# CS Capstone Technology Review

November 21, 2017

# Inferno

Prepared for

## OSU - Phoenix Solar Racing


Prepared by

## Group 43

## Inferno

Logan Kling



**Abstract**

This document is a detailed comparison between different technical options behind the programming of a simulation for a solar powered car. I research different options for the math library, the physics framework, the aerodynamics and computational fluid dynamics (CFD) packages, and the statistical model. For each of these sections, excluding the last, I have given an overview of what the piece is, an overview of what I'm looking for, at least three options for that piece, a discussion which compares and contrasts those pieces, and then a final conclusion on what piece will be used. The statistical model is a special case, as it's something I don't have a choice on. Given that, I simply had a long description explaining why I only had one choice with the statistical model.

## CONTENTS

# 1 Math Library

## 1.1 Overview

While all the calculations we will need to do could probably be done in native C++11, a math library will make coding this application at least two times easier, and should decrease development time by up to 50%.

## 1.2 Criteria

At a bare minimum this library should support exponential and logarithmic functions, along with floats with over 24 decimal points of accuracy. Hopefully, this library would also support basic calculus functions, as well as allow us to do anything a scientific calculator can do with ease. Another bonus for a math library would be support of vector calculus and linear algebra. While multithreading and support for numbers larger than 64 bits would be helpful for our program, they are very far from essential.

## 1.3 Potential Choices

### *1.3.1 ArrayFire*

ArrayFire describes itself as a high performance software library for parallel computing with an easy-to-use API. Its array based function set makes parallel programming more accessible. It is also very optimized for GPU (Graphical Processing Unit or Graphics Card) computations (as a GPU often has dozens of processing units with dozens of "threads" each. This library is also the only one that's developed by a private corporation. Despite this, ArrayFire is open source. [1]

### *1.3.2 Boost*

The Boost library is a general C++ library which tries to expand and increase the functionality of C++. Boost is a truly massive set of libraries. Amid all of this, it also has support for larger numbers and a few exponential and logarithmic functions. [2]

### *1.3.3 GNU MP Bignum C++ Interface*

This is simply a library which expands on all the types, integer, and float. More specifically, it allows a user to create those variables with the only limit of size being the amount of free memory the computer has. It also adds rational numbers and its own number generator. [3]

### *1.3.4 NTL - A Library for doing Number Theory*

NTL is a high-performance, portable C++ library providing data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields. It's developed by one researcher, Victor Shoup. [4]

### *1.3.5 G+Smo cross-platform library for isogeometric analysis*

G+Smo (Geometry + Simulation Modules, pronounced "gismo") is an open-source C++ library that brings together mathematical tools for geometric design and numerical simulation. It is developed mainly by researchers and PhD students. [5]

## 1.4 Discussion

While the boost library has the most generalized functionality, I would argue that it is the least suitable for this task. The Boost library is far more than a math library and we aren't looking for much more than a math library. Boosts generality adds overhead and on top of that, it's nowhere near as optimized as ArrayFire, the GNU Bignum library, NTL, or even G+Smo. G+Smo is also pretty well optimized, but it just isn't the best solution for this. Ideally, we want a math library which integrates well with our physics library. G+Smo is a geometry library. The GNU Bignum library is very good at what it does. It allows one to create numbers with as much size as there is space on the heap. It's also surprisingly fast with such massive numbers, especially given that from everything I could find it's single threaded. It adds whole new number types along with its own ultra-efficient functions. I'm partial to this as I've used it before, but it was originally designed for C90—which is what I used it with. It would be better to use a library with more functionality and one that's truly designed for C++11.

