example, the QPU cannot perform 32-bit addition and 32-bit ANDing at the same time.

- The GPU architecture is relatively simpler than that of CPU and therefore not only lacks the ability to execute some instructions seen on CPU, but also lacks the advanced abilities seen on CPU, such as dynamic out-of-order execution and speculation. When there is any hazard, the GPU will stall [13, p. 34].
- Data allocation cannot be performed by GPU; it must be allocated by CPU before the GPU can use it. The GPU also has no stack, so it cannot perform subroutine calls. All routines in shader language will be flattened during compiling.
- The GPU “driver” running on the CPU side must address the above issues prior to sending the data and commands to the GPU.

4.1.4 Choice of Platform

As discussed above, there are three different potential platforms that may be used to implement the proposed algorithm. Since the application requires large throughput, a low-cost CPU solution would be far too slow for this algorithm. Although an FPGA satisfies the performance requirement, the unit cost and development cost are too high. The middle ground solution of a GPU provides adequate throughput for this application, as well as remaining relatively low unit cost.

The popular Raspberry Pi 4 will be used to implement this algorithm as it is based on a quad-core ARM A72 CPU with an integrated VideoCore 6 GPU. Unfortunately, at the time of writing, there is no publicly available documentation for VideoCore 6, however, it is assumed that it is a natural progression of the VideoCore 4 and will not be radically different.

4.2 Comparison of Different GPU APIs

There are vendor specific APIs (CUDA and DirectX) and open APIs (OpenCL, Open GL, OpenGL ES, and Vulkan) which can be used to execute a program on a GPU. Each has their advantages and disadvantages.

4.2.1 Vendor Specific GPU APIs

Vendor specific APIs are designed for a specific model of GPU or a specific series of GPUs. It can be used to empower proprietary computation modules in a GPU hence increasing the performance of the implementation. For example, Nvidia CUDA is an API that can be used on certain Nvidia GPUs for parallel computing acceleration but cannot be used on other GPUs from other vendors. While DirectX will operate on Nvidia and other hardware, it is limited to Microsoft platforms only (Windows, Xbox). Some APIs are compatible with different hardware and software platforms but are attached to

specific software licensing. Vendor specific support often deprecates quickly over time as newer hardware is introduced.

Given that the proposed algorithm is intended to operate on low-cost devices, using vendor specific APIs is a poor choice.

4.2.2 OpenCL

OpenCL (Open Computing Language) *is an open, royalty-free standard for cross-platform, parallel programming of diverse accelerators found in supercomputers, cloud servers, personal computers, mobile devices, and embedded platforms. OpenCL greatly improves the speed and responsiveness of a wide spectrum of applications in numerous market categories including professional creative tools, scientific and medical software, vision processing, and neural network training and inferencing.* [14]

OpenCL was designed to be used for general purpose computation situations which are a suitable option for the proposed algorithm, however, most low-end GPUs commonly found on embedded devices do not support OpenCL.

4.2.3 OpenGL

OpenGL (Open Graphic Library) *is the most widely adopted 2D and 3D graphics API in the industry, bringing thousands of applications to a wide variety of computer platforms. It is window-system and operating-system independent as well as network-transparent. OpenGL enables developers of software for PC, workstation, and supercomputing hardware to create high-performance, visually compelling graphics software applications, in markets such as CAD, content creation, energy, entertainment, game development, manufacturing, medical, and virtual reality. OpenGL exposes all the features of the latest graphics hardware.* [15]

The main purpose of OpenGL is for rendering 2D and 3D graphics. The first OpenGL version only provides a set of fixed rendering pipelines that are dedicated to 2D and 3D rendering. Later, in 2004, OpenGL 2.0 officially introduced the OpenGL Shading Language (GLSL) [16, p. 2]. This gave developers the possibility to define their own shader language to precisely control the 2D and 3D rendering process. Although the main purpose of GLSL was to define custom 2D and 3D rendering processes, it was used to define general purpose computation programs as well. Input data was sent to GPU shader program as vertices and textures, and computation results saved in framebuffer were read back. In 2012, OpenGL 4.3 introduced compute shaders that was more designed for general purpose computation. Instead of using vertices, textures, and framebuffers to transfer data, now developers can use SSBO (shader storage buffer object) to perform computation in the compute shaders. Compared to the prior methods, the compute shader

eliminates the need of performing computations separately in the vertex shader and fragment shader and the reduces the overhead of graphic related operations. The introducing of custom GLSL made general purpose computation possible on OpenGL while the introduction of compute shaders reduces the general computation overhead. Most importantly, OpenGL has the highest compatibility rate among all other APIs.

4.2.4 OpenGL ES

OpenGL ES (Open Graphic Library for Embedded System) is a subset of OpenGL intended for embedded systems. The limited space, power, and cost constraints have reflected on the hardware available for full OpenGL compatibly, so this subset was created. For example, the latest OpenGL ES standard (3.2 from 2015) requires all GPUs to support single precision floating-point number [17, p. 7] and a minimal 2D texture size of 2048 x 2048 [18, p. 486] ,however the OpenGL 4.5 standard (from 2014) requires all GPUs to support both single and double precision floating-point numbers [19, p. 9] and a minimal 2D texture size of 16384 x 16384 pixels [20, p. 614].

As such, embedded GPUs and OpenGL ES should be chosen for lower cost and complexity applications.

Note that the versioning of OpenGL and OpenGL ES do not line up as they are different standards.

4.2.5 Vulkan

Vulkan is the most state-of-art graphic API designed to replace OpenGL and OpenGL ES. Vulkan provides direct interface between the user program and the GPU hardware reducing the interfacing overhead and resulting in higher performance compared to OpenGL and OpenGL ES. As with any new approach to solving a problem, Vulkan is focused on the higher end desktop systems, and consequently there is limited current support for embedded systems.

4.2.6 Choice of API

Given that the intended platform is to be an embedded system, the best choice is OpenGL ES as it is currently available on a variety of embedded CPUs. Vulkan is a potential second choice, but there may be issues with driver availability.

4.3 Optimizations for OpenGL ES GPUs

There are several algorithms that can measure object speed using high performance GPUs, but these do not scale well on embedded GPUs. The proposed algorithm is

designed by examining and mitigating the limitations of these GPUs. This section briefly covers some of these limitations.

4.3.1 Hazards

Unlike serialized CPUs, GPU architecture is generally simpler but with greater parallelism. In CPUs, advanced strategies implemented in hardware compensate for the potential performance loss when hazards are encountered. In GPUs, maximizing the number of parallel cores is prioritized, and space cannot be wasted on such circuits, as such a GPU will simply stall in case of a hazard which effects performance [13, p. 34]. Although the GPU driver will attempt to optimize the shader code to avoid such hazards, skillful coding can significantly help as well.

Branch instructions contribute significantly to control hazards, and as such, they should be avoided, but this can be difficult. In some cases, an arithmetic operation can be used to replace the branch operation completely. Consider the following example of a conditional addition using a general GPU (with data forwarding between subsequence execute stages) but lacks any branch prediction capability.

```
; if (r3 < 500) r1 += r2;

use_branch:
cmp    r3, 500 ; Compare r3 with value 500
bgeq   PC+1    ; branch if greater or equal (skip next instruction)
add    r1, r2   ; r1 += r2

no_branch:
sub    r3, 500 ; Subtract r3 - 500; If r3 < 500, will overflow: MSB <- 1
                    ; Otherwise, MSB will be 0
msr    r3, 31  ; Mathematical shift right: extend MSB. Result = 0xFFFFFFFF
                    ; if overflow (r3 < 500), 0 if not
and    r3, r2   ; Bit-wise ANDing, get r2 if previous result is 0xFFFFFFFF
                    ; get 0 if previous result is 0x00000000
add    r1, r3   ; r1 += r3, r3 = r2 if r3 < 500, r3 = 0 otherwise
```

Instr.	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7	Cycle 8	Cycle 9
cmp	fetch	decode	execute	writeback					
bgeq		fetch	decode	execute	writeback				
add						fetch	decode	execute	writeback

Figure 27 GPU Control Hazard with Branch

Figure 27 shows the “use_branch” instruction code in the GPU pipeline (assuming data forwarding is enabled). The pipeline stalls after the **bgeq** (branch if greater or equal) instruction as it determines address of the next instruction.

Instr.	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7	Cycle 8	Cycle 9
sub	fetch	decode	execute	writeback					
msr		fetch	decode	execute	writeback				
and			fetch	decode	execute	writeback			
add				fetch	decode	execute	writeback		

Figure 28 GPU Control Hazard without Branch

Figure 28 shows the alternate “no_branch” instruction code in the GPU pipeline. Although this code executes more instructions than the previous method, it never suffers from any stalls as no branches are introduced in the pipeline.

One of the most common situations coding operations which potentially produces many branch conditions is the “for” loop, however this is often mitigated by “loop unrolling”. Some OpenGL drivers allow developers the ability to unconditionally instruct the compilers to unroll loops, and some do not. Given that there is no standardization on this feature, it is often best for the developer to manually unroll the loops. [21]

A common operation used in image processing is kernel filter which adds multiple pixels with some weight applied. This operation can be described using the following C code:

```
typedef struct Pixel {
    float r, g, b, a;
} Pixel;

void use_branch() {
    Pixel p = {0, 0, 0, 0};
    for (int i = 0; i < 8; i++) {
        p += buffer[i] * mask[i];
    }
}

void no_branch() {
    Pixel p = {0, 0, 0, 0};
    p += buffer[0] * mask[0];
    p += buffer[1] * mask[1];
    p += buffer[2] * mask[2];
    p += buffer[3] * mask[3];
    p += buffer[4] * mask[4];
    p += buffer[5] * mask[5];
    p += buffer[6] * mask[6];
    p += buffer[7] * mask[7];
}
```

The “`use_branch()`” method uses a standard for-loop to load the pixel from the buffer, and the index `i` is checked in every iteration which causes multiple stalls. Whereas the “`no_branch()`” method manually unrolls the for-loop to eliminate the branch operation which hence eliminates the stall and maximizes the processing throughput.

Although loop unrolling eliminates the branch operations, it increases the code size and for larger loops this may decrease performance as the instruction cache may be exceeded causing possible cache misses.

4.3.2 Data storage size

Minimizing data representations will often result in maximizing data throughput. For example, 64-bit SIMD processors can compute one 64-bit value, or two 32-bit values, or four 16-bit values, or eight 8-bit values at the same time. A compromise in precision can lead to substantial improvements in throughput.

OpenGL ES [17, p. 7] (but not OpenGL [19, p. 9]) allows the developer to define the precision of the data storage using `lowp`, `mediump` and `highp`. In most case, `lowp` represents an 8-bit integer or floating-point number, `mediump` represents 16-bit integer or floating-point number, and `highp` represents a 32-bit integer or IEEE754 32-bit floating-point number.

All OpenGL and OpenGL ES objects require a 4-channel vector data with same type (e.g. `int[4]`, `float[4]`). Since the GPU is optimized for floating point numbers, it makes more sense to use an 8-bit or 16-bit floating-point vector to describe integer data as, overall, it improves the throughput to the system. For this algorithm, 32-bit or 64-bit floating point numbers are not required as the 8-bit and 16-bit floating point numbers provide sufficient precision.

4.3.3 Asynchronized Operation

OpenGL and OpenGL ES are logically synchronized (i.e. the command and data send to GPU are executed and manipulated in order), however, the underlying communication between the GPU and CPU are asynchronized (i.e. the GPU may lag the CPU calls). Asynchronization is used to prevent the GPU and memory controller from stalling the CPU, hence the GPU and CPU can perform other tasks instead of waiting on each other.

As the OpenGL wiki states: *The OpenGL specification usually defines the behavior of commands such that they will execute in the order given, and that every subsequent command must behave as if all prior commands have completed and their contents are visible. However, it is important to keep in mind that this is how OpenGL is specified and behaves, not how it is implemented.* [22]

Data is uploaded to the GPU by the driver which places it in queue, and when the GPU is idle a DMA is initiated. GPU commands are similarly executed by placing the command in queue, and then executed in order as previous commands are completed. Each of these functions are non-blocking (i.e. return immediately).

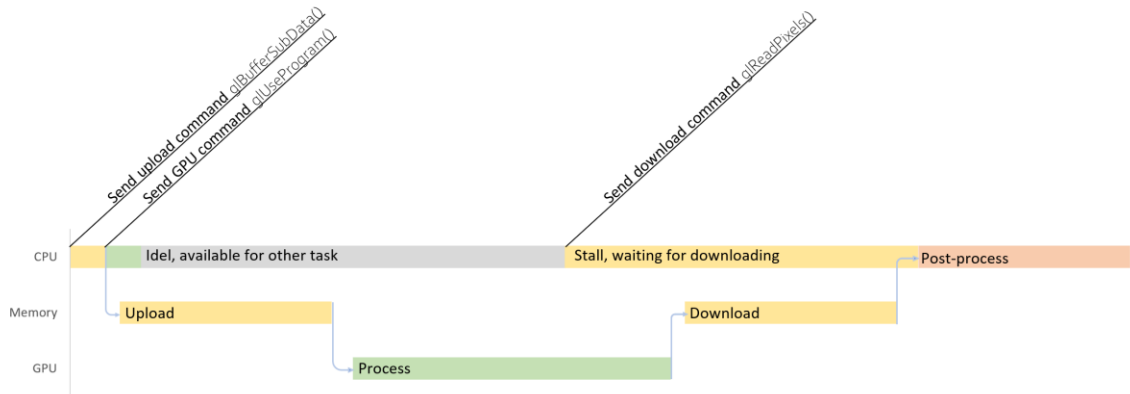


Figure 29 OpenGL Asynchronous Operation

Figure 29 shows an example of the overall workflow when uploading data, executing an operation in the GPU, and downloading back for future processing.

In this example, CPU calls the `glTexSubImage()` or `glBufferSubData()` functions to place data in the uploading queue, and then calls `glUseProgram()` to invoke the shader program to run on the GPU. Neither of these functions are blocking so the CPU may proceed onto other tasks. On the GPU side, the shader program will be blocked until the DMA is completed.

At some point, the CPU will want to download the processing results from the GPU, this can be done by calling the blocking `glReadPixels()` function, which will stall the CPU and wait for the GPU to completely finish. Other blocking functions like `glFinish()` or `glClientWaitSync()` can be used to force the CPU to stall until all previous GPU commands are fully executed, however, frequently explicit synchronization has negative impact on overall performance.

4.3.4 Task Management

When using a GPU in a CPU host, poor inter-processor communication can result in significant performance loss. Uploading data from the CPU side to the GPU side and vice versa can take time.

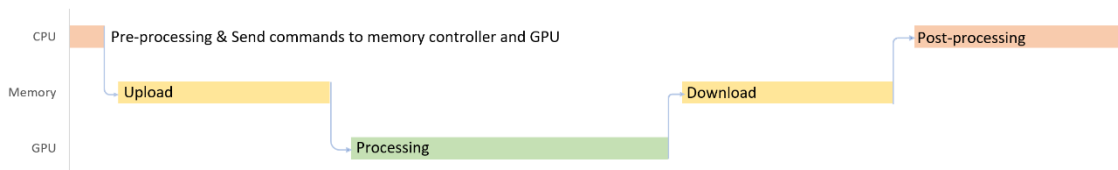


Figure 30 Unoptimized Program CPU-GPU Memory Transfer

Figure 30 shows an example of how an unoptimized CPU-to-GPU-to- CPU program may run. The 5 discrete stages are separated by the asynchronous CPU-to-GPU transactions, so each previous stage must be fully completed before moving on to the next:

- 1 The CPU pre-processes the data.
- 2 The memory controller uploads the data from the CPU side to the GPU side.
- 3 The GPU processes the data, hence step 2 must be complete.
- 4 The memory controller downloads the data from the GPU side to the CPU side.
- 5 The CPU post-processes the data.

During these stages, only one device is active while the other two are idle. This results in long stall times, and low utilization in both CPU and GPU; hence, the program has low performance and throughput.

Device	Early Stage	Later Stage
CPU	Pre-process data[t+1] and GPU control	Post-process data[t-1]
Memory	Download data[t-1] and Upload data[t+1]	
GPU	Process data[t]	

Figure 31 Optimized Program CPU-GPU Memory Data

To minimize idling and increase throughput, tasks must be carefully managed. Figure 31 shows an example of a more optimized workflow with two primary stages, the early stage, and the later stage. During the early stage, the CPU can pre-process the next data segment, and the memory controller can download the previous data segment from the GPU side memory to the CPU side memory. Once completed, the previous data segment is available in the CPU memory, and the CPU has pre-processed the next data segment, the program enters the later stage. Now, in the later stage, the CPU can post-process the previous data segment, while the memory controller can upload the previous data segment. This optimization eliminates stalls by keeping the CPU, the GPU, and the memory controller fully utilized; hence, increases the throughput of this program.

4.3.5 Double Buffering

Double buffering is a general scheme that is frequently used to eliminate stalls caused by back-to-back task management. It uses a pair of identical sized buffers (one called the front buffer, the other called the back buffer) that allows task A to process the front buffer, and task B to process the back buffer. After both processes are complete, the buffer pointers swap so that in the next iteration task A will process the back buffer (which contains the result from task B) and task B will process the front buffer (which contains the result from task A). This technique can be applied to more buffers depending on the number of resources available to perform the necessary tasks.

Double buffering can be easily applied to the GPU by manually creating these two buffers. The GPU will perform operations on the front buffer while the memory controller updates the back buffer. When updating the back buffer, the buffer on the GPU-side should be mapped to a memory space on the CPU-side using the `glMapBufferRange()` function. When invoked with the `GL_MAP_UNSYNCHRONIZED_BIT` flag, the OpenGL driver will assume the developer has performed all the necessary synchronization; hence it will not perform any internal synchronization.

4.4 Implementation Details

The following section discusses the OpenGL ES and GPU implementation details.

4.4.1 Task Scheduling

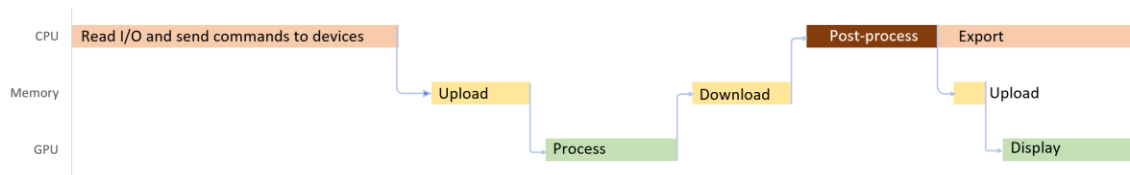


Figure 32 Unoptimized Algorithm Workflow

Figure 32 shows the unoptimized workflow of both the GPU and the CPU in this algorithm.

- 1 The CPU reads and pre-processes the raw video data from the video source. The pre-processing includes changing the color format and aligning the data. The data provided by video device can be any arbitrary format such as BGR and YUV; however, OpenGL yields the highest performance when the data is in RGBA format (4 word aligned). This pre-processing stage converts data into this format and saves it in a TBO (Texture Buffer Object).
- 2 The memory controller uploads the TBO from the CPU side to the GPU side.
- 3 The GPU processes the data and renders the result into an FBO (Framebuffer Object).
- 4 The memory controller downloads the FBO from the GPU side to the CPU side.
- 5 The CPU post-processes the FBO to extract the vehicle speed data.
- 6 The CPU exports the results to an external destination such as database, and the GPU may display the result on screen in the development environment.

Without any optimizations, the algorithm has poor resource utilization. At any given time, only one of the CPU, GPU, or memory controller is working, leaving the other two idling. To increase the utilization, a processing pipeline is used to stream the data from the CPU-side to the GPU-side through the memory controller to keep them fully utilized.

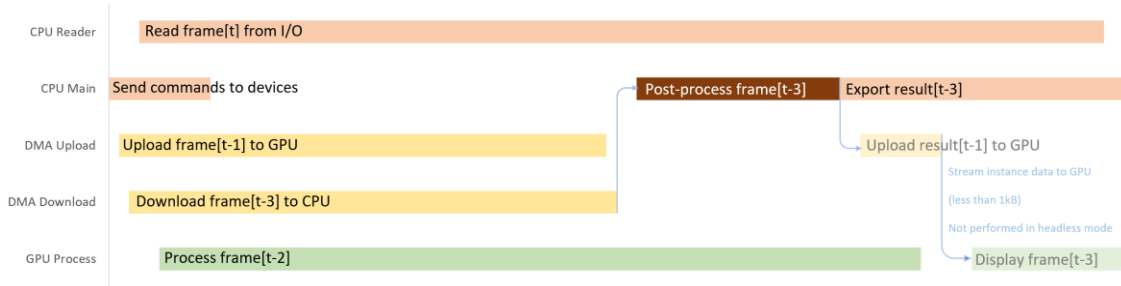


Figure 33 Optimized Algorithm Workflow

Figure 33 shows the optimized workflow where two CPU threads, the GPU, and the memory controller are processing different frames of the incoming video.

- The CPU reader thread reads and pre-processes the raw video data from the I/O device. Since the I/O device is almost always slower compared to the CPU, reading the I/O device will generally block the CPU thread. To solve this, a dedicated reader thread reads the video data from the I/O device. When using a live camera as an input device, the camera requires the entire frame time (`frame[t]`) to scan the sensor and stream the video data.
- Early in the frame window, the CPU main thread sends commands to GPU which require only a fraction of the processing window time. Later, the CPU main thread will post-process the data of 3 frames earlier (`frame[t-3]`).
- While CPU and GPU are processing the frames, the memory controller will be uploading `frame[t-1]` and downloading `frame[t-3]`.
- The GPU is processing the data uploaded by the memory controller in the previous frame, which is the data from 2 frames earlier (`frame[t-2]`) which requires the entire frame window.

4.4.2 Individual Stages

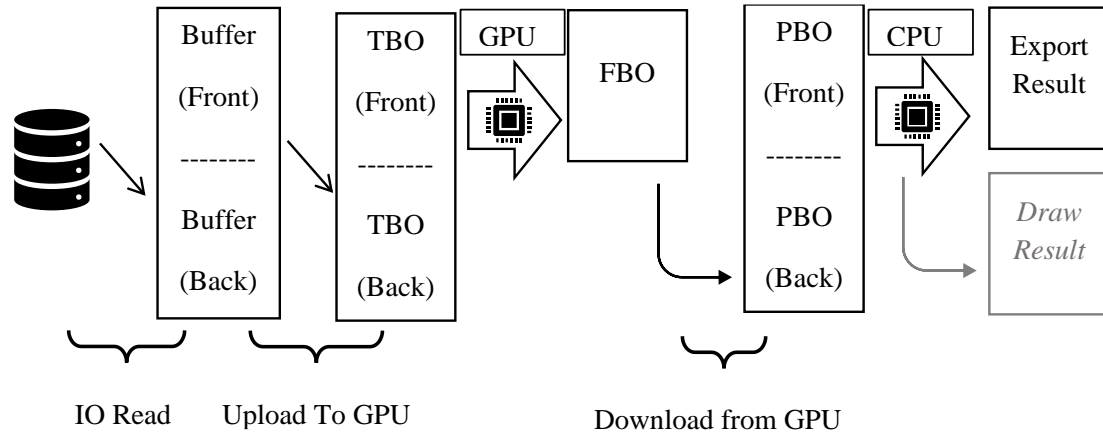


Figure 34 Task Scheduling Overview

Figure 34 shows a general overview of the data flow between processing blocks. Following is a detailed description about the memory management used in this algorithm in each stage.

Stage 1: Read the Raw Video Stream

Reading data from I/O (the camera) is slower than the CPU as the video stream must send a complete whole frame in a fixed interval of time. Waiting for this data will block the process and stall the whole program. This problem can be solved by using non-blocking I/O (process doesn't stop on system calls) or asynchronized I/O (a callback routine is used when data is available), or by a dedicated thread.

For this algorithm, both the double buffering scheme and an additional reader thread is used to perform I/O operation while the main thread is processing data and controlling OpenGL. For the double buffer, the front buffer is used by the main thread, the back buffer is used by the reader thread.

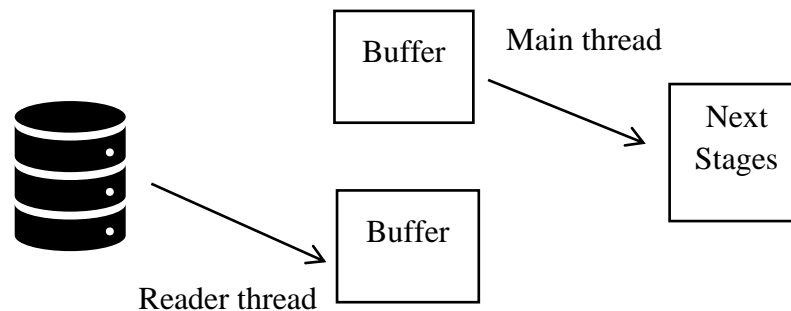


Figure 35 Memory Management Reader Buffer 1

Figure 35 shows the double buffer layout. At the beginning of each frame, the main thread will pass the address of the back buffer to the reader thread, so that it can begin to read raw video data in. Meanwhile, the main thread is working on the front buffer.

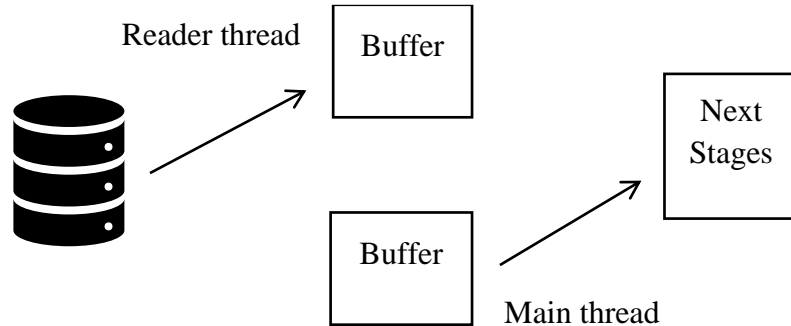


Figure 36 Memory Management Reader Buffer 2

At the end of each frame, the main thread will first wait for the reader thread to finish the filling of back buffer. Once done, the main thread will swap the front and back buffer, as shown in Figure 36. The back buffer of the current frame now becomes the front buffer of the next frame as this buffer contains new content, which can be uploaded to GPU in the next frame. The front buffer of current frame becomes the back buffer of next frame which contains data already uploaded to GPU and can now be discarded. In the next frame, this buffer will be used by the reader thread for new data filling.

Stage 2: Upload Data from the CPU to the GPU

There are two ways to perform the data uploading to GPU. However, there is no significant performance difference between the two. This is probably the case as the size of each frame is small (3.6M pre frame for 720p RGBA8). Both are described below as the performance may change if larger images sizes are used, or if there is some other architectural change in the system.

The conventional memory space (heap) method uses memory allocated by the `malloc()` system call shown in Figure 37.

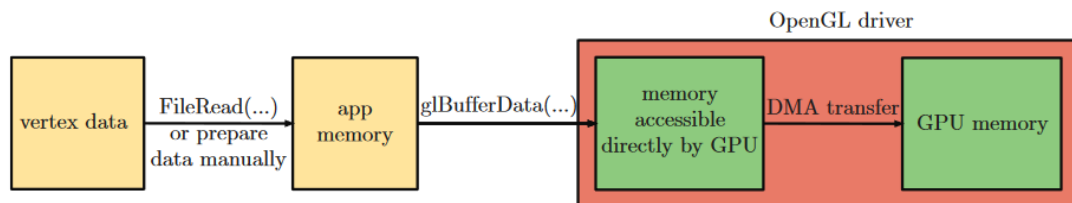


Figure 37 Conventional Upload [23, p. 394]

Here the reader thread will upload the data into the program's address space. Then, a `glTexSubImage()` or `glBufferSubData()` call in the main thread will copy the

content to a pinned memory in OpenGL driver's memory space that is directly accessible to the GPU. Note that, this pinned memory space is on the CPU side; the data is not uploaded to the GPU yet. A DMA controlled by OpenGL driver and the hardware will later perform the actual uploading to GPU [23, p. 393].

The OpenGL PBO (Pixel Buffer object) uses a pinned memory space as shown in Figure 38.

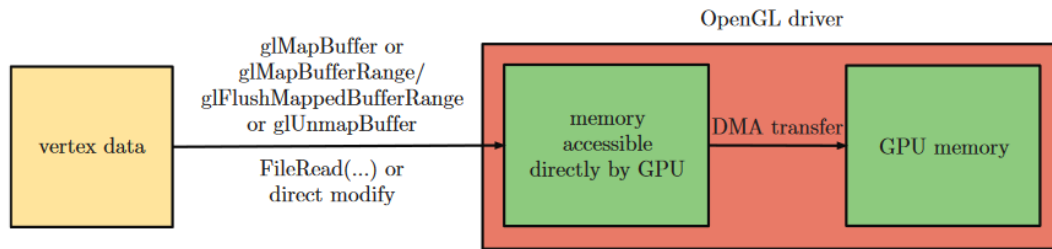


Figure 38 PBO Upload [23, p. 394]

Here the main thread will request the OpenGL driver to map a PBO to a CPU-side GPU-accessible pinned memory by a `glMapBufferRange()` call. Then, the reader thread can read data directly into this pinned memory. Lastly, a `glUnmapBuffer()` call followed by a `glTexSubImage()` or `glBufferSubData()` call by the main thread will allow the DMA controller to transfer the data from PBO (memory on CPU side) to actual texture buffer in GPU (memory on GPU side) [23, p. 394]. This method avoids the extra memory copy used in the conventional memory approach.

Stage 3: Receive Data from CPU on GPU Side

Once the video frame data is loaded into the GPU-accessible pinned memory space, the transfer can be initiated, and the GPU starts the actual data processing. This may not happen in a timely manner as the DMA may be doing other work.

Figure 39 shows an example of unoptimized data uploading to the GPU. Here the CPU will initiate the data upload followed by the GPU commands (invoking shader programs). However, there is no free DMA controller at this moment and the GPU ends up stalling.

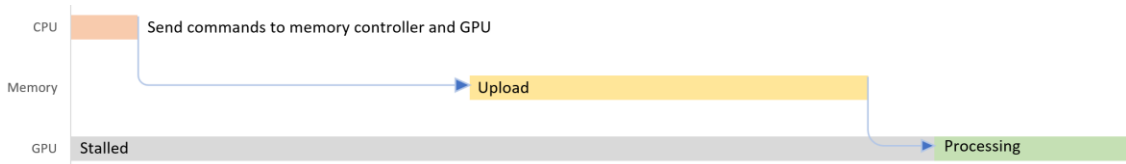


Figure 39 Unoptimized GPU Data Upload

To solve this issue, the double buffer scheme (`texture_orignalBuffer[2]` in the code), shown in Figure 40, is used to store the video data in the front texture for the

actual processing on the GPU side (accessible to shader programs) and the back texture is for the data uploading (accessible to asynchronous DMA).

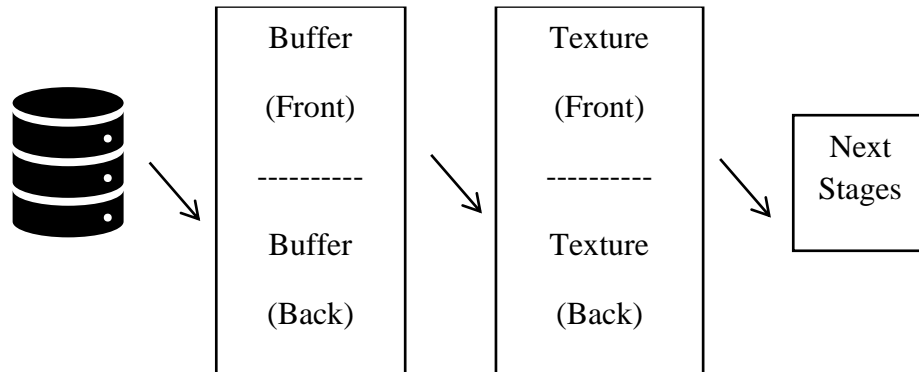


Figure 40 Render Buffer

At the beginning of each frame, the program will upload the video data of the current frame to the back texture. While the OpenGL driver and the hardware perform the DMA on the back texture, the GPU manipulates data of current frame saved in the front texture.

At the end of each frame (after the video data of current frame is ready in the GPU texture buffer, and the GPU has finished processing data of pervious frame) the program will swap the front and back texture. The current frame video data in the back texture becomes front buffer in next frame, accessible to the GPU. The previous frame video data in the front texture can be discarded so the space can be reused for uploading in next frame in the background.

Stage 4: Downloading the Speed Data from the GPU to the CPU

Once the video data has been processed in the GPU, the speed information of the objects found is saved in an FBO (`fb_speed` in the code) in the GPU. Since this is the last stage needed in processing the frame, the CPU processing is no longer needed, so it can be blocked while waiting for the final data from the GPU. Once received, the CPU will build the `speedData` structure and output the results to the file. When the CPU processing is allowed to continue, it will be one frame behind what the GPU processed; see Figure 41.

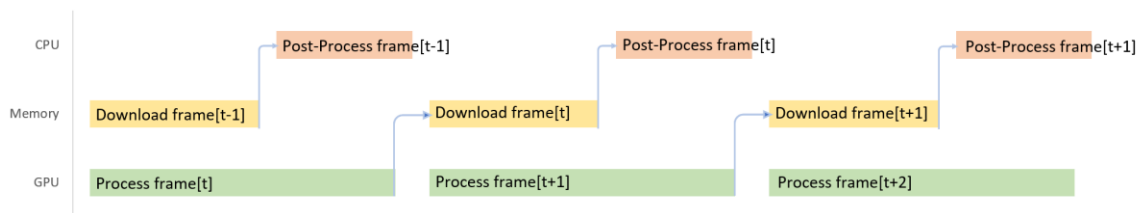


Figure 41 Processing Data Flow

Stage 5: Display Results on Screen

When in development mode, the CPU program will construct a mesh (based on the number of objects) that contains the position and name of glyphs representing the speed according to the speed data in buffer `speedData`. Next, a `glBufferSubData()` call with `STREAM_DRAW` flag is used to upload the VAO which draws the glyphs on the screen.

For graphic programs, including OpenGL programs, the window system normally provides a double buffer for drawing. The front buffer is used by the program for drawing, the back buffer is used by the window system for displaying. When the program is drawing on the buffer, the window system displays the result of the previous frame. After the program finishes the drawing, the window system will swap the front and back buffer (Figure 42).

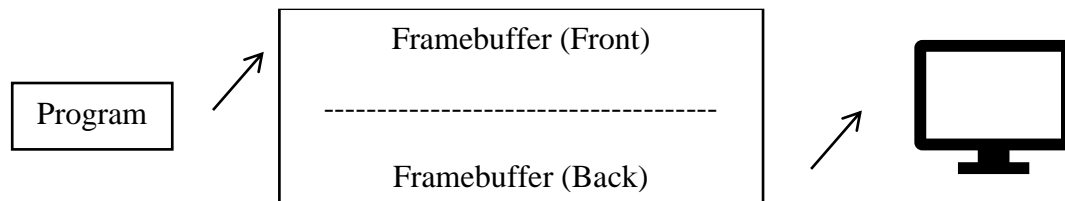


Figure 42 Window System Framebuffer

Using a double framebuffer is beneficial as it allows the program to draw on a memory space that is exclusively for the program instead of window framebuffer that is shared with the display adapter. It also prevents the user from seeing the intermediate result of the drawing (i.e. screen tearing), hence it guarantees a stable image in the displayed window. However, double framebuffer does introduce a one frame of latency.

Stage Summary

Double buffer is heavily used in this algorithm to minimize stalls and improve throughput; however, it increases latency. The result displayed on the screen does not reflect the speed of object of current frame, instead, it reflects the speed of object in a few frames earlier.

Figure 43 shows the data in the processing pipeline. Here `frame[t]` refers to the data for current frame, which is the data just feed in by an IO device; `frame[t-1]` refers to the data from the previous frame.

Operation	Input/Output Frame
CPU reader thread - reading	Frame[t]
DMA 1: CPU to GPU	Frame[t-1]
GPU - processing	Frame[t-2]
DMA 2: GPU to CPU	Frame[t-3]
CPU main thread – post processing	Frame[t-3]
CPU main thread – result display uploading	Frame[t-3]
GPU – draw result in system framebuffer	Frame[t-3]
Window system display	Frame[t-4]

Figure 43 Algorithm Pipeline

4.4.3 Pre-defined Roadmap

As described in previous chapter, this algorithm converts an object's screen-domain position into a road-domain location. Performing this operation on-the-fly would require large computation power using the special hardware SFUs. Since there is only one SFU in each slice in the VideoCore 4 architecture, this becomes a substantial bottom neck.

Since the associated road-domain location of each pixel on the screen never changes, the relationship of pixels between perspective and rectified views also remain the same. Therefore, a lookup table method is more favourable, which can adapt to more complex road surfaces, an example is shown in Figure 44. In this algorithm, a pre-defined lookup table call roadmap is stored in a texture object which stores the associated road-domain location of each pixel on the screen, and the relationship of each pixel between perspective view and rectified view. When this data is required, the GPU will perform a texture lookup using one of the TMUs (VC4 provides one or two TMUs [13, p. 14]) in the slice. When there is no free TMU, the QPU waiting to use the TMU must stall.

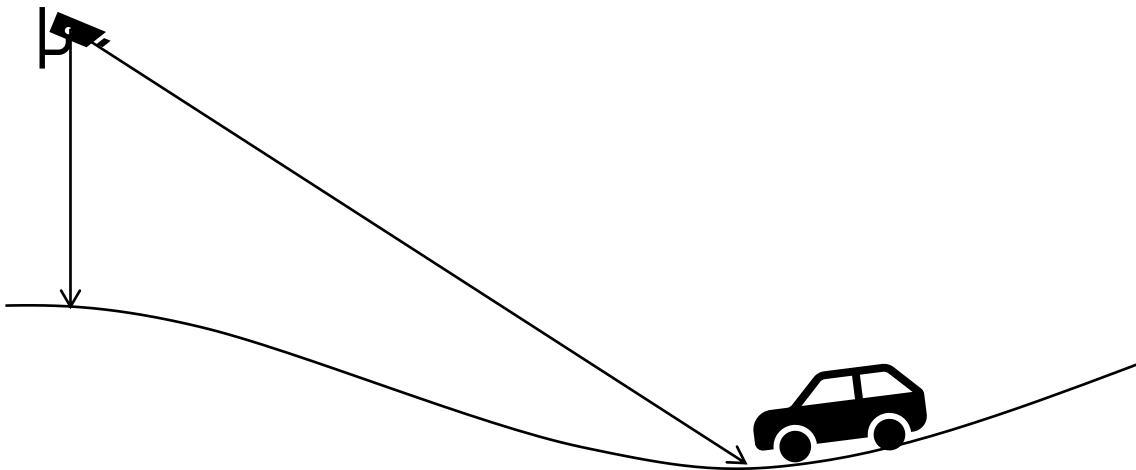


Figure 44 Trigonometry Error on Uneven surface

Using a pre-defined roadmap increases the memory footprint which means cache miss penalty. When the required texture is not presented in the cache, the operation must stall for the hardware to load the required memory into cache. There are two ways to minimize the cache miss rate: to optimize the order of memory access to increase locality, & to use smaller data size so more data can be saved in the cache.

The associated road-domain location of a specific pixel on the screen requires two real values, and the relationship of perspective and rectified view conversion requires two integer values, for example:

```
struct Road {  
    float x, y; //Road-domain coord  
    int p2o, o2p; //perspective - rectified  
}
```

OpenGL and OpenGL ES always use the RGBA format to store a value in a TBO. From OpenGL's perspective, a RGBA color is simply a 4-channel vector. Any types of data, including but not limited to colors, are stored in the RGBA format. If the data has less than four channels, it still consumes the four-channel data size. For the above example, storing these two different data types would require two RGBA entries stored in two textures. A better solution is to use the same data type for all elements, in this case, have the integer rectification and inverse rectification projection data represented as a float. Now, when performing the projection, the system can use NTC (which uses floats) to address pixels instead of absolute pixel address (using integer):

```
struct Road {  
    float x, y; //Road-domain coord  
    float p2o, o2p; //perspective - rectified  
}
```

Since the frame width and height are under 2048 pixels, an 11-bit precision number is sufficient to address any pixel in the frame; therefore, 16-bit floating point is adequate. Furthermore, since data on road (such as road-domain position, width of pixel and perspective/ rectified view projection coordinates) are continuous, it is possible to save data in the roadmap and use interpolation. Doing so can reduce the memory footprint of the roadmap, which reduces the chance of a cache miss. The interpolation is performed by the TMU hardware [13, p. 39]; however, the simple linear interpolation will produce slightly different result than using a full-sized roadmap. When comparing the advantage of lower cache miss and the disadvantage of slightly different data mapping, it seems using smaller roadmap can further boost the performance of this algorithm.

To fetch data from texture or TBO, the `texture()` or `texelFetch()` functions can be used which will read all 4 channels of a pixel. However, some data for this algorithm, such as the moving blob detection result and the speed data, only have one meaningful channel, hence these functions waste bandwidth. OpenGL 3.1 introduces a new function

`textureGather()` which reads one channel of four neighbor pixels while quadrupling the throughput.

CHAPTER 5

Experimental Results

Although the intent of the proposed algorithm is to operate in real time with live camera, it was tested with two pre-recorded videos obtained from a smart phone. The video files were then decompressed, down sampled to 720p resolution at 10 FPS frame rate and saved in a flat RGBA8 file format. These files were fed into the main program, alternatively a live video stream could be used as well with small changes.



Figure 45 Experimental Result - Raw Stage – QEW



Figure 46 Experimental Result - Raw Stage – Hwy3

Two example scenes with multiple vehicles, shown in Figure 45 (QEW, even road) and Figure 46 (Hwy3, uneven road), will be used to demonstrate the results of this algorithm.

