Oregon State University
College of Engineering

CS Capstone Technology Review

November 21, 2017

100K Spaceport America Demonstration Rocket Project

Prepared for

School of MIME

Nancy Squires _____ _____
Signature *Date*

Prepared by

Group 42

Glenn Upthagrove _____ _____
Signature *Date*

Abstract

In this technology review various solutions are explored and compared to solve three sections of the 100k Rocketry project for the computer science sub-team. The first explored is three solutions for making a 3D trace of the flight path. The second is how to log the data coming in from the hardware on the ground station. The final piece is an exploration of how to handle the data coming infrom the hardware before it is passed off to logging.

CONTENTS

1 Role in Project

For the most part, I shall be focused on software running on the ground station. The data shall be travelling through a pipeline, starting at collection on the rocket, and ending at visualization. I sit mostly in the middle, where the data is collected from the hardware, formatted, logged and passed further down the pipeline. I also will handle a small protion of the visualization, as I shall be doing the 3D trace program.

2 Overview of Project

In this project we are, at a high level, trying to track a rocket during its flight. This implies that our job is more specifically, to clean up the data of noise before transmission from the rocket, to gather the data as it comes in from hardware on the groundstation, handle and format the data, log it, and display it in a meaningful way. The sections covered in this document are the handling of the data, the logging of the data, and the 3D aspect of the visualization.

3 3D Trace

This section covers possible solutions for the 3D trace of the flight path. Author: Glenn Upthagrove.

3.1 Criteria

The tools used to make the 3D trace must meet certain criteria. The first being that it must be capable of running on as reduced a hardware profile as would be seen on a Raspberry Pi, as it is very possible that will make our ground station. It must be capable of running on many different graphics accelerators, not specifically to any on emanufacturer. The final criteria is that it must be easy easy enough to program in, sych that it is reasonable to complete in the reduced time window alloted to our group.

3.2 OpenGL

The OpenGL Application Programming Interface (API) is a simple high level abstraction of graphics card interaction controlled by the Khronos group [1]. This allows the programmer to leverage the Graphics Processing Unit (GPU) is rendering 3D scenes. The API is simple, and allows significant results in a much shorter time frame than other graphics libraries. It is also a cross-platform solution, allowing implementation on many devices, such as the Nvidia Jetson TK1 or most desktops [1] [2]. It is also the most widely used graphics API in industry, giving us access to a robust development community [1]. The library can also be accessed through most programming languages [1]. There are faster options on the market, such as Vulkan, but are usually much harder and more specialized than OpenGL[3]. This library also has bonuses in personal familiarity and simplicity, making implementation time much shorter, and eliminating most if not all time to learn the required materials for the trace. This is an important consideration since our full software suit must be working by the end of winter term, in time for the practice launch.

3.3 WebGL

WebGL, also of the Khronos group, is an API for using the reduced form of OpenGL, known as OpenGL ES, on the web, through ECMAScript and the HTML5 canvas element [4]. This API allows for a more basic 3D experience, but capable of being run on less powerful hardware. The requirements of the 3D trace of the flight path are such that these reduced capabilities should cause no degradation in quality. This library also should work well on a Raspberry Pi, if

that so happens to become the ground station, as it is supported by web browsers that Linux can run, such as Firefox and Chrome, and the underlying OpenGL ES is supported by the Raspberri Pi [4] [5]. WebGL also has the added bonus of being very similar to OpenGL, making crossover fairly straightforward [4].

3.4 Vulkan

A much younger graphics library under the Khronos group, is Vulkan. Unlike OpenGL it is not a high level abstraction, but instead is meant to be a much lower level interaction layer, that can leverage much more power out of the system, while higher level APIs can stay relatively unchanged [3]. Vulkan can achieve much of this by clever uses of parallelism in the pipeline [3]. While this can allow for a noticeable improvement in performance for many graphically intense applications, this would be wasted on such a simple program such as our 3D trace. Vulkan is also much more verbose and requires a much longer development time, especially considering it being very unfamiliar to us.

3.5 Conclusion

Considering the requirements of the 3D trace, and the time constraints of our project, I believe OpenGL is the best solution. It is reasonable to believe that due to the reduced hardware profile of a Raspberry Pi, WebGL would be a better solution, however OpenGL is supported on the Raspberry Pi as well [6]. Since the scope of our project is fairly substantial, and we have to be done within one term to meet the date of the test launch, it is more important that we can get all the software completed in time rather than make it as absolutely efficient as possible. Since time is such a limiting factor, the familiarity of OpenGL and the vastly simpler code, makes it a far superior choice for this project than Vulkan. OpenGL is far simpler than Vulkan, more familiar than either Vulkan or WebGL, and more than capable of rendering the trace, OpenGL is the appropriate choice for this project[7].

4 LOGGING

This section covers options for formatting the data for storage in non-volatile memory. Author: Glenn Upthagrove.

4.1 Criteria

The form in which we output the data is critical for parts further down the pipeline. The data must be easily accessable to the pther parts of the system downstream, and thus it is important that it is either stored as it will be used, or at least in such a maner as to minimize reprocessing by later modules. The output should also be human readable, as there shall be an output to a log file.

4.2 Pipe Delimited

This would be the simplest way to format and store the data, where each float is separated by a pipe character, and each set of floats is ended by a newline character. This would allow for a very minimalistic formatting process to simply break up the data and write it to a file. The data could be later opened by any of the other portions of our suite for later use. While this is both easy to implement and would minimize processor usage, it would mean that any other program accessing the data may have to do additional formatting to make it useful for that program.