This brings us to NTL and ArrayFire which seem to have a lot in common. These are both array based, like the GNU Bignum library, but they're more up to date and have more functions and parallel programing as a result. I actually couldn't find any mention of the NTL library being optimized for, or even supporting GPU processing. This project will allow us to split up our calculations into dozens, if not hundreds of processes. Even a very low-end GPU will be at least twice as fast as an eight core CPU. Almost no one on the Phoenix Solar Team has an eight core CPU anyways. ArrayFire also appears to be far more actively developed than NTL, and certainly has a better marketing team than NTL. Of course, this difference in development cycle could mean NTL is far more stable. It could also mean NTL is dead, though. NTL's last update was on the 18th of November 2016. That's over a year ago at this point.

### 1.5 Conclusion

Ultimately, we decided to go with ArrayFire because we're young and reckless. In all seriousness though, ArrayFire appears to be a *far* better choice. It has more documentation, and it was designed with speed and developers in mind. NTL appears to have been a research project and actually has slightly less functionality than ArrayFire. While it's highly unlikely we will need all or even most of ArrayFire's functionality, it will be nice to know that we have it there. In the worst case scenario in which ArrayFire doesn't work out for us, we can start using NTL. We welcome any feedback on this, though, especially critical feedback.

## 2 Physics Framework

### 2.1 Overview

A physics framework will be almost essential for this project. The physics in this is probably going to be far more complicated than the physics in Physics 211  213. A physics framework will hopefully stop us from pulling *all* our hair out over this project.

### 2.2 Criteria

Our physics engine should be pretty basic. While there are real-time physics engines and high-performance physics engines, our engine should be below both of those in accuracy. We need to simulate a 2000 mile drive in under 30 minutes, after all. Honestly, we hope to be able to simulate a 2000 mile drive in under 10 minutes, but that's an extremely ambitious goal. Our car should be an object with rigid body dynamics, and the complexity of our physics calculations should allow for us to simulate at least half a mile driven in a second or less.

### 2.3 Potential Choices

#### *2.3.1 Advanced Simulation Library*

Advanced Simulation Library is a free and open-source hardware-accelerated Multiphysics simulation platform which enables users to write customized numerical solvers in C++ and deploy them on a variety of massively parallel architectures. This means that this software is meant to be run on powerful computers with high processor count CPUs and powerful GPUs. ASL is meant for very high accuracy physics simulations with a lot of detail. [6]

#### *2.3.2 Bullet*

Bullet is a cross-platform physics engine coded in C++ which can simulate collision detection, and soft and rigid body dynamics. It's a somewhat older engine which has been used in a variety of video games and movies. Its most recent improvements have been focused on robotics, reinforcement learning, and VR. [7]

#### *2.3.3 Newton Game Dynamics*

Newton Game Dynamics is a cross-platform physics engine written in C++ for realistically simulating rigid bodies in games and other real-time applications. Its solver is deterministic and not based on traditional Linear Complementarity Problem (LCP) or iterative methods. As troubling as the spelling error on this engines home page is, its ability to be split up into multiple processes (with some work from the programmer) and permissive Zlib license are appealing. [8]

#### *2.3.4 Open Dynamics Engine*

Open Dynamics Engine is a cross-platform physics engine written in C/C++ which consists of two main components: a rigid body dynamics simulation engine, and a collision detection engine. It supports a variety of basic shapes and is a popular choice for robotics simulation applications. [9]

#### *2.3.5 Project Chrono*

Project Chrono is a cross-platform physics engine written in C/C++ developed by University of Wisconsin-Madison and University of Parma. A few of the things it supports are simulating rigid and soft body dynamics, collision detection, and vehicle dynamics. Project Chrono was initially created to be used as a multibody simulation tool for robotics and biomechanics applications. [10]

### 2.4 Discussion

The Advanced Simulation library is far more than what's needed for our project. One of its advertised uses is simulating the conflicting flow of two fluids in great detail. That's not really something that can be accelerated, and it's definitely not something we need. Bullet would be pretty good for our project if we were able to use a previous version. The newest changes to Bullet have all been changes we do not need. Now, the Newton Game Dynamics, Open Dynamics, and Project Chrono engines all have a lot in common, but Project Chrono still has far more than we need. We won't need collision detection, or soft body dynamics. In fact, because of those engines similarities, even the Newton Game Dynamics, and the Open Dynamics Engines both have a little more than we need. OpenDynamics also has less than we need, though. OpenDynamics engine was initially created in 2001. Because of this it is explicitly iterative, and doesn't natively support complex shapes, unlike Newton Game Dynamics. The reason we need support for complex shapes is that the car will ultimately be shaped like a teardrop. Even Bullet takes an explicitly iterative approach, which is disappointing considering how reliable its diversity of uses show it to be.