5.1 Intermediate Results

5.1.1 Moving Blob Detection

Figure 47 and Figure 48 show the results of detecting moving groups of pixels (blobs) in the scene. Note that this stage does not detect objects, just the moving pixels; so, it may give multiple blobs for each object.

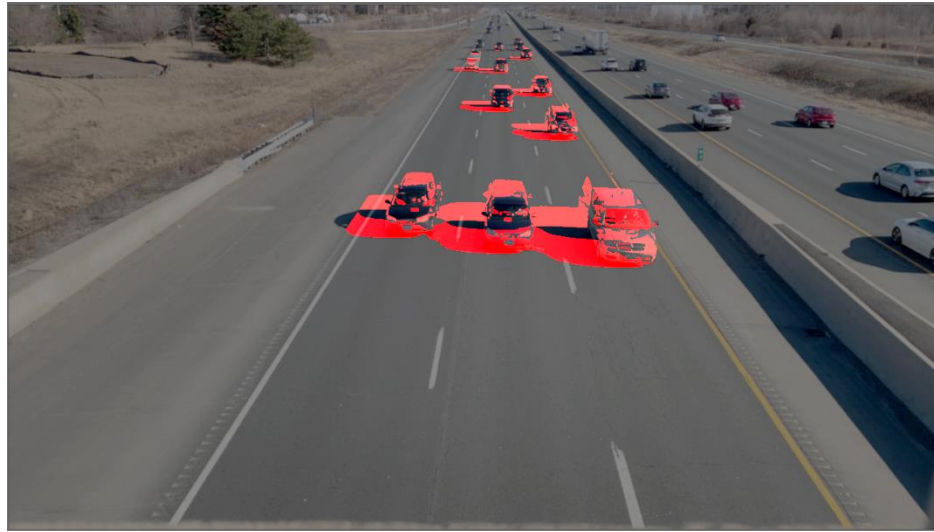


Figure 47 Experimental Result - Moving Detection Stage - QEW



Figure 48 Experimental Result - Moving Detection Stage – Hwy3

5.1.2 Group Blobs to Objects

Figure 49 and Figure 50 show how the blobs were grouped to form individual objects. Note that when multiple objects are close to each other, more than one object may be grouped together.

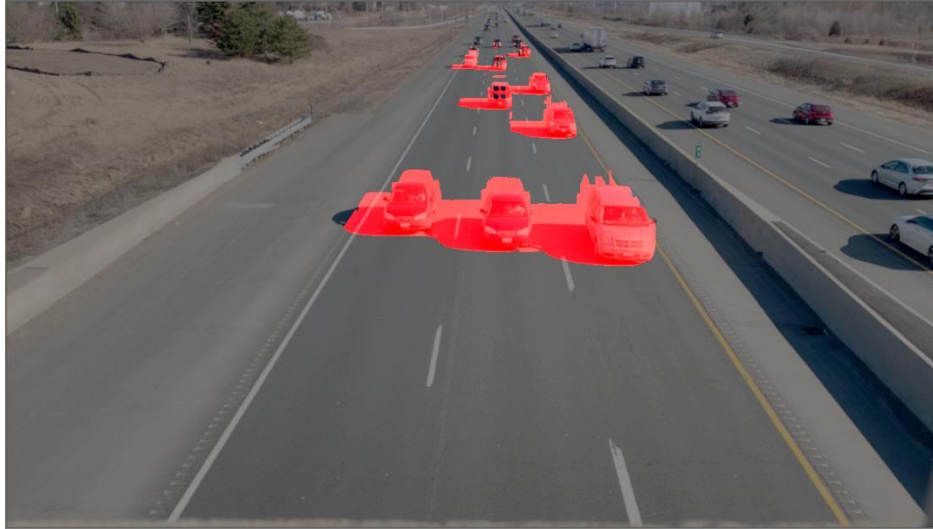


Figure 49 Experimental Result - Moving Detection Fixed Stage - QEW

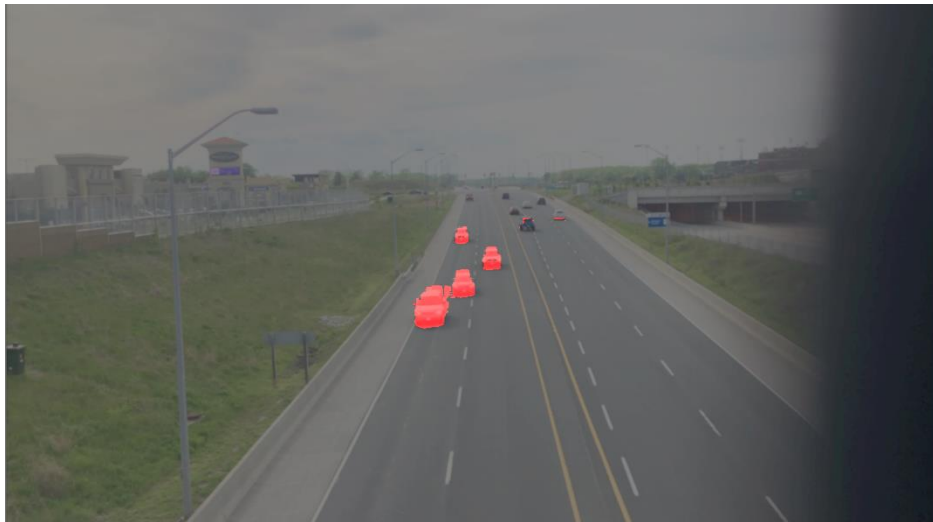


Figure 50 Experimental Result - moving Detection Fixed Stage – Hwy3

5.1.3 Center Lower Edge Detection

Before performing tracking, the algorithm must first detect the lower edge of each object in the scene, then find the center portion of that lower edge; only these objects are tracked. The results are shown in Figure 51 and Figure 52. The objects are classified into four types (`no_object`, `object`, `lower_edge`, and `center_lower_edge`). To

better illustrate the results, the shader program has a development mode which applies distinct colors to the objects for readability, red, yellow, and light red shadows are used to represent the center lower edge, lower edge, and object respectively. Note the “noise” introduced by the shadows in the previous blob grouping stage, which will be classified as false objects as they do not have a proper center lower edge and therefore will not be tracked.

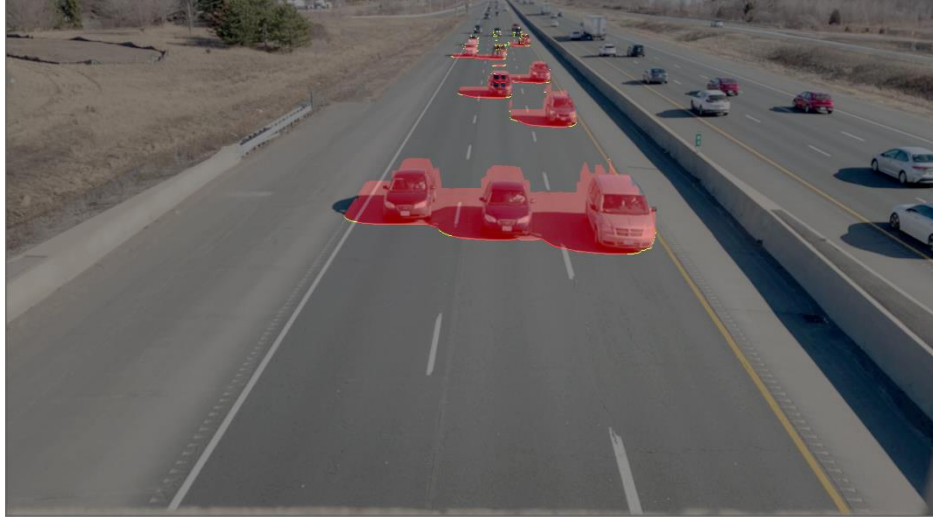


Figure 51 Experimental Result - Edge Detection – QEW



Figure 52 Experimental Result - Edge Detection – Hwy3

5.1.4 Object Tracking and Speed Measurement

The algorithm next projects the scene from perspective view to rectified view to perform the tracking in the vertical direction. The tracking and speed measurement is performed on each pixel of the center lower edge, therefore each of these pixels will have a result in this stage. Once completed, the scene is projected back to perspective view, however, some results may be lost in this back-projection.

Next, the algorithm measures the object's screen-domain displacement and uses the roadmap to calculate their road-domain speed. Figure 53 and Figure 54 show just one result each from the location of the crosshair cursor. Each pixel has two values, the first value represents the road-domain speed, and the second value represents the screen-domain displacement in the unit of pixels encoded in 2's complement. The last two channels should be ignored as the shader program does not use them.

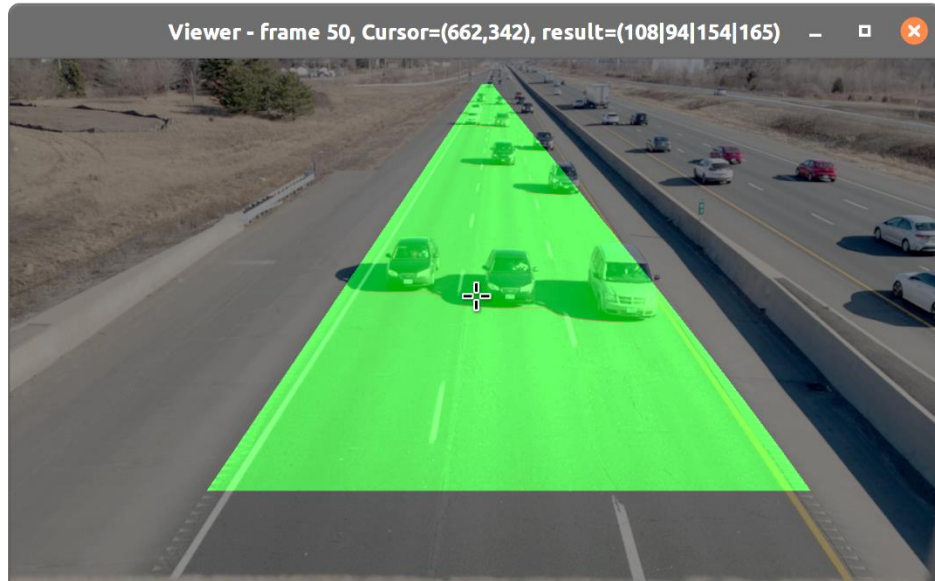


Figure 53 Experimental Result - Measurement Stage - QEW



Figure 54 Experimental Result - Measurement Stage – Hwy3

For example, in Figure 54, the screen-domain position (x=577px, y=451px) has results 85 and 119 (-8 pixels when converted from 2's complement form). Therefore, the object traveled 8 pixels downward in the screen-domain which translates to 85 km/h when using the roadmap translation with an earlier frame.

5.1.5 Grouping Results for All the Objects

The last step is to group the result on each pixel of each object to generate one result for each object. These results are transferred from the GPU to the CPU and exported Figure 55 in the following text file format:

```
F $frame_number $object_count
O $speed_kph $road_domain_xy $screen_domain_xy $deltay
```

Following is the output generated on the QEW scene at frame 50 (Figure 53):

```
F 50 10
O 86 -0.80,108.03 630,91 -3
O 92 1.70,105.53 660,93 -3
O 61 4.89,76.35 721,126 -6
O 109 0.85,65.34 656,146 -7
O 111 3.91,50.65 737,186 -12
O 92 -2.26,28.56 542,320 -26
O 94 -1.42,28.56 578,320 -26
O 108 0.51,26.59 663,342 -34
O 112 2.50,25.78 758,352 -39
```

O 115 3.51,25.16 810,360 -39

Following is the output of the Hwy3 scene at frame 330 (Figure 54):

F 329 3

O 71 -3.14,84.65 667,372 -5

O 81 -5.41,68.22 623,410 -8

O 85 -6.75,54.98 581,451 -10

5.1.6 Display on Screen

In development mode, the speed results can be shown with the associated objects on the screen; see Figure 55 and Figure 56.



Figure 55 Experimental Result - Display Stage – QEW



Figure 56 Experimental Result - Display Stage – Hwy3

5.2 Accuracy

The accuracy of this algorithm is highly dependent on the resolution and quality of the input images. If the resolution of the camera is low, the accuracy of the algorithm will be low. If the quality of the input video is poor, such as in foggy and rainy weather, the algorithm may produce false result as it cannot reliably detect objects in the scene. Furthermore, due to the nature of the perspective view, objects further from the camera will have a lower resolution (pixel to meter ratio) which will impact the measured result against the object's road-domain speed. These issues are common limitation with video-based algorithms.

A poorly generated roadmap will provide inaccurate road-domain information which will also negatively affect the speed calculations. In very windy environment, the camera may shift causing the roadmap to misalign with the input video, however, radar-based systems are also susceptible to variations in alignment.

The accuracy will be discussed by investigating the screen-domain displacement measurement versus manual counting. The algorithm has a development mode to draw the change in the y-position in the screen-domain instead of the road-domain speed in the display stage. Each figure specifically shows the region where the vehicles are as well as including a ruler tick representing 10 pixels in the vertical direction (e.g. brown at y=510px, red at y=520px, orange at y=530px, etc.)

The following Hwy3 example shows the change in the y-position from the current frame (Figure 57 at frame 330) to the immediate previous frame (Figure 58 at frame 329).

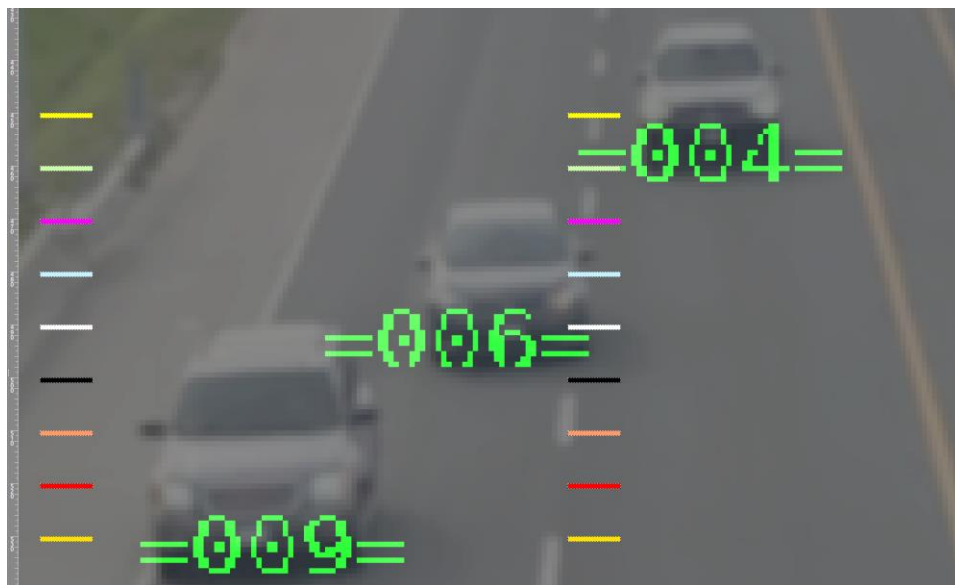


Figure 57 Accuracy Benchmark – Between frame 330 and 329



Figure 58 Accuracy Benchmark – Between frame 329 and 328

The result here shows that the van (vehicle on the bottom left) moves 9 pixels in one frame. To verify this, the license plate is used as a reference point. At frame 330 (Figure 57) the lower edge of the license plate is at the same vertical position as the yellow strip, and at frame 329 (Figure 58) the license plate moves to 2 pixels below the red strip; a difference of 8 pixels between two frames. The algorithm finds a difference of 9 pixels (a 11% deviation from the manually measured results). This deviation can be attributed to aliasing of the camera resolution and errors in previous stages, such as edge detection and

moving blob detection. As noted previously, the low resolution and high sampling rate negatively affect the accuracy of this algorithm. The result of this algorithm states that this vehicle is travelling at 88 km/h (see Figure 61). If that is the case, a 1 pixel deviation translates to a $88 \div 9 = 9.8$ km/h measurement error.

The same example can be calculated again showing the change in the y-position from current frame (Figure 59 at frame 330) and (Figure 60 at frame 327); a 3-frame difference.



Figure 59 Accuracy Benchmark – Between frame 330 and 327

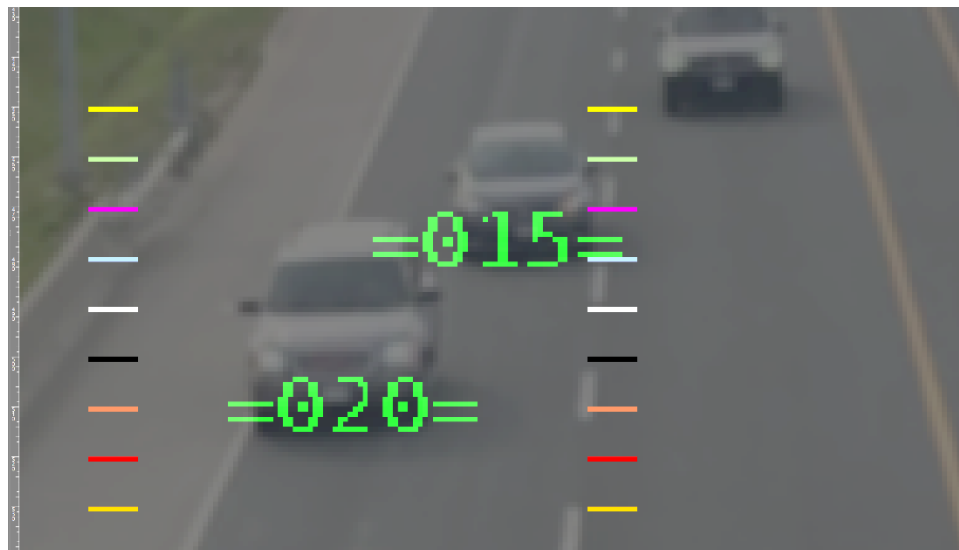


Figure 60 Accuracy Benchmark – Between frame 327 and 324

Now the van moves 23 pixels in three frames according to the algorithm. Using the license plate again as a reference, the van starts from the same vertical position as the

yellow strip at frame 330 (Figure 59) and ends 3 pixels above the orange strip at frame 327 (Figure 60); a 23 pixel difference in one frame.

This manually measured result is the same as the result generated by the algorithm. There may still be error due to aliasing, however, a 1-pixel deviation now translates to a $88 \div 23 = 3.8$ km/h measurement error.

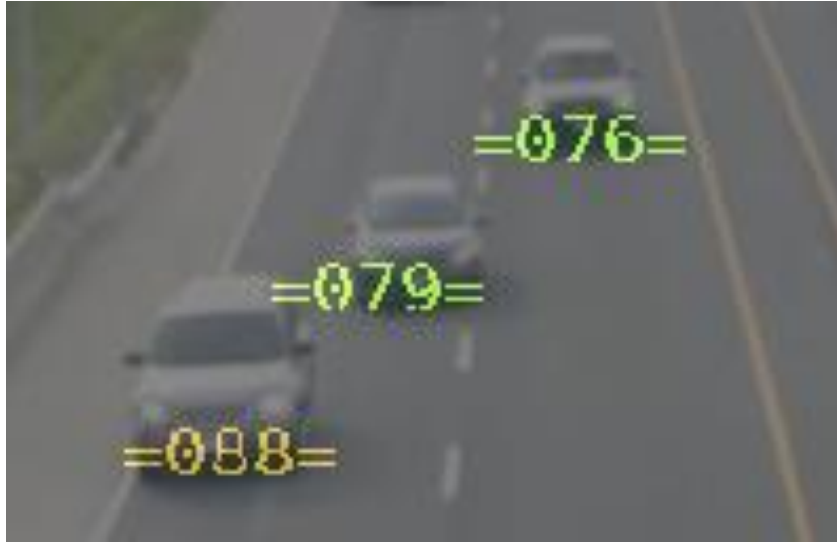


Figure 61 Accuracy Benchmark - Speed Measurement between frame 330 and 327

Figure 61 shows the above result after applying the roadmap translation. The three vehicles in Figure 59 move 23, 16 and 10 pixels respectively in 3 frames. By applying the roadmap, the program determines their speed to be 88 km/h, 79 km/h and 76 km/h respectively.

As seen above, the measuring error of the y-position displacement in the screen-domain may slightly deviate from the manually determined result, but this error is usually small, and therefore acceptable. The quality of the roadmap file and the resolution of the camera are major contributing factors in maximizing the accuracy of the road-domain speed measurement.

The most accurate results are located at the center portion of the scene as further objects have poor accuracy due to the low resolution in the perspective view. Closer objects which are moving too fast tend to generate blurred images on low-speed cameras which also gives poor results in the lower edge detection stage of the algorithm. The algorithm will also generate incorrect results at the frames when the object either enters or exits the ROI due to lack of lookback frames and partial detection for the specific object.

5.3 Results

To visually demonstrate the proposed algorithm, a series of frames from the Hwy3 scene are processed; Figure 62 shows the first frame in this series. Here, the three vehicles travelling towards the camera on the lower left will be discussed in detail. The value of 1 inside the box refers to their beginning position later in Figure 63.

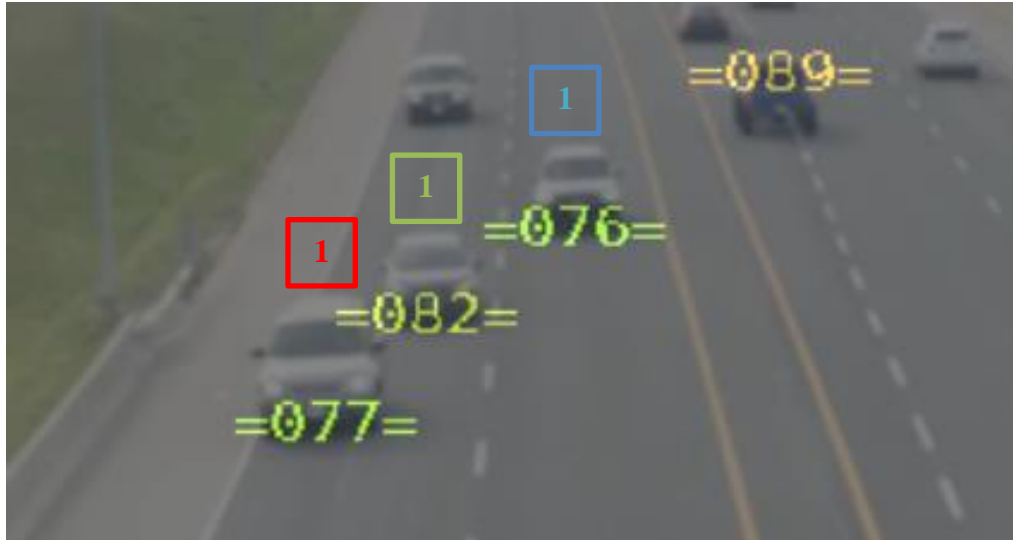
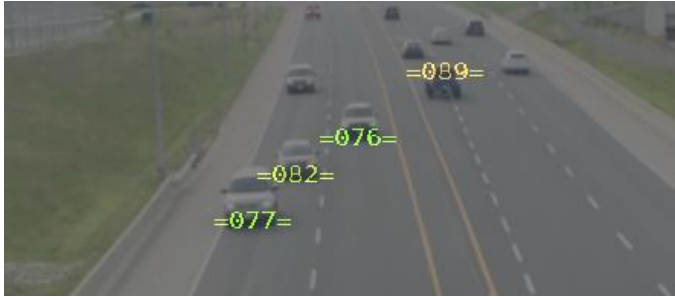

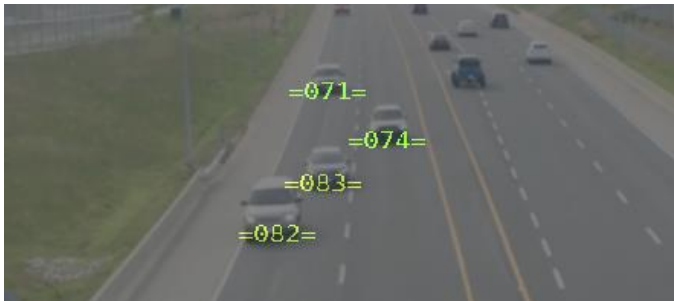
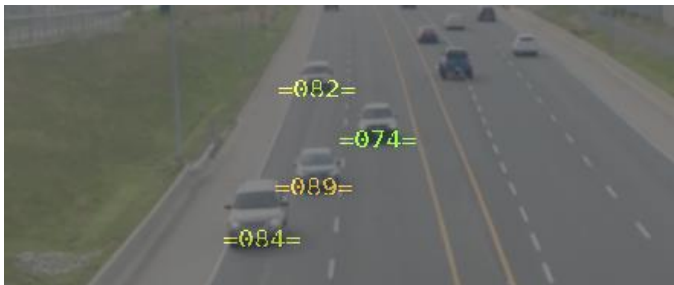
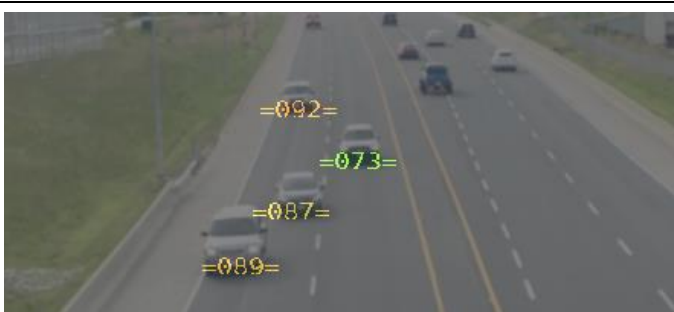
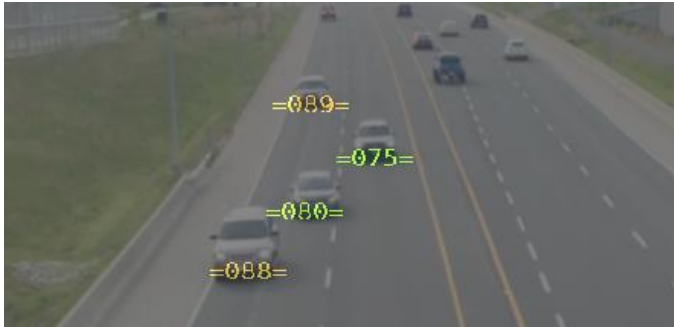

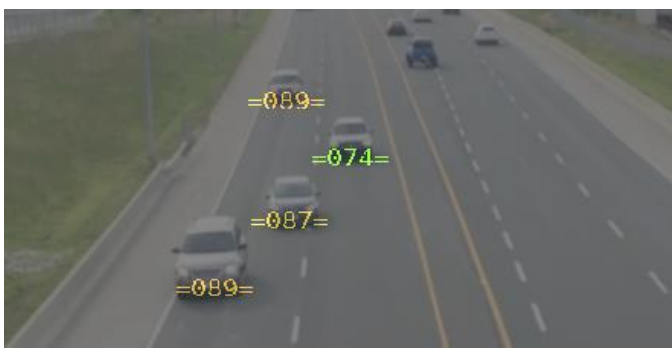
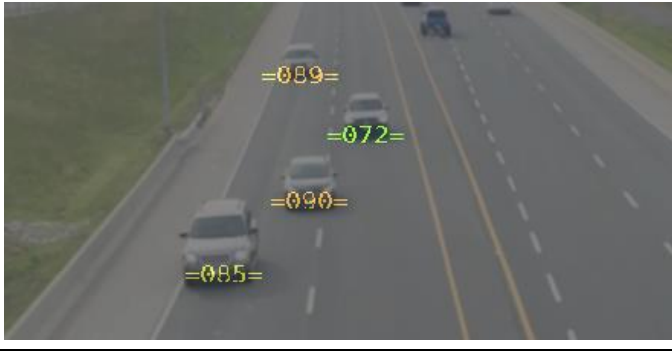
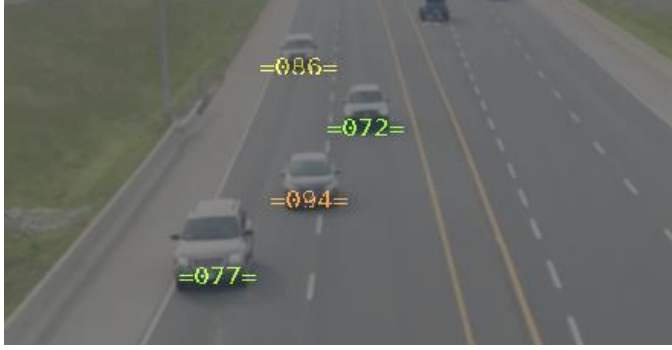


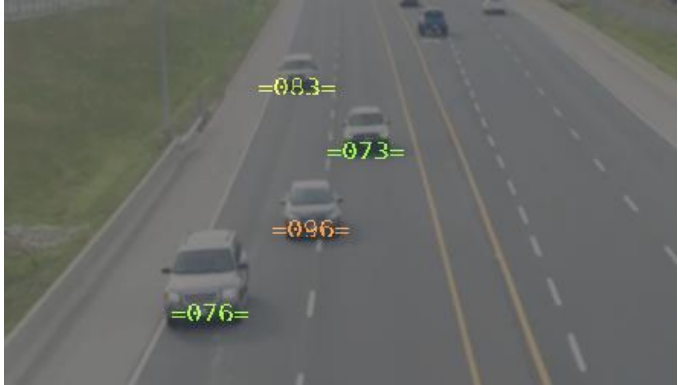
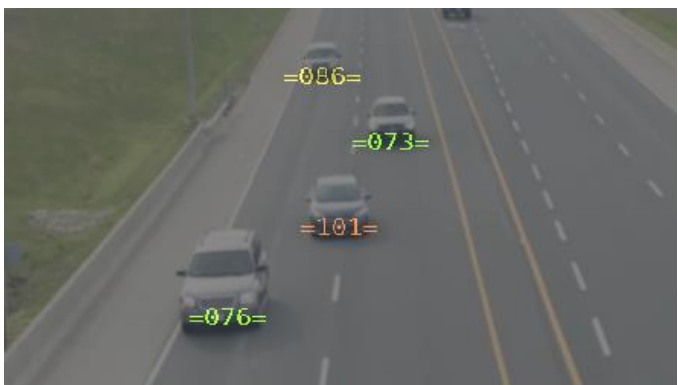

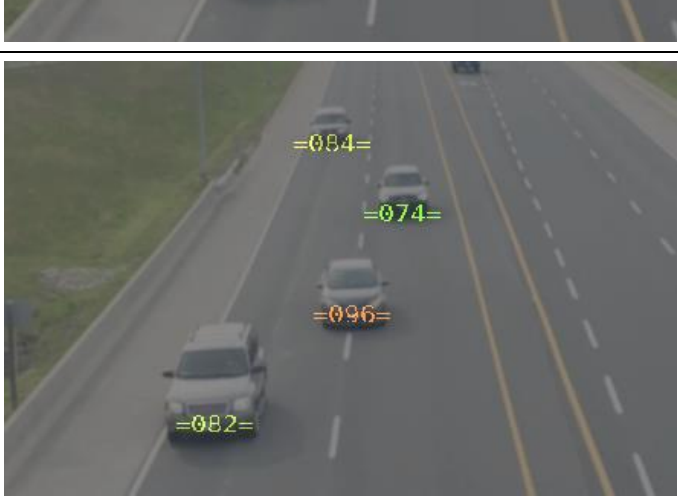
Figure 62 Result

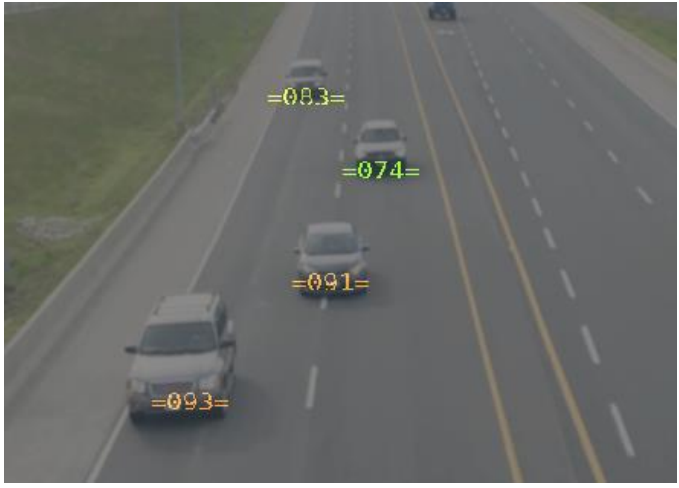


Table 1 is a list of the frames with speed displayed (as shown in development mode) and machine-readable output and some comments:

Display Speed shown on object	Machine Output
	<pre> F \$frame_number \$object_count O \$speed_kph \$road_domain_xy \$screen_domain_xy \$deltay </pre>
	<pre> F 323 4 O 89 2.89,139.76 723,311 5 O 76 -3.16,95.74 666,354 -8 O 82 -5.79,81.66 625,378 -13 O 77 -6.84,68.93 597,408 -16 </pre>
<p>At frame 323, 4 objects are detected. Object 1: speed 89km/h, road-domain location (2.89m,139.76m), screen-domain position (723px,311px), and moves 5 pixels up.</p>	

	<p>F 324 3 O 77 -3.17,94.40 666,356 -9 O 80 -5.77,79.16 624,383 -11 O 78 -6.76,66.79 596,414 -18</p>
	<p>F 325 4 O 71 -6.57,119.39 627,327 -6 O 74 -3.19,92.40 666,359 -9 O 83 -5.71,77.38 624,387 -15 O 82 -6.74,64.74 594,420 -19</p>
	<p>F 326 4 O 82 -6.51,117.31 627,329 -6 O 74 -3.13,90.50 667,362 -10 O 89 -5.56,74.72 625,393 -16 O 84 -6.75,62.37 591,427 -20</p>
	<p>F 327 4 O 92 -6.46,115.23 627,331 -7 O 73 -3.20,88.73 666,365 -10 O 87 -5.67,72.51 622,398 -16 O 89 -6.72,60.01 589,434 -22</p>
<p>When the object is far from the camera, this low resolution (pixel/meter) causes inaccurate measurement. The speed reading of objects will oscillate.</p>	
	<p>F 328 4 O 89 -6.59,113.15 625,333 -7 O 75 -3.14,86.95 667,368 -10 O 80 -5.63,70.72 621,403 -16 O 88 -6.79,57.64 584,442 -23</p>

	<p>F 329 4</p> <p>O 87 -6.53,110.02 625,336 -8</p> <p>O 74 -3.14,84.65 667,372 -11</p> <p>O 86 -5.27,68.58 626,409 -18</p> <p>O 90 -6.75,54.98 581,451 -25</p>
	<p>F 330 4</p> <p>O 89 -6.49,108.30 625,338 -7</p> <p>O 74 -3.13,83.16 667,375 -11</p> <p>O 87 -5.12,66.09 627,416 -20</p> <p>O 89 -6.72,52.80 578,460 -27</p>
	<p>F 331 4</p> <p>O 89 -6.53,106.59 624,340 -8</p> <p>O 72 -3.12,81.16 667,379 -12</p> <p>O 90 -4.90,64.06 630,422 -21</p> <p>O 85 -6.71,50.86 574,470 -29</p>
	<p>F 332 4</p> <p>O 86 -6.55,104.02 623,343 -8</p> <p>O 72 -3.12,79.16 667,383 -12</p> <p>O 94 -4.83,61.36 630,430 -23</p> <p>O 77 -6.70,48.93 570,480 -31</p>

	<p>F 333 4</p> <p>O 83 -6.59,102.43 622,345 -8</p> <p>O 73 -3.12,77.38 667,387 -13</p> <p>O 96 -4.70,58.82 631,438 -24</p> <p>O 76 -6.68,46.65 565,493 -34</p>
	<p>F 334 4</p> <p>O 86 -6.52,100.11 622,348 -9</p> <p>O 73 -3.12,75.60 667,391 -13</p> <p>O 101 -4.52,56.16 633,447 -27</p> <p>O 76 -6.64,44.70 560,506 -37</p>
	<p>F 335 4</p> <p>O 82 -6.53,97.78 621,351 -9</p> <p>O 73 -3.12,73.83 667,395 -14</p> <p>O 102 -4.44,53.57 633,456 -28</p> <p>O 78 -6.56,42.60 556,520 -41</p>
	<p>F 336 4</p> <p>O 84 -6.48,95.74 621,354 -9</p> <p>O 74 -3.12,71.79 667,400 -14</p> <p>O 96 -4.32,51.63 634,466 -31</p> <p>O 82 -6.68,39.89 544,538 -46</p>

	<p>F 337 4</p> <p>O 83 -6.58,93.74 619,357 -10</p> <p>O 74 -3.04,70.00 668,405 -15</p> <p>O 91 -4.20,49.31 635,478 -33</p> <p>O 93 -6.46,37.19 543,556 -52</p>
<p>When the object is close to the camera, it is blurred due to its high speed. Measurement will be affected.</p>	
	<p>F 338 4</p> <p>O 82 -6.61,91.73 618,360 -10</p> <p>O 72 -3.08,68.22 667,410 -16</p> <p>O 84 -4.08,47.10 636,490 -36</p> <p>O 104 -6.62,34.04 528,577 -58</p>
	<p>F 339 5</p> <p>O 80 -6.54,89.91 618,363 -10</p> <p>O 72 -3.01,66.09 668,416 -17</p> <p>O 84 -3.94,45.00 637,504 -40</p> <p>O 96 -6.70,32.08 519,590 -53</p> <p>O 100 -6.24,32.08 536,590 -55</p>

Object exits the scene; an incomplete object on the ROI border produces multiple center lower edges with multiple reading generated. However, both readings have similar results, so the outcome is not significantly affected.



F 340 3
O 80 -6.61,88.14 616,366 -11
O 73 -3.01,64.40 668,421 -18
O 85 -3.80,42.60 638,520 -44



F 341 3
O 76 -6.59,85.77 615,370 -11
O 79 -3.00,62.03 668,428 -19
O 91 -3.66,39.89 639,538 -50



F 342 3
 O 80 -6.57,83.65 614,374 -12
 O 81 -3.00,59.71 668,435 -20
 O 101 -3.50,36.89 641,558 -56



F 343 3
 O 76 -6.55,81.66 613,378 -12
 O 83 -2.98,57.64 668,442 -21
 O 113 -3.34,33.44 643,581 -63



F 344 3
 O 78 -6.60,79.66 611,382 -13
 O 85 -2.91,55.27 669,450 -23
 O 98 -3.25,32.08 645,590 -55



F 345 2
 O 75 -6.57,77.82 610,386 -13
 O 84 -2.90,53.18 669,458 -25


	F 346 2 O 80 -6.53,75.60 609,391 -14 O 78 -2.89,51.44 669,467 -26
---	---

Table 1 Result by Frame

Figure 63 shows the vehicles' locations on a Google Map satellite photo at frame 323 (1), 328 (2), 333(3), 338 (4), and 343(5). A ruler is provided on the left edge of the road (shown in white).

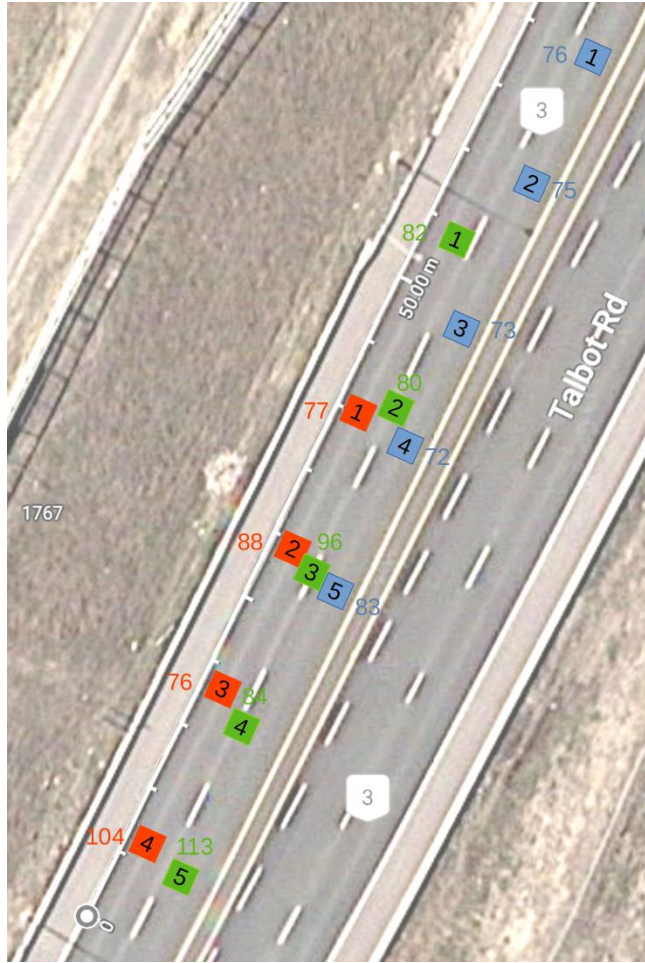


Figure 63 Result on Satellite Photo

The first vehicle (the van, red in satellite photo) travels about 34 meters in 1.5 seconds (15 frames), which is equivalent to 81.6 km/h; whereas the second vehicle (the sedan changing lanes, green in satellite photo) and the third vehicle (the sedan on the passing lane, blue in satellite photo) travel about 49 meters and 42 meters in 2 seconds (20 frames), respectively, equivalent to 88.2 km/h and 75.6 km/h respectively.

Table 2 was generated to compare the results of the algorithm and the manually measured speed on the satellite photo. It shows the derived and average speed of the three vehicles in the scene as well as the manually measured speed on the satellite photo. Because the objects are blurred when they are close to the camera, those results will be discarded from the average speed calculation.

