Semester Project: Trajectory Prediction in Particle Colliders

Numerical Computing

Formulation of Algorithms and Optimization Techniques to Preserve Storage Space for Collision Data in Particle Colliders

Author

Name: Syed Mustafa Ahmed

Registration Number: BCS183102

Section: 03

Instructor

Dr. Masroor Ahmed

Department of Computer Science

Capital University of Science & Technology, Islamabad

Pakistan

Table of Contents

Abstract

The Particle collider at CERN bombards particles close to the speed of light. This generates many subatomic particles and the 8 different particle detectors collect data about them throughout the process. This data is immense in size (25 petabytes per year) and thus bottlenecks the number of collisions the facility can perform in a given period of time.

An algorithm that predicts the supposed trajectory of particles can focus on collecting data in more focused regions. Additionally, techniques that support better storage efficiency need to be implemented alongside the algorithm for better results. This can decrease the amount of space required per collision to save its data hence possibly aiding in costs, frequency of runs, data management etc. Additionally, numerous other optimization steps can be added to this methodology to ensure the best results.

Keywords: Trajectory Prediction, Optimization, Particle Colliders, Algorithm Creation

Introduction

About

CERN is the largest particle physics laboratory in the world. It contains numerous stages of accelerators and detectors. These detectors are placed at different locations and vary in what particles they can detect and how they do it. The project's concern is the collision of protons i.e. Hydrogen ions. Still, the calculations are general enough to accommodate for changes for different masses, velocities, inertia etc. of other particles and manner of detection.

Specification

CERN has eight total detectors i.e. ALICE, ATLAS, CMS, TOTEM, LHCb, LHCf, MoEDAL and FASER **[1]**. Specifically, the project's concern is with the ATLAS project which is a general-purpose detector at CERN. The detector itself consists of many sensitive instruments such as a muon detector and two types of calorie-meters (electromagnetic and hadronic) to measure energies. The hadronic calorie-meter is responsible for detecting protons and neutrons while the electrons and photons are detected using the electromagnetic calorie-meter. The project is mostly concerned with the hadronic calorie-meter but the same principles of collisions can be attributed to collisions of/resulting in electrons and photons.

The detector is barrel-shaped and not spherical. This will make it easier when the 3D space is to be projected onto a 2D plane for noise maps.

Methodology

Existing Methods

LHC currently keeps lossless data records archived in Tape form, distributed by File Transfer System (FTS) and handled by the EOS hard disk-based open-source system. The CERN Advanced Storage Manager (CASTOR) is responsible for keeping these high volume 6 million annual files in a hierarchical storage model. The possible improvement to this model mentioned in the project is the introduction of more accurate and low-level decimals. **[2]**