4.3 JSON

Java Script Object Notation (JSON) is a widely used format for storing and transmitting data [8]. The data can be stored as a variable, written as plain text and then read back into a variable with the same structure. The syntax is familiar to most languages that are derived from C, and thus can be easily manipulated in many of these languages, and most have libraries already available to do this [8]. Python has a library readily available to handle JSON, which could be used to make the logging into a file very easy, especially with how well Python handles strings [9]. JSON would also make the data more easily available to the 2D visualization, and the 3D trace could be written to use it as well, adding no conversion step in between the log and these two other modules.

4.4 YAML

YAML (YAML Aint Markup Language) is a superset of JSON [10]. It uses very similar syntax, and is in fact more capable, as it can do all the things JSON can do but with extensions on that syntax. It is also similar in size and complexity, but is more human readable, at the expense of parsing ease. When compared to JSON, YAML is less well adapted to our other modules, and unlike JSON, a YAML library for Python would need to be separately installed [11].

4.5 Conclusion

The log shall be stored in JSON. This format is slightly larger and more complex than the pipe delimiter, but also more human readable, and more easily integrates with other portions of the project. JSON is also less readable than YAML, but easier to parse, has more readily available libraries, and will integrate more easily with the rest of the project. These factors make JSON by far the best option. Additional logs could also be easily added in future upon the request of another member of this or any other sub-team[7].

5 Data Handling

This section shall discuss options for which language or combination thereof to handle incoming packets on the ground station. Author: Glenn Upthagrove.

5.1 Criteria

The language choice for the module handlign the data once it is recieved by the hardware is critical. this implies that the langauge must forst be able to access the data from the hardware. The language should also consider how easily it can manipulate said data once it is retrieved. The language should also take into account any libraries that are availabel to make these probles easier, such as libraries that can turn data into the chooses output format. Familiarity, while not a deciding factor, should also be factored in, as the time window we have is not broad, and familiarity makes bugs easier to find and fix, which is critical on software that must be running in near real time.

5.2 C/C++

The C language is a high level language, but allows one to get very close to hardware, and even drop directly into assembly if need be. C++ is very similar to C, and can for the most part even use C code inside it, but allows for more features, as it is an object oriented language. C and C++ could both be very good options for dealing with the data as it is recovered from hardware, as it allows for very close memory access and manipulation. C and C++ are also very familiar languages to us all, and would reduce the time taken to develop and debug. Storing the data and passing it to other sections would be the downside of C, as it is not as easy to manipulate strings in C as in some other languages.

5.3 Python

The Python language is a high level scripting language, with similar syntax to C. Python is on average slower at run time than C/C++, as it is not compiled. However Python also is generally somewhat shorter and easier to write than C/C++ and could speed up development time. Python also is often better at string manipulation than C/C++ and could make the passing of data to the rest of the project modules easier. There is also a readily available library for using the JSON format we intend on using in other sections of this program [9]. The weak spot however is that Python being generally more abstracted than C/C++ would make the retrieval of the data from the hardware much more difficult to do, on top of running far slower than a C/C++ solution.

5.4 C/C++ and Python

As seen above both a pure C/C++ solution and a pure Python solution, seem to have certain strengths and weaknesses that correspond quite well. In simpler terms, they complement one another, and the obvious solution is to combine them, which luckily can be done with relative ease. The retrieval of the data from the hardware could be easily done in C/C++, then this solution could either be called by a Python program which takes the data from the C/C++ as a return, or they could be separate modules that are continuously running, and the C/C++ solution passes its data to the Python module over a pipe or through a file. The Python then converts the data to JSON and then passes it along to any other parts of the suite that need the data. The C/C++ portion could also speed up the program more than a pure Python solution.

5.5 Conclusion

As can be clearly seen from the above discussions, the best solution is the combine the speed and superior hardware access of C/C++ with the better string and JSON handling of Python. Together they can make a very elegant solution for retrieving, formatting, and storing the data as it comes into the ground station. The speed of the ground station hardware will also be superior to the rocket, and thus should have plenty of time to run a simple Python script, especially since part of the work will be offloaded onto a fast C/C++ solution[7].

REFERENCES

[1] T. K. Group. Opengl overview. [Online]. Available: https://www.opengl.org/about/

[2] Nvidia. Jetson tk1. [Online]. Available: http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

[3] T. K. Group. Vulkan overview. [Online]. Available: https://www.khronos.org/assets/uploads/developers/library/overview/vulkanoverview.pdf

[4] ——. Webgl specification. [Online]. Available: https://www.khronos.org/registry/webgl/specs/latest/1.0/

[5] R. P. Foundation. Raspberry pi 3 model b. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[6] S. long. Another new raspbian release. [Online]. Available: https://www.raspberrypi.org/blog/another-new-raspbian-release/

[7] e. a. Anisimova. Groundstation: Technology review. [Online]. Available: https://github.com/100k-rocketry/groundstation/blob/master/doc/technologyreview/technology-review-version-2.pdf

[8] R. Data. Json - introduction. [Online]. Available: https://www.w3schools.com/js/js_json_intro.asp

[9] P. S. Foundation. 18.2. json json encoder and decoder. [Online]. Available: https://docs.python.org/2/library/json.html

[10] e. a. Oren Ben-Kiki. Yaml aint markup language (yaml) version 1.2. [Online]. Available: http://www.yaml.org/spec/1.2/spec.html#id2759572

[11] C. C. Evans. Yaml 1.2. [Online]. Available: http://yaml.org/