Frame	Van	Lane Changing Sedan	Passing Lane Sedan
323	77	82	76
324	78	80	77
325	82	83	74
326	84	89	74
327	89	87	73
328	88	80	75
329	90	86	74
330	89	87	74
331	85	90	72
332	77	94	72
333	76	96	73
334	76	101	73
335	78	102	73
336	82	96	74
337	93	91	74
338	104 (discard)	84	72
339	100, 96 (discard)	84	72
340	-	85	73
341	-	91	79
342	-	101 (discard)	81
343	-	113 (discard)	83
Average	82.6	88.4	74.6
Satellite result	81.6	88.2	75.6
Difference	1.0 (1.23%)	0.2 (0.23%)	1.0 (1.32%)

Table 2 Result - Algorithm Speed by Frame vs Satellite Speed

The speed difference shown in the table is less than 1.0 km/h or below 1.5% which shows the algorithm performs well over several averaged results.

5.4 Benchmarking Performance

The proposed algorithm is implemented in “C”. GLEW (OpenGL Extension Wrangler) and GLFW (OpenGL FrameWork) are used to load the OpenGL 3.1 ES API, of which only the core profile is used. As an example of a low-end embedded system, a Raspberry Pi 4B with 4GB RAM and integrated VideoCore 6 GPU is used. The RPi Linux image is used with default device tree; no overclock is used.

5.4.1 Scene Information

Scene	Hwy3	QEW
Input Video Format	1280x720x10FPS RGBA8	1280x720x10FPS RGBA8
Input Video Length	518 frames (51.8s, 1.9GB)	331 frames (33.1s, 1.2GB)
Number of Vehicles	16	40
Frame size	921600px	921600px
Perspective ROI size	143892px (15.61%)	245529px (26.64%)
Rectified ROI size	236293px (25.64%)	463484px (50.29%)
Tracking Frame Lookback	3	3
Speed Threshold	200 km/h	200 km/h

Table 3 Performance Benchmark – Scene Information

Table 3 shows the information about the two test scenes. The resolution and FPS for both scenes are same: 1280 pixels wide by 720 pixels high at 10 FPS, that is 921600 pixels each frame and 9.216 mega pixels each second.

The Hwy3 scene had moderate traffic conditions, and 16 vehicles were recorded during the 51.8 seconds of video (518 frames). The QEW scene had busier traffic conditions, and 40 vehicles had been recorded during the 33.1 seconds of video (331 frames). Furthermore, the perspective and rectified ROI of the QEW scenes are 245529 pixels (occupies 26.64% of the screen) and 463484 pixels (occupies 50.29% of the screen), respectively. The ROI of the QEW scene is about double that of the Hwy3 scene, 143892 pixels (occupies 15.61% of the screen) for perspective view and 236293 pixels (occupies 25.64% of the screen) for rectified view. The ROIs sizes of QEW yield higher system load than the Hwy3 scene; the longer process time can be expected due to this reason. However, the higher number of objects in the QEW scene may not yield a higher system load, as this load is affected by the number of pixels representing objects (not the number objects), and the chance of the blob searching being intercepted in the speed measurement stage. The blob grouping stage is performed on pixels that do not represent an object, so a busier scene probably yields less processing time if the ROI sizes of both scenes are the same.

For the speed measurement stage, the frame lookback is set to 3 meaning the algorithm will lookback 3 frames when finding the displacement of an object. The speed threshold for both scenes is 200 km/h. Note that, due to the distance of the camera to the road, the actual thresholds in the unit of pixels are different.

5.4.2 Overall Performance Benchmark

Scene	Process Time	Average Frame Time
Hwy3 (Production)	28.743 s	55.489 ms
Hwy3 (DevMode)	34.019 s	65.673 ms
QEW (Production)	33.636 s	101.62 ms
QEW (DevMode)	36.563 s	110.46 ms

Table 4 Performance Benchmark – Process Time

Table 4 shows the time for this algorithm to process the scenes and the average time spent in each frame of the scenes. The time measured for each run is shown in Appendix D.1 Performance Benchmarks – Process Time. The production scenes represent the time for the algorithm to read, upload, process, download, post-process and export the data; whereas the “DevMode” (or Development Mode) scenes represent the time to do the above works plus uploading and displaying the results on the screen using a human-readable format for an operator to inspect the result.

To minimize any variation on the process time, the program has been executed ten times on the test machine, to determine the average result. The time spent in each stage is listed in Appendix D.1 Performance Benchmarks – Process Time. This time only measures the time to process the input data (time to stream a test video file to the program); and does not include time to start the program, load the OpenGL API and compile the shader program.

For both scenes, the display stage (Development mode) consumes about 10 milliseconds of the process time. This feature is useful for an operator to manually monitoring the output. For production environment, the algorithm uses a machine-readable format to export the results; hence this stage can be removed to further increase the algorithm performance.

Since the input video is at 10 FPS, the algorithm has 100 milliseconds to process each frame. This constraint is satisfied by the Hwy3 scene; however, due to high workload of the QEW scene, it fails to meet the time constraint for some frames. The production version is slightly over the time constraint, whereas the development mode version is about 10% over the time constraint. The production version is expected to be the one in normal use.

5.4.3 Stage Performance Benchmark

To measure the time spent in each stage of the algorithm, sync objects are placed in the GPU pipeline to measure the time when each stage of this algorithm finished. The sync objects and time get system call are placed in the following way:

```

for (each frame) {
    start = getTime();
    uploadRawFrameToGPU();
    getRawFrameFromReaderThread();
    startDownloadProcessedFrame();
    dmaInit = getTime();
    placeProgramQueueGPU(shader_blur);
    placeSync(sync_blur);
    placeProgramQueueGPU(shader_changingBlobDetection);
    placeSync(sync_changingBlobDetection);
    placeProgramQueueGPU(shader_changingBlobGrouping);
    placeSync(sync_changingBlobGrouping);
    placeProgramQueueGPU(shader_lowerEdgeDetection);
    placeSync(sync_lowerEdgeDetection);
    placeProgramQueueGPU(shader_projectToRectified);
    placeSync(sync_projectToRectified);
    placeProgramQueueGPU(shader_pixelSpeedMeasurement);
    placeSync(sync_pixelSpeedMeasurement);
    placeProgramQueueGPU(shader_projectToPerspective);
    placeSync(sync_projectToPerspective);
    placeProgramQueueGPU(shader_speedPixelGrouping);
    placeSync(sync_speedPixelGrouping);
    wait(sync_blur);
    blur = getTime();
    wait(sync_changingBlobDetection);
    changingBlobDetection = getTime();
    wait(sync_changingBlobGrouping);
    changingBlobGrouping = getTime();
    wait(sync_lowerEdgeDetection);
    lowerEdgeDetection = getTime();
    wait(sync_projectToRectified);
    projectToRectified = getTime();
    wait(sync_pixelSpeedMeasurement);
    pixelSpeedMeasurement = getTime();
    wait(sync_projectToPerspective);
    projectToPerspective = getTime();
    wait(sync_speedPixelGrouping);
    speedPixelGrouping = getTime();
    startDownloadProcessedFrame();
    downloadFinished = getTime();
    postProcess();
    postProcess = getTime();
    if (displayMode) {
        uploadResultToGPU();
        placeProgramQueueGPU(shader_display);
    }
    waitReadThread();
    display = getTime();
}

```

The `getTime()` function will request a system `MONOTONIC` clock and record a timestamp at that point. To get the time used by a specific stage, that timestamp is subtracted from its previous timestamp (e.g. time to project the scene to rectified view will be `projectToRectified - lowerEdgeDetection`).

Scene (unit: ms)	Hwy3 Production	Hwy3 DevMode	QEW Production	QEW DevMode
Start	0.444	3.456	0.608	3.765
DMA Init	22.441	33.185	24.232	37.188
Blur	10.327	10.128	11.337	11.217
Changing Blob Detection	1.359	1.665	1.861	2.308
Changing Blob Grouping	20.062	19.829	53.922	54.491
Lower Edge Detection	3.557	3.349	6.249	6.843
Project to Rectified	1.610	1.457	2.829	2.688
Pixel Speed Measurement	2.487	2.075	8.375	7.099
Project to Perspective	2.798	2.195	5.344	4.133
Speed Pixel Grouping	2.499	1.949	4.081	3.065
Download Finished	0.018	0.016	0.020	0.020
Post-Process	7.413	7.194	13.988	14.460
Display	0.069	0.411	0.123	0.704
Total Frame Time	75.083	86.909	132.972	147.987

Table 5 Performance Benchmark - Stage Time

Table 5 shows a summary of the time spent in each stage of the algorithm. This algorithm is executed for multiple times to avoid any variations, and the time recorded in each run is shown in Appendix D.2 Performance Benchmarks – Stage Time. Due to the extra time polling, “gettime” system call, and explicitly synchronization, the program needs more time to process the same test scenes, and the total frame time shown in this table will be higher than Table 4. Since the process pipeline is partially empty in the first few frames, this benchmark will only count the time from the 5th frame (first frame is frame 0) to the end of the test video.

The “Start” and “DMA Init” stage for both scenes use a similar amount of time. The “DevMode” requires the algorithm upload the results back to GPU and display them (copy content from user-space framebuffer to system-space screen buffer); hence more time is required by the system (stall the “Start” stage) and more bandwidth is required by the memory controller (affects the “DMA Init” time).

The “Blur” stage is applied to the entire scene instead of just the ROI, therefore this stage consumes the same amount of time for both scenes. The remaining stages are shader only and applied on the ROI. Since the QEW scene has larger ROI than the Hwy3 scene, it requires more time to process these stages.

The “Changing Blob Grouping” stage consumes most process time (about 30% of the total process time).

Since the algorithm uses PBOs to perform the download, the actual download is applied on the data from previous frame, and it is performed in the background. The “Download Finished” stage is in fact a simple pointer swap operation showing that it only takes a fraction of time.

The “Post Processing” performed by CPU depends on the number of objects in the scene. Given that the QEW scene is busier than the Hwy3 scene, more time is used in the QEW scene.

The “Display” stage only places a command in the command queue (asynchronized operation) and does not block the pipeline. Only a fraction of time is consumed by this stage.

It is worth noting that this benchmark has its limitation. Since using the sync object introduces extra time in the pipeline, it negatively affects the accuracy of this benchmark. For example, there are 3 processes X , Y and Z in series each requiring 3 milliseconds to complete, and 2 sync object P and Q (which are placed in between) each requiring 1 millisecond to complete.

- The sync object is measured on the CPU side, but the process is executed on the GPU side, so the time used by sync objects will overlap with the processes. Here the benchmark started at 0 milliseconds, P finished at 4 milliseconds, Q finished at 7 milliseconds, and the process finished at 9 milliseconds. The result will be X uses 4 milliseconds ($P-0$), Y uses 3 milliseconds ($Q-P$) and Z uses 2 milliseconds ($9-Q$).
- Adding sync objects makes the process on CPU main thread and GPU consume more time, which will give other thread (e.g. CPU reader thread and memory controller) extra time to finish their work. For example, if the main thread (and the GPU) only requires 50 milliseconds to process the data and the memory controller needs 70 milliseconds to transfer the data, the memory bandwidth will be the bottleneck of this algorithm. With the sync object, the main thread (and the GPU) now requires 100 milliseconds to process the data and perform the explicitly synchronization, the memory bandwidth will no longer be the bottleneck.

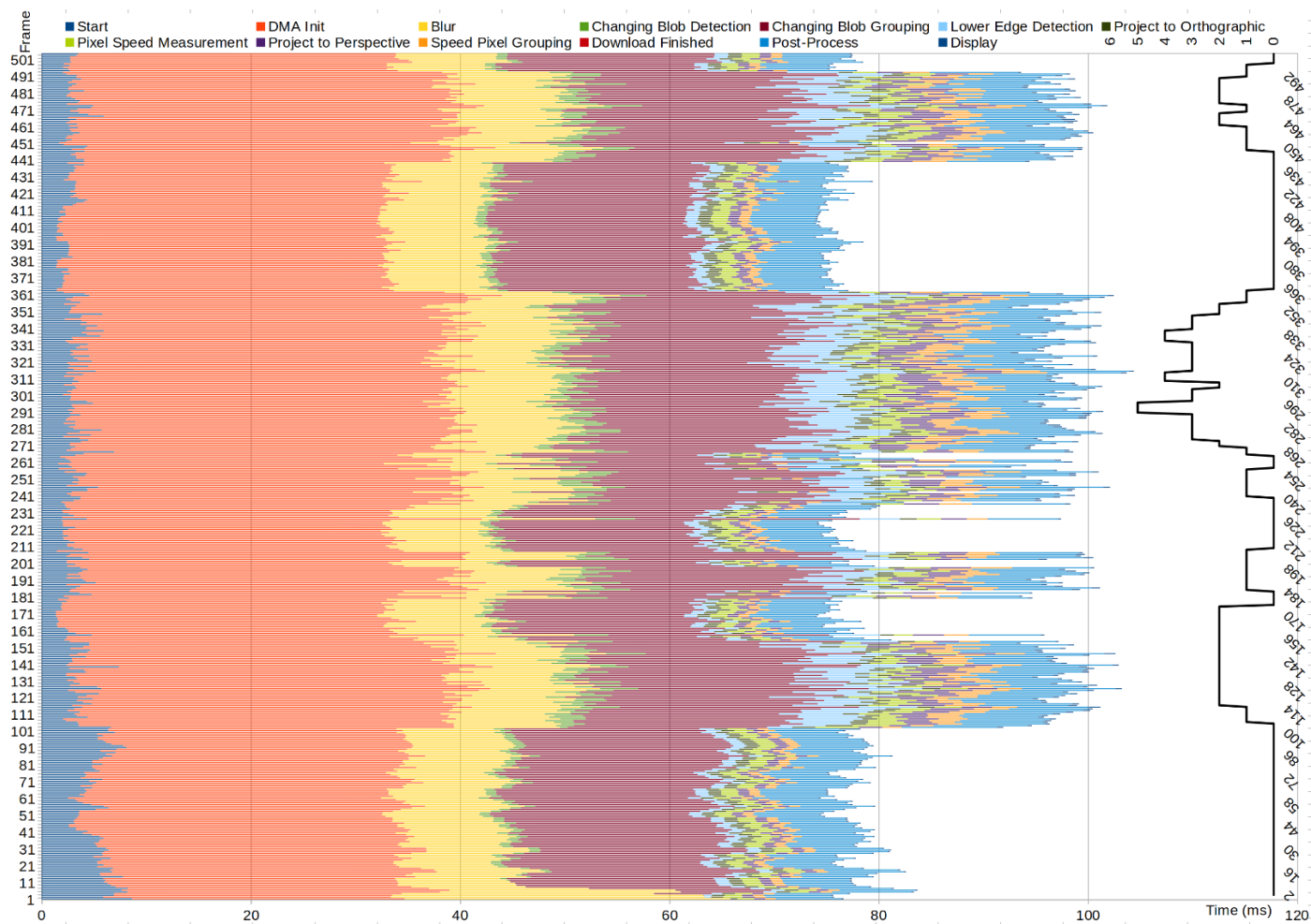


Figure 64 Performance Benchmark – Hwy3 Stage Time vs Objects

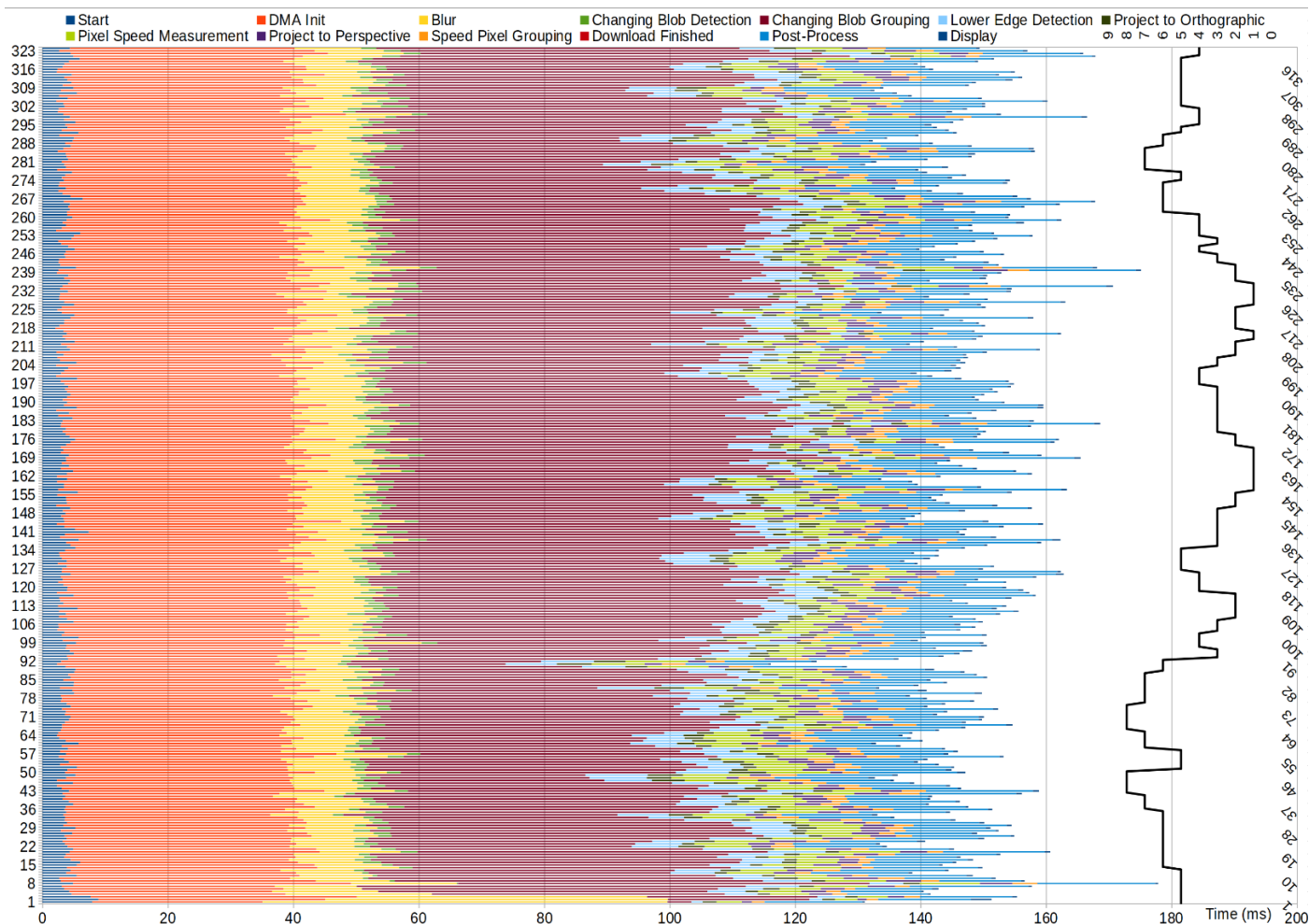


Figure 65 Performance Benchmark — QEW Stage Time vs Objects

Figure 64 and Figure 65 show the average time used by each stage verses number of objects in the Hwy3 and QEW scenes in each frame including the Display stage for development mode. The distinct colors are given to the time consumed by each stage of the algorithm; whereas the number of objects is shown in the line chart located at right of the graph (note: x-axis has larger value in left direction and smaller value in the right direction).

For the Hwy3 scene, there is no object presented in the scene at some time and the maximum number of objects at any moment in the scene is 6. It is clear to say that the number of objects is directly contributed to the process time, especially when there is no object in the scene.

For the QEW scene, the road is constantly occupied by 4 to 6 objects (for most of the time). The figure shows that there is no significant difference in processing time contributed by number of objects in the scene.

CHAPTER 6

Conclusion and Future Work

6.1 Conclusions

This thesis presented a low-cost video-based speed measurement algorithm. It uses the industry standard non-vendor specific OpenGL ES so it can be implemented on any platform with OpenGL ES support. It operates on low-end embedded system as it uses simple and basic foundations based on a few assumptions to lower the overall implementation complexity in OpenGL ES. Furthermore, it does not require any special peripheral devices, so existing infrastructure or low-cost cameras can be used with minimal modification. Several optimizations were discussed in this thesis that further increased the system capability.

The proposed algorithm does come with some limitations. It requires a preciously generated roadmap aligned with the input image to rectify the image and to calculate the displacement of the target objects. It also assumes vehicles move along the direction of the road lanes with limited lane changing ($1/3$ of vehicle width). The quality and resolution of the input image highly affect the accuracy and precision of the result.

The algorithm was tested on two real world traffic videos (720p video at 10 FPS) and successfully extracts the speed data of road vehicles in view on a low-end embedded system (Raspberry Pi 4).

6.2 Future Work

6.2.1 Automatically Roadmap Generation

Currently, the roadmap mapping the road-domain location and the perspective & rectification projection information must be generated manually for a scene. Having the roadmap automatically generated during system initialization can make the system off-the-shelf and ready-to-use.

6.2.2 Spherical Roadmap with Gyroscope

Cameras mounted on road infrastructure are subject to strong winds. Any movement of camera, especially in changing of its orientation, will misalign the roadmap and input image of the scene. To solve this issue, a gyroscope and spherical roadmap can be used to read the orientation of the camera. At the beginning of each frame, the system can read the gyroscope in addition to camera input to get the real orientation of the camera. It can

then use this data to crop a rectangular section of the spherical roadmap, which contains the road-domain data for that orientation in that frame.

Gyroscopes are commonly used in embedded systems (i.e. smart phones) with relatively high accuracy and low cost. For example, the MPU-6050 gyroscope cost between \$5 to \$10 and it is capable of providing acceleration with a 16-bit resolution at +/- 2G range in all three axis, which translates to $\frac{2^{*90}}{2^{15}} = 0.0055$ degree per bit, at the rate of 40Hz [24]. For a 2 Ki x 2 Ki pixels input image with a 60 degree FOV, each pixel covers a 0.029 degree region, which is easily detected by this gyroscope.

Alternatively, a software image stabilizer can be prepended to the proposed algorithm to feed a stabilized video to compensate any camera movement. This solution can reduce the system cost by eliminating the gyroscope but will increase the software processing load.

6.2.3 Secondary System on the Next Stage

The proposed algorithm only extracts the vehicle speed data from the video. A secondary system can be appended to further analysis the overall speed across a number of frames (i.e. like the results in Table 2). For example, this secondary system can also read the license plate of any vehicle that excess speed limit and upload all of this to a centralized vehicle server.

6.2.4 Using Vulkan API

The latest version of OpenGL ES is 3.2, which was developed in 2015. In 2017, Khronos announced that there will be no newer core version of OpenGL ES; instead, the new Vulkan API will be used as its replacement [25].

Khronos stated that Vulkan provides lower system access, more predictable behaviour, and higher performance than OpenGL and OpenGL ES. Vulkan may significantly improve the performance of this algorithm, however, it currently has poor support on embedded systems. Although Khronos said any device compatible with OpenGL ES 3 is Vulkan compatible, that does not mean there is a Vulkan driver available for that device. It is recommended to continue with OpenGL ES as many embedded devices support it. Only when Vulkan support is more widely available, should the code be evolved to adapt for it.

6.3 Other Considerations

For OpenGL and OpenGL ES, Khronos only specifies an interface specification between the GPU vendor/driver provider and user program developer. The behaviour of the

hardware is not well defined by this specification, making it unpredictable on different platforms with different drivers. OpenGL and OpenGL ES are highly abstract, and the model used in the API is often far different from the actual architecture of a modern GPU. GPU vendors and drivers are free to do things that are not specified. As such, the same program may work on one platform but not on another. Some functions require the use of constants on some platforms, but not for others (for example, `textureOffset()` with variable offset parameter can be compiled with Nvidia desktop GPUs but not with the VideoCore 6 GPU found in the Raspberry Pi 4).

OpenGL and OpenGL ES only specify the minimal accuracy of a given data type (e.g. `mediump float`) [26, p. 87]. GPU vendors are free to use higher precision when manipulating data, however they are not required to implement high precision computation in the fragment shader. This can generate different results.

Lastly, any optimization for a specific platform may end up decreasing performance on another.

REFERENCE

- [1] Government of Canada, "Canadian Motor Vehicle Traffic Collision Statistics: 2020," Government of Canada, 1 February 2022. [Online]. Available: <https://tc.canada.ca/en/road-transportation/statistics-data/canadian-motor-vehicle-traffic-collision-statistics-2020>. [Accessed 1 June 2022].
- [2] National Safety Council, "Speeding," National Safety Council, 2 April 2022. [Online]. Available: <https://injuryfacts.nsc.org/motor-vehicle/motor-vehicle-safety-issues/speeding/>. [Accessed 1 June 2022].
- [3] S. Zeadally, J. Guerrero and J. Contreras, " A tutorial survey on vehicle-to-vehicle communications," *Telecommunication systems*, pp. 469-489, 4 December 2019.
- [4] U.S. Department Of Transportation National Highway Traffic Safety, "FMVSS No. 150 Vehicle-To-Vehicle Communication Technology For Light Vehicles," November, 2016.
- [5] P. Anna, "Multiple Object Tracking in Realtime," OpenCV, 27 October 2020. [Online]. Available: <https://opencv.org/multiple-object-tracking-in-realtime/>. [Accessed 14 August 2022].
- [6] J. Peter, K. Karel, D. Tomas and S. Istvan, "Comparison of tracking algorithms implemented in OpenCV," *MATEC Web of Conferences*, 2016.
- [7] Z. Wang, 3D reconstruction using a stereo vision system with simplified inter-camera geometry, 2010.
- [8] Dicklyon, "A de Havilland Canada Dash 8 propeller, with severe rolling-shutter distortion from a Pixel 3 camera," 13 July 2021. [Online]. Available: https://en.wikipedia.org/wiki/Rolling_shutter#/media/File:Propellor_with_rolling-shutter_artifact.jpg. [Accessed 15 August 2022].
- [9] U.S. Department of Transportation, Traffic Detector Handbook: Third Edition - Volume I, McLean, Virginia: Research, Development, and Technology Turner-Fairbank Highway Research Center, 2006.
- [10] S. Javadi, M. Dahl and M. I. Pettersson, "Vehicle speed measurement model for video-based systems," *Computers & electrical engineering*, pp. 238-248, 2019.

- [11] J. Sochor, R. Juránek and A. Herout, "Traffic surveillance camera calibration by 3D model bounding box alignment for accurate vehicle speed measurement," *Computer vision and image understanding*, pp. 87-98, August 2017.
- [12] D. Carbonera Luvizon, B. T. Nassu and R. Minetto, "A Video-Based System for Vehicle Speed Measurement in Urban Roadways," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, pp. 1393-1404, June 2017.
- [13] Broadcom, "VideoCore® IV 3D Architecture Guide," 16 September 2013. [Online]. Available: <https://docs.broadcom.com/doc/12358545>. [Accessed 1 July 2022].
- [14] Khronos, "OpenCL," Khronos Group, 2022. [Online]. Available: <https://www.khronos.org/opencl/>. [Accessed 18 June 2022].
- [15] Khronos, "OpenGL About," Khronos Group, 2022. [Online]. Available: <https://www.khronos.org/opengl/>. [Accessed 18 June 2022].
- [16] J. Kessenich, D. Baldwin and R. Rost, "The OpenGL Shading Language Version 1.10," 2004 April 2004. [Online]. Available: <https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.1.10.pdf>. [Accessed 19 June 2022].
- [17] Khronos, "OpenGL ES 3.2 Quick Reference Card," 2015. [Online]. Available: <https://www.khronos.org/files/opengles32-quick-reference-card.pdf>. [Accessed 19 June 2022].
- [18] J. Leech, "OpenGL ES 3.2 Spec," 5 May 2022. [Online]. Available: https://www.khronos.org/registry/OpenGL/specs/es/3.2/es_spec_3.2.pdf. [Accessed 19 June 2022].
- [19] Khronos, "OpenGL 4.5 Quick Reference Card," 2014. [Online]. Available: <https://www.khronos.org/files/opengl45-quick-reference-card.pdf>. [Accessed 19 June 2022].
- [20] M. Segal and K. Akeley, "The OpenGL Graphics System: A specification Version 4.5 Core Profile," 29 June 2017. [Online]. Available: <https://www.khronos.org/registry/OpenGL/specs/gl/glspec45.core.pdf>. [Accessed 19 June 2022].

- [21] Stack overflow users, "GLSL shader not unrolling loop when needed," 1 September 2013. [Online]. Available: <https://stackoverflow.com/questions/18557694/glsl-shader-not-unrolling-loop-when-needed>. [Accessed 3 July 2022].
- [22] Khronos, "Synchronization," 4 April 2021. [Online]. Available: <https://www.khronos.org/opengl/wiki/Synchronization>. [Accessed 2 July 2022].
- [23] C. Patrick and R. Christophe, "Asynchronous Buffer Transfers," in *OpenGL Insights*, CRC Press.
- [24] InvenSense Inc., "MPU-6000 and MPU-6050 Product Specification," 19 August 2013. [Online]. Available: https://product.tdk.com/system/files/dam/doc/product/sensor/motion-inertial/imu/data_sheet/mpu-6000-datasheet1.pdf. [Accessed 12 May 2023].
- [25] H. Tobias, "OpenGL ES Update," 2017. [Online]. Available: https://www.khronos.org/assets/uploads/developers/library/2017-siggraph/06_3D-BOF-SIGGRAPH_Aug17.pdf. [Accessed 1 September 2022].
- [26] Q. (. J. K. G. (. D. B. a. R. R. (. 1. A. Robert J. Simpson, "The OpenGL ES® Shading Language Version 3.20.6," 10 July 2019. [Online]. Available: https://registry.khronos.org/OpenGL/specs/es/3.2/GLSL_ES_Specification_3.20.pdf. [Accessed 6 April 2023].

APPENDICES

Appendix A Pre-process

The following are programs used to prepare the video file and generate necessary instruction data for the main program.

Appendix A.1 Video decoder

This Python script is used to convert a compressed video file (such as mp4) to a plain gray8, RGB8 or RGBA8 file format. Since the main program has no video decoder, the video file must be processed by this Python script before using.

```
'''
Video To BitmapVideo Convertor
This simple mini program is used to convert common video file to
readable plain bitmaps.

THIS PROGRAM IS NOT THE CORE PART OF THIS PROJECT! THIS PROGRAM IS USED
TO PREPARE TEST DATA FOR THIS PROJECT.
A 1-minute-long 1080p 30FPS RGB24 video contains about 10GB of data.
Common video files (like .mp4, .avi, .mov) usually compress the video
data to minimize storage space and memory bandwidth
requirements. Compression and decompression (codec) algorithm are
complex, and different format may
use different algorithms. The codec is not what we are studying here.
We do not want to deal with codec or
compressed video in this project; instead, we will only deal with easy-
to-read plain bitmaps.
By "plain", it means:
- A file contains multiple frames. Frame 1 first, then frame 2, then
frame 3...
- A frame has width * height pixels. The order is left-to-right, then
top-to-bottom, like how we write in most languages (e.g. English,
morden Chinese).
- A pixel has 1, 3 or 4 bytes, which is mono, RGB or RGBA.
So, a 1 minute long 1080p 30FPS RGB24 video is about 60MB in .mov
format, but with RGB scheme there are
60s * 30fps * (1920*1080)px/frame * 3byte/px = 11,197,440,000 bytes in
plain bitmap video format.
That's a lot of data need to put on the disk or in our memory. Since we
are now developing the algorithm on a high-end
platform, this is not a big issue. But, when we implement this program
in our product, we need to switch to
streaming mode (so there is only few frames in memory buffer).
'''

import sys
import cv2
```

```

import numpy

src = 'on3.mov' #The input file, a video file in common video
format such as .mp4
dest = '../..on3-720v10.data' #Where to save the plain bitmap video
width = 1280 #Output frame size in pixel
height = 720
fpsSkip = 2 #Skip frames. E.g.: if src is 30fps and skip is 2, dest
will be 30 / (1+2) = 10fps
scheme = cv2.COLOR_BGR2RGBA #The default scheme of openCV is BGR, the
output can be RGBA, RGB or GRAY

'''
Most video contains RGB 3 channels.
RGBA consumes extra 25% space, but align pixels in 32-bit words, which
makes GPU uploading faster.
Gray/Mono consumes least amount of memory, but lacks color.
'''

count = 0
outputFile = open(dest, 'wb')
video = cv2.VideoCapture(src)
if video.isOpened():
    meta = (video.get(cv2.CAP_PROP_FRAME_WIDTH),
video.get(cv2.CAP_PROP_FRAME_HEIGHT), video.get(cv2.CAP_PROP_FPS))
    print('Src video size: {}px * {}px @ {}fps'.format(*meta))
    print('Dest video size: {}px * {}px @ {}fps'.format(width, height,
meta[2]/(1+fpsSkip)))
while True:
    ret, frame = video.read()
    if frame is None or count >= 2400:
        print('Done, end of video reached or limit reached, {} frames
processed.'.format(video.get(cv2.CAP_PROP_FRAME_COUNT)))
        break
    if count % (fpsSkip + 1) == 0:
        frame = cv2.cvtColor(frame, scheme)
        frame = cv2.resize(frame, (width, height))
        outputFile.write(bytearray(frame))
        count = count + 1
outputFile.close()

```

Appendix A.2 Roadmap Generator – Key Point Based

This C program is used to generate a roadmap file based on key points on the road. Linear interpolation is used to generate non-key points.

```

/** This program is used to generate roadmap file.
 * See the readme.md for detail.
 */

#include <stdio.h>
#include <stdlib.h>

```

```

#include <stdarg.h>
#include <inttypes.h>
#include <errno.h>
#include <math.h>
#include <string.h>

/** Output file format:
 * Section I: File header.
 * Section II: Table 1: Road-domain geographic data in perspective and
orthographic views.
 * Section III: Table 2: Search distance, perspective and orthographic
projection map.
 * Section IV: Focus region points.
 * Section V: Meta data: ASCII meta data, for reference, ignored by program.
 */

// Section I: File header
typedef struct FileHeader {
    uint32_t width, height; //Frame size in pixel
    uint32_t pCnt; //Number of road points (perspective view)
    uint32_t fileSize; //Size of roadmap file without meta data, in byte
} header_t;

// Section II: Table 1: Road-domain geographic data in perspective and
orthographic views
typedef struct FileDataTable1 {
    float px, py; //Road-domain geographic data in perspective view
    float ox; //Road-domain geographic data in orthographic view, y-coord of
orthographic view is same as y-coord of perspective view
    float pw; //Perspective pixel width
} data1_t;

// Section III: Table 2: Search distance, perspective and orthographic
projection map
typedef struct FileDataTable2 {
    float searchLimitUp, searchLimitDown; //Up/down search limit y-coord
    float lookupXp2o, lookupXo2p; //Projection lookup table. Y-coord in
orthographic and perspective views are same
} data2_t;

// Section IV: Focus region points
typedef struct Point_t {
    float rx, ry; //Road-domain location (m)
    float sx, sy; //Screen domain position (NTC)
} point_t;

```

```

#define DELI "%*[^0-9.-]" //Delimiter
#define err(format, ...) {fprintf(stderr, "ERROR: "format __VA_OPT__(,)
__VA_ARGS__);}
#define info(format, ...) {fprintf(stderr, "INFO: "format __VA_OPT__(,)
__VA_ARGS__);}

//Return 0 if X is closer to A, 1 if X is closer to B, -1 if X is smaller than
A or greater than B
int isBetween(float x, float a, float b) {
    if (x < a || x > b)
        return -1;
    return ((x - a) <= (b - x)) ? 0 : 1;
}

//Find the first-order equation to describe the line connect two points: y = x
* cof[1] + cof[0]
void polyFit2(float x1, float x2, float y1, float y2, float* cof) {
    cof[1] = (y2 - y1) / (x2 - x1);
    cof[0] = y1 - x1 * cof[1];
}

//Read a line of string, return 0 if token not found, 1 if token found and
argument count matched, -1 if argument count doesn't match
int readline(const char* src, const char* token, const unsigned int argc,
const char* format, ...) {
    if (strncmp(token, src, strlen(token)))
        return 0;

    int result = 1; //Success
    va_list args;
    va_start(args, format);
    if (vsscanf(src, format, args) != argc) //If mismatch
        result = -1; //Change to fail
    va_end(args);

    return result;
}

header_t header;
data1_t* t1;
data2_t* t2;
point_t(* p)[2];

void new() __attribute__((constructor));

```

```

void new() {
    header.width = 0;
    header.height = 0;
    header.pCnt = 0;
    header.fileSize = 0;
    t1 = NULL;
    t2 = NULL;
    p = NULL;
}

void destroy() __attribute__((destructor));
void destroy() {
    free(t1);
    free(t2);
    free(p);
}

int main(int argc, char* argv[]) {
    if (argc != 3) {
        err("ERROR: Bad arg: Use this width height < info.txt > output.map\n");
        err("\twidth and height is the size of frame in px\n");
        err("\tinfo.txt (stdin): an ASCII file contains human-readable road\n");
        err("\toutput.map (stdout): a binary roadmap file\n");
        return EXIT_FAILURE;
    }

    /* Get size */ {
        info("Get frame size and allocate buffer for tables\n");
        header.width = atoi(argv[1]);
        header.height = atoi(argv[2]);

        if (header.width > 2048 || header.width < 320 || (unsigned
int)header.width & (unsigned int)0b111) { //float16 interval > 0 if value >
2048 (abs)
            err("Width cannot be greater than 2048 or less than 320, and must be
multiply of 8\n");
            return EXIT_FAILURE;
        }
        if (header.height > 2048 || header.height < 240 || (unsigned
int)header.height & (unsigned int)0b111) {
            err("Height cannot be greater than 2048 or less than 240, and must be
multiply of 8\n");
            return EXIT_FAILURE;
        }
        /* Note:

```

```

Float16 use 10 bits frac.
For absolute coord, when value > 2048, interval > 1.
For NTC, interval = 2^-11, which is 1/2048 resolution.
*/

t1 = malloc(sizeof(data1_t) * header.height * header.width);
t2 = malloc(sizeof(data2_t) * header.height * header.width);
if (!t1 || !t2) {
    err("Cannot allocate memory for table storage (errno=%d)\n", errno);
    return EXIT_FAILURE;
}

/* Get config from stdin */ {
    info("Process road points\n");
    char buffer[500];
    int res; //Result of readline() 0 = token not match, 1 = ok, -1 = argument
count mismatch
    float sy, sxl, sxr, ry, rxl, rxr; //Screen/road point y/x-left/x-right
coord
    unsigned pPairCnt = 0;

    /* Road points - read for persp*/
    while (fgets(buffer, sizeof(buffer), stdin)) {
        if (res = readline(buffer, "POINT", 6,
DELI"%f"DELI"%f"DELI"%f"DELI"%f"DELI"%f"DELI"%f", &sy, &sxl, &sxr, &ry, &rxl,
&rxr)) {
            if (res == -1) {
                err("Cannot read POINT: bad format, require
'...float...float...float...float...float...float'\n");
                return EXIT_FAILURE;
            }
            if (!(p = realloc(p, (pPairCnt+1) * sizeof(point_t) * 2)) {
                err("Cannot allocate memory for points storage when reading info
file (errno=%d)\n", errno);
                return EXIT_FAILURE;
            }

            point_t* current = p[pPairCnt];
            current[0] = (point_t){.sx = sxl, .sy = sy, .rx = rxl, .ry = ry};
            current[1] = (point_t){.sx = sxr, .sy = sy, .rx = rxr, .ry = ry};
            unsigned int sxl = current[0].sx * header.width, sxr = current[1].sx *
header.width, sy = current[0].sy * header.height;
            info("POINT: PL pos (%.4f,%.4f) <%upx,%upx> @ loc (%.4f,%.4f)\n",
current[0].sx, current[0].sy, sxl, sy, current[0].rx, current[0].ry);

```

```

        info("POINT: PR pos (%.4f,%.4f) <%upx,%upx> @ loc (%.4f,%.4f)\n",
current[1].sx, current[1].sy, sxr, sy, current[1].rx, current[1].ry);

        if (pPairCnt) { //This check does not apply on first road point pair
            point_t* previous = p[pPairCnt - 1];
            if (current[0].sy <= previous[0].sy) {
                err("POINT %u - Screen y-coord should be in top-to-bottom order,
current y-coord (%.4f) should be greater than last y-coord (%.4f)\n",
pPairCnt, current[0].sy, previous[0].sy);
                return EXIT_FAILURE;
            }
            if (current[0].ry >= previous[0].ry) {
                err("POINT %u - Road point at bottom position should be closer,
current y-coord (%.4f) should be lower than last y-coord (%.4f)\n",pPairCnt,
current[0].ry, previous[0].ry);
                return EXIT_FAILURE;
            }
            if (current[1].sx <= current[0].sx) {
                err("POINT %u - Right point (%.4f) should have greater screen x-
coord than left point (%.4f)\n", pPairCnt, current[1].sx, current[0].sx);
                return EXIT_FAILURE;
            }
            if (current[1].rx <= current[0].rx) {
                err("POINT %u - Right point (%.4f) should have greater road x-
coord than left point (%.4f)\n", pPairCnt, current[1].rx, current[0].rx);
                return EXIT_FAILURE;
            }
        }

        pPairCnt++;
    } else {
        err("Unknown info: %s\n", buffer);
        return EXIT_FAILURE;
    }
}

/* Road points - gen for ortho */
float sol = p[0][0].sx, sor = p[0][0].sx, sot = p[0][0].sy, sob =
p[0][0].sy, rol = p[0][0].rx, ror = p[0][0].rx, rot = p[0][0].ry, rob =
p[0][0].ry;
for (unsigned int i = 0; i < pPairCnt; i++) {
    if (p[i][0].sx < sol) {
        sol = p[i][0].sx;