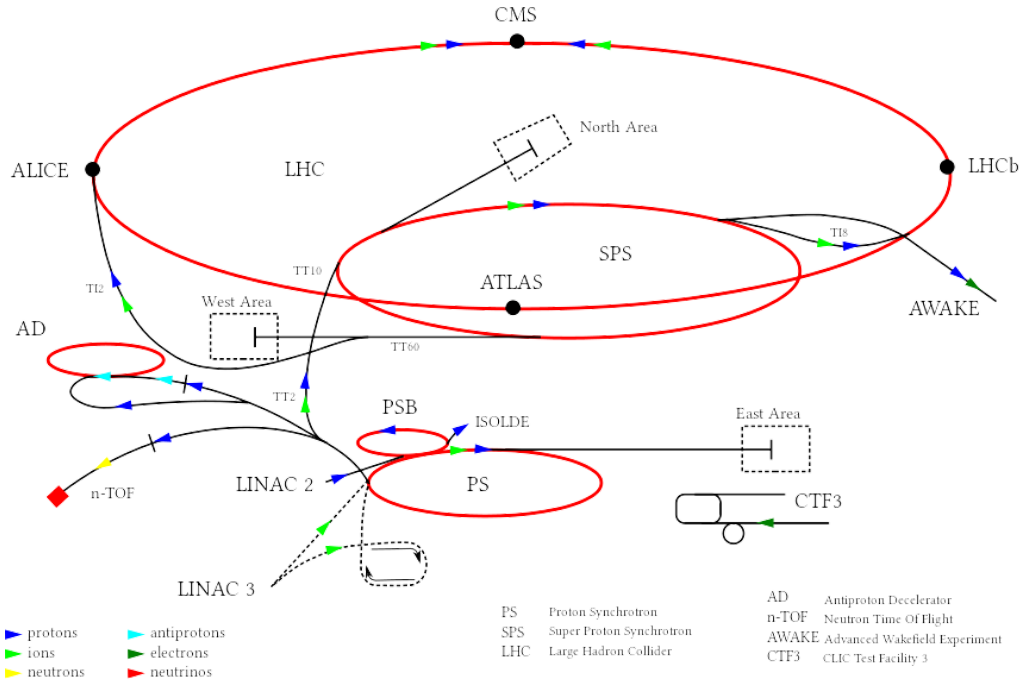

Figure 1: How CERN Operates

On the other hand, collisions themselves do not have a working algorithm for storage saving measures since compression is detrimental to the data itself. Therefore, techniques only limited to archival are applied to the data. This, in conclusion, means that this project represents techniques that can possibly be extended as applications at LHC.

Metrics

For the calculations regarding trajectory calculations, we need to take into account the factors that are general and not specific to certain particles. These influences dictate how well the actual system works. The basic measurements are as follows:

m	-	mass of particle
m_n	-	mass of each subatomic particle
n	-	number of subatomic particles
v	-	velocity of particles
v_n	-	velocity of each subatomic particle
q	-	charge on particle
q_n	-	charge on each subatomic particle
s	-	spin of particle
s_n	-	spin of subatomic particle
p	-	momentum of particles
p_n	-	momentum of subatomic particle

In this list, m is a huge influence while n tells the algorithm to scale itself and recalculate itself. The v of particles will also dictate v_n since that same velocity will be split into energies for the subatomic particles to move. Here, q is a weak influencing force that at such high velocities barely matters but is a necessary factor since the algorithm needs to correct for distribution according to the remaining factors. Alongside charge, spin is a factor in determining how the particle will curve and the location where it will fall onto the detector **[3]**. Additionally, the manner of the composition of the colliding particle itself

For protons specifically,

m	=	$1.67262192369 * 10^{-27}$		
q	=	$1.602176634 * 10^{-19}$ C	or	+1e
s	=	1/2		

The velocity of the proton in the particle accelerator is close to the speed of light i.e.

$v \approx 299792458$ m/s

It should be noted here that since the theory of special relativity doesn't allow the particle to move at the speed of light, it's better to take into account the particle's energy or momentum rather than the speed. In a proton, n is known to be 3 i.e. 2 up quarks (u) and two down quarks (d). So,

$n = 3$ (per proton)

The up quark has,

$m_{up} = 1.7 - 3.3$ MeV $=$

$q_{up} = +2/3e$

$s_{up} = 1/2$

while the down quark has,

$m_{down} = 4.1\text{-}5.8$ MeV $=$

$q_{down} = -1/3e$

$s_{down} = 1/2$

We can see that here the mass of the quarks is written in MeV (Mega Electron Volts). This is also a unit of mass since the units for both mass and energy and often used interchangeably in particle physics. Where, if $e = mc^2$, then, $m = e/c^2$

$1 \text{ eV} / c^2 = (1.602176634 * 10^{-19} \text{ C}) . 1\text{V} / (2.99792458 * 10^8 \text{ m/s})^2$

$1 \text{ eV} / c^2 = 1.78266192 * 10^{-36} \text{ kg}$

The figures for the measurements are not rounded off since the full values are needed for them to undergo a special process of almost perfectly accurate calculations later on. The resulting answers will have minimal errors. At that point, the error will be so low that the sheer frequency of the experiments themselves will remove any ambiguity in the data. This data needs to be as accurate as it can be and needs to be preserved that way. The data cannot be rounded off or altered in any manner since that goes against the nature of operations at the LHC.

Concepts & Mathematics

There are two types of collisions at CERN regarding protons. One does collisions of particles with each other and the other one, specific to protons and neutrons requires them to be bombarded onto a piece of lead. With regards to calculating their trajectories after collisions, the collision with lead is easier as the protons collide with a static object. Here, we do not need to account for subatomic particles exceeding a scatter area of 180 degrees and the probability of each zone can be calculated with the help of immense amounts of data already present. The algorithm can be trained on this data which is harder to do with a more random scattering in collisions of particles themselves.