### 2.5 Conclusion

We will be choosing the Newtonian Game Dynamics Engine because it has about as few components as the Open Dynamics Engine, but its algorithms are explicitly deterministic. The developers of the Newtonian Game Dynamics Engine are even working on an engine that can use multiple cores. Given this, it's clear that the Newtonian Game Dynamics Engine was designed speed and optimization in mind.

## 3 Aerodynamics and CFD Packages

### 3.1 Overview

CFD stands for Computational Fluid Dynamics. These packages simply determine how air (and possibly even rain) resistance factor into the power draw of the car. With any luck, we won't need these packages, but there's a small chance that we will.

### 3.2 Criteria

Whatever solution we use for this, it must be able to calculate the air resistance on the solar car with at least 70% accuracy of the actual air resistance on the car in the real world. The Phoenix Solar Racing Team has aerodynamic constants from solid works. There's a pretty good chance that those numbers will be more accurate than anything we could calculate. If they're off from the real-world numbers by more than 30%, we will need to come up with our own aerodynamic and CFD packages in order to produce more accurate results.

### 3.3 Potential Choices

#### *3.3.1 OpenFOAM*

OpenFOAM is a UNIX/Linux C++ toolbox for the development of customized numerical solvers, and pre-/post-processing utilities for the solution of continuum mechanics problems, including computational fluid dynamics. OpenFOAMs most distinguishing feature is its syntax. Its use of operator overloaders makes it ideal for custom solvers—provided one never needs to delve to deeply into the many forks of its library. [11]

#### *3.3.2 OVERFLOW*

The OVERset grid FLOW solver is a UNIX/Linux software package for simulating fluid flow around solid bodies using computational fluid dynamics. This software was originally developed in the early 1990's. For this reason, it has been widely used to better understand the aerodynamic forces on vehicles moving at enormous speeds by evaluating the flow field surrounding those vehicles. [12]

#### *3.3.3 SU2*

The Stanford University Unstructured (SU2) suite is a cross-platform and open-source collection of software tools written in C++ for performing Partial Differential Equation (PDE) analysis and solving PDE-constrained optimization problems. The toolset is designed with computational fluid dynamics and aerodynamic shape optimization in mind. It is designed to be extremely flexible. As such it has also been used for several multi-physics simulations, including but not limited to: combustion modeling, two-phase flow simulations, magnetohydrodynamics simulation, and multi-species thermochemical non-equilibrium flow analysis. [13]

### 3.4 Discussion

These are all fairly similar. OpenFOAM is the lightest of these packages, but it is still expandable. While the development of OpenFOAM has recently stagnated, that's because of fragmentation due to many forked projects. This means there are a lot of small libraries that can be added onto OpenFOAM if they're needed. Like OVERFLOW, OpenFOAM is Unix/Linux only OVERFLOW was designed by, and for, NASA. Given this information, it really isn't optimal for our purposes as this was designed for rockets which frequently fly at over twice the speed of sound. SU2 is sort of like a newer version of OpenFOAM that can run on Windows, Mac, and Linux. Though, as can be seen from its ostentatious description, it can also run a ton of simulations, many of which have almost nothing to do with aerodynamics. However, SU2 is quite large and it contains far more packages than we need.

### 3.5 Conclusion

We would have decided to choose OpenFOAM for this because it's smaller and lighter than SU2, but OpenFOAM is Linux/Unix only. So, we will be using SU2 if the aerodynamic constants from solid works provided by the Phoenix Solar Racing Team are too inaccurate.