```

```

        rol = p[i][0].rx;
    }
    if (p[i][1].sx > sor) {
        sor = p[i][1].sx;
        nor = p[i][1].rx;
    }
    if (p[i][1].sy < sot) {
        sot = p[i][1].sy;
        rot = p[i][1].ry;
    }
    if (p[i][1].sy > sob) {
        sob = p[i][1].sy;
        rob = p[i][1].ry;
    }
}
if (!(p = realloc(p, (pPairCnt+2) * sizeof(point_t) * 2)) {
    err("Cannot allocate memory for points storage when generting outer box
(errno=%d)\n", errno);
    return EXIT_FAILURE;
}
p[pPairCnt+0][0] = (point_t){.sx = sol, .sy = sot, .rx = rol, .ry = rot};
p[pPairCnt+0][1] = (point_t){.sx = sor, .sy = sot, .rx = nor, .ry = rot};
p[pPairCnt+1][0] = (point_t){.sx = sol, .sy = sob, .rx = rol, .ry = rob};
p[pPairCnt+1][1] = (point_t){.sx = sor, .sy = sob, .rx = nor, .ry = rob};
info("POINT: TL pos (%.4f,%.4f) @ loc (%.4f,%.4f)\n", p[pPairCnt+0][0].sx,
p[pPairCnt+0][0].sy, p[pPairCnt+0][0].rx, p[pPairCnt+0][0].ry);
info("POINT: TR pos (%.4f,%.4f) @ loc (%.4f,%.4f)\n", p[pPairCnt+0][1].sx,
p[pPairCnt+0][1].sy, p[pPairCnt+0][1].rx, p[pPairCnt+0][1].ry);
info("POINT: BL pos (%.4f,%.4f) @ loc (%.4f,%.4f)\n", p[pPairCnt+1][0].sx,
p[pPairCnt+1][0].sy, p[pPairCnt+1][0].rx, p[pPairCnt+1][0].ry);
info("POINT: BR pos (%.4f,%.4f) @ loc (%.4f,%.4f)\n", p[pPairCnt+1][1].sx,
p[pPairCnt+1][1].sy, p[pPairCnt+1][1].rx, p[pPairCnt+1][1].ry);
pPairCnt += 2;

header.pCnt = pPairCnt * 2;
header.fileSize = sizeof(header_t) + header.width * header.height *
(sizeof(data1_t) + sizeof(data2_t)) + header.pCnt * sizeof(point_t);
}

/* Find road data */ {

/* Perspective */ {
    info("Generate perspective view roadmap\n");

```

```

float csl[2], csr[2], crl[2], crr[2], cry[2], crx[2]; //Coefficient
screen/road left-x/right-x of focus region mesh, road y relative to screen y-
coord, coefficient road x to screen x
unsigned int currentPointIdx = -1; //0xFFFFFFFF...
for (unsigned int y = 0; y < header.height; y++) {
    float yNorm = (float)y / header.height; //Normalized y-coord
    unsigned int requestPointIndex;
    if (yNorm < p[0][0].sy) {
        requestPointIndex = 0;
    } else if (yNorm > p[header.pCnt/2-3][0].sy) {
        requestPointIndex = header.pCnt/2 - 4;
    } else {
        for (unsigned int i = 0; i < header.pCnt/2 - 4; i++) {
            if (isBetween(yNorm, p[i][0].sy, p[i+1][0].sy) >= 0) {
                requestPointIndex = i;
                break;
            }
        }
    }
}

if (currentPointIdx != requestPointIndex) {
    unsigned int up = requestPointIndex, down = requestPointIndex + 1;
    //      info("Y pos %u/%.4f loc %.4f): Update: point_up: idx%u %.4f;
point_down: idx%u %.4f\n", y, yNorm, (float)y / size.height,
requestPointIndex, p[up][0].sy, requestPointIndex+1, p[down][0].sy);
    point_t pTL = p[up][0], pTR = p[up][1];
    point_t pBL = p[down][0], pBR = p[down][1];
    polyFit2(pTL.sy, pBL.sy, pTL.sx, pBL.sx, csl); //Equation: left
point screen x-coord relative to screen y-coord (left of the focus region
mesh)
    polyFit2(pTR.sy, pBR.sy, pTR.sx, pBR.sx, csr); //Right point screen
x-coord relative to screen y-coord
    polyFit2(pTL.sy, pBL.sy, pTL.rx, pBL.rx, crl); //Left point road x-
coord relative to screen y-coord
    polyFit2(pTR.sy, pBR.sy, pTR.rx, pBR.rx, crr); //Right point road x-
coord relative to screen y-coord
    polyFit2(pTL.sy, pBL.sy, pTL.ry, pBL.ry, cry); //Road y-ccord
relative to screen y-coord
    currentPointIdx = requestPointIndex;
}

float slNorm = yNorm * csl[1] + csl[0], rl = yNorm * crl[1] + crl[0];
float srNorm = yNorm * csr[1] + csr[0], rr = yNorm * crr[1] + crr[0];
float ry = yNorm * cry[1] + cry[0];
unsigned int sl = slNorm * header.width, sr = srNorm * header.width;

```

```

        if (y % 16 == 0) info("Y pos %u/%.4f loc %.4f, left at pos %u/%.4f
loc %.4f, right pos %u/%.4f loc %.4f\n", y, yNorm, ry, sl, slNorm, rl, sr,
srNorm, rr);
        polyFit2(slNorm, srNorm, rl, rr, crx); //Road x-coord relative to
screen x-coord
        for (unsigned int x = 0; x < header.width; x++) {
            float xNorm = (float)x / header.width; //Normalized x-coord
            unsigned int i = y * header.width + x;
            t1[i].px = crx[1] * xNorm + crx[0];
            t1[i].py = ry;
        }
    }
}

/* Orthographic */ {
    info("Generate orthographic view roadmap\n");
    float cx[2];
    unsigned int baselineY = p[header.pCnt/2-1][1].sy * header.height;
    float baselineLeft = t1[ baselineY * header.width + 0 ].px,
baselineRight = t1[ baselineY * header.width + header.width - 1 ].px;
    polyFit2(0, header.width - 1, baselineLeft, baselineRight, cx);
    for (unsigned int y = 0; y < header.height; y++) {
        for (unsigned int x = 0; x < header.width; x++) {
            unsigned int i = y * header.width + x;
            t1[i].ox = x * cx[1] + cx[0];
            /* Y-axis in orthographic view is same as in perspective view (we
project in x-axis only) */
        }
    }
}

/* Perspective pixel width */ {
    for (unsigned int y = 0; y < header.height; y++) {
        unsigned int i = y * header.width;
        unsigned int j = i + 1;
        t1[i].pw = t1[i+1].px - t1[i].px;
        for (unsigned int k = header.width - 1; k; k--) {
            t1[i].pw = t1[i+1].px - t1[i].px;
            i++;
        }
    }
}

/* Calculate search distance */ {

```

```

    info("Calculate search distance\n");
    /* Notes:
     * 1 - Use limit instead of distance:
     * Some GL driver (eg RPI) support texelFetchOffset() with const offset
    only; therefore GLSL code
     * "for (ivec2 OFFSET; ...; offset += DISTANCE) { xxx =
    texelFetchOffset(sampler, base, lod, OFFSET); }"
     * will not work. Although
     * "for (ivec2 OFFSET; ...; offset += DISTANCE) { xxx =
    texelFetchOffset(sampler, base + OFFSET, lod); }"
     * does work, but introduce extra computation per iteration. Then we have
    to use GLSL code
     * "for (ivec2 IDX; idx < LIMIT; idx++) { xxx = texelFetch(sampler, IDX,
    lod); }".
     * 2 - Use NTC instead of absolute coord:
     * For some reasons, RPI seems not work with isampler.
     * 3 - Let the program decide the limit:
     * The search distance is the maximum distance that an object could
    reasonably travel between two frames.
     * Since the input video FPS is not known, it doesn't make sense defining
    the limit now. However, what we
     * know now is the up and down edge of the focus region.
    */
    float limitLeft = p[header.pCnt/2-2][0].sx, limitRight = p[header.pCnt/2-
    2][1].sx, limitUp = p[header.pCnt/2-2][1].sy, limitDown = p[header.pCnt/2-
    1][1].sy;
    info("Limit: left %.4f right %.4f; top %.4f bottom %.4f\n", limitLeft,
    limitRight, limitUp, limitDown);
    data2_t* ptr = t2;
    for (unsigned int y = 0; y < header.height; y++) {
        float yNorm = (float)y / header.height;
        if ( yNorm < limitUp || yNorm > limitDown ) {
            for (unsigned int x = header.width; x; x--) {
                ptr->searchLimitUp = yNorm;
                ptr->searchLimitDown = yNorm;
                ptr++;
            }
        } else {
            for (unsigned int x = 0; x < header.width; x++) {
                float xNorm = (float)x / header.width;
                if ( xNorm < limitLeft || xNorm > limitRight ) {
                    ptr->searchLimitUp = yNorm;
                    ptr->searchLimitDown = yNorm;
                } else {
                    ptr->searchLimitUp = limitUp;

```

```

        ptr->searchLimitDown = limitDown;
    }
    ptr++;
}
}
}
}

/* Find lookup table */ {
    info("Finding view transform lookup table\n");
    for (unsigned int y = 0; y < header.height; y++) {
        unsigned int left = y * header.width, right = left + header.width - 1;
        for (unsigned int x = 0; x < header.width; x++) {
            unsigned int current = left + x;

            /* Perspective to Orthographic */ {
                float road = t1[current].ox;
                if (road <= t1[left].px) {
                    t2[current].lookupXp2o = 0.0;
                } else if (road >= t1[right].px) {
                    t2[current].lookupXp2o = 1.0;
                } else {
                    for (unsigned int i = 0; i < header.width - 2; i++) {
                        int check = isBetween(road, t1[left+i].px, t1[left+i+1].px);
                        if (check != -1) {
                            t2[current].lookupXp2o = (float)(i + check) / header.width;
                            break;
                        }
                    }
                }
            }

            /* Orthographic to Perspective */ {
                float road = t1[current].px;
                if (road <= t1[left].ox) {
                    t2[current].lookupXo2p = 0;
                } else if (road >= t1[right].ox) {
                    t2[current].lookupXo2p = 1.0;
                } else {
                    for (unsigned int i = 0; i < header.width - 2; i++) {
                        int check = isBetween(road, t1[left+i].ox, t1[left+i+1].ox);
                        if (check != -1) {
                            t2[current].lookupXo2p = (float)(i + check) / header.width;
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    }
    }
}

/* Write to file */ {
    info("Writing to file\n");
    fwrite(&header, sizeof(header), 1, stdout);
    fwrite(t1, sizeof(data1_t), header.height * header.width, stdout);
    fwrite(t2, sizeof(data2_t), header.height * header.width, stdout);
    fwrite(p, sizeof(point_t), header.pCnt, stdout);

    fputs("\n\n== Metadata:
=====\\n",
stdout);
    fprintf(stdout, "Camera resolution: %upx x %upx (%usqpx)\\n", header.width,
header.height, header.width * header.height);
    for (size_t i = 0; i < header.pCnt / 2; i++) {
        point_t left = p[i][0];
        point_t right = p[i][1];
        unsigned int sxl = left.sx * header.width, sxr = right.sx *
header.width, sy = left.sy * header.height;
        fprintf(stdout, "POINT: PL pos (%.4f,%.4f) <%upx,%upx> @ loc
(%.4f,%.4f)\\n", left.sx, left.sy, sxl, sy, left.rx, left.ry);
        fprintf(stdout, "POINT: PR pos (%.4f,%.4f) <%upx,%upx> @ loc
(%.4f,%.4f)\\n", right.sx, right.sy, sxr, sy, right.rx, right.ry);
    }
}

info("Done!\\n");
return EXIT_SUCCESS;
}

```

A text file containing road points should be fed to this program. An example text file is given below:

```

POINT 0.4176, 0.4724, 0.6208, 156.3, -10.0, 12.0
POINT 0.4259, 0.4698, 0.6276, 147.5, -10.0, 12.0
POINT 0.4306, 0.4682, 0.6323, 141.3, -10.0, 12.0
POINT 0.4361, 0.4661, 0.6375, 135.0, -10.0, 12.0
POINT 0.4426, 0.4641, 0.6432, 129.0, -10.0, 12.0
POINT 0.4500, 0.4615, 0.6484, 122.5, -10.0, 12.0
POINT 0.4583, 0.4583, 0.6557, 116.3, -10.0, 12.0
POINT 0.4667, 0.4557, 0.6620, 110.0, -10.0, 12.0

```

```

POINT 0.4769, 0.4526, 0.6698, 103.7, -10.0, 12.0
POINT 0.4880, 0.4490, 0.6776, 97.5, -10.0, 12.0
POINT 0.5009, 0.4453, 0.6880, 91.3, -10.0, 12.0
POINT 0.5157, 0.4396, 0.7010, 85.0, -10.0, 12.0
POINT 0.5324, 0.4339, 0.7125, 79.0, -10.0, 12.0
POINT 0.5528, 0.4260, 0.7302, 72.5, -10.0, 12.0
POINT 0.5769, 0.4161, 0.7490, 66.3, -10.0, 12.0
POINT 0.6028, 0.4057, 0.7708, 60.0, -10.0, 12.0
POINT 0.6324, 0.3932, 0.7943, 53.7, -10.0, 12.0
POINT 0.6769, 0.3745, 0.8313, 47.5, -10.0, 12.0
POINT 0.7370, 0.3479, 0.8766, 41.0, -10.0, 12.0
POINT 0.8213, 0.3115, 0.9490, 34.5, -10.0, 12.0

```

To generate the road map, execute command:

```
./a.out 1280 720 < on3.txt > ../../on3-720.map
```

Appendix A.3 Roadmap Generator – Trigonometry Based

This C program is used to generate a roadmap file based on camera FOV, height and pitch angle. Trigonometry is used to calculate the data on each road point.

```

/** This program is used to generate roadmap file.
 * See the readme.md for detail.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <inttypes.h>
#include <errno.h>
#include <math.h>
#include <string.h>

/** Output file format:
 * Section I: File header.
 * Section II: Table 1: Road-domain geographic data in perspective and
orthographic views.
 * Section III: Table 2: Search distance, perspective and orthographic
projection map.
 * Section IV: Focus region points.
 * Section V: Meta data: ASCII meta data, for reference, ignored by program.
 */

// Section I: File header
typedef struct FileHeader {
    uint32_t width, height; //Frame size in pixel

```

```

    uint32_t pCnt; //Number of road points (perspective view)
    uint32_t fileSize; //Size of roadmap file without meta data, in byte
} header_t;

// Section II: Table 1: Road-domain geographic data in perspective and
orthographic views
typedef struct FileDataTable1 {
    float px, py; //Road-domain geographic data in perspective view
    float ox; //Road-domain geographic data in orthographic view, y-coord of
orthographic view is same as y-coord of perspective view
    float pw; //Perspective pixel width
} data1_t;

// Section III: Table 2: Search distance, perspective and orthographic
projection map
typedef struct FileDataTable2 {
    float searchLimitUp, searchLimitDown; //Up/down search limit y-coord
    float lookupXp2o, lookupXo2p; //Projection lookup table. Y-coord in
orthographic and perspective views are same
} data2_t;

// Section IV: Focus region points
typedef struct Point_t {
    float rx, ry; //Road-domain location (m)
    float sx, sy; //Screen domain position (NTC)
} point_t;

#define DELI "%[^0-9.-]" //Delimiter
#define err(format, ...) {fprintf(stderr, "ERROR: "format __VA_OPT__(,)
__VA_ARGS__);}
#define info(format, ...) {fprintf(stderr, "INFO: "format __VA_OPT__(,)
__VA_ARGS__);}

//Return 0 if X is closer to A, 1 if X is closer to B, -1 if X is smaller than
A or greater than B
int isBetween(float x, float a, float b) {
    if (x < a || x > b)
        return -1;
    return ((x - a) <= (b - x)) ? 0 : 1;
}

//Find the first-order equation to describe the line connect two points: y = x
* cof[1] + cof[0]
void polyFit2(float x1, float x2, float y1, float y2, float* cof) {
    cof[1] = (y2 - y1) / (x2 - x1);

```

```

    cof[0] = y1 - x1 * cof[1];
}

//Read a line of string, return 0 if token not found, 1 if token found and
//argument count matched, -1 if argument count doesn't match
int readline(const char* src, const char* token, const unsigned int argc,
const char* format, ...) {
    if (strncmp(token, src, strlen(token)))
        return 0;

    int result = 1; //Success
    va_list args;
    va_start(args, format);
    if (vsscanf(src, format, args) != argc) //If mismatch
        result = -1; //Change to fail
    va_end(args);

    return result;
}

header_t header;
data1_t* t1;
data2_t* t2;
point_t(* p)[2];
struct {
    float fov[2]; //in deg
    float height, pitch; //in meter and deg
} camera = {
    .fov = {60.0f, 45.0f},
    .height = 10.0f,
    .pitch = 0.0f
};

void new() __attribute__((constructor));
void new() {
    header.width = 0;
    header.height = 0;
    header.pCnt = 0;
    header.fileSize = 0;
    t1 = NULL;
    t2 = NULL;
    p = NULL;
}

void destroy() __attribute__((destructor));

```

```

void destroy() {
    free(t1);
    free(t2);
    free(p);
}

int main(int argc, char* argv[]) {
    if (argc != 3) {
        err("ERROR: Bad arg: Use this width height < info.txt > output.map\n");
        err("\twidth and height is the size of frame in px\n");
        err("\tinfo.txt (stdin): an ASCII file contains human-readable road\n");
        err("\toutput.map (stdout): a binary roadmap file\n");
        return EXIT_FAILURE;
    }

    /* Get size */ {
        info("Get frame size and allocate buffer for tables\n");
        header.width = atoi(argv[1]);
        header.height = atoi(argv[2]);

        if (header.width > 2048 || header.width < 320 || (unsigned
int)header.width & (unsigned int)0b111) { //float16 interval > 0 if value >
2048 (abs)
            err("Width cannot be greater than 2048 or less than 320, and must be
multiply of 8\n");
            return EXIT_FAILURE;
        }
        if (header.height > 2048 || header.height < 240 || (unsigned
int)header.height & (unsigned int)0b111) {
            err("Height cannot be greater than 2048 or less than 240, and must be
multiply of 8\n");
            return EXIT_FAILURE;
        }
        /* Note:
        Float16 use 10 bits frac.
        For absolute coord, when value > 2048, interval > 1.
        For NTC, interval = 2^-11, which is 1/2048 resolution.
        */

        t1 = malloc(sizeof(data1_t) * header.height * header.width);
        t2 = malloc(sizeof(data2_t) * header.height * header.width);
        if (!t1 || !t2) {
            err("Cannot allocate memory for table storage (errno=%d)\n", errno);
            return EXIT_FAILURE;
        }
    }
}

```

```

}

/* Get config from stdin */ {
    info("Process road points\n");
    char buffer[500];
    int res; //Result of readline() 0 = token not match, 1 = ok, -1 = argument
count mismatch
    float sy, sxl, sxr; //Screen point y/x-left/x-right coord
    unsigned pPairCnt = 0;

    /* Road points - read for persp*/
    while (fgets(buffer, sizeof(buffer), stdin)) {
        if (res = readline(buffer, "POINT", 3, DELI"%f"DELI"%f"DELI"%f", &sy,
&sxl, &sxr)) {
            if (res == -1) {
                err("Cannot read POINT: bad format, require
'...float...float...float'\n");
                return EXIT_FAILURE;
            }
            if (!(p = realloc(p, (pPairCnt+1) * sizeof(point_t) * 2)) ) {
                err("Cannot allocate memory for points storage when reading info
file (errno=%d)\n", errno);
                return EXIT_FAILURE;
            }

            point_t* current = p[pPairCnt];
            current[0] = (point_t){.sx = sxl, .sy = sy, .rx = 0, .ry = 0};
            current[1] = (point_t){.sx = sxr, .sy = sy, .rx = 0, .ry = 0};
            unsigned int sxl = current[0].sx * header.width, sxr = current[1].sx *
header.width, sy = current[0].sy * header.height;
            info("POINT: PL pos (%.4f,%.4f) <%upx,%upx> PR pos (%.4f,%.4f)
<%upx,%upx>\n", current[0].sx, current[0].sy, sxl, sy, current[1].sx,
current[1].sy, sxr, sy);

            if (pPairCnt) { //This check does not apply on first road point pair
                point_t* previous = p[pPairCnt - 1];
                if (current[0].sy <= previous[0].sy) {
                    err("POINT %u - Screen y-coord should be in top-to-bottom order,
current y-coord (%.4f) should be greater than last y-coord (%.4f)\n",
pPairCnt, current[0].sy, previous[0].sy);
                    return EXIT_FAILURE;
                }
                if (current[1].sx <= current[0].sx) {
                    err("POINT %u - Right point (%.4f) should have greater screen x-
coord than left point (%.4f)\n", pPairCnt, current[1].sx, current[0].sx);

```

```

        return EXIT_FAILURE;
    }
}

    pPairCnt++;
} else if (res = readline(buffer, "FOV", 2, DELI"%f"DELI"%f",
camera.fov, camera.fov+1)) {
    if (res == -1) {
        err("Cannot read FOV: bad format, require '...float...float'\n");
        return EXIT_FAILURE;
    }
    info("FOV = {%f, %f}\n", camera.fov[0], camera.fov[1]);
} else if (res = readline(buffer, "POS", 2, DELI"%f"DELI"%f",
&camera.height, &camera.pitch)) {
    if (res == -1) {
        err("Cannot read POS: bad format, require '...float...float'\n");
        return EXIT_FAILURE;
    }
    info("Height = %f, pitch = %f\n", camera.height, camera.pitch);
} else {
    err("Unknown info: %s\n", buffer);
    return EXIT_FAILURE;
}
}

/* Road points - gen for ortho */
float sol = p[0][0].sx, sor = p[0][0].sx, sot = p[0][0].sy, sob =
p[0][0].sy;
for (unsigned int i = 0; i < pPairCnt; i++) {
    if (p[i][0].sx < sol) {
        sol = p[i][0].sx;
    }
    if (p[i][1].sx > sor) {
        sor = p[i][1].sx;
    }
    if (p[i][1].sy < sot) {
        sot = p[i][1].sy;
    }
    if (p[i][1].sy > sob) {
        sob = p[i][1].sy;
    }
}
if (!(p = realloc(p, (pPairCnt+2) * sizeof(point_t) * 2)) ) {
    err("Cannot allocate memory for points storage when generting outer box
(errno=%d)\n", errno);
}

```

```

        return EXIT_FAILURE;
    }
    p[pPairCnt+0][0] = (point_t){.sx = sol, .sy = sot};
    p[pPairCnt+0][1] = (point_t){.sx = sor, .sy = sot};
    p[pPairCnt+1][0] = (point_t){.sx = sol, .sy = sob};
    p[pPairCnt+1][1] = (point_t){.sx = sor, .sy = sob};
    info("POINT: TL pos (%.4f,%.4f)\n", p[pPairCnt+0][0].sx,
p[pPairCnt+0][0].sy);
    info("POINT: TR pos (%.4f,%.4f)\n", p[pPairCnt+0][1].sx,
p[pPairCnt+0][1].sy);
    info("POINT: BL pos (%.4f,%.4f)\n", p[pPairCnt+1][0].sx,
p[pPairCnt+1][0].sy);
    info("POINT: BR pos (%.4f,%.4f)\n", p[pPairCnt+1][1].sx,
p[pPairCnt+1][1].sy);
    pPairCnt += 2;

    header.pCnt = pPairCnt * 2;
    header.fileSize = sizeof(header_t) + header.width * header.height *
(sizeof(data1_t) + sizeof(data2_t)) + header.pCnt * sizeof(point_t);
}

/* Find road data */ {

    /* Perspective */ {
        info("Generate perspective view roadmap\n");
        double pitchTop = (90.0 + camera.pitch + 0.5 * camera.fov[1]) * M_PI /
180;
        double pitchBottom = (90.0 + camera.pitch - 0.5 * camera.fov[1]) * M_PI
/ 180;
        double pitchStep = (pitchBottom - pitchTop) / (header.height - 1);
        double pitchCurrent = pitchTop;

        data1_t* ptr = t1;
        for (int y = 0; y < header.height; y++) {
            double yawSpan = camera.height / cos(pitchCurrent) * sin(0.5 *
camera.fov[0] * M_PI / 180); //Half
            double yawStep = 2 * yawSpan / (header.width - 1);
            double yawCurrent = -yawSpan;
            double posY = tan(pitchCurrent) * camera.height;
            for (int x = 0; x < header.width; x++) {
                *(ptr++) = (data1_t){
                    .px = yawCurrent,
                    .py = posY
                };
                yawCurrent += yawStep;
            }
        }
    }
}

```

```

    }
    pitchCurrent += pitchStep;
}

/* Orthographic */ {
    info("Generate orthographic view roadmap\n");
    float cx[2];
    unsigned int baselineY = p[header.pCnt/2-1][1].sy * header.height;
    float baselineLeft = t1[ baselineY * header.width + 0 ].px,
baselineRight = t1[ baselineY * header.width + header.width - 1 ].px;
    polyFit2(0, header.width - 1, baselineLeft, baselineRight, cx);
    for (unsigned int y = 0; y < header.height; y++) {
        for (unsigned int x = 0; x < header.width; x++) {
            unsigned int i = y * header.width + x;
            t1[i].ox = x * cx[1] + cx[0];
            /* Y-axis in orthographic view is same as in perspective view (we
project in x-axis only) */
        }
    }
}

/* Perspective pixel width */ {
    for (unsigned int y = 0; y < header.height; y++) {
        unsigned int i = y * header.width;
        unsigned int j = i + 1;
        t1[i].pw = t1[i+1].px - t1[i].px;
        for (unsigned int k = header.width - 1; k; k--) {
            t1[i].pw = t1[i+1].px - t1[i].px;
            i++;
        }
    }
}

/* Calculate search distance */ {
    info("Calculate search distance\n");
    /* Notes:
    * 1 - Use limit instead of distance:
    * Some GL driver (eg RPI) support texelFetchOffset() with const offset
only; therefore GLSL code
    * "for (ivec2 OFFSET; ...; offset += DISTANCE) { xxx =
texelFetchOffset(sampler, base, lod, OFFSET); }"
    * will not work. Although

```

```

    * "for (ivec2 OFFSET; ...; offset += DISTANCE) { xxx =
texelFetchOffset(sampler, base + OFFSET, lod); }"
    * does work, but introduce extra computation per iteration. Then we have
to use GLSL code
    * "for (ivec2 IDX; idx < LIMIT; idx++) { xxx = texelFetch(sampler, IDX,
lod); }".
    * 2 - Use NTC instead of absolute coord:
    * For some reasons, RPI seems not work with isampler.
    * 3 - Let the program decide the limit:
    * The search distance is the maximum distance that an object could
reasonably travel between two frames.
    * Since the input video FPS is not known, it doesn't make sense defining
the limit now. However, what we
    * know now is the up and down edge of the focus region.
    */
    float limitLeft = p[header.pCnt/2-2][0].sx, limitRight = p[header.pCnt/2-
2][1].sx, limitUp = p[header.pCnt/2-2][1].sy, limitDown = p[header.pCnt/2-
1][1].sy;
    info("Limit: left %.4f right %.4f; top %.4f bottom %.4f\n", limitLeft,
limitRight, limitUp, limitDown);
    data2_t* ptr = t2;
    for (unsigned int y = 0; y < header.height; y++) {
        float yNorm = (float)y / header.height;
        if ( yNorm < limitUp || yNorm > limitDown ) {
            for (unsigned int x = header.width; x; x--) {
                ptr->searchLimitUp = yNorm;
                ptr->searchLimitDown = yNorm;
                ptr++;
            }
        } else {
            for (unsigned int x = 0; x < header.width; x++) {
                float xNorm = (float)x / header.width;
                if ( xNorm < limitLeft || xNorm > limitRight ) {
                    ptr->searchLimitUp = yNorm;
                    ptr->searchLimitDown = yNorm;
                } else {
                    ptr->searchLimitUp = limitUp;
                    ptr->searchLimitDown = limitDown;
                }
                ptr++;
            }
        }
    }
}

```

```

/* Find lookup table */ {
    info("Finding view transform lookup table\n");
    for (unsigned int y = 0; y < header.height; y++) {
        unsigned int left = y * header.width, right = left + header.width - 1;
        for (unsigned int x = 0; x < header.width; x++) {
            unsigned int current = left + x;

            /* Perspective to Orthographic */ {
                float road = t1[current].ox;
                if (road <= t1[left].px) {
                    t2[current].lookupXp2o = 0.0;
                } else if (road >= t1[right].px) {
                    t2[current].lookupXp2o = 1.0;
                } else {
                    for (unsigned int i = 0; i < header.width - 2; i++) {
                        int check = isBetween(road, t1[left+i].px, t1[left+i+1].px);
                        if (check != -1) {
                            t2[current].lookupXp2o = (float)(i + check) / header.width;
                            break;
                        }
                    }
                }
            }

            /* Orthographic to Perspective */ {
                float road = t1[current].px;
                if (road <= t1[left].ox) {
                    t2[current].lookupXo2p = 0;
                } else if (road >= t1[right].ox) {
                    t2[current].lookupXo2p = 1.0;
                } else {
                    for (unsigned int i = 0; i < header.width - 2; i++) {
                        int check = isBetween(road, t1[left+i].ox, t1[left+i+1].ox);
                        if (check != -1) {
                            t2[current].lookupXo2p = (float)(i + check) / header.width;
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

```

/* Write to file */ {

```

```

    info("Writing to file\n");
    fwrite(&header, sizeof(header), 1, stdout);
    fwrite(t1, sizeof(data1_t), header.height * header.width, stdout);
    fwrite(t2, sizeof(data2_t), header.height * header.width, stdout);
    fwrite(p, sizeof(point_t), header.pCnt, stdout);

    fputs("\n\n== Metadata:
=====\\n",
stdout);
    fprintf(stdout, "Camera resolution: %upx x %upx (%usqpx)\\n", header.width,
header.height, header.width * header.height);
    fprintf(stdout, "Camera install height: %.2f m, pitch %.2f deg,
FOV %.2fdeg x %.2fdeg\\n", camera.height, camera.pitch, camera.fov[0],
camera.fov[1]);
    for (size_t i = 0; i < header.pCnt / 2; i++) {
        point_t left = p[i][0];
        point_t right = p[i][1];
        unsigned int sx1 = left.sx * header.width, sxr = right.sx *
header.width, sy = left.sy * header.height;
        fprintf(stdout, "POINT: PL pos (%.4f,%.4f) <%upx,%upx> @ loc
(%.4f,%.4f)\\n", left.sx, left.sy, sx1, sy, left.rx, left.ry);
        fprintf(stdout, "POINT: PR pos (%.4f,%.4f) <%upx,%upx> @ loc
(%.4f,%.4f)\\n", right.sx, right.sy, sxr, sy, right.rx, right.ry);
    }
}

info("Done!\\n");
return EXIT_SUCCESS;
}

```

A text file containing road points should be fed to this program. An example text file is given below:

```

FOV      60.0, 35.0
POS      9.5, -17.5
POINT    0.0472, 0.5016, 0.5484
POINT    0.9833, 0.0500, 0.9500

```

To generate the road map, execute command:

```
./a.out 1280 720 < 403s.txt > ../../403s-720.map
```

Appendix B Main program

The following are code files used to compile the main program. All code files must be placed in the same directory. GCC comes with Ubuntu 20.04 desktop version is used. In addition, GLEW and GLFW are used.

To compile the main program, execute command:

```
gcc *.c -D_FILE_OFFSET_BITS=64 -DVERBOSE -O3 -IX11 -IGLEW -IGL -lglfw3 -lthread -lm -ldl
```

Following command will execute this program:

```
./a.out 1280 720 10 40123 ../on3-720.map
```

Where “./a.out” is the executable program; “1280 720 10” gives the resolution and FPS of the input video; “40123” tells the reader thread the color scheme of the input video (4 channels, red the first, green the second, blue the third and alpha the fourth. For BGR, use 3210: 3 channels, red the third, green the second, blue the first); “../on3-720.map” is the roadmap file. Executing this program will create and listen to a named pipe called “tmpframefifo.data”. Use another command to feed the video file to this pipe, example command is shown below:

```
pv ../on3-720v10.data > tmpframefifo.data
```

Appendix B.1.1 common.h

```
/** Common config and define
 */

#ifndef INCLUDE_COMMON_H
#define INCLUDE_COMMON_H

#include <inttypes.h>

/** Size of 2D object (image): width and height in pixel
 */
typedef struct Size2D_t {
    union { size_t width; size_t x; };
    union { size_t height; size_t y; };
} size2d_t;

/** Size of 3D object (image): width, height and depth
 */
typedef struct Size3D_t {
    union { size_t width; size_t x; };
```

```

    union { size_t height; size_t y; };
    union { size_t depth; size_t z; };
} size3d_t;

/** Float/Int vectors (GPU types)
 */
typedef struct IntVec1 {
    union { int width; int x; int r; };
} ivec1;
typedef struct IntVec2 {
    union { int width; int x; int r; };
    union { int height; int y; int g; };
} ivec2;
typedef struct IntVec3 {
    union { int width; int x; int r; };
    union { int height; int y; int g; };
    union { int depth; int z; int b; };
} ivec3;
typedef struct IntVec4 {
    union { int width; int x; int r; };
    union { int height; int y; int g; };
    union { int depth; int z; int b; };
    union { int time; int w; int a; };
} ivec4;
typedef struct Vec1 {
    union { float width; float x; float r; };
} vec1;
typedef struct Vec2 {
    union { float width; float x; float r; };
    union { float height; float y; float g; };
} vec2;
typedef struct Vec3 {
    union { float width; float x; float r; };
    union { float height; float y; float g; };
    union { float depth; float z; float b; };
} vec3;
typedef struct Vec4 {
    union { float width; float x; float r; };
    union { float height; float y; float g; };
    union { float depth; float z; float b; };
    union { float time; float w; float a; };
} vec4;

/** Get length of an array (static allocated)
 * @param array The array

```

```

    * @return Number of elements
    */
#define arrayLength(array) ( sizeof(array) / sizeof(array[0]) )

/** Get the current machine time in nanosecond
    * @return Current time
    */
uint64_t nanotime();

/** Check if a point is inside a box.
    * @param x Test point x-coord
    * @param y Test poiny y-coord
    * @param left Left x-coord of the box
    * @param right Right x-coord of the box
    * @param top Top y-coord of the box
    * @param bottom Bottom y-coord of the box
    * @param strict If 0, point is considered inside if on border; if positive,
    ptr is outside if on border; if negative, ptr is inside if on left-top border
    but outside if on right-bottom border (similar to 2D for loop checking)
    * @return True-equivalent (in most case 1) if point is inside; false-
    equivalent (0) if point is outside
    */
int inBox(const int x, const int y, const int left, const int right, const int
top, const int bottom, const int strict);

#endif /* #ifndef INCLUDE_COMMON_H */

```

Appendix B.1.2 common.c

```

#include <time.h>
#include <stdlib.h>
#include <string.h>

#include "common.h"

uint64_t nanotime() {
    struct timespec t;
    clock_gettime(CLOCK_MONOTONIC, &t);
    return t.tv_sec * 1000000000LLU + t.tv_nsec;
}

int inBox(const int x, const int y, const int left, const int right, const int
top, const int bottom, const int strict) {
    if (strict == 0)
        return x >= left && x <= right && y >= top && y <= bottom;
    if (strict > 0)
        return x > left && x < right && y > top && y < bottom;
    return x >= left && x < right && y >= top && y < bottom;
}

```

Appendix B.1.3a gl.h

```
/** Class - GL.class. Captdam's OpenGL helper class.
 * Start, load, config, manage and destroy OpenGL engine.
 * GLFW and GLEW are used as backend of this class.
 * Only one GL class and one window is allowed.
 * Use this class to access OpenGL engine.
 */

#ifndef INCLUDE_GL_H
#define INCLUDE_GL_H

/* == Common
=====
== */

/** Init config */
typedef struct GL_Config {
    uint8_t vMajor;      //OpenGL version major
    uint8_t vMinor;      //OpenGL version minor
    uint8_t gles;        //OpenGL-ES (1) or OpenGL (0)
    uint8_t vsynch;      //Limit FPS to v-synch points
    uint16_t winWidth, winHeight; //Window size
    char* winName;        //Window name
} gl_config;

/** Usage frequency hint for driver */
typedef enum GL_Usage {
    gl_usage_stream = 0, //Set once then use for a few times
    gl_usage_static = 1, //Set once, use many times: More likely saved in GPU
    gl_usage_dynamic = 2, //Set and use rapidly: More likely saved in CPU memory
    gl_usage_placeholderEnd} gl_usage;

/* == Window and driver management
===== */

/** Cursor and window & framebuffer size */
typedef struct GL_WinsizeCursor {
    int winsize[2]; //Window size
    int framesize[2]; //Framebuffer size
    double curPos[2]; //Cursor pos in NTC
    int curPosWin[2]; //Cursor pos in px respect to window
    int curPosFrame[2]; //Cursor pos in px respect to framebuffer
} gl_winsizeNcursor;

typedef void* gl_synch; //Synch point
/** Synch status */
typedef enum GL_Synch_Status {
    gl_synch_error = -1,
    gl_synch_timeout = 0, //Command cannot be finished before timeout
    gl_synch_done = 1, //Command already finished before this call
    gl_synch_ok = 2 //Command finished before timeout
} gl_synch_status; //Synch status
```

```

/* == Shader and shader param data types, UBO
===== */

typedef unsigned int gl_program; //Shader program
#define GL_INIT_DEFAULT_PROGRAM (gl_program)0

typedef int gl_param; //Shader program argument handler
/** Shader program argument data type*/
typedef enum GL_DataType {
    gl_datatype_float = 0, //32-bit float
    gl_datatype_int = 1, //32-bit int
    gl_datatype_uint = 2, //32-bit unsigned int
    gl_datatype_placeholderEnd } gl_datatype;
typedef unsigned int gl_ubo; //Uniform buffer object
#define GL_INIT_DEFAULT_UBO (gl_ubo)0

/** Shader types */
typedef enum GL_ProgramSourceCodeType {
    gl_programSrcType_vertex = 0, //Vertex shader
    gl_programSrcType_fragment = 1, //Fragment shader
    gl_programSrcType_geometry = 2, //Geometry shader (optional)
    gl_programSrcType_placeholderEnd } gl_programSrcType;
/** Shader code source */
typedef enum GL_ProgramSourceCodeLocation {
    gl_programSrcLoc_mem = 0, //Source code content saved in memory
    gl_programSrcLoc_file = 1, //Source code content saved in file
    gl_programSrcLoc_placeholderEnd } gl_programSrcLoc;
/** Shader source code attributes */
typedef struct GL_ProgramSourceCode {
    const gl_programSrcType type; //Shader type: gl_programSrcType_*
    const gl_programSrcLoc loc; //Source code location: gl_programSrcLoc_*
    const char* str; //Source code content or the source code file directory
} gl_programSrc;

/** Shader program argument types */
typedef enum GL_ProgramArgType {
    gl_programArgType_normal = 0, //Uniform
    gl_programArgType_ubo = 1, //UBO
    gl_programArgType_placeholderEnd } gl_programArgType;
/** OpenGL shader program argument (to get the ID of uniform or UBO) */
typedef struct GL_ProgramArg {
    const gl_programArgType type; //Argument type: gl_programArgType_*
    const char* name; //Name of uniform/UBO in shader code
    gl_param id; //Pass-by-reference, the id will be returned back here
} gl_programArg;

/* == Mesh (vertices)
===== */

/** Mesh geometry type */
typedef unsigned int gl_meshmode;
#define gl_meshmode_points ((gl_meshmode)0x0000)
#define gl_meshmode_lines ((gl_meshmode)0x0001)
#define gl_meshmode_lineLoop ((gl_meshmode)0x0002)
#define gl_meshmode_lineStrip ((gl_meshmode)0x0003)

```

```

#define gl_meshmode_triangles ((gl_meshmode)0x0004)
#define gl_meshmode_triangleStrip ((gl_meshmode)0x0005)
#define gl_meshmode_triangleFan ((gl_meshmode)0x0006)
// #define gl_meshmode_quads ((gl_meshmode)0x0007)
// #define gl_meshmode_quadStrip ((gl_meshmode)0x0008)
// #define gl_meshmode_polygon ((gl_meshmode)0x0009)

/** Geometry objects (mesh) */
typedef struct GL_Mesh {
    unsigned int vao, vbo; //Vertex array object (the mesh), vertex buffer
    object (vertices form the mesh)
    unsigned int ebo, ibo; //Indices buffer object (for indexed draw); Instance
    object bound to this mesh for instanced draw
    unsigned int drawSize; //Number of vertices or indices in the mesh
    gl_meshmode mode;
} gl_mesh;
#define GL_INIT_DEFAULT_MESH (gl_mesh){0, 0, 0, 0, 0, 0}

typedef float gl_vertex_t; //Mesh vertices
typedef unsigned int gl_index_t; //Mesh index

/* == Texture, PBO for texture transfer and FBO for off-screen rendering
===== */

/** Texture data format */
typedef enum GL_TexFormat {
    gl_texformat_R8 = 0, gl_texformat_RG8, gl_texformat_RGB8,
    gl_texformat_RGBA8,
    gl_texformat_R8I, gl_texformat_RG8I, gl_texformat_RGB8I,
    gl_texformat_RGBA8I,
    gl_texformat_R8UI, gl_texformat_RG8UI, gl_texformat_RGB8UI,
    gl_texformat_RGBA8UI,
    gl_texformat_R16F, gl_texformat_RG16F, gl_texformat_RGB16F,
    gl_texformat_RGBA16F,
    gl_texformat_R16I, gl_texformat_RG16I, gl_texformat_RGB16I,
    gl_texformat_RGBA16I,
    gl_texformat_R16UI, gl_texformat_RG16UI, gl_texformat_RGB16UI,
    gl_texformat_RGBA16UI,
    gl_texformat_R32F, gl_texformat_RG32F, gl_texformat_RGB32F,
    gl_texformat_RGBA32F,
    gl_texformat_R32I, gl_texformat_RG32I, gl_texformat_RGB32I,
    gl_texformat_RGBA32I,
    gl_texformat_R32UI, gl_texformat_RG32UI, gl_texformat_RGB32UI,
    gl_texformat_RGBA32UI,
    gl_texformat_d16, gl_texformat_d24, gl_texformat_d32f,
    gl_texformat_d24s8, gl_texformat_d32fs8, gl_texformat_s8,
    gl_texformat_placeholderEnd} gl_texformat;