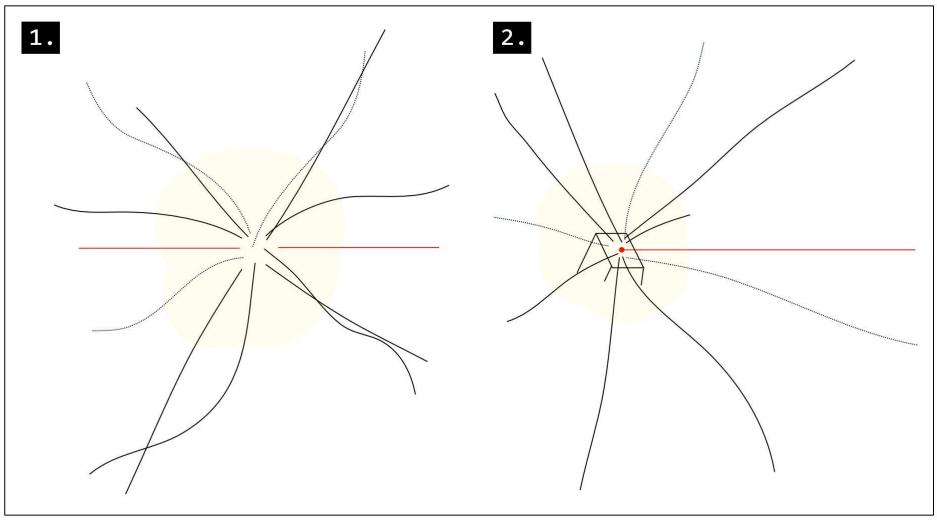

Figure 2: Types Of Collisions

The two approaches we can take here for trajectory calculations are from the point of collision to the detector surface and from the detector surface to the point of collision. The first method requires us to account for variables mentioned before and correct the calculations in terms of spread and curvature by training on actual experimental data. The latter method is better for faster data recording since fewer calculations are to be performed and the main concern is the preservation of storage space on servers.

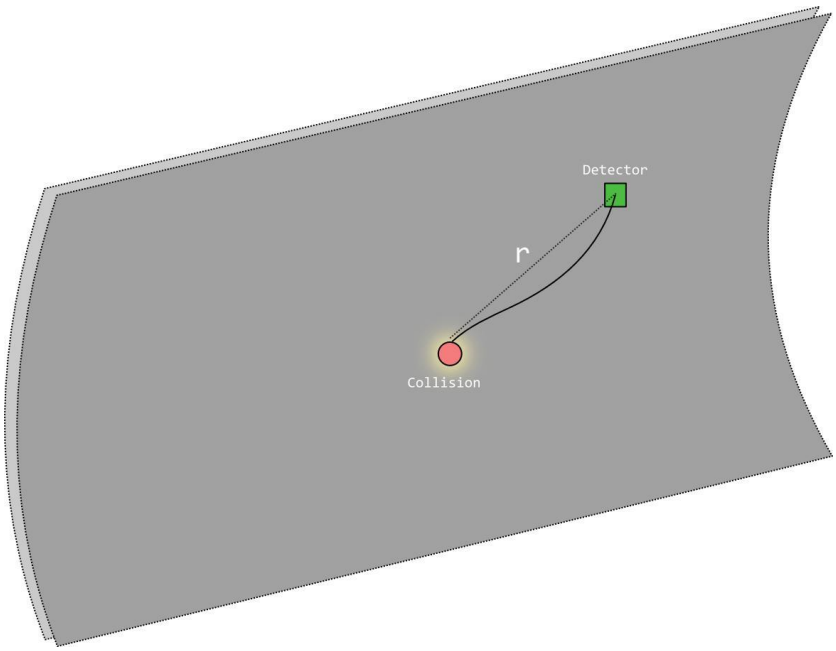

Figure 3: The Path to-from Collision & Detector

For collisions with lead, we have to calculate for components of a single proton. This lets us restrict calculations to half of the collisions between two particles since they will generate twice the subatomic particles. Collision with lead also lets us calculate for semi-spherical scatter rather than a full scatter because the flat lead surface doesn't allow the scatter to be greater than 90° degrees on either side meaning a total of 180° degrees.