## 4 Statistical Model

### 4.1 Overview

A statistical models' definition is a little counterintuitive. A statistical model isn't how we graph or store our data, but how our variables correlate with each other.

### 4.2 Criteria

Unfortunately, this means I have no choice between statistical models. The laws of physics are constant anywhere near this scale. The only statistical model I will have is that of the real world.

### 4.3 Discussion/Conclusion

Our statistical model is pretty simple from a general perspective, but it can get a little complicated if one dives into the actual equations. Because of that, I'm not going to give anything more than a general perspective. The idea behind calculating how much energy the car will use is based on how much energy it takes to increase or maintain the momentum of the car with its coefficient of friction. Momentum is the product of mass and velocity, so the heavier the car, the more energy it will take to speed it up. The car will also lose more energy going up a hill, and less energy going down. Somewhat unexpectedly, though, the car will lose more energy going up a hill the heavier it is, but will lose almost the same amount of energy going down a hill, no matter its weight, for angles less than $\approx 15^\circ$. The car will also have some of its power loss offset by how much energy it generates from its solar panels. That's just basic arithmetic. The car also has regenerative braking. This probably won't be in our simulation based on how detailed our map would need to be. In order to take the regenerative braking into account, we have to be able to accurately predict how much cumulative velocity will be lost from braking. In order to do that, we need to know which intersections have stop signs and which ones have lights. Due to this, it's very unlikely we will simulate any energy being saved with the regenerative brakes.

## References


[1] Pavan Yalamanchili, Umar Arshad, Brian Kloppenborg, Miguel Lloreda, Ghislain Antony Vaillant, Cedric Nugteren, Filipe Maia, Gatan Lehmann, Nathan Jackson, Richard Klemm, Andreas Schuh, and Vardan Akopian, "Arrayfire." [Online]. Available: https://arrayfire.com/

[2] BOOST Team, "Boost (c++ libraries)." [Online]. Available: http://www.boost.org/

[3] GNU Project, "Gnu multiple precision arithmetic library." [Online]. Available: https://gmplib.org/

[4] Victor Shoup, "Number theory library." [Online]. Available: http://shoup.net/ntl/

[5] Andrea Bressan, Florian Buchegger, Antonella Falini, Michael Haberleitner, Christoph Hofer, Clemens Hofreither, Mario Kapl, Gabor Kiss, Stefan Kleiss, Angelos Mantzaflaris, Svetlana Matculevich, Stephen Moore, Dominik Mokris, Felix Scholz, Jarle Sogn, Jaka Speh, Stefan Takacs, Jrgen Vogl, and Harald Weiner, "G+smo." [Online]. Available: http://gs.jku.at/trac/gismo

[6] Avtech Scientific, "Advanced simulation library." [Online]. Available: http://asl.org.il/

[7] Erwin Coumans, Benjamin Ellenberger, Victor Nicolaichuk, and Oleg Klimov, "Bullet." [Online]. Available: http://bulletphysics.org

[8] Julio Jerez and Alain Suero, "Newton game dynamics." [Online]. Available: http://newtondynamics.com/forum/newton.php

[9] Russell Smith, "Opendynamicsengine." [Online]. Available: http://www.ode.org/

[10] Radu Serban, Alessandro Tasora, Hammad Mazhar, Dario Mangoni, and Conlain K., "Project chrono." [Online]. Available: https://projectchrono.org/

[11] Henry Weller and CFD Direct, "Openfoam." [Online]. Available: https://openfoam.org/

[12] NASA, "Overflow." [Online]. Available: https://www.nas.nasa.gov/about/staff/tpulliam.html

[13] Heather Kline, Tim Albring, Brendan Tracey, Ruben Sanchez, Salvatore Vitale, David Thomas, Francisco D. Palacios, Antonio Rubino, and Samet Cakmakcioglu, "Su2." [Online]. Available: https://su2code.github.io/