/** Texture type */
typedef unsigned int gl_textype;
#define gl_textype_2d ((gl_textype)0)
#define gl_textype_3d ((gl_textype)1)
#define gl_textype_2dArray ((gl_textype)2)
#define gl_textype_renderBuffer ((gl_textype)10)

```

```

/** Texture object */
typedef struct GL_Texture {
    unsigned int texture;
    unsigned int width, height;
    union { unsigned int depth; unsigned int layer; };
    gl_texformat format;
    gl_textype type : 8;
    unsigned int mipmap : 1;
} gl_tex;
#define GL_INIT_DEFAULT_TEX (gl_tex){0, 0, 0, 0, 0, 0, 0}

/** Texture constructor argument */
typedef unsigned int gl_tex_dimWrapping;
#define gl_tex_dimWrapping_repeat ((gl_tex_dimWrapping)0x2901)
#define gl_tex_dimWrapping_edge ((gl_tex_dimWrapping)0x812F)
#define gl_tex_dimWrapping_mirrorRepeat ((gl_tex_dimWrapping)0x8370)

typedef unsigned int gl_tex_dimFilter;
#define gl_tex_dimFilter_nearest ((gl_tex_dimFilter)0x2600)
#define gl_tex_dimFilter_linear ((gl_tex_dimFilter)0x2601)
#define gl_tex_dimFilter_nearestMipmapNearest ((gl_tex_dimFilter)0x2700)
#define gl_tex_dimFilter_linearMipmapNearest ((gl_tex_dimFilter)0x2701)
#define gl_tex_dimFilter_nearestMipmapLinear ((gl_tex_dimFilter)0x2702)
#define gl_tex_dimFilter_linearMipmapLinear ((gl_tex_dimFilter)0x2703)

typedef struct GL_Texture_Dim {
    unsigned int size; //Size in pixel
    gl_tex_dimWrapping wrapping;
} gl_tex_dim;

/** Pixel buffer object for texture data transfer */
typedef unsigned int gl_pbo;
#define GL_INIT_DEFAULT_PBO (gl_pbo)0

/** Framebuffer object for off-screen rendering */
typedef unsigned int gl_fbo;
#define GL_INIT_DEFAULT_FBO (gl_fbo)0

/** Framebuffer attachment types */
typedef enum GL_FramebufferAttachment {
    gl_fboattach_depth = -3, gl_fboattach_stencil = -2,
    gl_fboattach_depth_stencil = -1,
    gl_fboattach_color0 = 0
} gl_fboattach;

/** Framebuffer clear mask */
#define gl_frameBuffer_clearColor 0x4000
#define gl_frameBuffer_clearDepth 0x0100
#define gl_frameBuffer_clearStencil 0x0400
#define gl_frameBuffer_clearAll 0x4500

/* == Window and driver management
===== */

```

```

/** Init the GL class.
 * @return 1 if success, 0 if fail
 */
int gl_init(gl_config config) __attribute__((cold));

/** Get the GLFW window object so user program can use GLFW lib to access the
window.
 * @return GLFW window object
 */
void* gl_getWindow();

/** Call to start a render loop.
 * Process all GLFW window events.
 */
void gl_drawStart();

/** Get window and framebuffer size, get cursor position respect to window and
framebuffer size respectively.
 * @return Window and framebuffer size, cursor position
 */
gl_winsizeNcursor gl_getWinsizeCursor();

/** Set the drawing viewport.
 * @param offset Offset in px {x,y}
 * @param size Size in px {width,height}
 */
void gl_setViewport(const unsigned int offset[static 2], const unsigned int
size[static 2]);

/** Call at the end of a render loop.
 * @param title New window title (pass NULL to use old title)
 */
void gl_drawEnd(const char* const title);

/** Set or check the window close flag.
 * A close flag request the GL to terminate the window, but it may not be done
right away.
 * @param close Pass positive value to set the close flag, pass 0 to unset,
negative value will have no effect (only get the flag)
 * @return Close flag: 0 = close flag unset, non-zero = close flag set
 */
int gl_close(const int close);

/** Destroy the GL class, kill the viewer window and release all the GL
resource.
 */
void gl_destroy();

/** Switch to line mode
 * @param weight Set the line weight and turn on line mode; 0 to turn off line
mode
 */
void gl_lineMode(const unsigned int weight);

/** Force the GL driver to sync.

```

```

    * Calling thread will be blocked until all previous GL calls executed
    completely.
    */
void gl_await();

/** Request the GL driver to sync.
 * Request to empty the command queue. Calling thread will not be blocked.
 */
void gl_rsync();

/** Set a point in the GPU command queue for a later gl_syncWait() call.
 * @return A synch point
 */
gl_sync gl_syncSet();

/** Wait GPU command queue.
 * @param s A synch point previously returned by gl_syncSet()
 * @param timeout Timeout in nano seconds
 * @return Synch status
 */
gl_sync_status gl_syncWait(const gl_sync s, const uint64_t timeout);

/** Delete a synch point if no longer need.
 * @param s A synch point previously returned by gl_syncSet()
 */
void gl_syncDelete(const gl_sync s);

/* == Shader and shader param data types, UBO
===== */

/** Set the header of all shader program.
 * All shaders should begin with "#version". This function is used to set this
common header so it is not required everytime a shader is created.
 * This lib will not keep a copy of this string, so it is the application's
responsibility to make sure this string is live when creating shader programs.
 * @param header Pointer to the string
 */
void gl_program_setCommonHeader(const char* const header);

/** Create a shader program from file or memory.
 * @param src A string representing the program, use "fDIR" (e.g.
"f./myshader.glsl") for file, use "mSRC" (e.g. "m@VS\nlayout...") for in-
memory code
 * @param args A list of gl_programArg used to return the ID of each param.
The last gl_programArg should have a NULL .name to indicate the end of list
 * @return Shader program
 */
gl_program gl_program_load(char* src, gl_programArg* const args);

/** Check a shader program.
 * @param program A shader program previously returned by gl_program_load()
 * @return 1 if good, 0 if not
 */
int gl_program_check(const gl_program* const program);

```

```

/** Use a shader program (bind a shader program to current)
 * @param program A shader program previously returned by gl_program_load()
 */
void gl_program_use(const gl_program* const program);

/** Set parameter of the current shader program.
 * Call gl_program_use() to bind the shader program before set the parameter.
 * @param paramId ID of parameter previously returned by gl_program_load()
 * @param length Number of vector in a parameter (vecX, can be 1, 2, 3 or 4,
use 1 for non-verctor params, e.g. sampler)
 * @param type Type of the data: gl_datatype_*
 * @param data Pointer to the data to be pass
 */
void gl_program_setParam(const gl_param paramId, const unsigned int length,
const gl_datatype type, const void* data);

/** Delete a shader program, the shader program will be reset to
GL_INIT_DEFAULT_SHADER.
 * @param program A shader program previously returned by gl_program_load()
 */
void gl_program_delete(gl_program* const program);

/** Create a uniform buffer object (UBO) and bind it to a binding point.
 * @param bindingPoint Binding point to bind
 * @param size Size of memory allocating to the buffer, in bytes
 * @param usage A hint to the driver about the frequency of usage, can be
gl_usage_*
 * @return UBO
 */
gl_ubo gl_uniformBuffer_create(const unsigned int bindingPoint, const unsigned
int size, const gl_usage usage);

/** Check an UBO
 * @param ubo An UBO previously returned by gl_uniformBuffer_create()
 * @return 1 if good, 0 if not
 */
int gl_uniformBuffer_check(const gl_ubo* const ubo);

/** Bind a shader program's block param to a uniform buffer in the binding
point.
 * @param bindingPoint Binding point to bind, same as the one used with
gl_uniformBuffer_create()
 * @param program A shader program previously returned by gl_program_load()
 * @param paramId ID of parameter previously returned by gl_program_load()
 */
void gl_uniformBuffer_bindShader(const unsigned int bindingPoint, const
gl_program* const program, const gl_param paramId);

/** Update a portion of uniform buffer.
 * @param ubo An UBO previously returned by gl_uniformBuffer_create()
 * @param start Starting offset of the update in bytes
 * @param len Length of the update in bytes
 * @param data Pointer to the update data
 */

```

```

void gl_uniformBuffer_update(const gl_ubo* const ubo, const unsigned int
start, const unsigned int len, const void* const data);

/** Delete an UBO.
 * @param ubo An UBO previously created by gl_uniformBuffer_create()
 */
void gl_uniformBuffer_delete(gl_ubo* const ubo);

/* == Mesh (vertices)
===== */

/** Create and bind gl_mesh object.
 * Each mesh has vCnt vertices, each vertex has Sigma(vStride) of numbers.
 * The order of layout location in shader program must be vertices first, then
instances, no gap between.
 * @param vCnt Number of vertices, each vertices contains sum(vSize) gl_vertex
 * @param eCnt Number of indices (NOT face, indices/face for strip is
undefined) for indexed draw or 0 for non-indexed draw, an index is an gl_index
 * @param iCnt Max number of instances or 0 for non-instanced draw, each
instance contains sum(iSize) gl_vertex
 * @param mode The mode of the mesh, can be gl_meshmode_*, this affects how
the GPU draw the mesh
 * @param vSize Zero terminated array, size of each attribute (number of
gl_vertex_t) in a vertex
 * @param iSize Zero terminated array, size of each attribute (number of
gl_vertex_t) in a instance, ignored if iBufSize is 0
 * @param vertices The vertice array, leave NULL if no actual data is ready
 * @param indices The indices array, leave NULL if no actual data is ready,
ignored if eCnt is 0, use index -1 for restart in strip
 * @param shareInstance Share instance buffer with another gl_mesh object.
Useful for model with multiple meshes, instead of creating new buffer, the
same buffer data can be reused
 * @return Mesh
 */
gl_mesh gl_mesh_create(
    const unsigned int vCnt, const unsigned int eCnt, const unsigned int iCnt,
    const gl_meshmode mode,
    const gl_index_t* const vSize, const gl_index_t* const iSize,
    const gl_vertex_t* const vertices, const gl_index_t* const indices, const
gl_mesh* const shareInstance
);

/** Check a mesh
 * @param mesh A mesh previously returned by gl_mesh_create()
 * @return 1 if good, 0 if not
 */
int gl_mesh_check(const gl_mesh* const mesh);

/** Update portion of a mesh's vertices array (VBO inside mesh).
 * @param mesh A mesh previously returned by gl_mesh_create()
 * @param vertice A pointer to the new vertices
 * @param start Offset of the update in vertices array (in unit of
gl_vertex_t)
 * @param offCnt Number of vertices in the array (in unit of gl_vertex_t)
 */

```

```

void gl_mesh_updateVertices(const gl_mesh* const mesh, const gl_vertex_t*
const vertices, const unsigned int start, const unsigned int len)
__attribute__((deprecated));

/** Update portion of a mesh's indices array (EBO bind to VAO inside mesh).
 * @param mesh A mesh previously returned by gl_mesh_create()
 * @param indices A pointer to the new indices
 * @param start Offset of the update in indices array (in unit of gl_index_t)
 * @param offCnt Number of indices in the array (in unit of gl_index_t)
 */
void gl_mesh_updateIndices(const gl_mesh* const mesh, const gl_index_t* const
indices, const unsigned int start, const unsigned int len) __attribute__((deprecated));

/** Update portion of a mesh's instances array (IBO inside mesh).
 * @param mesh A mesh previously returned by gl_mesh_create()
 * @param instances A pointer to the new instances
 * @param start Offset of the update in instances array (in unit of
gl_vertex_t)
 * @param offCnt Number of instances in the array (in unit of gl_vertex_t)
 */
void gl_mesh_updateInstances(const gl_mesh* const mesh, const gl_vertex_t*
const instances, const unsigned int start, const unsigned int len);

/** Draw a mesh once or for a few times.
 * If the mesh is created with instance, that instance will be used; if
without instance, only draw the mesh once.
 * @param mesh A mesh previously returned by gl_mesh_create()
 * @param vSize Number of vertices/indices used to draw, pass 0 to draw all
(size when create)
 * @param iSize Number of instance to draw
 */
void gl_mesh_draw(const gl_mesh* const mesh, const unsigned int vSize, const
unsigned int iSize);

/** Delete a mesh.
 * @param mesh A mesh previously returned by gl_mesh_create()
 */
void gl_mesh_delete(gl_mesh* const mesh);

/* == Texture, PBO for texture transfer and FBO for off-screen rendering
===== */

/** Create a texture or renderbuffer whit empty content.
 * A render buffer is like texture, but with no content and on accessible for
framebuffer.
 * Render buffer has no sampler, so using renderbuffer in FBO is favored if
texture is not required. Renderbuffer has limited format support.
 * In favor of GLES, no 1D texture.
 * Mipmap will be generated if minFilter request mipmap; otherwise, no mipmap
is used. (magFilter cannot use mipmap).
 * @param format Format of the texture, can be gl_texformat_*
 * @param type Texture type, can be gl_textype_*
 * @param minFilter Minify filter, can be gl_tex_dimFilter_*
 * @param magFilter Magnify filter, can be gl_tex_dimFilter_* (no mipmap)

```

```

    * @param dim Array of 3 dimensions properties, one member for each dimension
    (always pass 3 members! higher dimension will be ignored); For array texture,
    only size is used for highest dimension
    * @return Texture
    */
gl_tex gl_texture_create(const gl_texformat format, const gl_textype type,
const gl_tex_dimFilter minFilter, const gl_tex_dimFilter magFilter, const
gl_tex_dim dim[static 3]);

/** Check a texture.
    * @param tex A texture previously returned by gl_texture_create()
    * @return 1 if good, 0 if not
    */
int gl_texture_check(const gl_tex* const tex);

/** Update a portion of a texture.
    * Do NOT use on renderbuffer.
    * Use uint8_t for X8/X8UI/S8; int8_t for X8I; uint6_t for X16UI; int16_t for
    X16I; uint32_t for X32UI; int32_t for X32I; f32_t for X16/X32.
    * @param tex A texture previously returned by gl_texture_create()
    * @param data Pointer to the texture data
    * @param offset Start point of texture to be update, z-coord is ignored if
    texture is 2D
    * @param size Size of update, z-coord is ignored if texture is 2D
    */
void gl_texture_update(const gl_tex* const tex, const void* const data, const
unsigned int offset[static 3], const unsigned int size[static 3]);

/** Bind a texture to OpenGL engine texture unit.
    * Do NOT use on renderbuffer.
    * @param tex A texture previously returned by gl_texture_create()
    * @param paramId ID of parameter previously returned by gl_program_load()
    * @param unit A OpenGL texture unit, a texture unit can only hold one texture
    at a time
    */
void gl_texture_bind(const gl_tex* const tex, const gl_param paramId, const
unsigned int unit);

/** Delete a texture.
    * @param tex A texture previously returned by gl_texture_create()
    */
void gl_texture_delete(gl_tex* const tex);

/** Transfer data from PBO to actual texture
    * @param pbo A PBO previously returned by gl_pixelBuffer_create()
    * @param tex Dest gl_tex object previously created by gl_texture_create()
    */
void gl_pixelBuffer_updateToTexture(const gl_pbo* const pbo, const gl_tex*
const tex);

/** Delete a PBO.
    * @param pbo A PBO previously returned by gl_pixelBuffer_create()
    */
void gl_pixelBuffer_delete(gl_pbo* const pbo);

```

```

/** Create a framebuffer object (FBO) for off-screen rendering.
 * Attach a list of texture to the frame buffer.
 * @param count Number of texture to attach
 * @param internalBuffer A list of texture to be attached, texture type must
be 2D texture or renderbuffer, at least 1 is required
 * @param type A list of attachment type used for each texture to be attached,
can be gl_fboattach_* or any int >= 0 for color attachment, must be less than
max number of attachment allowed by GL driver
 * @return FBO
 */
gl_fbo gl_frameBuffer_create(const unsigned int count, const gl_tex
internalBuffer[static 1], const gl_fboattach type[static 1]);

/** Check a FBO
 * @param fb A FBO previously returned by gl_frameBuffer_create()
 * @return 1 if good, 0 if not
 */
int gl_frameBuffer_check(const gl_fbo* const fbo);

/** Bind a FBO to current for drawing.
 * To bind the default buffer (display window), pass NULL.
 * @param fbo A FBO previously returned by gl_frameBuffer_create(), or NULL
for display window
 * @param clear Bit mask used to clear buffers, use gl_frameBuffer_clear*
 */
void gl_frameBuffer_bind(const gl_fbo* const fbo, const int clear);

/** Download a portion of frame buffer from GPU.
 * @param fbo A FBO previously created by gl_frameBuffer_create()
 * @param dest Where to save the data, the memory space should be enough to
hold the download content
 * @param format Format of data downloading
 * @param id Which attached texture to download
 * @param offset Start point of framebuffer to be update {x,y}
 * @param size Size of framebuffer to be download {width,height}
 */
void gl_frameBuffer_download(const gl_fbo* const fbo, void* const dest, const
gl_texformat format, const unsigned int attachment, const unsigned int
offset[static 2], const unsigned int size[static 2]);

/** Delete a FBO.
 * @param fb A FBO previously returned by gl_frameBuffer_create()
 */
void gl_frameBuffer_delete(gl_fbo* const fbo);

/** Create a pixel buffer object (PBO) that is used to manually transfer
texture data.
 * @param size Size of the PBO/texture data in bytes (number_of_pixel *
bytes_per_pixel)
 * @param type 0 for upload (unpack), 1 for download (pack)
 * @param usage A hint to the driver about the frequency of usage, can be
gl_usage_*
 */
gl_pbo gl_pixelBuffer_create(const unsigned int size, const int type, const
gl_usage usage);

```

```

/** Check a PBO.
 * @param pbo A PBO previously returned by gl_pixelBuffer_create()
 * @return 1 if good, 0 if not
 */
int gl_pixelBuffer_check(const gl_pbo* const pbo);

/** Start a CPU-to-GPU transfer, obtain the pointer to the GPU memory.
 * @param pbo A PBO previously returned by gl_pixelBuffer_create()
 * @param size Size of the PBO/texture data in bytes
 */
void* gl_pixelBuffer_updateStart(const gl_pbo* const pbo, const unsigned int
size);

/** Finish the data transfer started by gl_pixelBuffer_updateStart().
 */
void gl_pixelBuffer_updateFinish();

/** Transfer data from PBO to actual texture
 * @param pbo A PBO previously returned by gl_pixelBuffer_create()
 * @param tex Dest gl_tex object previously created by gl_texture_create()
 */
void gl_pixelBuffer_updateToTexture(const gl_pbo* const pbo, const gl_tex*
const tex);

/** Start a GPU-to-CPU transfer, download texture of an framebuffer to PBO.
 * No other download operation is allowed between
gl_pixelBuffer_downloadStart()-gl_pixelBuffer_downloadFinish()-
gl_pixelBuffer_downloadDiscard() calls.
 * @param pbo A PBO previously returned by gl_pixelBuffer_create()
 * @param fbo A FBO previously returned by gl_frameBuffer_create()
 * @param format Format of data downloading
 * @param attachment Which attached texture to download (0 to MAX_ATTACHMENT),
not valid for depth and stencil buffer
 * @param offset Start point of framebuffer to be update {x,y}
 * @param size Size of framebuffer to be download {width,height}
 */
void gl_pixelBuffer_downloadStart(const gl_pbo* pbo, const gl_fbo* const fbo,
const gl_texformat format, const unsigned int attachment, const unsigned int
offset[static 2], const unsigned int size[static 2]);

/** Obtain the data address after finishing the data transfer started by
gl_pixelBuffer_updateStart().
 * A synch object can be used to test if the download has been finished.
 * This function will map the internal buffer to user space, call
gl_pixelBuffer_downloadDiscard() to unmap after the data has been processed.
 * @param pbo A PBO previously returned by gl_pixelBuffer_create()
 * @param size Size of the PBO/texture data in bytes
 * @return Address of the downloaded data
 */
void* gl_pixelBuffer_downloadFinish(const unsigned int size);

/** Discard the pointer returned by gl_pixelBuffer_downloadFinish()
 */
void gl_pixelBuffer_downloadDiscard();

```

```

/** Delete a PBO.
 * @param pbo A PBO previously returned by gl_pixelBuffer_create()
 */
void gl_pixelBuffer_delete(gl_pbo* const pbo);

#endif /* #ifndef INCLUDE_GL_H */

```

Appendix B1.3b gl.c

```

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

#include <GL/glew.h>
#include <GL/glwf3.h>

#include "gl.h"

#if !defined(__arm__) && !defined(__aarch64__)
#define SHADER_HEADER_SUPPORTTEXT
#endif

/* == Window and driver management
===== */

GLFWwindow* window = NULL; //Display window object

void __gl_windowCloseCallback(GLFWwindow* window); //Event callback when
window closed by user (X button or kill)
void __gl_glfwErrorCallback(int code, const char* desc); //GLFW error log
void GLAPIENTRY __gl_glErrorCallback(GLenum src, GLenum type, GLuint id,
GLenum severity, GLsizei length, const GLchar* message, const void*
userParam); //OpenGL log

#define __gl_elog(format, ...) (fprintf(stderr, "[GL] Err:\t"format"\n"
__VA_OPT__(,) __VA_ARGS__))
#ifdef VERBOSE
#define __gl_log(format, ...) (fprintf(stderr, "[GL] Log:\t"format"\n"
__VA_OPT__(,) __VA_ARGS__))
#else
#define __gl_log(format, ...)
#endif

int gl_init(gl_config config) {
    __gl_log("Init OpenGL");

    /* init GLFW */
    if (!glfwInit()) {
#ifdef VERBOSE
        __gl_elog("\tFail to init GLFW");
#endif
        gl_destroy();
    }
}

```

```

    return 0;
}

/* Start and config OpenGL, init window */
glfwWindowHint(GLFW_CLIENT_API, config.gles ? GLFW_OPENGL_ES_API :
GLFW_OPENGL_API);
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, config.vMajor);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, config.vMinor);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
// glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#ifdef VERBOSE
    glfwWindowHint(GLFW_OPENGL_DEBUG_CONTEXT, GL_TRUE);
#endif
window = glfwCreateWindow(config.winWidth, config.winHeight, config.winName,
NULL, NULL);
if (!window){
    #ifdef VERBOSE
        __gl_eelog("\tFail to open window");
    #endif
    gl_destroy();
    return 0;
}
glfwMakeContextCurrent(window);

glfwSwapInterval(config.vsynch);

/* Init GLEW */
glewExperimental = GL_TRUE;
GLenum glewInitError = glewInit();
if (glewInitError != GLEW_OK) {
    #ifdef VERBOSE
        __gl_eelog("\tFail init GLEW: %s", glewGetErrorString(glewInitError));
    #endif
    gl_destroy();
    return 0;
}

/* Driver and hardware info */
__gl_log("OpenGL driver and hardware info:");
int textureSize, textureSizeArray, textureSize3d, textureImageUnit,
textureImageUnitVertex;
glGetIntegerv(GL_MAX_TEXTURE_SIZE, &textureSize);
glGetIntegerv(GL_MAX_ARRAY_TEXTURE_LAYERS, &textureSizeArray);
glGetIntegerv(GL_MAX_3D_TEXTURE_SIZE, &textureSize3d);
glGetIntegerv(GL_MAX_TEXTURE_IMAGE_UNITS, &textureImageUnit);
glGetIntegerv(GL_MAX_VERTEX_TEXTURE_IMAGE_UNITS, &textureImageUnitVertex);
__gl_log("\t- Vendor: %s", glGetString(GL_VENDOR));
__gl_log("\t- Renderer: %s", glGetString(GL_RENDERER));
__gl_log("\t- Version: %s", glGetString(GL_VERSION));
__gl_log("\t- Shader language version: %s",
glGetString(GL_SHADING_LANGUAGE_VERSION));
__gl_log("\t- Max texture size: %d", textureSize);
__gl_log("\t- Max texture layers: %d", textureSizeArray);
__gl_log("\t- Max 3D texture size: %d", textureSize3d);
__gl_log("\t- Max texture Units: %d", textureImageUnit);

```

```

__gl_log("\t- Max Vertex texture units: %d", textureImageUnitVertex);

/* Event control - callback */
glfwSetWindowCloseCallback(window, __gl_windowCloseCallback);
glfwSetErrorCallback(__gl_glfwErrorCallback);
glDebugMessageCallback(__gl_glErrorCallback, NULL);

/* OpenGL config */
// glEnable(GL_PROGRAM_POINT_SIZE); glPointSize(10.0f);
// glEnable(GL_DEPTH_TEST); glDepthFunc(GL_LESS);
// glEnable(GL_CULL_FACE); glCullFace(GL_BACK); glFrontFace(GL_CCW);
// glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glfwSetCursor(window, glfwCreateStandardCursor(GLFW_CROSSHAIR_CURSOR));

return 1;
}

void* gl_getWindow() {
    return (void*)window;
}

void gl_drawStart() {
    glfwPollEvents();
}

gl_winsizeNcursor gl_getWinsizeCursor() {
    gl_winsizeNcursor x;
    glfwGetCursorPos(window, x.curPos, x.curPos+1); //In fact, this call returns
    cursor pos respect to window
    glfwGetWindowSize(window, x.winsize, x.winsize+1);
    glfwGetFramebufferSize(window, x.framesize, x.framesize+1); //Window size
    and framebuffer size may differ if DPI is not 1
    x.curPosWin[0] = x.curPos[0]; x.curPosWin[1] = x.curPos[1];
    x.curPos[0] /= x.winsize[0]; x.curPos[1] /= x.winsize[1];
    x.curPosFrame[0] = x.curPos[0] * x.framesize[0];
    x.curPosFrame[1] = x.curPos[1] * x.framesize[1];
    return x;
}

void gl_setViewport(const unsigned int offset[static 2], const unsigned int
size[static 2]) {
    glViewport(offset[0], offset[1], size[0], size[1]);
}

void gl_drawEnd(const char* const title) {
    if (title)
        glfwSetWindowTitle(window, title);

    glfwSwapBuffers(window);
}

int gl_close(const int close) {
    if (close == 0) {
        __gl_log("Request to keep window");
    }
}

```

```

        glfwSetWindowShouldClose(window, 0);
        return 0;
    } else if (close > 0) {
        __gl_log("Request to close window");
        glfwSetWindowShouldClose(window, 1);
        return 1;
    } else {
        return glfwWindowShouldClose(window);
    }
}

void gl_destroy() {
    __gl_log("Destroy OpenGL");
    gl_close(1);
    glfwTerminate();
}

void gl_lineMode(const unsigned int weight) {
    if (weight) {
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        glLineWidth(weight);
    } else {
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    }
}

void gl_ensure() {
    glFinish();
}

void gl_rsync() {
    glFlush();
}

gl_sync gl_synthSet() {
    gl_sync s = glFenceSync(GL_SYNC_GPU_COMMANDS_COMPLETE, 0);
    return s;
}

gl_sync_status gl_synthWait(const gl_sync s, uint64_t const timeout) {
    gl_sync_status statue;
    switch (glClientWaitSync(s, GL_SYNC_FLUSH_COMMANDS_BIT, timeout)) {
        case GL_ALREADY_SIGNALED:
            statue = gl_synth_done;
            break;
        case GL_CONDITION_SATISFIED:
            statue = gl_synth_ok;
            break;
        case GL_TIMEOUT_EXPIRED:
            statue = gl_synth_timeout;
            break;
        default:
            statue = gl_synth_error;
            break;
    }
    return statue;
}

```

```

}
void gl_synchDelete(const gl_synch s) {
    glDeleteSync(s);
}

void __gl_windowCloseCallback(GLFWwindow* window) {
    __gl_log("Window close event fired");
    glfwSetWindowShouldClose(window, 1);
}
void __gl_glfwErrorCallback(int code, const char* desc) {
    __gl_elog("GLFW error %d: %s", code, desc);
}
void GLAPIENTRY __gl_glErrorCallback(GLenum src, GLenum type, GLuint id,
GLenum severity, GLsizei length, const GLchar* message, const void* userParam)
{
    if (type == GL_DEBUG_TYPE_ERROR)
        __gl_elog("GL Driver: type %x, severity %x, message: %s", type, severity,
message);
    else
        __gl_log("GL Driver: type %x, severity %x, message: %s", type, severity,
message);
}

/* == Shader and shader param data types, UBO
===== */

char* shaderCommonHeader = ""; //Default is empty, with 0-terminator, NOT NULL

void gl_program_setCommonHeader(const char* const header) {
    shaderCommonHeader = (char*)header;
}

gl_program gl_program_load(char* src, gl_programArg* const args) {
    //Create a pointer to src code
    char* code = NULL;
    if (src[0] == 'f') { //From file (1)
        __gl_log("Init shader program from file: %s", src+1);
        FILE* fp = fopen(src+1, "r");
        if (!fp) {
            __gl_elog("Cannot open shader source code file: %s", src+1);
            return GL_INIT_DEFAULT_PROGRAM;
        }
        fseek(fp, 0, SEEK_END);
        long int len = ftell(fp);
        if (len < 0) {
            fclose(fp);
            __gl_elog("Cannot get shader source code file length: %s", src+1);
            return GL_INIT_DEFAULT_PROGRAM;
        }
        __gl_log(" - File size: %ld", len);
        code = alloca(len + 1); //Shader code is normally small
        rewind(fp);
        !fread(code, 1, len, fp);
        code[len] = '\0';
        fclose(fp);
    }
}

```

```

} else { //From memory (0)
    __gl_log("Init shader program from memory @ %p", src);
    code = src + 1;
}

const unsigned int shaderTypes[] = {GL_VERTEX_SHADER, GL_FRAGMENT_SHADER,
GL_GEOMETRY_SHADER};
const char* shaderTokens[] = {"@VS\n", "@FS\n", "@GS\n"};
const unsigned int shaderCnt = sizeof(shaderTypes) / sizeof(shaderTypes[0]);
struct {
    GLuint name; //GL shader name
    unsigned int line; //Shader line number in file
    char* ptr; //Pointer to start of each src code
} shaders[shaderCnt];

GLuint status;
GLuint msgLength, program = glCreateProgram();
if (!program) {
    __gl_elog("Cannot create new empty shader program");
    return GL_INIT_DEFAULT_PROGRAM;
}

for (int i = 0; i < shaderCnt; i++) { //Pass 1: Get ptr to specific shader
and line number
    shaders[i].ptr = strstr(code, shaderTokens[i]);
    if (shaders[i].ptr) {
        shaders[i].line = 1;
        for (char* x = code; x < shaders[i].ptr; x++) { if (*x == '\n')
shaders[i].line++; }
        __gl_log(" - Get shader %.2s, line %u, offset %lu @ %p",
shaderTokens[i]+1, shaders[i].line, shaders[i].ptr - code, shaders[i].ptr);
    }
}

for (int i = 0; i < shaderCnt; i++) { //Pass 2: Replace toekn to '\0'
    if (shaders[i].ptr)
        memset(shaders[i].ptr, 0, strlen(shaderTokens[0]));
}

for (int i = 0; i < shaderCnt; i++) { //Pass 3: Create shader
    if (!shaders[i].ptr)
        continue;
    if (!( shaders[i].name = glCreateShader(shaderTypes[i]) )) {
        __gl_elog("Cannot create new empty %.2s shader", shaderTokens[i]+1);
        glDeleteProgram(program);
        return GL_INIT_DEFAULT_PROGRAM;
    }
    char numberLine[32];
    snprintf(numberLine, 32, "#line %u\n", shaders[i].line);
    const GLchar* x[3] = {shaderCommonHeader, numberLine, shaders[i].ptr +
strlen(shaderTokens[0])};
    glShaderSource(shaders[i].name, 3, x, NULL);
    glCompileShader(shaders[i].name);
    glGetShaderiv(shaders[i].name, GL_COMPILE_STATUS, &status);
    if (!status) {
        glGetShaderiv(shaders[i].name, GL_INFO_LOG_LENGTH, &msgLength);
        char msg[msgLength];

```

```

        glGetShaderInfoLog(shaders[i].name, msgLength, NULL, msg);
        if (src[0] == 'f')
            __gl_elog("Error in %.2s shader from file \"%s\":\n%s",
shaderTokens[i]+1, src+1, msg);
        else
            __gl_elog("Error in %.2s shader @ %p:\n%s", shaderTokens[i]+1, src,
msg);
        glDeleteShader(shaders[i].name); //Delete current shader
        glDeleteProgram(program); //Delete program and all previous shaders,
previous shaders have attached and flag for delete
        return GL_INIT_DEFAULT_PROGRAM;
    }
    glAttachShader(program, shaders[i].name); //Attach current shader, so we
can flag them delete for automatical delete
    glDeleteShader(shaders[i].name); //Shader compiled and attached, flag the
shader for delete when the program is delete
}

glLinkProgram(program);
glGetProgramiv(program, GL_LINK_STATUS, &status);
if (!status) {
    glGetProgramiv(program, GL_INFO_LOG_LENGTH, &msgLength);
    char msg[msgLength];
    glGetProgramInfoLog(program, msgLength, NULL, msg);
    if (src[0] == 'f')
        __gl_elog("Shader program from file \"%s\" link fail:\n%s", src+1, msg);
    else
        __gl_elog("Shader program @ %p link fail:\n%s", src, msg);
    glDeleteProgram(program);
    return GL_INIT_DEFAULT_PROGRAM;
}

/* Bind arguments */
for (gl_programArg* a = args; a->name; a++) {
    switch (a->type) {
        case gl_programArgType_normal:
            a->id = glGetUniformLocation(program, a->name);
            break;
        case gl_programArgType_ubo:
            a->id = glGetUniformBlockIndex(program, a->name);
            break;
    }
    if (a->id == -1) {
        if (src[0] == 'f')
            __gl_elog("Shader program argument bind fail: Argument \"%s\" is not
in shader code from file \"%s\"", a->name, src+1);
        else
            __gl_elog("Shader program argument bind fail: Argument \"%s\" is not
in shader code @ %p", a->name, src);
        glDeleteProgram(program);
        return GL_INIT_DEFAULT_PROGRAM;
    }
    __gl_log(" - Shader program argument \"%s\" bind to %d", a->name, a->id);
}

```

```

    GLuint binSize;
    glGetProgramiv(program, GL_PROGRAM_BINARY_LENGTH, &binSize);
    __gl_log(" - Shader program (%u) create success, size %u", program,
binSize);
    return program;
}

int gl_program_check(const gl_program* const program) {
    return *program != GL_INIT_DEFAULT_PROGRAM;
}

void gl_program_use(const gl_program* const program) {
    glUseProgram(*program);
}

void gl_program_setParam(const gl_param paramId, const unsigned int length,
const gl_datatype type, const void* const data) {
    if (length - 1 > 3) { //If pass 0, 0-1 get 0xFFFFFFFF
        __gl_elog("Param set fail: GL supports vector size 1 to 4 only");
        return;
    }

    if (type == gl_datatype_int) {
        const int* d = data;
        if (length == 4)
            glUniform4i(paramId, d[0], d[1], d[2], d[3]);
        else if (length == 3)
            glUniform3i(paramId, d[0], d[1], d[2]);
        else if (length == 2)
            glUniform2i(paramId, d[0], d[1]);
        else
            glUniform1i(paramId, d[0]);
    } else if (type == gl_datatype_uint) {
        const unsigned int* d = data;
        if (length == 4)
            glUniform4ui(paramId, d[0], d[1], d[2], d[3]);
        else if (length == 3)
            glUniform3ui(paramId, d[0], d[1], d[2]);
        else if (length == 2)
            glUniform2ui(paramId, d[0], d[1]);
        else
            glUniform1ui(paramId, d[0]);
    } else if (type == gl_datatype_float) {
        const float* d = data;
        if (length == 4)
            glUniform4f(paramId, d[0], d[1], d[2], d[3]);
        else if (length == 3)
            glUniform3f(paramId, d[0], d[1], d[2]);
        else if (length == 2)
            glUniform2f(paramId, d[0], d[1]);
        else
            glUniform1f(paramId, d[0]);
    } else
        __gl_elog("Param set fail: GL supports date type int, uint and float
only");
}

```

```

}

void gl_program_delete(gl_program* const program) {
    glDeleteProgram(*program);
    *program = GL_INIT_DEFAULT_PROGRAM;
}

gl_ubo gl_uniformBuffer_create(const unsigned int bindingPoint, const unsigned
int size, const gl_usage usage) {
    const GLenum usageLookup[] = {GL_STREAM_DRAW, GL_STATIC_DRAW,
GL_DYNAMIC_DRAW};
    if (usage < 0 || usage >= gl_usage_placeholderEnd)
        return GL_INIT_DEFAULT_UBO;

    gl_ubo ubo = GL_INIT_DEFAULT_UBO;

    glGenBuffers(1, &ubo);
    glBindBuffer(GL_UNIFORM_BUFFER, ubo);
    glBufferData(GL_UNIFORM_BUFFER, size, NULL, usageLookup[usage]);
    glBindBufferBase(GL_UNIFORM_BUFFER, bindingPoint, ubo);
    glBindBuffer(GL_UNIFORM_BUFFER, 0);

    return ubo;
}

int gl_uniformBuffer_check(const gl_ubo* const ubo) {
    return *ubo != GL_INIT_DEFAULT_UBO;
}

void gl_uniformBuffer_bindShader(const unsigned int bindingPoint, const
gl_program* const program, const gl_param paramId) {
    glUniformBlockBinding(*program, paramId, bindingPoint);
}

void gl_uniformBuffer_update(const gl_ubo* const ubo, const unsigned int
start, const unsigned int len, const void* const data) {
    glBindBuffer(GL_UNIFORM_BUFFER, *ubo);
    glBufferSubData(GL_UNIFORM_BUFFER, start, len, data);
    glBindBuffer(GL_UNIFORM_BUFFER, 0);
}

void gl_uniformBuffer_delete(gl_ubo* const ubo) {
    glDeleteBuffers(1, ubo);
    *ubo = GL_INIT_DEFAULT_UBO;
}

/* == Mesh (vertices)
===== */

gl_mesh gl_mesh_create(
    const unsigned int vCnt, const unsigned int eCnt, const unsigned int iCnt,
    const gl_meshmode mode,
    const gl_index_t* const vSize, const gl_index_t* const iSize,
    const gl_vertex_t* const vertices, const gl_index_t* const indices, const
gl_mesh* const shareInstance

```

```

) {
    __gl_log("Create mesh, %s%M%u", eCnt ? "Indices " : "", iCnt ? "Instances
" : "", mode);
    gl_mesh mesh = {
        .vao = 0, .vbo = 0, .ebo = 0, .ibo = 0,
        .drawSize = eCnt ? eCnt : vCnt, //Set to face count if applied, otherwise
use vertex size
        .mode = mode
    };
    glGenVertexArrays(1, &mesh.vao);
    glBindVertexArray(mesh.vao);
    __gl_log(" - VAO Id = %u", mesh.vao);

    gl_index_t aIdx = 0, vboStride = 0, iboStride = 0; //Get attribute index in
shader program, get vertex stride (in unit of number, not byte) for vbo and
ibo
    for (const gl_index_t* x = vSize; *x; x++)
        vboStride += *x;
    if (iCnt) {
        for (const gl_index_t* x = iSize; *x; x++)
            iboStride += *x;
    }

    /* Set vertices */
    glGenBuffers(1, &mesh.vbo);
    glBindBuffer(GL_ARRAY_BUFFER, mesh.vbo);
    __gl_log(" - VBO Id = %u", mesh.vbo);
    glBufferData(GL_ARRAY_BUFFER, vCnt * vboStride * sizeof(gl_vertex_t),
vertices, GL_STATIC_DRAW); //Mesh (model) is infrequently updated
    for (unsigned int i = 0, offset = 0; vSize[i]; i++) {
        glEnableVertexAttribArray(aIdx);
        glVertexAttribPointer(aIdx, vSize[i], GL_FLOAT, GL_FALSE, vboStride *
sizeof(gl_vertex_t), (GLvoid*)(offset * sizeof(gl_vertex_t)));
        offset += vSize[i];
        aIdx++;
    }
    __gl_log(" - Vertices attributes bind");

    /* Set indices, if applied */
    if (eCnt) {
        glGenBuffers(1, &mesh.ebo);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, mesh.ebo);
        __gl_log(" - EBO Id = %u", mesh.ebo);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, eCnt * sizeof(gl_index_t), indices,
GL_STATIC_DRAW);
    }

    /* Set instance, if applied */
    if (iCnt) {
        if (shareInstance) {
            mesh.ibo = shareInstance->ibo;
            glBindBuffer(GL_ARRAY_BUFFER, mesh.ibo);
            __gl_log(" - IBO Id = %u (shared with mesh VAO %u)", mesh.ibo,
shareInstance->vao);
        } else {

```