The data we collect from the detection surface to the point of contact only require a special method of storage discussed in the later chapter of supporting mechanisms and optimization of structures. The heavier concern is the calculations for the trajectory calculations from point of contact to the detection surface

Algorithm & Supporting Mechanisms

The logarithm itself is based upon numerous techniques and principles, each with their own benefits. So this project will discuss them all.

Before moving onto techniques, it is a better idea to quantify the benefits of each technique. For this purpose, we can spread out the three-dimensional cylindrical detector surface to a two-dimensional rectangle as shown here. The cylinder can be split from the seam.

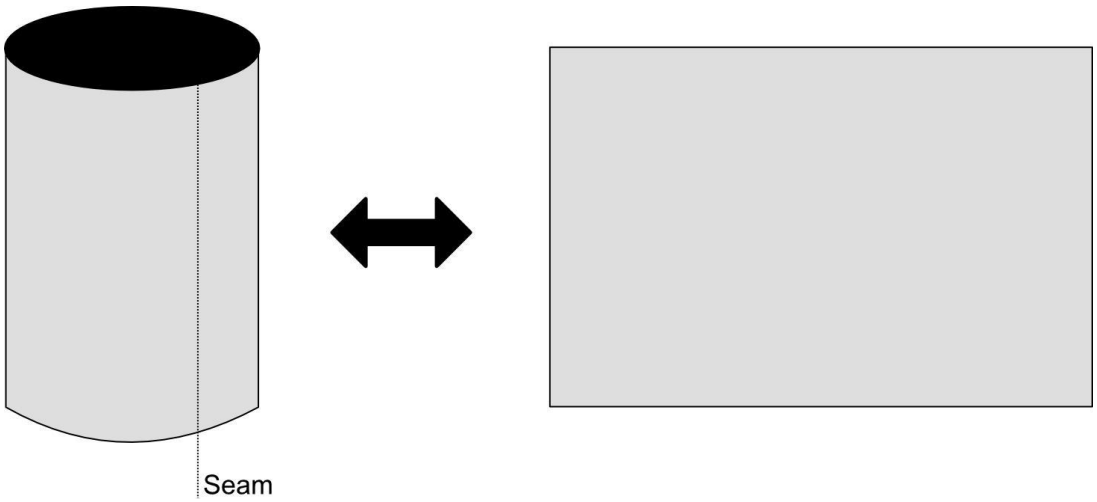

Figure 4: Unwrapping a Cylinder to 2D Plane

This will let us visualize scattering better and create indexed data on the x and y plane only. This technique is commonly used in computer graphics for texturing and is called unwrapping. This unwrapping creates a two-dimensional rectangular “island”. From here, we can project the detected locations for the subatomic particles onto the plane.