```

        glGenBuffers(1, &mesh.ibo);
        glBindBuffer(GL_ARRAY_BUFFER, mesh.ibo);
        __gl_log(" - IBO Id = %u", mesh.ibo);
        glBufferData(GL_ARRAY_BUFFER, iCnt * iboStride * sizeof(gl_vertex_t),
NULL, GL_STREAM_DRAW); //Instance of mesh is frequently updates
    }
    for (unsigned int i = 0, offset = 0; iSize[i]; i++) {
        glEnableVertexAttribArray(aIdx);
        glVertexAttribPointer(aIdx, iSize[i], GL_FLOAT, GL_FALSE, iboStride *
sizeof(gl_vertex_t), (GLvoid*)(offset * sizeof(gl_vertex_t)));
        glVertexAttribDivisor(aIdx, 1);
        offset += iSize[i];
        aIdx++;
    }
    __gl_log(" - Instances attributes bind");
}

return mesh;
}

int gl_mesh_check(const gl_mesh* const mesh) {
    return mesh->vao && mesh->vbo;
}

void gl_mesh_updateVertices(const gl_mesh* const mesh, const gl_vertex_t*
const vertices, const unsigned int start, const unsigned int len) {
    glBindBuffer(GL_ARRAY_BUFFER, mesh->vbo);
    glBufferSubData(GL_ARRAY_BUFFER, start * sizeof(gl_vertex_t), len *
sizeof(gl_vertex_t), vertices);
}

void gl_mesh_updateIndices(const gl_mesh* const mesh, const gl_index_t* const
indices, const unsigned int start, const unsigned int len) {
    glBindVertexArray(mesh->vao);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, mesh->ebo);
    glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, start * sizeof(gl_index_t), len *
sizeof(gl_index_t), indices);
}

void gl_mesh_updateInstances(const gl_mesh* const mesh, const gl_vertex_t*
const instances, const unsigned int start, const unsigned int len) {
    glBindBuffer(GL_ARRAY_BUFFER, mesh->ibo);
    glBufferSubData(GL_ARRAY_BUFFER, start * sizeof(gl_vertex_t), len *
sizeof(gl_vertex_t), instances);
}

void gl_mesh_draw(const gl_mesh* const mesh, const unsigned int vSize, const
unsigned int iSize) {
    glBindVertexArray(mesh->vao);
    if (mesh->ibo) {
        if (mesh->ebo)
            glDrawElementsInstanced(mesh->mode, vSize ? vSize : mesh->drawSize,
GL_UNSIGNED_INT, 0, iSize);
        else

```

```

        glDrawArraysInstanced(mesh->mode, 0, vSize ? vSize : mesh->drawSize,
iSize);
    } else {
        if (mesh->ebo)
            glDrawElements(mesh->mode, vSize ? vSize : mesh->drawSize,
GL_UNSIGNED_INT, 0);
        else
            glDrawArrays(mesh->mode, 0, vSize ? vSize : mesh->drawSize);
    }
    glBindVertexArray(GL_INIT_DEFAULT_MESH.vao);
}

void gl_mesh_delete(gl_mesh* const mesh) {
    glBindVertexArray(mesh->vao); //Remove reference to EBO in VAO
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
    glBindVertexArray(0);

    glDeleteVertexArrays(1, &mesh->vao);
    glDeleteBuffers(1, &mesh->vbo);
    glDeleteBuffers(1, &mesh->ebo);
    glDeleteBuffers(1, &mesh->ibo);
    *mesh = GL_INIT_DEFAULT_MESH;
}

/* == Texture, PBO for texture transfer and FBO for off-screen rendering
===== */

//OpenGL texture format lookup table
const struct {
    gl_texformat eFormat;
    GLint internalFormat;
    GLenum format;
    GLenum type;
} __gl_texformat_lookup[gl_texformat_placeholderEnd] = {
    {gl_texformat_R8,    GL_R8,    GL_RED,    GL_UNSIGNED_BYTE },
    {gl_texformat_RG8,   GL_RG8,   GL_RG,     GL_UNSIGNED_BYTE },
    {gl_texformat_RGB8,  GL_RGB8,  GL_RGB,    GL_UNSIGNED_BYTE },
    {gl_texformat_RGBA8, GL_RGBA8, GL_RGBA,   GL_UNSIGNED_BYTE },
    {gl_texformat_R8I,   GL_R8I,   GL_RED_INTEGER, GL_BYTE },
    {gl_texformat_RG8I,  GL_RG8I,  GL_RG_INTEGER, GL_BYTE },
    {gl_texformat_RGB8I, GL_RGB8I, GL_RGB_INTEGER, GL_BYTE },
    {gl_texformat_RGBA8I, GL_RGBA8I, GL_RGBA_INTEGER, GL_BYTE },
    {gl_texformat_R8UI,  GL_R8UI,  GL_RED_INTEGER, GL_UNSIGNED_BYTE },
    {gl_texformat_RG8UI, GL_RG8UI, GL_RG_INTEGER, GL_UNSIGNED_BYTE },
    {gl_texformat_RGB8UI, GL_RGB8UI, GL_RGB_INTEGER, GL_UNSIGNED_BYTE },
    {gl_texformat_RGBA8UI, GL_RGBA8UI, GL_RGBA_INTEGER, GL_UNSIGNED_BYTE },
    {gl_texformat_R16F,  GL_R16F,  GL_RED,    GL_FLOAT },
    {gl_texformat_RG16F, GL_RG16F, GL_RG,     GL_FLOAT },
    {gl_texformat_RGB16F, GL_RGB16F, GL_RGB,    GL_FLOAT },
    {gl_texformat_RGBA16F, GL_RGBA16F, GL_RGBA,   GL_FLOAT },
    {gl_texformat_R16I,  GL_R16I,  GL_RED_INTEGER, GL_SHORT },
    {gl_texformat_RG16I, GL_RG16I, GL_RG_INTEGER, GL_SHORT },
    {gl_texformat_RGB16I, GL_RGB16I, GL_RGB_INTEGER, GL_SHORT },
    {gl_texformat_RGBA16I, GL_RGBA16I, GL_RGBA_INTEGER, GL_SHORT },
    {gl_texformat_R16UI, GL_R16UI, GL_RED_INTEGER, GL_UNSIGNED_SHORT },

```

```

    {gl_texformat_RG16UI, GL_RG16UI, GL_RG_INTEGER, GL_UNSIGNED_SHORT },
    {gl_texformat_RGB16UI, GL_RGB16UI, GL_RGB_INTEGER, GL_UNSIGNED_SHORT },
    {gl_texformat_RGBA16UI, GL_RGBA16UI, GL_RGBA_INTEGER, GL_UNSIGNED_SHORT
},
    {gl_texformat_R32F, GL_R32F, GL_RED, GL_FLOAT },
    {gl_texformat_RG32F, GL_RG32F, GL_RG, GL_FLOAT },
    {gl_texformat_RGB32F, GL_RGB32F, GL_RGB, GL_FLOAT },
    {gl_texformat_RGBA32F, GL_RGBA32F, GL_RGBA, GL_FLOAT },
    {gl_texformat_R32I, GL_R32I, GL_RED_INTEGER, GL_INT },
    {gl_texformat_RG32I, GL_RG32I, GL_RG_INTEGER, GL_INT },
    {gl_texformat_RGB32I, GL_RGB32I, GL_RGB_INTEGER, GL_INT },
    {gl_texformat_RGBA32I, GL_RGBA32I, GL_RGBA_INTEGER, GL_INT },
    {gl_texformat_R32UI, GL_R32UI, GL_RED_INTEGER, GL_UNSIGNED_INT },
    {gl_texformat_RG32UI, GL_RG32UI, GL_RG_INTEGER, GL_UNSIGNED_INT },
    {gl_texformat_RGB32UI, GL_RGB32UI, GL_RGB_INTEGER, GL_UNSIGNED_INT },
    {gl_texformat_RGBA32UI, GL_RGBA32UI, GL_RGBA_INTEGER, GL_UNSIGNED_INT
},
    {gl_texformat_d16, GL_DEPTH_COMPONENT16, GL_DEPTH_COMPONENT,
GL_UNSIGNED_INT },
    {gl_texformat_d24, GL_DEPTH_COMPONENT24, GL_DEPTH_COMPONENT,
GL_UNSIGNED_INT },
    {gl_texformat_d32f, GL_DEPTH_COMPONENT32F, GL_DEPTH_COMPONENT,
GL_FLOAT },
    {gl_texformat_d24s8, GL_DEPTH24_STENCIL8, GL_DEPTH_STENCIL,
GL_UNSIGNED_INT_24_8 },
    {gl_texformat_d32fs8, GL_DEPTH32F_STENCIL8, GL_DEPTH_STENCIL,
GL_FLOAT_32_UNSIGNED_INT_24_8_REV},
    {gl_texformat_s8, GL_STENCIL_INDEX8, GL_STENCIL_INDEX, GL_UNSIGNED_BYTE }
};

```

```

gl_tex gl_texture_create(const gl_texformat format, const gl_textype type,
const gl_tex_dimFilter minFilter, const gl_tex_dimFilter magFilter, const
gl_tex_dim dim[static 3]) {
    GLuint tex;

    unsigned int mipmap = (unsigned int)minFilter & (unsigned int)0x0100 ? 1 :
0; //mipmap=0x2700-0x2703, non-mipmap=0x2600-0x2601, so check bit 8

    switch (type) {
        case gl_textype_2d:
            glGenTextures(1, &tex);
            glBindTexture(GL_TEXTURE_2D, tex);
            glTexImage2D(
                GL_TEXTURE_2D, 0, __gl_texformat_lookup[format].internalFormat,
                dim[0].size, dim[1].size,
                0, __gl_texformat_lookup[format].format,
                __gl_texformat_lookup[format].type, NULL
            );
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, minFilter);
//GL_LINEAR
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, magFilter);
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, dim[0].wrapping);
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, dim[1].wrapping);
            break;
        case gl_textype_2dArray:

```

```

        glGenTextures(1, &tex);
        glBindTexture(GL_TEXTURE_2D_ARRAY, tex);
        glTexImage3D(
            GL_TEXTURE_2D_ARRAY, 0, __gl_texformat_lookup[format].internalFormat,
            dim[0].size, dim[1].size, dim[2].size,
            0, __gl_texformat_lookup[format].format,
            __gl_texformat_lookup[format].type, NULL
        );
        glTexParameterf(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MIN_FILTER, minFilter);
        glTexParameterf(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_MAG_FILTER, magFilter);
        glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_S,
            dim[0].wrapping);
        glTexParameteri(GL_TEXTURE_2D_ARRAY, GL_TEXTURE_WRAP_T,
            dim[1].wrapping);
        break;
    case gl_texttype_3d:
        glGenTextures(1, &tex);
        glBindTexture(GL_TEXTURE_3D, tex);
        glTexImage3D(
            GL_TEXTURE_3D, 0, __gl_texformat_lookup[format].internalFormat,
            dim[0].size, dim[1].size, dim[2].size,
            0, __gl_texformat_lookup[format].format,
            __gl_texformat_lookup[format].type, NULL
        );
        glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, minFilter);
        glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, magFilter);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, dim[0].wrapping);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, dim[1].wrapping);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, dim[2].wrapping);
        break;
    case gl_texttype_renderBuffer:
        glGenRenderbuffers(1, &tex);
        glBindRenderbuffer(GL_RENDERBUFFER, tex);
        glRenderbufferStorage(GL_RENDERBUFFER,
            __gl_texformat_lookup[format].internalFormat, dim[0].size, dim[1].size);
        break;
    }

    return (gl_tex){tex, dim[0].size, dim[1].size, dim[2].size, format, type,
        mipmap};
}

int gl_texture_check(const gl_tex* const tex) {
    return tex->texture != GL_INIT_DEFAULT_TEX.texture;
}

void gl_texture_update(const gl_tex* const tex, const void* const data, const
    unsigned int offset[static 3], const unsigned int size[static 3]) {
    switch (tex->type) {
        case gl_texttype_2d:
            glBindTexture(GL_TEXTURE_2D, tex->texture);
            glTexSubImage2D(
                GL_TEXTURE_2D, 0,
                offset[0], offset[1],
                size[0], size[1],

```

```

        __gl_texformat_lookup[tex->format].format,
__gl_texformat_lookup[tex->format].type, data
    );
    if (tex->mipmap)
        glGenerateMipmap(GL_TEXTURE_2D);
    break;
case gl_textype_2dArray:
    glBindTexture(GL_TEXTURE_2D_ARRAY, tex->texture);
    glTexSubImage3D(
        GL_TEXTURE_2D_ARRAY, 0,
        offset[0], offset[1], offset[2],
        size[0], size[1], size[2],
        __gl_texformat_lookup[tex->format].format,
__gl_texformat_lookup[tex->format].type, data
    );
    if (tex->mipmap)
        glGenerateMipmap(GL_TEXTURE_2D_ARRAY);
    break;
case gl_textype_3d:
    glBindTexture(GL_TEXTURE_3D, tex->texture);
    glTexSubImage3D(
        GL_TEXTURE_3D, 0,
        offset[0], offset[1], offset[2],
        size[0], size[1], size[2],
        __gl_texformat_lookup[tex->format].format,
__gl_texformat_lookup[tex->format].type, data
    );
    if (tex->mipmap)
        glGenerateMipmap(GL_TEXTURE_3D);
    break;
}
}

void gl_texture_bind(const gl_tex* const tex, const gl_param paramId, const
unsigned int unit) {
    const GLuint typeLookup[] = {GL_TEXTURE_2D, GL_TEXTURE_3D,
GL_TEXTURE_2D_ARRAY};
    glActiveTexture(GL_TEXTURE0 + unit);
    glBindTexture(typeLookup[tex->type], tex->texture);
    glUniform1i(paramId, unit);
}

void gl_texture_delete(gl_tex* const tex) {
    if (tex->type == gl_textype_renderBuffer)
        glDeleteRenderbuffers(1, &tex->texture );
    else
        glDeleteTextures(1, &tex->texture);
    tex->texture = GL_INIT_DEFAULT_TEX.texture;
}

gl_fbo gl_frameBuffer_create(const unsigned int count, const gl_tex
internalBuffer[static 1], const gl_fboattach type[static 1]) {
    GLuint fbo;
    glGenFramebuffers(1, &fbo);
    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, fbo);

```

```

__gl_log("Create FBO, id = %u", fbo);
unsigned int targetCnt = 0;
for (uint i = 0; i < count; i++) {
    if (type[i] == gl_fboattach_depth) {
        if (internalBuffer[i].type == gl_texttype_renderBuffer)
            glFramebufferRenderbuffer(GL_DRAW_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_RENDERBUFFER, internalBuffer[i].texture);
        else
            glFramebufferTexture2D(GL_DRAW_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
GL_TEXTURE_2D, internalBuffer[i].texture, 0);
        __gl_log(" - Attach texture / renderbuffer to FBO, id = %u, type
Depth", internalBuffer[i].texture);
    } else if (type[i] == gl_fboattach_stencil) {
        if (internalBuffer[i].type == gl_texttype_renderBuffer)
            glFramebufferRenderbuffer(GL_DRAW_FRAMEBUFFER, GL_STENCIL_ATTACHMENT,
GL_RENDERBUFFER, internalBuffer[i].texture);
        else
            glFramebufferTexture2D(GL_DRAW_FRAMEBUFFER, GL_STENCIL_ATTACHMENT,
GL_TEXTURE_2D, internalBuffer[i].texture, 0);
        __gl_log(" - Attach texture / renderbuffer to FBO, id = %u, type
Stencil", internalBuffer[i].texture);
    } else if (type[i] == gl_fboattach_depth_stencil) {
        if (internalBuffer[i].type == gl_texttype_renderBuffer)
            glFramebufferRenderbuffer(GL_DRAW_FRAMEBUFFER,
GL_DEPTH_STENCIL_ATTACHMENT, GL_RENDERBUFFER, internalBuffer[i].texture);
        else
            glFramebufferTexture2D(GL_DRAW_FRAMEBUFFER,
GL_DEPTH_STENCIL_ATTACHMENT, GL_TEXTURE_2D, internalBuffer[i].texture, 0);
        __gl_log(" - Attach texture / renderbuffer to FBO, id = %u, type Depth
+ Stencil", internalBuffer[i].texture);
    } else {
        if (internalBuffer[i].type == gl_texttype_renderBuffer)
            glFramebufferRenderbuffer(GL_DRAW_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
GL_RENDERBUFFER, internalBuffer[i].texture);
        else
            glFramebufferTexture2D(GL_DRAW_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 +
type[i], GL_TEXTURE_2D, internalBuffer[i].texture, 0);
        __gl_log(" - Attach texture / renderbuffer to FBO, id = %u, type
color %u", internalBuffer[i].texture, type[i] & 0xFF);
        targetCnt++;
    }
}

/*__gl_log(" - %u color render targets bound", targetCnt);
GLuint target[targetCnt];
for (uint i = 0; i < count; i++) {
    if (type[i] >= 0) {
        target[i] = GL_COLOR_ATTACHMENT0 + type[i];
    }
}
glDrawBuffers(targetCnt, target);*/

return fbo;
}

```

```

int gl_frameBuffer_check(const gl_fbo* const fbo) {
    return *fbo != GL_INIT_DEFAULT_FBO;
}

void gl_frameBuffer_bind(const gl_fbo* const fbo, const int clear) {
    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, fbo ? *fbo : 0);
    if (clear) {
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        glClear(clear);
    }
}

void gl_frameBuffer_download(const gl_fbo* const fbo, void* const dest, const
gl_texformat format, const unsigned int attachment, const unsigned int
offset[static 2], const unsigned int size[static 2]) {
    if (format < 0 || format >= gl_texformat_placeholderEnd)
        return;

    glBindFramebuffer(GL_READ_FRAMEBUFFER, *fbo);
    glReadBuffer(GL_COLOR_ATTACHMENT0 + attachment);
    glReadPixels(offset[0], offset[1], size[0], size[1],
__gl_texformat_lookup[format].format, __gl_texformat_lookup[format].type,
dest);
}

void gl_frameBuffer_delete(gl_fbo* const fbo) {
    glDeleteFramebuffers(1, fbo);
    *fbo = GL_INIT_DEFAULT_FBO;
}

gl_pbo gl_pixelBuffer_create(const unsigned int size, const int type, const
gl_usage usage) {
    const GLenum usageLookupDraw[] = {GL_STREAM_DRAW, GL_STATIC_DRAW,
GL_DYNAMIC_DRAW};
    const GLenum usageLookupRead[] = {GL_STREAM_READ, GL_STATIC_READ,
GL_DYNAMIC_READ};
    if (usage < 0 || usage >= gl_usage_placeholderEnd)
        return GL_INIT_DEFAULT_PBO;

    gl_pbo pbo;
    glGenBuffers(1, &pbo);
    if (type) {
        glBindBuffer(GL_PIXEL_PACK_BUFFER, pbo);
        glBufferData(GL_PIXEL_PACK_BUFFER, size, NULL, usageLookupRead[usage]);
        glBindBuffer(GL_PIXEL_PACK_BUFFER, GL_INIT_DEFAULT_PBO);
    } else {
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pbo);
        glBufferData(GL_PIXEL_UNPACK_BUFFER, size, NULL, usageLookupDraw[usage]);
        glBindBuffer(GL_PIXEL_UNPACK_BUFFER, GL_INIT_DEFAULT_PBO);
    }
    return pbo;
}

int gl_pixelBuffer_check(const gl_pbo* const pbo) {

```

```

    return *pbo != GL_INIT_DEFAULT_PBO;
}

void* gl_pixelBuffer_updateStart(const gl_pbo* const pbo, const unsigned int
size) {
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, *pbo);
    return glMapBufferRange(GL_PIXEL_UNPACK_BUFFER, 0, size, GL_MAP_WRITE_BIT/*
| GL_MAP_INVALIDATE_BUFFER_BIT*/ | GL_MAP_UNSYNCHRONIZED_BIT);
}

void gl_pixelBuffer_updateFinish() {
    glUnmapBuffer(GL_PIXEL_UNPACK_BUFFER);
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, GL_INIT_DEFAULT_PBO);
}

void gl_pixelBuffer_updateToTexture(const gl_pbo* const pbo, const gl_tex*
const tex) {
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, *pbo);
    switch (tex->type) {
        case gl_texture_2d:
            glBindTexture(GL_TEXTURE_2D, tex->texture);
            glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, tex->width, tex->height,
__gl_texformat_lookup[tex->format].format,
__gl_texformat_lookup[tex->format].type, 0);
            glBindTexture(GL_TEXTURE_2D, GL_INIT_DEFAULT_TEX.texture);
            break;
        case gl_texture_2dArray:
            glBindTexture(GL_TEXTURE_2D_ARRAY, tex->texture);
            glTexSubImage3D(GL_TEXTURE_2D_ARRAY, 0, 0, 0, 0, tex->width,
tex->height, tex->depth, __gl_texformat_lookup[tex->format].format,
__gl_texformat_lookup[tex->format].type, 0);
            glBindTexture(GL_TEXTURE_2D_ARRAY, GL_INIT_DEFAULT_TEX.texture);
            break;
        case gl_texture_3d:
            glBindTexture(GL_TEXTURE_3D, tex->texture);
            glTexSubImage3D(GL_TEXTURE_2D_ARRAY, 0, 0, 0, 0, tex->width,
tex->height, tex->depth, __gl_texformat_lookup[tex->format].format,
__gl_texformat_lookup[tex->format].type, 0);
            glBindTexture(GL_TEXTURE_3D, GL_INIT_DEFAULT_TEX.texture);
            break;
    }
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER, GL_INIT_DEFAULT_PBO);
}

void gl_pixelBuffer_downloadStart(const gl_pbo* pbo, const gl_fbo* const fbo,
const gl_texformat format, const unsigned int attachment, const unsigned int
offset[static 2], const unsigned int size[static 2]) {
    if (format < 0 || format >= gl_texformat_placeholderEnd)
        return;

    glBindFramebuffer(GL_READ_FRAMEBUFFER, *fbo);
    glReadBuffer(GL_COLOR_ATTACHMENT0 + attachment);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, *pbo);
    glReadPixels(offset[0], offset[1], size[0], size[1],
__gl_texformat_lookup[format].format, __gl_texformat_lookup[format].type, 0);
}

```

```

}

void* gl_pixelBuffer_downloadFinish(const unsigned int size) {
    void* ptr = glMapBufferRange(GL_PIXEL_PACK_BUFFER, 0, size,
    GL_MAP_READ_BIT);
    return ptr;
}

void gl_pixelBuffer_downloadDiscard() {
    glUnmapBuffer(GL_PIXEL_PACK_BUFFER);
    glBindBuffer(GL_PIXEL_PACK_BUFFER, GL_INIT_DEFAULT_PBO);
}

void gl_pixelBuffer_delete(gl_pbo* const pbo) {
    glDeleteBuffers(1, pbo);
    *pbo = GL_INIT_DEFAULT_PBO;
}

```

Appendix B.1.4a roadmap.h

```

/** Class - Roadmap.class.
 * Road mesh also indicates focus region.
 * Road information: road points associated to screen-domain pixel.
 * Search limit for measure.
 * Project lookup used to transform image between prospective and orthographic
view.
 */

#ifndef INCLUDE_ROADMAP_H
#define INCLUDE_ROADMAP_H

#include <inttypes.h>

/** Roadmap class object data structure
 */
typedef struct Roadmap {
    struct Roadmap_Header{
        uint32_t width, height; //Frame size in pixel
        uint32_t pCnt; //Number of road points
        uint32_t fileSize; //Size of roadmap file without meta data, in byte
    } header;
    struct RoadPoint { //[1...5]Region of interest vertices {screen-x, screen-
y} in prespective view [-4..-1] in orthographic view
        float sx;
        float sy;
    }* roadPoints;
    struct Roadmap_Table1 { //Road-domain geographic data in perspective view
and orthographic view
        float px, py; //Road-domain geographic data in perspective view
        float ox; //Road-domain geographic data in orthographic view, y-coord of
orthographic view is same as y-coord of perspective view
        float pw; //Perspective pixel width (inverse): distance * this = number of
pixels for that distance
    }* t1;
}

```

```

    struct Roadmap_Table2 { //Search limit, perspective and orthographic
projection map
    float searchLimitUp, searchLimitDown; //Up/down search limit y-coord for
displacement measure, Single frame limit on left pixel, multi frame limit on
right pixel, based on param of roadmap_post()
    float lookupXp2o, lookupXo2p; //Projection lookup table. Y-crood in
orthographic and perspective views are same
    }* t2;
} roadmap;

#define ROADMAP_DEFAULTSTRUCT {.roadPoints = NULL}

/** Init a roadmap object, allocate memory for data read map file.
 * @param roadmapFile Directory to a binary coded file contains road-domain
data
 * @param statue Return-by-reference error message if this function fail, NULL
if success
 * @return roadmap class object upon success
 */
roadmap roadmap_init(const char* const roadmapFile, char** const statue);

/** Post processing the table data.
 * The table provides scene dependent data, but some data is config depedent.
 * Some data can be further optimized for the program.
 * Generate the search limit in roadmap table 2, used for object displacement
measurement.
 * @param this This roadmap class object
 * @param limitSingle Object max road-domain displacement (meter) in one frame
 * @param limitMulti Object max road-domain displacement (meter) in multiple
frame
 */
void roadmap_post(roadmap* this, float limitSingle, float limitMulti);

/** Destroy this roadmap class object, frees resources (all tables).
 * @param this This roadmap class object
 */
void roadmap_destroy(roadmap* this);

#endif /* #ifndef INCLUDE_ROADMAP_H */

```

Appendix B1.4b roadmap.c

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "roadmap.h"

roadmap roadmap_init(const char* const roadmapFile, char** const statue) {
    roadmap this = {
        .header = {
            .width = 0,
            .height = 0,

```

```

        .pCnt = 0,
        .fileSize = 0
    },
    .roadPoints = NULL,
    .t1 = NULL,
    .t2 = NULL
};

FILE* fp = fopen(roadmapFile, "rb");
if (!fp) {
    *statue = "Fail to open roadmap file";
    return this;
}
if (!fread(&this.header, sizeof(this.header), 1, fp)) {
    *statue = "Error in roadmap file: Cannot read header";
    fclose(fp);
    return this;
}

unsigned int pixelCount = this.header.width * this.header.height;

void* buffer = malloc( //Compact all data into one buffer
    pixelCount * sizeof(struct Roadmap_Table1) +
    pixelCount * sizeof(struct Roadmap_Table2) +
    this.header.pCnt * sizeof(struct RoadPoint)
);
if (!buffer) {
    *statue = "Fail to allocate buffer memory for roadmap data";
    fclose(fp);
    return this;
}
this.roadPoints = buffer;
this.t1 = buffer +
    this.header.pCnt * sizeof(struct RoadPoint);
this.t2 = buffer +
    this.header.pCnt * sizeof(struct RoadPoint) +
    pixelCount * sizeof(struct Roadmap_Table1);

if (
    !fread(this.t1, sizeof(struct Roadmap_Table1), pixelCount, fp) ||
    !fread(this.t2, sizeof(struct Roadmap_Table2), pixelCount, fp)
) {
    *statue = "Error in roadmap file: Cannot read tables";
    fclose(fp);
    free(buffer);
    this.roadPoints = NULL; //Invalid the data memory ptr
    return this;
}
for (struct RoadPoint* p = this.roadPoints; p < this.roadPoints +
this.header.pCnt; p++) {
    float data[4]; //{Road-domain location (m) x, y, Screen domain position
(0.0 for left and top, 1.0 for right and bottom) x, y}
    if (!fread(data, sizeof(data), 1, fp)) {
        *statue = "Error in roadmap file: Cannot read focus region";
        fclose(fp);
    }
}

```

```

        free(buffer);
        this.roadPoints = NULL;
        return this;
    }
    *p = (struct RoadPoint){ .sx = data[2], .sy = data[3] };
}

*statue = NULL;
fclose(fp);
return this;
}

void roadmap_post(roadmap* this, float limitSingle, float limitMulti) {
    unsigned int width = this->header.width, height = this->header.height;
    for (unsigned int y = 0; y < height; y++) {
        for (unsigned int x = 0; x < width; x++) {
            unsigned int idx = y * width + x;
            float limit = x & 1 ? limitMulti : limitSingle; //Single frame limit on
left pixel, multi frame limit on right pixel

            // Object max road-domain displacement (meter) in one frame
            float self = this->t1[idx].py;
            int limitUp = y, limitDown = y;
            float limitUpNorm = (float)y / height, limitDownNorm = (float)y /
height;
            while(1) {
                if (limitUp <= 0) //Not exceed table size
                    break;
                if (limitUpNorm <= this->t2[idx].searchLimitUp) //Not excess max
distance provided by roadmap
                    break;
                float dest = this->t1[ limitUp * width + x ].py;
                if (dest - self > limit) //Not excess distance limit
                    break;
                limitUp--;
                limitUpNorm = (float)limitUp / height;
            }
            while(1) {
                if (limitDown >= height - 1)
                    break;
                if (limitDownNorm >= this->t2[idx].searchLimitDown)
                    break;
                float dest = this->t1[ limitDown * width + x ].py;
                if (self - dest > limit)
                    break;
                limitDown++;
                limitDownNorm = (float)limitDown / height;
            }
            this->t2[ y * width + x ].searchLimitUp = limitUpNorm;
            this->t2[ y * width + x ].searchLimitDown = limitDownNorm;

            // Inverse pixel width to avoid division: distance * 1/pixelWidth =
number of pixels for that distance
            this->t1[idx].pw = 1 / this->t1[idx].pw;
        }
    }
}

```

```

    }
}

void roadmap_destroy(roadmap* this) {
    if (!this->roadPoints)
        return;

    free(this->roadPoints);
    this->roadPoints = NULL;
}

```

Appendix B.1.5a speedometer.h

```

/** Class - Speedometer.class.
 * A class used to process and display speed on screen.
 * This class is used to sample the speed data and generate human readable
 * output.
 */

#ifndef INCLUDE_SPEEDOMETER_H
#define INCLUDE_SPEEDOMETER_H

#include "common.h"

/** Speedometer class object data structure
 */
typedef struct Speedometer_ClassDataStructure* Speedometer;

/** Init a speedometer object, allocate memory, read speedometer glyph from
 * bitmap file.
 * The glyph bitmap file must contains 256 glyphs, 16 rows in total, each row
 * contains 16 glyphs.
 * Order of glyphs must be left-to-right, top-to-bottom.
 * This bitmap must be in RGBA8 (uint32_t/pixel) format.
 * There is no restriction on the size of the each glyph, but all of them
 * should have same size.
 * @param glyphs Directory to a glyphs file, must be in plain RGBA8 raw data
 * format
 * @param statue If not NULL, return error message in case this function fail
 * @return $this(Opaque) speedometer class object upon success. If fail, free
 * all resource and return NULL
 */
Speedometer speedometer_init(const char* const glyphs, char** const statue);

/** Convert the glyph to array.
 * By default, the glyph map layout is 16 * 16; in another word, a flat
 * texture contains 256 small texture. This works for 2D texture.
 * By calling this method, the flat 2D glyph map will be converted into 2D
 * array texture; which means, there will be 256 different 2D textures in the
 * array.
 * Each of the 256 glyph can be access by using index (z-axis). This works for
 * 2D array texture.
 * @param this This speedometer class object
 */

```

```

void speedometer_convert(const Speedometer const this);

/** Get a pointer to the glyph.
 * @param this This speedometer class object
 * @param size Width and height of each glyph, pass-by-reference
 * @return Pointer to the bitmap glyph
 */
uint32_t* speedometer_getGlyph(const Speedometer const this, unsigned int
size[static 2]);

/** Destroy this speedometer class object, frees resources.
 * @param this This speedometer class object
 */
void speedometer_destroy(const Speedometer this);

#endif /* #ifndef INCLUDE_SPEEDOMETER_H */

```

Appendix B.1.5b speedometer.c

```

#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>
#include <string.h>

#ifdef VERBOSE
#include <errno.h>
#endif

#include "speedometer.h"

struct Speedometer_ClassDataStructure {
    uint8_t size[2];
    uint32_t* glyph;
};

Speedometer speedometer_init(const char* const glyphs, char** const statue) {
    Speedometer this = malloc(sizeof(struct Speedometer_ClassDataStructure));
    if (!this) {
        if (statue)
            *statue = "Fail to create speedometer class object data structure";
        return NULL;
    }
    *this = (struct Speedometer_ClassDataStructure){ .glyph = NULL };

    FILE* fp = fopen(glyphs, "rb");
    if (!fp) {
        if (statue)
            *statue = "Fail to open speedometer glyph file";
        speedometer_destroy(this);
        return NULL;
    }
}

```

```

    if (!fread(&this->size, 1, 2, fp)) {
        if (statue)
            *statue = "Error in speedometer glyph file: Cannot read size";
        fclose(fp);
        speedometer_destroy(this);
        return NULL;
    }

    this->glyph = malloc(256 * this->size[0] * this->size[1] *
sizeof(uint32_t));
    if (!this->glyph) {
        if (statue)
            *statue = "Fail to create buffer for speedometer glyph";
        fclose(fp);
        speedometer_destroy(this);
        return NULL;
    }

    if (!fread(this->glyph, sizeof(uint32_t), 256 * this->size[0] *
this->size[1], fp)) {
        if (statue)
            *statue = "Error in speedometer glyph file: Cannot read glyph";
        fclose(fp);
        speedometer_destroy(this);
        return NULL;
    }

    fclose(fp);
    return this;
}

void speedometer_convert(const Speedometer const this) {
    unsigned int gWidth = this->size[0], gHeight = this->size[1];
    uint32_t temp[16][gHeight][gWidth]; //For a row, 16 char

    /* Original format
    Ch0      Ch1      Ch2      Ch3      Ch4      Ch6      ... 16 characters
    AAAAAAA BBBB BBBB CCCCCC DDDDDDD EEEEEEE FFFFFFF ... - 0
    AAAAAAA BBBB BBBB CCCCCC DDDDDDD EEEEEEE FFFFFFF ... h y
    AAAAAAA BBBB BBBB XCCCCC DDDDDDD EEEEEEE FFFFFFF ... | |
    AAAAAAA BBBB BBBB CCCCCC DDDDDDD EEEEEEE FFFFFFF ... _ h
    |--w--| |--w--| |--w--| |--w--| |--w--| |--w--|
    0 x w
    X (Ch, y, x) = y * (w * 16) + Ch * w + x
    */

    for (unsigned int row = 0; row < 16; row++) { //16 rows * 16 characters/row
        uint32_t* rowPtr = this->glyph + row * 16 * gHeight * gWidth;
        for (unsigned int ch = 0; ch < 16; ch++) {
            for (unsigned int y = 0; y < gHeight; y++) {
                uint32_t* src = rowPtr + y * gWidth * 16 + ch * gWidth;
                memcpy(temp[ch][y], src, gWidth * sizeof(uint32_t));
            }
        }
        memcpy(rowPtr, temp, sizeof(temp));
    }
}

```

```

    }
}

uint32_t* speedometer_getGlyph(const Speedometer const this, unsigned int
size[static 2]) {
    size[0] = this->size[0];
    size[1] = this->size[1];
    return this->glyph;
}

void speedometer_destroy(const Speedometer this) {
    if (!this)
        return;

    free(this->glyph);
    free(this);
}

```

Appendix B.1.6a th_reader.h

```

struct th_reader_arg {
    unsigned int size; //Number of pixels in one frame
    const char* colorScheme; //Color scheme of the input raw data
    (numberOfChannel[1,3or4],orderOfChannelRGBA) RGB=3012, RGBA=40123, BGR=3210
};

/** Reader thread init.
 * Prepare semaphores, launch thread.
 * @param size Size of video in px
 * @param colorScheme A string represents the color format
 * @param statue If not NULL, return error message in case this function fail
 * @param ecode If not NULL, return error code in case this function fail
 * @return If success, return 1; if fail, release all resources and return 0
 */
int th_reader_init(const unsigned int size, const char* colorScheme, char**
statue, int* ecode);

/** Call this function when the main thread issue new address for video
uploading.
 * Reader will begin data uploading after this call.
 * @param addr Address to upload video data
 */
void th_reader_start(void* addr);

/** Call this function to block the main thread until reader thread finishing
video reading.
 */
void th_reader_wait();

/** Terminate reader thread and release associate resources
 */
void th_reader_destroy();

```

Appendix B.1.6b th_reader.c

```
#include <pthread.h>
#include <semaphore.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>

#include "th_reader.h"

#define BLOCKSIZE 64 //Height and width are multiple of 8, so frame size is
//multiple of 64. Read a block (64px) at once can increase performance
#define FIFONAME "tmpframefifo.data" //Video input stream

#define reader_info(format, ...) {fprintf(stderr, "[Reader] Log:\t"format"\n"
__VA_OPT__(,) __VA_ARGS__); } //Write log
#define reader_error(format, ...) {fprintf(stderr, "[Reader] Err:\t"format"\n"
__VA_OPT__(,) __VA_ARGS__); } //Write error log

void* th_reader(void* arg); //Reader thread private function - reader thread
int th_reader_readLuma(); //Reader thread private function - read luma video
int th_reader_readRGB(); //Reader thread private function - read RGB video
int th_reader_readRGBA(); //Reader thread private function - read RGBA video
int th_reader_readRGBADirect(); //Reader thread private function - read RGBA
//video with channel = RGBA (in order)

int valid = 0; //Thread has been init successfully
pthread_t tid; //Reader thread ID
sem_t sem_readerStart; //Fired by main thread: when pointer to pbo is ready,
//reader can begin to upload
sem_t sem_readerDone; //Fired by reader thread: when uploading is done, main
//thread can use
volatile void volatile* rawDataPtr; //Video raw data goes here. Main thread
//write pointer here, reader thread put data into this address
struct { unsigned int r, g, b, a; } colorChannel; //Private, for reader read
//function
unsigned int blockCnt; //Private, for reader read function
FILE* fp; //Private, for reader read function

int th_reader_init(const unsigned int size, const char* colorScheme, char**
statue, int* ecode) {
    if (sem_init(&sem_readerStart, 0, 0)) {
        if (ecode)
            *ecode = errno;
        if (statue)
            *statue = "Fail to create main-reader master semaphore";
        return 0;
    }

    if (sem_init(&sem_readerDone, 0, 0)) {
        if (ecode)
```

```

        *ecode = errno;
    if (statue)
        *statue = "Fail to create main-reader secondary semaphore";
    sem_destroy(&sem_readerStart);
    return 0;
}

pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
struct th_reader_arg arg = {.size = size, .colorScheme = colorScheme};
int err = pthread_create(&tid, &attr, th_reader, &arg);
if (err) {
    if (ecode)
        *ecode = err;
    if (statue)
        *statue = "Fail to create thread";
    sem_destroy(&sem_readerDone);
    sem_destroy(&sem_readerStart);
    return 0;
}
sem_wait(&sem_readerStart); //Wait for the thread start. Prevent this
function return before thread ready

    valid = 1;
    return 1;
}

void th_reader_start(void* addr) {
    rawDataPtr = addr;
    sem_post(&sem_readerStart);
}

void th_reader_wait() {
    sem_wait(&sem_readerDone);
}

void th_reader_destroy() {
    if (!valid)
        return;
    valid = 0;

    sem_destroy(&sem_readerDone);
    sem_destroy(&sem_readerStart);

    pthread_cancel(tid);
    pthread_join(tid, NULL);
}

void* th_reader(void* arg) {
    struct th_reader_arg* this = arg;
    unsigned int size = this->size;
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
    int (*readFunction)();

```

```

    reader_info("Frame size: %u pixels. Number of color channel: %c", size,
this->colorScheme[0]);

    if (this->colorScheme[1] != '\0') {
        colorChannel.r = this->colorScheme[1] - '0';
        if (this->colorScheme[2] != '\0') {
            colorChannel.g = this->colorScheme[2] - '0';
            if (this->colorScheme[3] != '\0') {
                colorChannel.b = this->colorScheme[3] - '0';
                if (this->colorScheme[4] != '\0')
                    colorChannel.a = this->colorScheme[4] - '0';
            }
        }
    }
    if (this->colorScheme[0] == '1') {
        reader_info("Color scheme: RGB = idx0, A = 0xFF");
        blockCnt = size / BLOCKSIZE;
        readFunction = th_reader_readLuma;
    } else if (this->colorScheme[0] == '3') {
        reader_info("Color scheme: R = idx%d, G = idx%d, B = idx%d, A = 0xFF",
colorChannel.r, colorChannel.g, colorChannel.b);
        blockCnt = size / BLOCKSIZE;
        readFunction = th_reader_readRGB;
    } else {
        reader_info("Color scheme: R = idx%d, G = idx%d, B = idx%d, A = idx%d",
colorChannel.r, colorChannel.g, colorChannel.b, colorChannel.a);
        if (colorChannel.r != 0 || colorChannel.g != 1 || colorChannel.b != 2 ||
colorChannel.a != 3) {
            blockCnt = size / BLOCKSIZE;
            readFunction = th_reader_readRGBA;
        } else {
            readFunction = th_reader_readRGBADirect;
            blockCnt = size;
        }
    }
}

unlink(FIFONAME); //Delete if exist
if (mkfifo(FIFONAME, 0777) == -1) {
    reader_error("Fail to create FIFO '"FIFONAME"' (errno = %d)", errno);
    return NULL;
}

reader_info("Ready. FIFO '"FIFONAME"' can accept frame data now"); //Ready
sem_post(&sem_readerStart); //Unblock the main thread

fp = fopen(FIFONAME, "rb"); //Stall until some data writed into FIFO
if (!fp) {
    reader_error("Fail to open FIFO '"FIFONAME"' (errno = %d)", errno);
    unlink(FIFONAME);
    return NULL;
}
reader_info("FIFO '"FIFONAME"' data received");

int readerShouldContinue = 1;