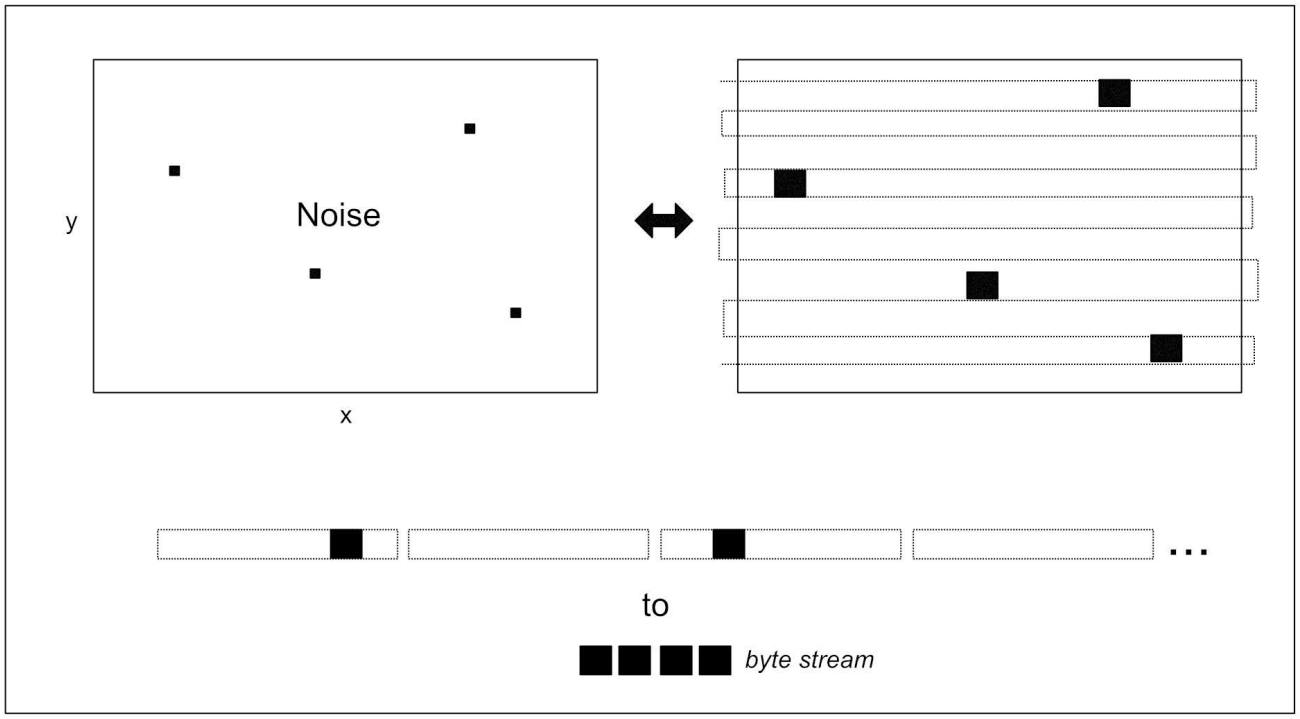

Figure 5: Converting Noise Map to Byte Stream

One more purpose of this method is to now create a stream of data. The plane is split into horizontal lines, each a pixel wide. The horizontal lines are laid out in front of each other, one after the other. This way the data has successfully been converted to one-dimensional data since we're reduced the y plane. The data is like a stream with points of interests that may occur in any segment. It should be noted that this stream of data is still recoverable back to the three-dimensional spatial data.

Now, since we know the location of each point of interest, we can retrieve the y coordinates at any time by,

(location on y coordinate) = (length to point of interest) / (length of each segment)

(location on x coordinate) = (length to point of interest) % (length of each segment)

Example:

Suppose,

L = 100 (length of the entire stream of data)

i = 75 (length to the point of interest) (this is also the index)

l = 10 (length of each segment)

Solution:

We can extrapolate more data from this byte stream. The two-dimensional lengths should be,

y-length = L / l = 100/10

y-length = 10

x-length = l (same as the length of each segment)

For location of the point of interest on the xy plane, we have to calculate the values like,

y = (length to point of interest) / (length of each segment)

y = 75 / 10

y = 7

x = (length to point of interest) % (length of each segment)

x = 75 % 10

x = 5

Final Coordinates = (x, y) = (5, 7)

The storage portion of the algorithm uses the exclusion technique to omit sections of the byte stream that CERN is not interested in. These sections are blank and waste space. Since we already have the value of "i" (unique for each point of interest) and it is possible to recreate the three-dimensional space from that data, we can create structures that index only those values.

For the index i, we need its value to be extremely precise. So, a special data structure is created to represent the value.

```
struct value{
unsigned int value;
bool value_sign;
short exponent;
}
```

The unsigned int value only takes into consideration the values from 0 to $2^{32} - 1 = 4294967295$. Making the value unsigned extends it's range while the sign value is handled by a boolean separately. The exponent is not unsigned and is short because it doesn't need to hold very large values. Its range is from -32768 to +32768. If the user does not want to use such a structure, the alternative option is double to extended-precision formats (40-80 bits) of floating-point numbers.

The next step is to feed the algorithm the indexed data or 2D noise maps. These noise maps are binary i.e. full black or white pixels and correct themselves with each sample and generation for n number of detection of subatomic particles. The 2D noise is Perlin noise since Perlin noise **[4]** can alternate itself with coupling data.