```

```

do {
    sem_wait(&sem_readerStart); //Wait until main thread issue new memory
address for next frame
    readerShouldContinue = readFunction();
    sem_post(&sem_readerDone); //Uploading done, allow main thread to use it
} while (readerShouldContinue);

fclose(fp);
unlink(FIFONAME);
reader_info("End of file or broken pipe! Please close the viewer window to
terminate the program");

while (1) { //Send dummy data to keep the main thread running
    sem_wait(&sem_readerStart); //Wait until main thread issue new memory
address for next frame
    memset((void*)rawDataPtr, 0, size * 4);
    sem_post(&sem_readerDone); //Uploading done, allow main thread to use it
}

return NULL;
}

int th_reader_readLuma() {
    uint8_t* dest = (uint8_t*)rawDataPtr;
    uint8_t luma[BLOCKSIZE];
    for (unsigned int i = blockCnt; i; i--) { //Block count = frame size / block
size
        if (!fread(luma, 1, BLOCKSIZE, fp)) {
            return 0; //Fail to read, or end of file
        }
        for (uint8_t* p = luma; p < luma + BLOCKSIZE; p++) {
            *(dest++) = *p; //R
            *(dest++) = *p; //G
            *(dest++) = *p; //B
            *(dest++) = 0xFF; //A
        }
    }
    return 1;
}

int th_reader_readRGB() {
    uint8_t* dest = (uint8_t*)rawDataPtr;
    uint8_t rgb[BLOCKSIZE * 3];
    for (unsigned int i = blockCnt; i; i--) {
        if (!fread(rgb, 3, BLOCKSIZE, fp)) {
            return 0;
        }
        for (uint8_t* p = rgb; p < rgb + BLOCKSIZE * 3; p += 3) {
            *(dest++) = p[colorChannel.r]; //R
            *(dest++) = p[colorChannel.g]; //G
            *(dest++) = p[colorChannel.b]; //B
            *(dest++) = 0xFF; //A
        }
    }
    return 1;
}

```

```

}

int th_reader_readRGBA() {
    uint8_t* dest = (uint8_t*)rawDataPtr;
    uint8_t rgba[BLOCKSIZE * 4];
    for (unsigned int i = blockCnt; i; i--) {
        if (!fread(rgba, 4, BLOCKSIZE, fp)) {
            return 0;
        }
        for (uint8_t* p = rgba; p < rgba + BLOCKSIZE * 4; p += 4) {
            *(dest++) = p[colorChannel.r]; //R
            *(dest++) = p[colorChannel.g]; //G
            *(dest++) = p[colorChannel.b]; //B
            *(dest++) = p[colorChannel.a]; //A
        }
    }
    return 1;
}

int th_reader_readRGBADirect() {
    if (!fread((void*)rawDataPtr, 4, blockCnt, fp)) //Block count = frame size /
    block size
        return 0;
    return 1;
}

```

Appendix B.1.7a th_output.h

```

#include <stdint.h>

typedef struct Output_Data {
    float rx, ry; //Road- and screen-domain coord of current frame
    uint16_t sx, sy;
    int16_t speed; //Speed (avged by shader)
    int16_t osy; //Change in y-coord
} output_data;

typedef struct Output_Header {
    int frame; //Frame number
    int count; //Number of data for current frame, use -1 to terminate thread
} output_header;

/** Output thread init.
 * Prepare communication pipe, launch thread.
 * @return If success, return 1; if fail, release all resources and return 0
 */
int th_output_init();

/** Pass data to output.
 * @param frame Frame number
 * @param count Number of data, use -1 to terminate
 * @param data An array of @param count data objects
 */
void th_output_write(const int frame, const int count, output_data* data);

```

```

/** Terminate Output thread and release associate resources.
 */
void th_output_destroy();

```

Appendix B.1.7b th_output.c

```

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

#include "th_output.h"

int _valid = 0;
int p[2] = {0, 0};
pthread_t tid; //Reader thread ID

struct itc_header {
    int frame;
    int count;
};

void* th_output(void* arg);

int th_output_init() {
    if (pipe(p) == -1) {
        fprintf(stderr, "Fail to create inter thread communication between main
thread and output thread\n");
        return 0;
    }

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    int err = pthread_create(&tid, &attr, th_output, NULL);
    if (err) {
        fprintf(stderr, "Fail to create output thread: %d\n", err);
        close(p[0]); p[0] = 0;
        close(p[1]); p[1] = 0;
        return 0;
    }

    _valid = 1;
    return 1;
}

void th_output_write(const int frame, const int count, output_data* data) {
    output_header header = {
        .frame = frame,
        .count = count
    };
    !write(p[1], &header, sizeof(output_header));
    !write(p[1], data, count * sizeof(output_data));
}

```

```

}

void th_output_destroy() {
    if (!_valid)
        return;
    _valid = 0;

    close(p[0]); p[0] = 0;
    close(p[1]); p[1] = 0;

    pthread_cancel(tid);
    pthread_join(tid, NULL);
}

void* th_output(void* arg) {
    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

    output_header header;
    for(;;) {
        !read(p[0], &header, sizeof(output_header));
        if (header.count == -1)
            break;

        output_data data[header.count];
        !read(p[0], data, header.count * sizeof(output_data));

        fprintf(stdout, "F %u %u\n", header.frame, header.count);
        for (output_data* ptr = data; ptr < data + header.count; ptr++) {
            fprintf(stdout, "O %d %.2f,%.2f %u,%u %d\n", ptr->speed, ptr->rx,
ptr->ry, ptr->sx, ptr->sy, ptr->osy);
        }
    }

    return NULL;
}

```

Appendix B.1.8 main.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <signal.h>

#include <GL/glew.h>
#include <GL/glFW3.h>

#include "common.h"
#include "gl.h"

```

```

#include "roadmap.h"
#include "speedometer.h"
#include "th_reader.h"
#include "th_output.h"

/* Program config */
#define MAX_SPEED 200 //km/h
#define GL_SYNC_TIMEOUT 500000000LLU //For gl sync timeout

/* Shader config */
// #define HEADLESS
#define SHADER_DIR "fshader/"
#define SHADER_MEASURE_INTERLACE 7 //Must be 2^n - 1 (1, 3, 7, 15...), this
create a 2^n level queue
#define SHADER_SPEED_DOWNLOADLATENCY 1 //Must be 2^n - 1 (1, 3, 7, 15...),
this create a 2^n level queue. Higher number means higher chance the FBO is
ready when download, lower stall but higher latency as well
#define SHADER_SPEEDOMETER_CNT 32 //Max number of speedometer

/* Speedometer */
#define SPEEDOMETER_FILE "./textmap.data"

/* Video data upload to GPU and processed data download to CPU */
// #define USE_PBO_UPLOAD //Not big gain: uploading is asynch op, driver will
copy data to internal buffer and then upload
#define USE_PBO_DOWNLOAD //Big gain: download is synch op

#define TEXUNIT_ROADMAP 15 //Reserve binding point for reference texture data
to reduce texture re-binding
#define TEXUNIT_SPEEDOLMETER 14

#define info(format, ...) {fprintf(stderr, "Log:\t"format"\n" __VA_OPT__(,)
__VA_ARGS__);} //Write log
#define error(format, ...) {fprintf(stderr, "Err:\t"format"\n" __VA_OPT__(,)
__VA_ARGS__);} //Write error log

int main(int argc, char* argv[]) {
    const uint zeros[4] = {0, 0, 0, 0}; //Zero array with 4 elements (can be
used as zeros[1], zeros[2] or zeros[3] as well, it is just a pointer in C)

    int status = EXIT_FAILURE;
    unsigned int frameCnt = 0;

    unsigned int sizeData[3]; //Number of pixels in width and height in the
input video, depth/leave is 0 (convenient for texture creation which requires
size[3])
    const char* color; //Input video color scheme
    unsigned int fps; //Input video FPS
    const char* roadmapFile; //File dir - roadmap file (binary)

    /* Program argument check */ {
        if (argc != 6) {
            error("Bad arg: Use 'this width height fps color roadmapFile'");
            error("\twhere color = ncccc (n is number of channel input, cccc is the
order of RGB[A])");

```

```

        error("\troadmapFile = Directory to a binary coded file contains road-
domain data");
        return status;
    }
    sizeData[0] = atoi(argv[1]);
    sizeData[1] = atoi(argv[2]);
    fps = atoi(argv[3]);
    color = argv[4];
    roadmapFile = argv[5];
    info("Start...\n");
    info("\tWidth: %upx, Height: %upx, Total: %usqpx", sizeData[0],
sizeData[1], sizeData[0] * sizeData[1]);
    info("\tFPS: %u, Color: %s", fps, color);
    info("\tRoadmap: %s", roadmapFile);

    if (sizeData[0] & (unsigned int)0b111 || sizeData[0] < 320 ||
sizeData[0] > 2048) {
        error("Bad width: Width must be multiple of 8, 320 <= width <= 2048");
        return status;
    }
    if (sizeData[1] & (unsigned int)0b111 || sizeData[1] < 240 ||
sizeData[1] > 2048) {
        error("Bad height: Height must be multiple of 8, 240 <= height <=
2048");
        return status;
    }
    if (color[0] != '1' && color[0] != '3' && color[0] != '4') {
        error("Bad color: Color channel must be 1, 3 or 4");
        return status;
    }
    if (strlen(color, 10) != color[0] - '0' + 1) {
        error("Bad color: Color scheme and channel count mismatched");
        return status;
    }
    for (int i = 1; i < strlen(color); i++) {
        if (color[i] < '0' || color[i] >= color[0]) {
            error("Bad color: Each color scheme must be in the range of [0,
channel-1 (%d)]", color[0] - '0' - 1);
            return status;
        }
    }
}

/* Program variables declaration */

//PBO or memory space for original video raw data uploading, and textures to
store original video data
#ifdef USE_PBO_UPLOAD
    gl_pbo pboUpload[2] = {GL_INIT_DEFAULT_PBO, GL_INIT_DEFAULT_PBO};
#else
    void* rawData[2] = {NULL, NULL};
#endif
    gl_tex texture_originalBuffer[2] = {GL_INIT_DEFAULT_TEX,
GL_INIT_DEFAULT_TEX}; //Front texture for using, back texture up updating

```

```

//Roadinfo, a mesh to store region of interest, and texture to store road-
domain data
roadmap roadmap = ROADMAP_DEFAULTSTRUCT;
gl_mesh mesh_persp = GL_INIT_DEFAULT_MESH;
gl_mesh mesh_ortho = GL_INIT_DEFAULT_MESH;
gl_tex texture_roadmap = GL_INIT_DEFAULT_TEX; //2D array texture: 1 - Geo
coord in persp and ortho views; 2 - Up and down search limit, P20 and O2P
project lookup
struct {
    float left, right, top, bottom;
} road_boxROI;

//To display human readable text on screen
gl_tex texture_speedometer = GL_INIT_DEFAULT_TEX; //Glyph
float* instance_speedometer_data = NULL; //Speed data to be draw (sx, sy,
speed)
gl_mesh mesh_display = GL_INIT_DEFAULT_MESH;

//Analysis and export data (CPU side)
uint8_t (* speedData)[sizeData[0]][2] = NULL; //FBO download,
height(sizeData[1]) * width(sizeData[0]) * RG8. Note: no performance
difference between RGBA8 and RG8 on VC6
#ifdef USE_PBO_DOWNLOAD
    gl_synch pboDownloadSynch = NULL;
    gl_pbo pboDownload = GL_INIT_DEFAULT_PBO;
#endif

//Final display on screen
gl_mesh mesh_final = GL_INIT_DEFAULT_MESH;

//Work stages: To save intermediate results during the process
typedef struct Framebuffer {
    gl_fbo fbo;
    gl_tex tex;
    gl_texformat format;
} fb;
#define DEFAULT_FB (fb){GL_INIT_DEFAULT_FBO, GL_INIT_DEFAULT_TEX,
gl_texformat_RGBA32F}
fb fb_raw[2] = { //Raw video data with minor pre-process
    [0 ... 1] = {GL_INIT_DEFAULT_FBO, GL_INIT_DEFAULT_TEX, gl_texformat_RGBA8}
//Input video, RGBA8
};
fb fb_object[SHADER_MEASURE_INTERLACE + 1] = { //Object detection of current
and previous frames
    [0 ... SHADER_MEASURE_INTERLACE] = {GL_INIT_DEFAULT_FBO,
GL_INIT_DEFAULT_TEX, gl_texformat_R8} //Enum < 256
};
fb fb_speed[SHADER_SPEED_DOWNLOADLATENCY + 1] = { //Speed measure result,
road-domain speed in km/h and screen-domain speed in px/frame
    [0 ... SHADER_SPEED_DOWNLOADLATENCY] = {GL_INIT_DEFAULT_FBO,
GL_INIT_DEFAULT_TEX, gl_texformat_RG8} //Range uint8 [0, 255]
};
fb fb_display = {GL_INIT_DEFAULT_FBO, GL_INIT_DEFAULT_TEX,
gl_texformat_RGBA8}; //Display human-readable text, video, RGBA8

```

```

    fb fb_stageA = {GL_INIT_DEFAULT_FBO, GL_INIT_DEFAULT_TEX, gl_texformat_RG8};
    //General intermediate data, normalized, max 2 ch
    fb fb_stageB = {GL_INIT_DEFAULT_FBO, GL_INIT_DEFAULT_TEX, gl_texformat_RG8};
    fb fb_check = {GL_INIT_DEFAULT_FBO, GL_INIT_DEFAULT_TEX,
gl_texformat_RGBA16F};

    //Program - Roadmap check
    struct { gl_program pid; } program_roadmapCheck = {.pid =
GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param src; gl_param mode; } program_project =
{.pid = GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param src; } program_blurFilter = {.pid =
GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param src; } program_edgeFilter = {.pid =
GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param current; gl_param previous; }
program_changingSensor = {.pid = GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param src; gl_param direction; }
program_objectFix = {.pid = GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param src; } program_edgeRefine = {.pid =
GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param current; gl_param hint; gl_param
previous; } program_measure = {.pid = GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param src; } program_sample = {.pid =
GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; } program_display = {.pid =
GL_INIT_DEFAULT_PROGRAM};
    struct { gl_program pid; gl_param original; gl_param result; } program_final
= {.pid = GL_INIT_DEFAULT_PROGRAM};

    /* Init OpenGL and viewer window */ {
        info("Init openGL...");
        if (!gl_init((gl_config){
            .vMajor = 3,
            .vMinor = 1,
            .gles = 1,
            .vsynch = 0,
            .winWidth = sizeData[0], .winHeight = sizeData[1],
            .winName = "Viewer"
        }))) {
            error("Cannot init OpenGL");
            goto label_exit;
        }
    }

    /* Use a texture to store raw frame data & Start reader thread */ {
        info("Prepare video upload buffer...");
        #ifdef USE_PBO_UPLOAD
            pboUpload[0] = gl_pixelBuffer_create(sizeData[0] * sizeData[1] * 4, 0,
gl_usage_stream); //Always use RGBA8 (good performance)
            pboUpload[1] = gl_pixelBuffer_create(sizeData[0] * sizeData[1] * 4, 0,
gl_usage_stream);
            if ( !gl_pixelBuffer_check(&(pboUpload[0]))
|| !gl_pixelBuffer_check(&(pboUpload[1])) ) {

```

```

        error("Fail to create pixel buffers for original frame data
uploading");
        goto label_exit;
    }
    #else
        rawData[0] = malloc(sizeData[0] * sizeData[1] * 4); //Always use RGBA8
(aligned, good performance)
        rawData[1] = malloc(sizeData[0] * sizeData[1] * 4);
        if (!rawData[0] || !rawData[1]) {
            error("Fail to create memory buffers for original frame data loading");
            goto label_exit;
        }
    #endif

    gl_tex_dim dim[3] = {
        { .size = sizeData[0], .wrapping = gl_tex_dimWrapping_edge},
        { .size = sizeData[1], .wrapping = gl_tex_dimWrapping_edge},
        { .size = 0, .wrapping = gl_tex_dimWrapping_edge}
    };
    texture_originalBuffer[0] = gl_texture_create(gl_texformat_RGBA8,
gl_textype_2d, gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim);
//RGBA8 Video use lowp
    texture_originalBuffer[1] = gl_texture_create(gl_texformat_RGBA8,
gl_textype_2d, gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim);
    if ( !gl_texture_check(&texture_originalBuffer[0])
|| !gl_texture_check(&texture_originalBuffer[1]) ) {
        error("Fail to create texture buffer for original frame data storage");
        goto label_exit;
    }

    info("Init reader thread...");
    char* statue;
    int code;
    if (!th_reader_init(sizeData[0] * sizeData[1], color, &statue, &code)) {
        error("Fail to create reader thread: %s. Error code %d", statue, code);
        goto label_exit;
    }
}

/* Roadmap: mesh for region of interest, road info, textures for road data
*/ {
    info("Load resource - roadmap \"%s\"...", roadmapFile);
    char* statue;
    roadmap = roadmap_init(roadmapFile, &statue);
    if (statue) {
        error("Cannot load roadmap: %s", statue);
        goto label_exit;
    }

    road_boxROI.left = roadmap.roadPoints[roadmap.header.pCnt-4].sx;
    road_boxROI.right = roadmap.roadPoints[roadmap.header.pCnt-1].sx;
    road_boxROI.top = roadmap.roadPoints[roadmap.header.pCnt-4].sy;
    road_boxROI.bottom = roadmap.roadPoints[roadmap.header.pCnt-1].sy;
}

```

```

roadmap_post(&roadmap, MAX_SPEED / 3.6 / fps, SHADER_MEASURE_INTERLACE *
MAX_SPEED / 3.6 / fps); //km/h to m/s to m/frame

const unsigned int sizeRoadmap[3] = {roadmap.header.width,
roadmap.header.height, 1};
const gl_index_t attributes[] = {2, 0};
mesh_persp = gl_mesh_create(roadmap.header.pCnt-4, 0, 0,
gl_meshmode_triangleStrip, attributes, NULL,
(gl_vertex_t*)(roadmap.roadPoints), NULL, NULL);
mesh_ortho = gl_mesh_create(4, 0, 0, gl_meshmode_triangleStrip,
attributes, NULL, (gl_vertex_t*)(roadmap.roadPoints + roadmap.header.pCnt-4),
NULL, NULL);
if ( !gl_mesh_check(&mesh_persp) || !gl_mesh_check(&mesh_ortho) ) {
roadmap_destroy(&roadmap);
error("Fail to create mesh for roadmap - Region of interest");
goto label_exit;
}

gl_tex_dim dim[3] = {
{.size = sizeRoadmap[0], .wrapping = gl_tex_dimWrapping_edge},
{.size = sizeRoadmap[1], .wrapping = gl_tex_dimWrapping_edge},
{.size = 2, .wrapping = gl_tex_dimWrapping_edge}
};
texture_roadmap = gl_texture_create(gl_texformat_RGBA16F,
gl_texture_2dArray, gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim);
//Mediump is good enough for road-domain geo locations, f16 (1/1024) is OK for
pixel indexing
if (!gl_texture_check(&texture_roadmap)) {
roadmap_destroy(&roadmap);
error("Fail to create texture buffer for roadmap");
goto label_exit;
}
gl_texture_update(&texture_roadmap, roadmap.t1, (const int[3]){0, 0, 0},
sizeRoadmap);
gl_texture_update(&texture_roadmap, roadmap.t2, (const int[3]){0, 0, 1},
sizeRoadmap);
}

/* Speedometer: speedometer glyph texture */ {
info("Load resource - speedometer \"%s\"...", SPEEDOMETER_FILE);
char* statue;
Speedometer speedometer = speedometer_init(SPEEDOMETER_FILE, &statue);
if (!speedometer) {
error("Cannot load speedometer: %s", statue);
goto label_exit;
}
speedometer_convert(speedometer);

unsigned int glyphSize[3] = {0,0,256};
uint32_t* glyph = speedometer_getGlyph(speedometer, glyphSize);
gl_tex_dim dim[3] = {
{.size = glyphSize[0], .wrapping = gl_tex_dimWrapping_edge},
{.size = glyphSize[1], .wrapping = gl_tex_dimWrapping_edge},
{.size = 256, .wrapping = gl_tex_dimWrapping_edge}
};
};

```

```

    texture_speedometer = gl_texture_create(gl_texformat_RGBA8,
gl_texture_2dArray, gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim);
//RGBA8 image use lowp
    if (!gl_texture_check(&texture_speedometer)) {
        speedometer_destroy(speedometer);
        error("Fail to create texture buffer for speedometer");
        goto label_exit;
    }
    gl_texture_update(&texture_speedometer, glyph, zeros, glyphSize);

    speedometer_destroy(speedometer); //Free speedometer memory after data
uploading finished

    instance_speedometer_data = malloc(SHADER_SPEEDOMETER_CNT * 3 *
sizeof(float));
    if (!instance_speedometer_data) {
        error("Fail to allocate memory for speedometer instance buffer");
        goto label_exit;
    }
    const gl_vertex_t vertices[] = {
        +0.02, -0.0125, 1.0f, 0.0f,
        +0.02, +0.0125, 1.0f, 1.0f,
        -0.02, +0.0125, 0.0f, 1.0f,
        -0.02, -0.0125, 0.0f, 0.0f
    };
    mesh_display = gl_mesh_create(4, 0, SHADER_SPEEDOMETER_CNT,
gl_meshmode_triangleFan, (gl_index_t[]){4, 0}, (gl_index_t[]){3, 0}, vertices,
NULL, NULL);
    if (!gl_mesh_check(&mesh_display)) {
        error("Fail to create mesh to store speedometer");
        goto label_exit;
    }
}

/* Create buffer for post process on CPU side & Start output thread for
result write */ {
    #ifdef USE_PBO_DOWNLOAD
        pboDownload = gl_pixelBuffer_create(sizeData[0] * sizeData[1] *
sizeof(speedData[0][0]), 1, gl_usage_stream);
        if (!gl_pixelBuffer_check(&pboDownload)) {
            error("Fail to create pixel buffer for speed downloading");
            goto label_exit;
        }
    #else
        speedData = malloc(sizeData[0] * sizeData[1] * sizeof(speedData[0][0]));
//FBO dump
        if (!speedData) {
            error("Fail to create buffer to download speed framebuffer");
            goto label_exit;
        }
    #endif

    info("Init output thread...");
    if (!th_output_init()) {
        error("Fail to create output thread");
    }
}

```

```

        goto label_exit;
    }
}

/* Create final display mesh */ {
    gl_vertex_t vertices[] = {
        1.0f, 0.0f,
        1.0f, 1.0f,
        0.0f, 1.0f,
        0.0f, 0.0f
    };
    mesh_final = gl_mesh_create(4, 0, 0, gl_meshmode_triangleFan, (const
gl_index_t[]){2, 0}, NULL, vertices, NULL, NULL);
    if (!gl_mesh_check(&mesh_final)) {
        error("Fail to create mesh for final display");
        goto label_exit;
    }
}

/* Create work buffer */ {
    info("Create GPU buffers...");

    gl_tex_dim dim[3] = {
        { .size = sizeData[0], .wrapping = gl_tex_dimWrapping_edge},
        { .size = sizeData[1], .wrapping = gl_tex_dimWrapping_edge},
        { .size = 0, .wrapping = gl_tex_dimWrapping_edge}
    };

    for (unsigned int i = 0; i < arrayLength(fb_raw); i++) {
        fb_raw[i].tex = gl_texture_create(fb_raw[i].format, gl_textype_2d,
gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim); //Video data
        if (!gl_texture_check(&fb_raw[i].tex)) {
            error("Fail to create texture to store raw video data (%u)", i);
            goto label_exit;
        }
        fb_raw[i].fbo = gl_frameBuffer_create(1, (const
gl_tex[]){fb_raw[i].tex}, (const gl_fboattach[]){gl_fboattach_color0});
        if (!gl_frameBuffer_check(&fb_raw[i].fbo) ) {
            error("Fail to create FBO to store raw video data (%u)", i);
            goto label_exit;
        }
    }

    for (unsigned int i = 0; i < arrayLength(fb_object); i++) {
        fb_object[i].tex = gl_texture_create(fb_object[i].format, gl_textype_2d,
gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim); //Object in video
        if (!gl_texture_check(&fb_object[i].tex)) {
            error("Fail to create texture to store object (%u)", i);
            goto label_exit;
        }
        fb_object[i].fbo = gl_frameBuffer_create(1, (const
gl_tex[]){fb_object[i].tex}, (const gl_fboattach[]){gl_fboattach_color0});
        if (!gl_frameBuffer_check(&fb_object[i].fbo) ) {
            error("Fail to create FBO to store object (%u)\n", i);
            goto label_exit;
        }
    }
}

```

```

    }
}

for (uint i = 0; i < arrayLength(fb_speed); i++) {
    fb_speed[i].tex = gl_texture_create(fb_speed[i].format, gl_texttype_2d,
gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim); //Speed data,
discrete
    if (!gl_texture_check(&fb_speed[i].tex)) {
        error("Fail to create texture to store speed (%u)", i);
        goto label_exit;
    }
    fb_speed[i].fbo = gl_frameBuffer_create(1, (const
gl_tex[]){fb_speed[i].tex}, (const gl_fboattach[]){gl_fboattach_color0});
    if (!gl_frameBuffer_check(&fb_speed[i].fbo) ) {
        error("Fail to create FBO to store speed (%u)", i);
        goto label_exit;
    }
}

fb_display.tex = gl_texture_create(fb_display.format, gl_texttype_2d,
gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim); //Video display to
user
    if (!gl_texture_check(&fb_display.tex)) {
        error("Fail to create texture to store speed display");
        goto label_exit;
    }
    fb_display.fbo = gl_frameBuffer_create(1, (const
gl_tex[]){fb_display.tex}, (const gl_fboattach[]){gl_fboattach_color0});
    if (!gl_frameBuffer_check(&fb_display.fbo) ) {
        error("Fail to create FBO to store speed display");
        goto label_exit;
    }
}

fb_stageA.tex = gl_texture_create(fb_stageA.format, gl_texttype_2d,
gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim); //General Data
    fb_stageB.tex = gl_texture_create(fb_stageB.format, gl_texttype_2d,
gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim);
    if ( !gl_texture_check(&fb_stageA.tex)
|| !gl_texture_check(&fb_stageB.tex) ) {
        error("Fail to create texture to store moving edge");
        goto label_exit;
    }
    fb_stageA.fbo = gl_frameBuffer_create(1, (const gl_tex[]){fb_stageA.tex},
(const gl_fboattach[]){gl_fboattach_color0});
    fb_stageB.fbo = gl_frameBuffer_create(1, (const gl_tex[]){fb_stageB.tex},
(const gl_fboattach[]){gl_fboattach_color0});
    if ( !gl_frameBuffer_check(&fb_stageA.fbo)
|| !gl_frameBuffer_check(&fb_stageB.fbo) ) {
        error("Fail to create FBO to store moving edge");
        goto label_exit;
    }
}

fb_check.tex = gl_texture_create(fb_check.format, gl_texttype_2d,
gl_tex_dimFilter_nearest, gl_tex_dimFilter_nearest, dim);

```

```

    fb_check.fbo = gl_frameBuffer_create(1, (const gl_tex[]){fb_check.tex},
    (const gl_fboattach[]){gl_fboattach_color0});
}

/* Load shader programs */ {
    info("Load shaders...");
    gl_program_setCommonHeader("#version 310 es\n");
    #define NL "\n"

    /* Create program: Roadmap check */ {
        gl_programArg arg[] = {
            {gl_programArgType_normal, "roadmap"},
            {.name = NULL}
        };

        if (!( program_roadmapCheck.pid =
gl_program_load(SHADER_DIR"roadmapCheck.glsl", arg) )) {
            error("Fail to create shader program: Roadmap check");
            goto label_exit;
        }

        gl_program_use(&program_roadmapCheck.pid);
        gl_texture_bind(&texture_roadmap, arg[0].id, TEXUNIT_ROADMAP);
    }

    /* Create program: Project P20 and P20 */ {
        gl_programArg arg[] = {
            {gl_programArgType_normal, "src"},
            {gl_programArgType_normal, "roadmap"},
            {gl_programArgType_normal, "mode"},
            {.name = NULL}
        };

        if (!( program_project.pid = gl_program_load(SHADER_DIR"project.glsl",
arg) )) {
            error("Fail to create shader program: Project perspective to
orthographic");
            goto label_exit;
        }
        program_project.src = arg[0].id;
        program_project.mode = arg[2].id;

        gl_program_use(&program_project.pid);
        gl_texture_bind(&texture_roadmap, arg[1].id, TEXUNIT_ROADMAP);
    }

    /* Create program: Blur and edge filter*/ {
        gl_programArg arg[] = {
            {gl_programArgType_normal, "src"},
            {.name = NULL}
        };

        if (!( program_blurFilter.pid =
gl_program_load(SHADER_DIR"blurFilter.glsl", arg) )) {
            error("Fail to create shader program: Blur filter");

```

```

        goto label_exit;
    }
    program_blurFilter.src = arg[0].id;

    if (!( program_edgeFilter.pid =
gl_program_load(SHADER_DIR"edgeFilter.glsl", arg) )) {
        error("Fail to create shader program: Edge filter");
        goto label_exit;
    }
    program_edgeFilter.src = arg[0].id;
}

/* Create program: Changing sensor */ {
    gl_programArg arg[] = {
        {gl_programArgType_normal, "current"},
        {gl_programArgType_normal, "previous"},
        {.name = NULL}
    };

    if (!( program_changingSensor.pid =
gl_program_load(SHADER_DIR"changingSensor.glsl", arg) )) {
        error("Fail to create shader program: Changing sensor");
        goto label_exit;
    }
    program_changingSensor.current = arg[0].id;
    program_changingSensor.previous = arg[1].id;
}

/* Create program: Object fix */ {
    gl_programArg arg[] = {
        {gl_programArgType_normal, "src"},
        {gl_programArgType_normal, "roadmap"},
        {gl_programArgType_normal, "direction"},
        {.name = NULL}
    };

    if (!( program_objectFix.pid =
gl_program_load(SHADER_DIR"objectFix.glsl", arg) )) {
        error("Fail to create shader program: Object fix");
        goto label_exit;
    }
    program_objectFix.src = arg[0].id;
    program_objectFix.direction = arg[2].id;

    gl_program_use(&program_objectFix.pid);
    gl_texture_bind(&texture_roadmap, arg[1].id, TEXUNIT_ROADMAP);
}

/* Create program: Edge refine */ {
    gl_programArg arg[] = {
        {gl_programArgType_normal, "src"},
        {gl_programArgType_normal, "roadmap"},
        {.name = NULL}
    };
};

```

```

        if (!( program_edgeRefine.pid =
gl_program_load(SHADER_DIR"edgeRefine.glsl", arg) )) {
            error("Fail to create shader program: Edge refine");
            goto label_exit;
        }
        program_edgeRefine.src = arg[0].id;

        gl_program_use(&program_edgeRefine.pid);
        gl_texture_bind(&texture_roadmap, arg[1].id, TEXUNIT_ROADMAP);
    }

    /* Create program: Measure */ {
        gl_programArg arg[] = {
            {gl_programArgType_normal, "current"},
            {gl_programArgType_normal, "hint"},
            {gl_programArgType_normal, "previous"},
            {gl_programArgType_normal, "roadmap"},
            {gl_programArgType_normal, "bias"},
            {.name = NULL}
        };

        if (!( program_measure.pid = gl_program_load(SHADER_DIR"measure.glsl",
arg) )) {
            error("Fail to create shader program: Measure");
            goto label_exit;
        }
        program_measure.current = arg[0].id;
        program_measure.hint = arg[1].id;
        program_measure.previous = arg[2].id;

        gl_program_use(&program_measure.pid);
        gl_texture_bind(&texture_roadmap, arg[3].id, TEXUNIT_ROADMAP);
        gl_program_setParam(arg[4].id, 1, gl_datatype_float, (const
float[1]){fps * 3.6f / SHADER_MEASURE_INTERLACE}); //m/Nframe to m/frame to
km/frame
    }

    /* Create program: Sample */ {
        gl_programArg arg[] = {
            {gl_programArgType_normal, "src"},
            {.name = NULL}
        };

        if (!( program_sample.pid = gl_program_load(SHADER_DIR"sample.glsl",
arg) )) {
            error("Fail to create shader program: Sample");
            goto label_exit;
        }
        program_sample.src = arg[0].id;
    }

    /* Create program: Display */ {
        gl_programArg arg[] = {
            {gl_programArgType_normal, "glyphmap"},
            {.name = NULL}
    }

```

```

    };

    if (!( program_display.pid = gl_program_load(SHADER_DIR"display.glsl",
arg) )) {
        error("Fail to create shader program: Display");
        goto label_exit;
    }

    gl_program_use(&program_display.pid);
    gl_texture_bind(&texture_speedometer, arg[0].id, TEXUNIT_SPEEDOLMETER);
}

/* Create program: Final display */ {
    gl_programArg arg[] = {
        {gl_programArgType_normal, "originalTexture"},
        {gl_programArgType_normal, "processedTexture"},
        {.name = NULL}
    };

    if (!( program_final.pid = gl_program_load(SHADER_DIR"final.glsl",
arg) )) {
        error("Fail to create shader program: Final");
        goto label_exit;
    }
    program_final.orginal = arg[0].id;
    program_final.result = arg[1].id;
}

}

void ISR_SIGINT() {
    info("INT");
    gl_close(1);
}

signal(SIGINT, ISR_SIGINT);

#ifdef VERBOSE_TIME
    uint64_t timestamp = 0;
#endif
info("Program ready!");
fprintf(stdout, "R %u*%u : I %u\n", sizeData[0], sizeData[1],
SHADER_MEASURE_INTERLACE);

/* Main process loop here */
uint current, previous; //Two level queue
uint current_obj, hint_obj, previous_obj; //Object queue
uint current_speed, previous_speed; //Speedmap queue
while(!gl_close(-1)) {
    gl_drawStart();
    char winTitle[200];

    gl_winsizeNcursor winsizeNcursor = gl_getWinsizeCursor();
    int cursorPosData[2] = {winsizeNcursor.curPos[0] * sizeData[0],
winsizeNcursor.curPos[1] * sizeData[1]};

```

```

    if (!linBox(cursorPosData[0], cursorPosData[1], 0, sizeData[0], 0,
sizeData[1], -1)) {
        current = (uint)frameCnt & (uint)0b1 ? 1 : 0; //Front
        previous = 1 - current; //Back
        current_obj = (uint)frameCnt & (uint)SHADER_MEASURE_INTERLACE;
        hint_obj = ( (uint)frameCnt - (uint)1 ) &
(uint)SHADER_MEASURE_INTERLACE;
        previous_obj = ( (uint)frameCnt + (uint)1 ) &
(uint)SHADER_MEASURE_INTERLACE;
        current_speed = (uint)frameCnt & (uint)SHADER_SPEED_DOWNLOADLATENCY;
        previous_speed = ( (uint)frameCnt + (uint)1 ) &
(uint)SHADER_SPEED_DOWNLOADLATENCY;

        // Asking the read thread to upload next frame while the main thread
        processing current frame
        void* reader_addr; // Note: 3-stage uploading scheme: reader thread -
        main thread uploading - GPU processing
        #ifdef USE_PBO_UPLOAD
            gl_pixelBuffer_updateToTexture(&pboUpload[current],
&texture_ordinalBuffer[previous]);
            reader_addr = gl_pixelBuffer_updateStart(&pboUpload[previous],
sizeData[0] * sizeData[1] * 4);
        #else
            gl_texture_update(&texture_ordinalBuffer[previous], rawData[current],
zeros, sizeData);
            reader_addr = rawData[previous];
        #endif
        th_reader_start(reader_addr);

        // Start Download data processed in previous frame (if use PBO), this
        call starts download in background, non-stall
        #ifdef USE_PBO_DOWNLOAD
            gl_pixelBuffer_downloadStart(&pboDownload,
&fb_speed[previous_speed].fbo, fb_speed->format, 0, zeros, sizeData);
            //pboDownloadSynch = gl_synchSet(); //See no difference, probably
            because PBO download must be synch
        #endif

        #ifdef VERBOSE_TIME
            uint64_t timestampRenderStart = nanotime();
        #endif
        //gl_rsync(); //Request the GL driver start the queue

        gl_setViewport(zeros, sizeData);

        // Debug use ONLY: Check roadmap
        /*gl_frameBuffer_bind(&fb_check.fbo, gl_frameBuffer_clearAll);
        gl_program_use(&program_roadmapCheck.pid);
        gl_mesh_draw(&mesh_final, 0, 0);*/

        // Blur the raw to remove noise
        gl_frameBuffer_bind(&fb_raw[current].fbo, gl_frameBuffer_clearAll);
        //Mesa: Clear buffer allows the driver to discard old buffer (the doc says it
        is faster)
        gl_program_use(&program_blurFilter.pid);

```

```

    gl_texture_bind(&texture_ordinalBuffer[current], program_blurFilter.src,
0);
    gl_mesh_draw(&mesh_final, 0, 0); //Process the entire scene. Although we
only need to process ROI, but we want to display the entire scene

    // Finding changing to detect moving object
    gl_frameBuffer_bind(&fb_stageA.fbo, gl_frameBuffer_clearAll);
    gl_program_use(&program_changingSensor.pid);
    gl_texture_bind(&fb_raw[current].tex, program_changingSensor.current,
0);
    gl_texture_bind(&fb_raw[previous].tex, program_changingSensor.previous,
1);
    gl_mesh_draw(&mesh_persp, 0, 0);

    // Fix object
    gl_frameBuffer_bind(&fb_stageB.fbo, gl_frameBuffer_clearAll);
    gl_program_use(&program_objectFix.pid);
    gl_program_setParam(program_objectFix.direction, 2, gl_datatype_float,
(const float[2]){1, 0}); //Has more gap pixels, needs better cache locality
(horizontal pixels are together in memory)
    gl_texture_bind(&fb_stageA.tex, program_objectFix.src, 0);
    gl_mesh_draw(&mesh_persp, 0, 0);

    gl_frameBuffer_bind(&fb_stageA.fbo, gl_frameBuffer_clearAll);
    gl_program_setParam(program_objectFix.direction, 2, gl_datatype_float,
(const float[2]){0, 1}); //Most gap removed by h-fix, less gap and higher
chance of intercepted
    gl_texture_bind(&fb_stageB.tex, program_objectFix.src, 0);
    gl_mesh_draw(&mesh_persp, 0, 0);

    // Refine edge, thinning the thick edge
    gl_frameBuffer_bind(&fb_stageB.fbo, gl_frameBuffer_clearAll);
    gl_program_use(&program_edgeRefine.pid);
    gl_texture_bind(&fb_stageA.tex, program_edgeRefine.src, 0);
    gl_mesh_draw(&mesh_persp, 0, 0);

    // Project from perspective to orthographic
    gl_frameBuffer_bind(&fb_object[current_obj].fbo,
gl_frameBuffer_clearAll);
    gl_program_use(&program_project.pid);
    gl_program_setParam(program_project.mode, 1, gl_datatype_int, (const
int[1]){2});
    gl_texture_bind(&fb_stageB.tex, program_project.src, 0);
    gl_mesh_draw(&mesh_ortho, 0, 0);

    // Measure the distance of edge moving between current frame and
previous frame
    gl_frameBuffer_bind(&fb_stageA.fbo, gl_frameBuffer_clearAll);
    gl_program_use(&program_measure.pid);
    gl_texture_bind(&fb_object[current_obj].tex, program_measure.current,
0);
    gl_texture_bind(&fb_object[hint_obj].tex, program_measure.hint, 1);
    gl_texture_bind(&fb_object[previous_obj].tex, program_measure.previous,
2);
    gl_mesh_draw(&mesh_ortho, 0, 0);

```

```

        // Project from orthographic to perspective
        gl_frameBuffer_bind(&fb_stageB.fbo, gl_frameBuffer_clearAll);
        gl_program_use(&program_project.pid);
        gl_program_setParam(program_project.mode, 1, gl_datatype_int, (const
int[1]){3});
        gl_texture_bind(&fb_stageA.tex, program_project.src, 0);
        gl_mesh_draw(&mesh_persp, 0, 0);

        // Sample measure result, get single point
        gl_frameBuffer_bind(&fb_speed[current_speed].fbo,
gl_frameBuffer_clearAll);
        gl_program_use(&program_sample.pid);
        gl_texture_bind(&fb_stageB.tex, program_sample.src, 0);
        gl_mesh_draw(&mesh_persp, 0, 0);

        // Download data from previous frame
#ifdef USE_PBO_DOWNLOAD //Background download starts at beginning of
frame
        //gl_synchWait(pboDownloadSynch, GL_SYNCH_TIMEOUT);
        //gl_synchDelete(pboDownloadSynch);
        speedData = gl_pixelBuffer_downloadFinish(sizeData[0] * sizeData[1] *
sizeof(speedData[0][0]));
        #else //Command queue of previous frame should be finished by now
        gl_frameBuffer_download(speedData, &fb_speed[previous_speed].fbo,
fb_speed->format, 0, zeros, sizeData); //Download current speed data so we can
process in next iteration (blocking op)
        #endif

        // Analysis the processed data
        output_data speedAnalysisObj[SHADER_SPEEDOMETER_CNT]; //Not too mush
data, OK to use stack

        output_data* objPtr = speedAnalysisObj;
        float* instancePtr = instance_speedometer_data;
        int limit = SHADER_SPEEDOMETER_CNT;
        unsigned int edgeTop = road_boxROI.top * sizeData[1], edgeBottom =
road_boxROI.bottom * sizeData[1];
        unsigned int edgeLeft = road_boxROI.left * sizeData[0], edgeRight =
road_boxROI.right * sizeData[0];
        for (unsigned int y = edgeTop; y <= edgeBottom && limit; y++) { //Get
number speed data from downloaded framebuffer
            for (unsigned int x = edgeLeft; x <= edgeRight && limit; x++) {
                int16_t speed = speedData[y][x][0];
                int16_t screenDy = speedData[y][x][1];
                if (speed > 1) {
                    vec2 coordNorm = { (float)x / sizeData[0] , (float)y /
sizeData[1] };
                    ivec2 coordScreen = {x,y};
                    ivec2 coordRoadmap = { x * roadmap.header.width / sizeData[0] , y
* roadmap.header.height / sizeData[1] };
                    struct Roadmap_Table1 geoData = roadmap.t1[ coordRoadmap.y *
roadmap.header.width + coordRoadmap.x ];
                    *objPtr++ = (output_data){
                        .rx = geoData.px,

```