Figure 6 shows the interpolation of data as it overlays over itself in places where there is a higher probability of detection from a collision. BMP files collect this data. PNG or JPEG wasn't preferred because there is no need for alpha values of PNG and JPEG has compression issues and graphical artefacts. **[4]**

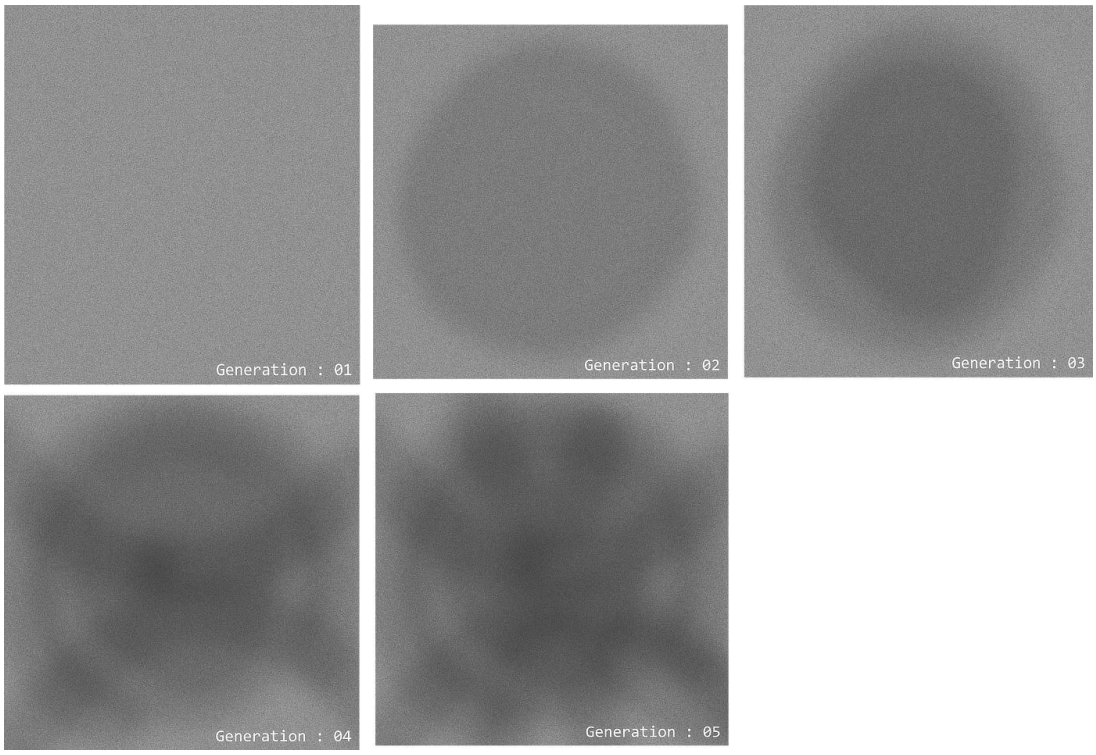

Figure 6: Noise Maps Generated By Trained Software

Graph generation algorithm is as follows,

n = 8 (number of subatomic particles)

for the demonstration, I've simulated millions because they are much easier to observe and see.

For the first sample,

black pixel (x, y) = black pixel (random(0, total_width), random (0, total_length))