```

        .ry = geoData.py,
        .sx = coordScreen.x,
        .sy = coordScreen.y,
        .speed = speed,
        .osy = screenDy - 128
    };
    *instancePtr++ = coordNorm.x;
    *instancePtr++ = coordNorm.y;
    *instancePtr++ = speed;
    limit--;
    }
}
}
#ifdef USE_PBO_DOWNLOAD
    gl_pixelBuffer_downloadDiscard();
#endif

// Render the analysis result and write to output
int objCnt = SHADER_SPEEDOMETER_CNT - limit;
#ifndef HEADLESS //Clean display, disabled in headless mode
    gl_frameBuffer_bind(&fb_display.fbo, gl_frameBuffer_clearAll);
#endif
if (objCnt) {
    #ifndef HEADLESS //Upload speed data to display, disabled in headless
mode
        gl_frameBuffer_bind(&fb_display.fbo, gl_frameBuffer_clearAll);
        gl_mesh_updateInstances(&mesh_display, instance_speedometer_data, 0,
objCnt * 3);
        gl_program_use(&program_display.pid);
        gl_mesh_draw(&mesh_display, 0, objCnt);
    #endif
    th_output_write(frameCnt - 1, objCnt, speedAnalysisObj);
}

// #define RESULT fb_stageA
// #define RESULT fb_stageB
// #define RESULT fb_raw[current]
// #define RESULT fb_object[current_obj]
// #define RESULT fb_speed[current_speed]
#define RESULT fb_display
// #define RESULT fb_check

// Draw final result on screen
#ifndef HEADLESS //Draw result on display, disabled in headless mode
    gl_setViewport(zeros, winSizeNcursor.framesize);
    gl_frameBuffer_bind(NULL, 0);
    gl_program_use(&program_final.pid);
    gl_texture_bind(&fb_raw[current].tex, program_final.original, 0);
    gl_texture_bind(&RESULT.tex, program_final.result, 1);
    gl_mesh_draw(&mesh_final, 0, 0);
#endif

#ifdef VERBOSE_TIME
    uint64_t timestampRenderEnd = nanotime();

```

```

        snprintf(winTitle, sizeof(winTitle), "Viewer - frame %u
- %.3lf/%.3lf", frameCnt, (timestampRenderEnd - timestampRenderStart) /
(double)1e6, (timestampRenderEnd - timestamp) / (double)1e6);
        timestamp = timestampRenderEnd;
    #else
        snprintf(winTitle, sizeof(winTitle), "Viewer - frame %u", frameCnt);
    #endif

    th_reader_wait(); //Wait reader thread finish uploading frame data
#ifdef USE_PBO_UPLOAD
    gl_pixelBuffer_updateFinish();
#endif

    frameCnt++;
} else {
    uint8_t x[4];
    float xf[4];
    gl_frameBuffer_download(&RESULT.fbo, x, RESULT.format, 0, cursorPosData,
(const uint[2]){1,1});
    snprintf(winTitle, sizeof(winTitle), "Viewer - frame %u, Cursor=(%d,%d),
result=(%d|%d|%d|%d)",
        frameCnt, cursorPosData[0], cursorPosData[1], x[0], x[1], x[2], x[3]
    );
    /*gl_frameBuffer_download(&RESULT.fbo, xf, RESULT.format, 0,
cursorPosData, (const uint[2]){1,1});
    snprintf(winTitle, sizeof(winTitle), "Viewer - frame %u, Cursor=(%d,%d),
result=(%.4f|%.4f|%.4f|%.4f)",
        frameCnt, cursorPosData[0], cursorPosData[1], xf[0], xf[1], xf[2],
xf[3]
    );*/
    usleep(50000);
}

#ifdef HEADLESS //Do not swap buffer if in headless mode
    gl_drawEnd(winTitle);
#endif
}

/* Free all resources, house keeping */
status = EXIT_SUCCESS;
label_exit:

gl_program_delete(&program_final.pid);
gl_program_delete(&program_display.pid);
gl_program_delete(&program_sample.pid);
gl_program_delete(&program_measure.pid);
gl_program_delete(&program_edgeRefine.pid);
gl_program_delete(&program_objectFix.pid);
gl_program_delete(&program_changingSensor.pid);
gl_program_delete(&program_edgeFilter.pid);
gl_program_delete(&program_blurFilter.pid);
gl_program_delete(&program_project.pid);
gl_program_delete(&program_roadmapCheck.pid);

gl_texture_delete(&fb_check.tex);

```

```

gl_frameBuffer_delete(&fb_check.fbo);
gl_texture_delete(&fb_stageB.tex);
gl_frameBuffer_delete(&fb_stageB.fbo);
gl_texture_delete(&fb_stageA.tex);
gl_frameBuffer_delete(&fb_stageA.fbo);
gl_texture_delete(&fb_display.tex);
gl_frameBuffer_delete(&fb_display.fbo);
for (uint i = arrayLength(fb_speed); i; i--) {
    gl_texture_delete(&fb_speed[i-1].tex);
    gl_frameBuffer_delete(&fb_speed[i-1].fbo);
}
for (unsigned int i = arrayLength(fb_object); i; i--) {
    gl_texture_delete(&fb_object[i-1].tex);
    gl_frameBuffer_delete(&fb_object[i-1].fbo);
}
for (unsigned int i = arrayLength(fb_raw); i; i--) {
    gl_texture_delete(&fb_raw[i-1].tex);
    gl_frameBuffer_delete(&fb_raw[i-1].fbo);
}

gl_mesh_delete(&mesh_final);

th_output_write(0, -1, NULL);
th_output_destroy();
#ifdef USE_PBO_DOWNLOAD
    gl_pixelBuffer_delete(&pboDownload);
//    gl_synchDelete(pboDownloadSynch);
#else
    free(speedData);
#endif

gl_mesh_delete(&mesh_display);
free(instance_speedometer_data);
gl_texture_delete(&texture_speedometer);

gl_texture_delete(&texture_roadmap);
gl_mesh_delete(&mesh_ortho);
gl_mesh_delete(&mesh_persp);
roadmap_destroy(&roadmap);

th_reader_destroy();
gl_texture_delete(&texture_ordinalBuffer[1]);
gl_texture_delete(&texture_ordinalBuffer[0]);
#ifdef USE_PBO_UPLOAD
    gl_pixelBuffer_delete(&pboUpload[1]);
    gl_pixelBuffer_delete(&pboUpload[0]);
#else
    free(rawData[0]);
    free(rawData[1]);
#endif

gl_destroy();

info("\n%u frames displayed.\n", frameCnt);
return status;

```

```
}
```

Appendix C Shader programs

The following are shader programs executed in GPU. All shaders must be placed in a subdirectory called “shader” under the main program’s directory.

OpenGL ES 3.1 is required. The program is tested on Ubuntu on Intel x64 platform with nVidia GPU and RPi Linux on Raspberry Pi 4 with Broadcom GPU.

Appendix C.1 roadmapCheck.glsl

@VS

```
layout (location = 0) in highp vec2 meshFocusRegion;
out highp vec2 pxPos;

void main() {
    gl_Position = vec4(meshFocusRegion.x * 2.0 - 1.0, meshFocusRegion.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshFocusRegion.xy;
}
```

@FS

```
uniform highp sampler2DArray roadmap;

in highp vec2 pxPos;
out highp vec4 result;

#define CH_ROADMAP1_PX x
#define CH_ROADMAP1_PY y
#define CH_ROADMAP1_OX z
#define CH_ROADMAP1_OY w
#define CH_ROADMAP2_SEARCH_UP x
#define CH_ROADMAP2_SEARCH_DOWN y
#define CH_ROADMAP2_LOOKUP_P20 z
#define CH_ROADMAP2_LOOKUP_O2P w

#define GRID_X 3.3
#define GRID_Y 10.0

void main() {
    highp ivec2 pxIdx = ivec2(vec2(textureSize(roadmap, 0)) * pxPos);
    highp ivec3 pxIdxT1 = ivec3(pxIdx, 0.0);
    highp ivec3 pxIdxT2 = ivec3(pxIdx, 1.0);

    highp vec4 roadPosCurrent = texelFetch(roadmap, pxIdxT1, 0);
    highp vec4 roadPosUp = texelFetchOffset(roadmap, pxIdxT1, 0, ivec2(0, -1));
}
```

```

    highp vec4 roadPosDown = texelFetchOffset(roadmap, pxIdxT1, 0, ivec2( 0,
+1));
    highp vec4 roadPosLeft = texelFetchOffset(roadmap, pxIdxT1, 0, ivec2(-
1, 0));
    highp vec4 roadPosRight = texelFetchOffset(roadmap, pxIdxT1, 0,
ivec2(+1, 0));
    highp vec4 roadInfoCurrent = texelFetch(roadmap, pxIdxT2, 0);

    highp vec4 color = vec4(0.0);

    //color = roadPosCurrent.xyzw;

    //color = roadInfoCurrent.xyzw;

    if ( sign(roadPosUp.CH_ROADMAP1_PY) != sign(roadPosDown.CH_ROADMAP1_PY) )
        color = vec4(1.0, 0.2, 0.2, 1.0);
    else if ( floor(roadPosLeft.CH_ROADMAP1_PX / GRID_X) !=
floor(roadPosRight.CH_ROADMAP1_PX / GRID_X) )
        color = vec4(0.1, 0.1, 0.8, 1.0);
    else if ( floor(roadPosUp.CH_ROADMAP1_PY / GRID_Y) !=
floor(roadPosDown.CH_ROADMAP1_PY / GRID_Y) )
        color = vec4(0.1, 0.8, 0.1, 1.0);

    /*if ( sign(roadPosUp.CH_ROADMAP1_OY) != sign(roadPosDown.CH_ROADMAP1_OY) )
        color = vec4(1.0, 0.2, 0.2, 1.0);
    else if ( floor(roadPosLeft.CH_ROADMAP1_OX / GRID_X) !=
floor(roadPosRight.CH_ROADMAP1_OX / GRID_X) )
        color = vec4(0.1, 0.1, 0.8, 1.0);
    else if ( floor(roadPosUp.CH_ROADMAP1_OY / GRID_Y) !=
floor(roadPosDown.CH_ROADMAP1_OY / GRID_Y) )
        color = vec4(0.1, 0.8, 0.1, 1.0);*/

    result = color;
}

```

Appendix C.2 project.glsl

@VS

```

layout (location = 0) in highp vec2 meshROI;
out mediump vec2 pxPos;

```

```

void main() {
    gl_Position = vec4(meshROI.x * 2.0 - 1.0, meshROI.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshROI.xy;
}

```

@FS

```

uniform mediump sampler2D src; //mediump for general data
uniform mediump sampler2DArray roadmap;
uniform lowp int mode; //RGBA index: 3=P2O 4=O2P

in mediump vec2 pxPos;

```

```

out mediump vec4 result; //mediump for general data

void main() {
    //Road surface is linear, so does the projection table, ok to use texture
    filter
    //Data may not be linear, so do not use filter

    mediump ivec2 srcSize = textureSize(src, 0);
    mediump vec2 srcSizeF = vec2(srcSize);

    mediump ivec2 projIdx = ivec2( srcSizeF * pxPos );
    projIdx.x = int( srcSizeF.x * texture(roadmap, vec3(pxPos, 1.0))[mode] );
    result = texelFetch(src, projIdx, 0);
}

```

Appendix C.3 blurFilter.glsl

@VS

```

layout (location = 0) in highp vec2 meshROI;
out mediump vec2 pxPos;

```

```

void main() {
    gl_Position = vec4(meshROI.x * 2.0 - 1.0, meshROI.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshROI.xy;
}

```

@FS

```

uniform lowp sampler2D src; //lowp for RGBA8 video

```

```

in mediump vec2 pxPos;
out lowp vec4 result; //lowp for RGBA8 video

```

```

//#define MONO Get gray-scale result
//#define BINARY 0.5 Get black/white image
//#define BINARY vec4(0.3, 0.4, 0.5, -0.1) //Get black/white image, use
different threshold for different channels

```

```

void main() {
    mediump ivec2 srcSize = textureSize(src, 0);
    mediump vec2 srcSizeF = vec2(srcSize);
    mediump ivec2 pxIdx = ivec2( srcSizeF * pxPos );

    //During accum, may excess lowp range, especially for edge filter when accum
    may get negative
    mediump vec4 accum = texelFetchOffset(src, pxIdx, 0, ivec2(-1,-1)) / 16.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2(-1, 0)) / 8.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2(-1,+1)) / 16.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2( 0,-1)) / 8.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2( 0, 0)) / 4.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2( 0,+1)) / 8.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2(+1,-1)) / 16.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2(+1, 0)) / 8.0;
}

```

```

    accum += texelFetchOffset(src, pxIdx, 0, ivec2(+1,+1)) / 16.0;

#ifdef MONO
    mediump float mono = dot(accum.rgb, vec3(0.299, 0.587, 0.114)) * accum.a;
    accum = vec4(vec3(mono), 1.0);
#endif

#ifdef BINARY
    accum = step(BINARY, accum);
#endif

    result = accum;
}

```

Appendix C.4 edgeFilter.glsl

@VS

```

layout (location = 0) in highp vec2 meshROI;
out mediump vec2 pxPos;

```

```

void main() {
    gl_Position = vec4(meshROI.x * 2.0 - 1.0, meshROI.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshROI.xy;
}

```

@FS

```

uniform lowp sampler2D src; //lowp for RGBA8 video

```

```

in mediump vec2 pxPos;
out lowp vec4 result; //lowp for RGBA8 video

```

```

//define MONO Get gray-scale result
//define BINARY 0.5 Get black/white image
//define BINARY vec4(0.3, 0.4, 0.5, -0.1) //Get black/white image, use
different threshold for different channels

```

```

void main() {
    mediump ivec2 srcSize = textureSize(src, 0);
    mediump vec2 srcSizeF = vec2(srcSize);
    mediump ivec2 pxIdx = ivec2( srcSizeF * pxPos );

    //During accum, may excess lowp range, especially for edge filter when accum
may get negative
    mediump vec4 accum = texelFetchOffset(src, pxIdx, 0, ivec2(0, -2)) * -1.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2(0, -1)) * -2.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2(0, 0)) * +6.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2(0, +1)) * -2.0;
    accum += texelFetchOffset(src, pxIdx, 0, ivec2(0, +2)) * -1.0;

#ifdef MONO
    mediump float mono = dot(accum.rgb, vec3(0.299, 0.587, 0.114)) * accum.a;
    accum = vec4(vec3(mono), 1.0);

```

```

#endif

#ifdef BINARY
    accum = step(BINARY, accum);
#endif

    result = accum;
}

```

Appendix C.5 changingSensor.fs.glsl

```

@VS

layout (location = 0) in highp vec2 meshROI;
out mediump vec2 pxPos;

void main() {
    gl_Position = vec4(meshROI.x * 2.0 - 1.0, meshROI.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshROI.xy;
}

@FS

uniform lowp sampler2D current; //lowp for RGBA8 video
uniform lowp sampler2D previous; //lowp for RGBA8 video

in mediump vec2 pxPos;
out lowp float result; //lowp for enum

#define THRESHOLD 0.1
//#define THRESHOLD vec4(0.05, 0.03, 0.05, -0.1) //Use different threshold for
different channels

void main() {
    lowp vec3 pb = texture(previous, pxPos).rgb;
    lowp vec3 pc = texture(current, pxPos).rgb;

    lowp vec3 d = abs(pc - pb);
    lowp float diff = dot(d, vec3(0.299, 0.587, 0.114));
    diff = step(THRESHOLD, diff);

    result = diff;
}

```

Appendix C.6 objectFix.fs.glsl

```

@VS

layout (location = 0) in highp vec2 meshROI;
out mediump vec2 pxPos;

void main() {
    gl_Position = vec4(meshROI.x * 2.0 - 1.0, meshROI.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshROI.xy;
}

```

```

}

@FS

uniform lowp sampler2D src; //lowp for enum, 1 ch
uniform mediump sampler2DArray roadmap; //mediump for object size
uniform lowp vec2 direction; //(1,0) for horizontal and (0,1) for vertical

in mediump vec2 pxPos;
out lowp float result; //lowp for enum

#define SEARCH_DISTANCE 0.7 //Should be object size / 2, or even less

bool search(mediump vec2 center, mediump vec2 step, mediump int cnt, lowp
sampler2D img) {
    bvec2 found = bvec2(false);
    mediump vec2 c1 = center, c2 = center;
    for (mediump int i = cnt; i != 0; i--) {
        c1 += step;
        if ( dot(textureGather(img, c1, 0), vec4(1.0)) > 1.0) {
            found.x = true;
            break;
        }
    }
    for (mediump int i = cnt; i != 0; i--) {
        c2 -= step;
        if ( dot(textureGather(img, c2, 0), vec4(1.0)) > 1.0) {
            found.y = true;
            break;
        }
    }
    return found.x && found.y;
}

void main() {
    lowp float ret = texture(src, pxPos).r;

    if (ret < 0.5) {
        mediump ivec2 srcSize = textureSize(src, 0);
        mediump vec2 srcSizeF = vec2(srcSize);

        mediump float pixelWidth = texture(roadmap, vec3(pxPos, 0.0)).w;
        mediump int searchDistancePx = int( 0.5 * SEARCH_DISTANCE * pixelWidth );
        //2px a time, so half the count

        mediump vec2 step = vec2(2.0) / srcSizeF; //2px a time for textureGather()
        ret = search(pxPos, direction * step, searchDistancePx, src) ? 0.7 : 0.0;
    }

    result = ret;
}

```

Appendix C.7 edgeRefine.fs.glsl

@VS

```
layout (location = 0) in highp vec2 meshROI;
out medium vec2 pxPos;
```

```
void main() {
    gl_Position = vec4(meshROI.x * 2.0 - 1.0, meshROI.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshROI.xy;
}
```

@FS

#define HUMAN

```
uniform lowp sampler2D src; //lowp for enum
uniform medium sampler2DArray roadmap;
```

```
in medium vec2 pxPos;
out lowp float result; //lowp for enum
```

```
#define SHADER_EDGEREFINE_BOTTOMDENOISE 0.2
#define SHADER_EDGEREFINE_SIDEMARGIN 0.6
```

```
#define RESULT_NOTOBJ 0.0
#define RESULT_OBJECT 0.3
#define RESULT_BOTTOM 0.6
#define RESULT_CBEDGE 1.0
```

```
// Return true if found valid pixel in limit
bool searchBottom(lowp sampler2D map, medium ivec2 idx, medium int limit) {
    for (medium ivec2 i = idx; i.y < limit; i.y++) {
        if (texelFetch(map, i, 0).r > 0.0)
            return true;
    }
    return false;
}
```

```
bvec2 minPath(lowp sampler2D map, medium ivec2 start, lowp float threshold,
medium int goal) {
    medium ivec2 lPtr = start, rPtr = start;
```

```
while (start.x - lPtr.x < goal) {
    medium ivec2 middle = lPtr + ivec2(-1, 0);
    medium ivec2 up = lPtr + ivec2(-1, -1);
    medium ivec2 down = lPtr + ivec2(-1, +1);
    if (texelFetch(map, middle, 0).r > threshold)
        lPtr = middle;
    else if (texelFetch(map, up, 0).r > threshold)
        lPtr = up;
    else if (texelFetch(map, down, 0).r > threshold)
        lPtr = down;
    else
        break;
}
```

```

}

while (rPtr.x - start.x < goal) {
    mediump ivec2 middle = rPtr + ivec2(+1, 0);
    mediump ivec2 up = rPtr + ivec2(+1, -1);
    mediump ivec2 down = rPtr + ivec2(+1, +1);
    if (texelFetch(map, middle, 0).r > threshold)
        rPtr = middle;
    else if (texelFetch(map, up, 0).r > threshold)
        rPtr = up;
    else if (texelFetch(map, down, 0).r > threshold)
        rPtr = down;
    else
        break;
}

return bvec2(start.x - lPtr.x >= goal, rPtr.x - start.x >= goal);
}

lowp float refine() {
    mediump ivec2 srcSize = textureSize(src, 0);
    mediump vec2 srcSizeF = vec2(srcSize);
    mediump ivec2 pxIdx = ivec2(srcSizeF * pxPos);

    //Not object
    if (texelFetch(src, pxIdx, 0).r < 0.1)
        return RESULT_NOTOBJ;

    mediump float pixelWidth = texture(roadmap, vec3(pxPos, 0.0)).w;
    mediump int limitSide = int(SHADER_EDGEREFINE_SIDEMARGIN * pixelWidth);
    mediump int limitBottom = int(SHADER_EDGEREFINE_BOTTOMDENOISE * pixelWidth);

    //Bottom clearence fail, so this is not bottom edge
    if (searchBottom( src , pxIdx + ivec2(0,1) , limitBottom + pxIdx.y + 1 ))
    //Atleast search 1, do not include self
        return RESULT_OBJECT;

    //If any side has space (edge not at center pertion), then this is not
    center portion
    if (!all( minPath(src, pxIdx, 0.5, limitSide) ))
        return RESULT_BOTTOM;

    //Centered bottom edge
    return RESULT_CBEDGE;
}

void main() {
    result = refine();
}

```

Appendix C.8 measure.fs.glsl

@VS

```

layout (location = 0) in highp vec2 meshROI;
out mediump vec2 pxPos;

void main() {
    gl_Position = vec4(meshROI.x * 2.0 - 1.0, meshROI.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshROI.xy;
}

@FS

uniform lowp sampler2D current; //lowp for enum
uniform lowp sampler2D hint; //lowp for enum
uniform lowp sampler2D previous; //lowp for enum
uniform mediump sampler2DArray roadmap;
uniform mediump float bias;

in mediump vec2 pxPos;
out lowp vec2 result; //Data: vec2(speed, target_yCoord_abs)

#define DEST_THRESHOLD_OBJ 0.3
#define DEST_THRESHOLD_EDGE 0.6
#define DEST_THRESHOLD_CENEDGE 1.0

mediump vec4 getLimit() {
    mediump vec4 limitGatherUp = textureGather(roadmap, vec3(pxPos, 1.0), 0);
    //Channel x for up limit
    mediump vec4 limitGatherDown = textureGather(roadmap, vec3(pxPos, 1.0), 1);
    //Channel y for down limit
    mediump float upHint = max(limitGatherUp[3], limitGatherUp[1]); //Up hint
    //should be closer to current y than up search (search upwards (-y), hint has
    //greater y)
    mediump float upSearch = min(limitGatherUp[3], limitGatherUp[1]);
    mediump float downHint = min(limitGatherDown[3], limitGatherDown[1]);
    //Search downwards (+y), hint has smaller y
    mediump float downSearch = max(limitGatherDown[3], limitGatherDown[1]);
    // Note:
    // the limit on (x,y) and (x,y+1) is similar, an extra compute to determine
    // y value (to choose from _j0 or _j1) is redundant
    return vec4(upHint, downHint, upSearch, downSearch);
}

void main() {
    mediump ivec2 videoSizeI = textureSize(current, 0);
    mediump vec2 videoSizeF = vec2(videoSizeI);

    mediump ivec2 pxIdx = ivec2(videoSizeF * pxPos);

    mediump ivec4 limit = ivec4(getLimit() * videoSizeF.y);

    mediump float displaceRoad = 0.0;
    mediump int displaceScreen = pxIdx.y;

    if (texelFetch(current, pxIdx, 0).r == DEST_THRESHOLD_CENEDGE) {

```

```

    mediump float currentPos = texture(roadmap, vec3(pxPos, 0.0)).y; //Roadmap
may have different size, use NTC; roadmap is linear, interpolation sample is
OK

    mediump float hintUpRoad = 0.0, hintDownRoad = 0.0;
    mediump int hintUpScreen = pxIdx.y, hintDownScreen = pxIdx.y;
    if (texelFetch(hint, pxIdx, 0).r < DEST_THRESHOLD_EDGE) { //If this pixel
is not edge (can be object or road), then the object has move
        for (mediump ivec2 idx = pxIdx; idx.y > limit.x; idx.y--) {
            if (texelFetch(hint, idx, 0).r >= DEST_THRESHOLD_EDGE) {
                mediump float roadPos = texture( roadmap ,
vec3(vec2(idx)/videoSizeF, 0.0) ).y;
                hintUpRoad = abs(currentPos - roadPos);
                hintUpScreen = idx.y;
                break;
            }
        }
        for (mediump ivec2 idx = pxIdx; idx.y < limit.y; idx.y++) {
            if (texelFetch(hint, idx, 0).r >= DEST_THRESHOLD_EDGE) {
                mediump float roadPos = texture( roadmap ,
vec3(vec2(idx)/videoSizeF, 0.0) ).y;
                hintDownRoad = abs(currentPos - roadPos);
                hintDownScreen = idx.y;
                break;
            }
        }
    }

    bool searchUp = false, searchDown = false;
    mediump int hintTargetY = pxIdx.y;
    if ( hintUpScreen != pxIdx.y && hintDownScreen != pxIdx.y ) { //If both
direction can reach object, use cloest direction
        if (hintUpRoad < hintDownRoad) {
            searchUp = true;
            hintTargetY = hintUpScreen;
        } else {
            searchDown = true;
            hintTargetY = hintDownScreen;
        }
    } else if (hintUpScreen != pxIdx.y) { //If only one direction can reach
object, go that direction
        searchUp = true;
        hintTargetY = hintUpScreen;
    } else if (hintDownScreen != pxIdx.y) {
        searchDown = true;
        hintTargetY = hintDownScreen;
    } else { //None direction can reach object, no search
        // Kepp both searchUp and searchDown flase, keep hintTargetY current
pixel y
    }

    mediump int searchTargetY = pxIdx.y;
    if (searchUp) {
        for (mediump ivec2 idx = pxIdx - ivec2(0, 1); idx.y > limit.z; idx.y--)
{

```

```

        if (texelFetch(previous, idx, 0).r >= DEST_THRESHOLD_EDGE) {
            mediump float roadPos = texture( roadmap ,
vec3(vec2(idx)/videoSizeF, 0.0) ).y;
            displaceRoad = abs(currentPos - roadPos);
            searchTargetY = idx.y;
            break;
        }
    }
} else if (searchDown) {
    for (mediump ivec2 idx = pxIdx + ivec2(0, 1); idx.y < limit.w; idx.y++)
    {
        if (texelFetch(previous, idx, 0).r >= DEST_THRESHOLD_EDGE) {
            mediump float roadPos = texture( roadmap ,
vec3(vec2(idx)/videoSizeF, 0.0) ).y;
            displaceRoad = abs(currentPos - roadPos);
            searchTargetY = idx.y;
            break;
        }
    }
}

displaceScreen = hintTargetY;
// displaceScreen = searchTargetY;
}

result = vec2(
    displaceRoad * bias / 255.0, //Normalize to [0,1], offset at 0.0
    0.5 + float(displaceScreen - pxIdx.y) / 255.0 //Normalize to [1,0], offset
at 0.5
);
}

```

Appendix C.9 sample.fs.glsl

@VS

```

layout (location = 0) in highp vec2 meshROI;
out mediump vec2 pxPos;

```

```

void main() {
    gl_Position = vec4(meshROI.x * 2.0 - 1.0, meshROI.y * 2.0 - 1.0, 0.0, 1.0);
    pxPos = meshROI.xy;
}

```

@FS

```

uniform lowp sampler2D src; //Data: Normalied speed [0,255] and y-target
offset at 128 [0,128,255]]

```

```

in mediump vec2 pxPos;
out lowp vec2 result; //Data: vec2(speed, target_yCoord), resolution is 1/256

```

```

#define MIN_SAMPLE_SIZE 2.0

```

```

#define SEARCH_TOTAL x
#define SEARCH_COUNT y
mediump vec2 search(lowp float threshold, mediump ivec2 start, lowp int dir) {
    mediump vec2 res = vec2(0.0); //Accumulation, use mediump
    mediump ivec2 idx = start;
    while (true) {
        mediump ivec2 dest;
        lowp float value;

        dest = idx + ivec2(dir, 0); //Middle
        value = texelFetch(src, dest, 0).x;
        if (value > threshold) {
            idx = dest;
            res.SEARCH_TOTAL += value;
            res.SEARCH_COUNT += 1.0;
            continue;
        }

        dest = idx + ivec2(dir, -1); //Up
        value = texelFetch(src, dest, 0).x;
        if (value > threshold) {
            idx = dest;
            res.SEARCH_TOTAL += value;
            res.SEARCH_COUNT += 1.0;
            continue;
        }

        dest = idx + ivec2(dir, +1); //Down
        value = texelFetch(src, dest, 0).x;
        if (value > threshold) {
            idx = dest;
            res.SEARCH_TOTAL += value;
            res.SEARCH_COUNT += 1.0;
            continue;
        }

        break; //Next, next up, and next down not found, stop search
    }
    return res;
}

void main() {
    mediump ivec2 pxIdx = ivec2(textureSize(src, 0)) * pxPos;

    mediump vec2 res = vec2(0.0);

    lowp vec2 current = texelFetch(src, pxIdx, 0).xy; //For pixels has speed
    mark
    if (current.x > 0.0) {
        mediump vec2 left = search(0.01, pxIdx, -1);
        mediump vec2 right = search(0.01, pxIdx, +1);
    }
}

```

```

        if ( min(left.SEARCH_COUNT, right.SEARCH_COUNT) > MIN_SAMPLE_SIZE &&
abs(0.5 + left.SEARCH_COUNT - right.SEARCH_COUNT) < 1.0 ) { //Remove small
sample (noise); Get center point, left == right or left == right - 1
        mediump float total = left.SEARCH_TOTAL + right.SEARCH_TOTAL;
        mediump float count = left.SEARCH_COUNT + right.SEARCH_COUNT;
        res.x = total / count; //Use avg speed
        res.y = current.y; //Use center pixel's y-value
    }
}

result = res;
}

```

Appendix C.10 display.glsl

@VS

```

layout (location = 0) in highp vec4 pos; //Local object pos, glyph texture pos
layout (location = 1) in highp vec3 speed; //Road pos, speed

out mediump vec3 gpos; //Glyph pos, mediump is good enough for small 2D 256
layers texture

void main() {
    gl_Position = vec4(
        (pos.x + speed.x) * 2.0 - 1.0,
        (pos.y + speed.y) * 2.0 - 1.0,
        0.0,
        1.0
    );
    gpos = vec3(pos.zw, speed.z);
}

```

@FS

```

uniform lowp sampler2DArray glyphmap; //lowp for RGBA8 texture

in mediump vec3 gpos;
out lowp vec4 result; //lowp for RGBA8 video

void main() {
    lowp vec4 color = texture(glyphmap, gpos);
    if (color.a == 0.0) discard;
    result = color;
}

```

Appendix C.11 final.glsl

@VS

```

layout (location = 0) in highp vec2 position;

out mediump vec2 pxPos;

```

```

void main() {
    gl_Position = vec4(position.x * 2.0 - 1.0, position.y * 2.0 - 1.0, 0.0,
1.0);
    pxPos = vec2(position.x, 1.0 - position.y); //Fix the difference in screen
and GL orientation: y-axis upside-down issue
}

@FS

uniform lowp sampler2D originalTexture; //lowp for RGBA8 video
uniform lowp sampler2D processedTexture;

in mediump vec2 pxPos;
out lowp vec4 result; //lowp for RGBA8 video

#define RAW_WEIGHT 0.8
#define PROCESSED_WEIGHT 0.9 //data.a

void main() {
    lowp vec4 data = texture(processedTexture, pxPos); //Use texture instead of
texelFetch because the window size may differ from data size
    lowp vec4 raw = texture(originalTexture, pxPos); //Therefore we need to use
texture sampling function provided by GPU
    result = RAW_WEIGHT * raw + PROCESSED_WEIGHT * data;
}

```

Appendix D Experimental Results

Appendix D.1 Performance Benchmarks – Process Time

Scene	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average	Frame Time
QEW Headless	31.598	33.461	34.338	33.376	33.419	33.989	33.907	33.679	34.33	34.259	33.6356	101.6181269
QEW Aux	36.346	36.244	36.469	36.236	37.406	36.259	36.295	36.314	37.884	36.172	36.5625	110.4607251
Hwy3 Headless	28.658	28.855	28.589	28.84	28.799	28.51	28.784	28.823	28.771	28.802	28.7431	55.48861004
Hwy3 Aux	34.292	34.104	33.984	33.794	33.859	34.277	33.818	34.174	34.326	33.557	34.0185	65.67277992

Appendix D.2 Performance Benchmarks – Stage Time

Scene	Start	DMA init	Blur	Change	Fix	Edge	p2o	measure	o2p	sample	Download	Finish	End w/o Aux	Frame
QEW	3.84917143	36.7292523	10.9906449	2.23025293	51.7201744	6.39580828	2.66738609	6.62230721	4.03052132	2.92932132	0.02009923	14.3069771	0.7831103	143.275027
	3.90451607	37.5981029	11.4163338	2.36293944	56.7380622	7.28527467	2.80251172	7.16058451	4.29127065	3.0618537	0.02008918	14.6018714	0.69393057	151.937341
	3.6734053	36.4313871	10.5882335	2.1191809	51.5429963	6.72465411	2.57201604	7.17731097	4.15662244	2.90777672	0.01923545	14.1307187	0.66279989	142.706337
	3.65839965	37.6984159	11.6355536	2.41120876	56.216717	6.92178011	2.72802141	7.13798008	4.08624596	3.10660846	0.02012696	14.4780721	0.69632703	150.795457
	3.70441322	37.3131751	11.4202549	2.42345118	56.4086997	6.9710337	2.72714163	7.14871353	4.00931805	3.11505629	0.02061983	14.8346343	0.74386843	150.84038
	3.74468249	37.59825	11.6734124	2.35703687	56.6649695	7.10379928	2.77187991	7.26336824	4.14255176	3.19638691	0.0226873	14.3736889	0.71347011	151.626184
	3.61261034	37.9737963	11.5105548	2.36524023	56.31953	7.26012988	2.71041715	7.29059247	4.25488405	3.04812563	0.02091141	14.9186218	0.73752326	152.022937
	3.84524743	37.4315924	11.4995779	2.41676623	56.2121109	6.98086716	2.73329619	7.12644476	4.19247404	3.12374265	0.01972648	14.4353689	0.71819265	150.735408
	3.9340187	36.3806341	10.7320841	2.10833023	51.4385839	6.36223867	2.52041162	7.24469818	4.06737538	3.07038751	0.02009762	14.3091293	0.67786301	142.865852
	3.73126322	36.7293744	10.7080504	2.28935473	51.6467273	6.43091952	2.64924531	6.82082614	4.10331653	3.09032311	0.0213359	14.218502	0.62191249	143.061151
Avg	3.76577278	37.188398	11.21747	2.30837615	54.4908571	6.84365054	2.68823271	7.09928261	4.13345802	3.06495823	0.02049294	14.4607585	0.70489977	147.986607
Scene	Start	DMA init	Blur	Change	Fix	Edge	p2o	measure	o2p	sample	Download	Finish	End w/o Aux	Frame

QEW headless	0.56021634	24.2743078	12.1147803	1.96240078	57.0573257	6.84478325	2.98294972	8.18063682	5.51871986	3.98724215	0.01964578	13.844433	0.1489366	137.496378
	0.59650187	24.5431408	12.3476601	2.06947077	57.4640466	6.95147136	2.934361	8.40618752	5.43493609	3.90866173	0.02032575	14.1575964	0.10505094	138.939411
	0.65990261	24.56988	12.2463482	2.09005584	57.0403899	7.03124885	2.94854194	8.36610806	5.18785433	3.74621818	0.0203592	13.9255817	0.12629493	137.958784
	0.64559797	24.4615598	10.6389439	1.67346274	51.3341747	5.78187214	2.80943219	8.41820783	5.5283019	4.41414578	0.02092392	14.1185684	0.09613073	129.941322
	0.61163756	23.9526605	10.6168661	1.69467579	51.2136476	5.65501871	2.78430514	8.51678236	5.30202882	4.31256589	0.02135551	14.1365094	0.12634611	128.944399
	Bad result, discard													
	0.66891793	23.9315799	12.1490218	2.06447815	57.6211256	6.88769709	2.92681995	8.29250866	5.46344292	3.88297741	0.02013532	13.8291501	0.14535331	137.883208
	0.61117869	24.1275722	10.6600436	1.69268746	51.1783876	5.54758491	2.7233769	8.56511138	5.24099031	4.29215701	0.02031644	14.196191	0.1168967	128.972494
	0.49104486	24.021931	10.567314	1.74915759	51.2276247	5.68316891	2.60312549	8.40090061	5.29094683	4.16308881	0.02008129	13.8630762	0.12984686	128.211307
	0.63066256	24.2076985	10.6949552	1.76008266	51.1680989	5.85577347	2.74806762	8.22592765	5.12763874	4.0256464	0.02043341	13.822391	0.11494397	128.40232
Avg	0.60840671	24.2322589	11.3373259	1.8618302	53.9227579	6.24873541	2.82899777	8.37470788	5.34387331	4.08141149	0.0203974	13.9881663	0.12331113	132.97218
Scene	Start	DMA init	Blur	Change	Fix	Edge	p2o	measure	o2p	sample	Download	Finish	End w/o Aux	Frame
Hwy3	3.81698433	33.4400359	10.093857	1.60197838	19.5696254	3.27563094	1.52319832	2.03754098	2.2614608	2.12249231	0.01682351	7.21363407	0.44341394	87.4166759
	3.37225032	33.1827714	10.0988882	1.76877383	19.685971	3.30551131	1.43626228	2.06479025	2.02945076	1.85340699	0.01587139	7.16456202	0.41547182	86.3939816
	3.87582238	32.8591404	10.0535595	1.78914122	19.7746294	3.35257381	1.4293172	2.00675309	2.07709528	1.84145143	0.01592245	7.24384123	0.40003647	86.7192838
	3.27197327	32.9929624	10.0399283	1.63325134	19.5114211	3.31612073	1.43675661	2.04935795	2.27423789	1.98356613	0.01619652	7.18900283	0.41035926	86.1251343
	3.50989695	33.3165544	10.0883044	1.66865662	19.6725328	3.38402535	1.43187587	2.08817131	2.23121615	1.9461461	0.01631169	7.21437254	0.42357693	86.9916411
	3.55290689	33.1602457	10.040456	1.69732511	19.6642142	3.21156156	1.4534315	2.14612813	2.07646248	1.87871911	0.01607228	7.1793912	0.40507692	86.4819911
	3.42820481	33.7656564	10.950556	1.70571088	21.5099015	3.55678601	1.4954309	2.18250874	2.32493042	2.04540028	0.01679537	7.26635437	0.39909208	90.6473277
	3.22854208	33.0708165	9.99766604	1.50077721	19.5440676	3.35221963	1.46302039	2.17047221	2.26360574	1.97323197	0.01636761	7.24194184	0.44155232	86.2642811
	3.26193895	32.9749782	9.935987	1.62188326	19.5884234	3.32202644	1.43146682	2.02497137	2.23453327	1.91223412	0.01571152	7.12848482	0.39409259	85.8467318
	3.23716926	33.0899036	9.98420015	1.66137029	19.7642701	3.41300538	1.47324302	1.98209038	2.17429519	1.93205938	0.01620526	7.09938691	0.37943854	86.2066375
Avg	3.45556893	33.1853065	10.1283403	1.66488681	19.8285056	3.34894612	1.45740029	2.07527844	2.1947288	1.94887078	0.01622776	7.19409718	0.41121109	86.9093686
Scene	Start	DMA init	Blur	Change	Fix	Edge	p2o	measure	o2p	sample	Download	Finish	End w/o Aux	Frame
Hwy3 headless	0.400544	22.0547548	10.0571428	1.27102625	19.2938698	3.35555189	1.61249585	2.51135169	2.66080769	2.46912349	0.01850567	7.48505913	0.08291046	73.2731436
	0.41488628	22.4222259	10.001415	1.32730826	19.3656305	3.32193363	1.60887621	2.45429671	2.55733101	2.43724166	0.01685061	7.44642303	0.0543496	73.4287684
	0.60102607	22.5135813	10.0674152	1.34722558	19.3785687	3.40289233	1.59189167	2.36394983	2.62949229	2.42649222	0.0174764	7.3180775	0.05949673	73.7175857
	0.48227269	22.5638939	10.0326938	1.36201963	19.2977114	3.47828376	1.49761049	2.38861247	2.73792092	2.39079072	0.01686228	7.30399895	0.06289715	73.6155682
	0.41252654	23.2197624	11.3454296	1.54770711	22.6267125	3.95788109	1.80814121	2.66878055	3.07791941	2.68987161	0.01822785	7.59042352	0.07267154	81.036055
	0.39884002	21.9497257	9.99480605	1.36304441	19.2983402	3.40171734	1.57836875	2.37118202	2.71234636	2.36345961	0.01726646	7.23597023	0.07650189	72.761569
	0.41594489	22.1715983	10.0083025	1.24118674	19.2411046	3.42837057	1.57614469	2.44705761	2.77336096	2.5254074	0.01779802	7.42500671	0.08806679	73.3593497
	0.44913331	23.2916224	11.805849	1.51638398	23.5341192	4.26621654	1.78614837	2.86681522	3.37392138	2.79379733	0.01831301	7.82282375	0.06916361	83.5943071
	0.44364653	21.8820149	9.96556416	1.31303255	19.2577649	3.41086859	1.53529173	2.47041154	2.68591268	2.49624704	0.01700245	7.294027	0.06482377	72.8366078
	0.41656694	22.3432305	9.99085602	1.30663471	19.3257905	3.54174438	1.50959058	2.32631598	2.76960968	2.39626218	0.01743641	7.21231142	0.05520614	73.2115554
Avg	0.44353873	22.441241	10.3269474	1.35955692	20.0619612	3.55654601	1.61045595	2.48687736	2.79786224	2.49886932	0.01757392	7.41341212	0.06860877	75.083451

VITA AUCTORIS

NAME:	Aiming Deng
PLACE OF BIRTH:	Chongqing, China
YEAR OF BIRTH:	1996
EDUCATION:	University of Windsor, B.ASc., Windsor, ON, 2019 University of Windsor, M.ASc., Windsor, ON, 2023