then set n = n – 1

Repeat until n is not 0

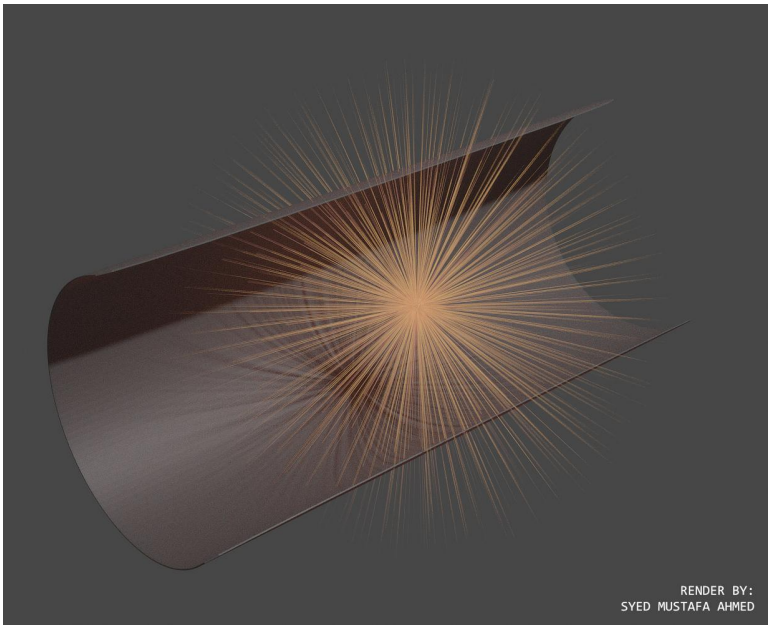

Figure 7: Running Simulations in Blender 3D

The other method is a simulation using software for initial data. Set m, v, s, p and q to actual values discussed in "metrics" section. Use simulation software such as MATLAB, Simul8, Blender etc for simulation.

Settings are,

Contact of rigid bodies at "0"

Collision triggers the emission. Emissions have their own mass, spin and momentum.

The network works by having an acceptable window that gets smaller and smaller each generation. The details increase and the current data is used again for comparison for the next calculation through interpolation.

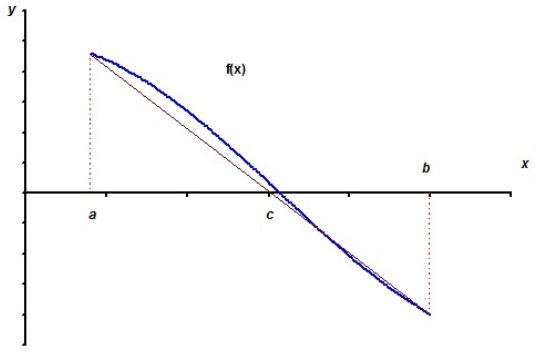

Figure 8: How Linear Interpolation Works

Here, linear interpolation works best since interpolation uses the data from the previous computation and also is quite optimized for straightening curves for easier calculations. For interpolation, we need to,

Let a and b be the two points (the detector and the point of collision)

$c = a - f(a) * (b-a) / (f(b) - f(a))$

For the next step,

if,

$f(c) * f(a) < 0$

then the value of b is equal to c

otherwise, the value of a is equal to c

This difference is overlayed on the noise map increasing the density of spotting where the probability is higher.

$P(collision) \rightarrow P(Collision) + P(c)$

Normally, this ends when we reach an acceptable level of error but here since we are training a network, we have to keep feeding the network each generation. With each generation, the probability of the path of trajectory becomes higher and narrower in the noise map.

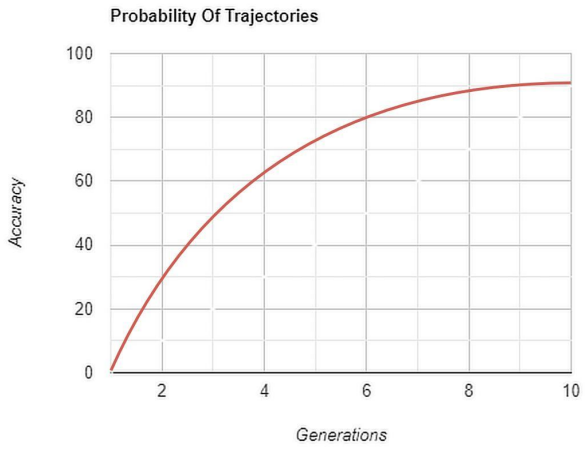

Figure 9: Increase & Tapering-off of Probability of Accuracy over Generations

The total Probability of any given points calculated to have a higher chance of collision detection increases but never to 1.

$$0 \quad <= \quad P(c) \quad < \quad 1$$

This is true for each equivalent pixel on the detector. If the detector has equivalent pixels then,

$$P(\text{per pixel}) \quad = \quad P(\text{total}) / 100$$

And,

P(total) which is always less than 1,

$$= \quad P(1) + P(2) + P(3) \ldots P(n) \quad < \quad 1$$

Optimization Of Model(s)

The main concern of this optimization is to increase the efficiency both in terms of speed and memory for the algorithm discussed in the previous sections. Firstly, the model needs to be understood in modules.

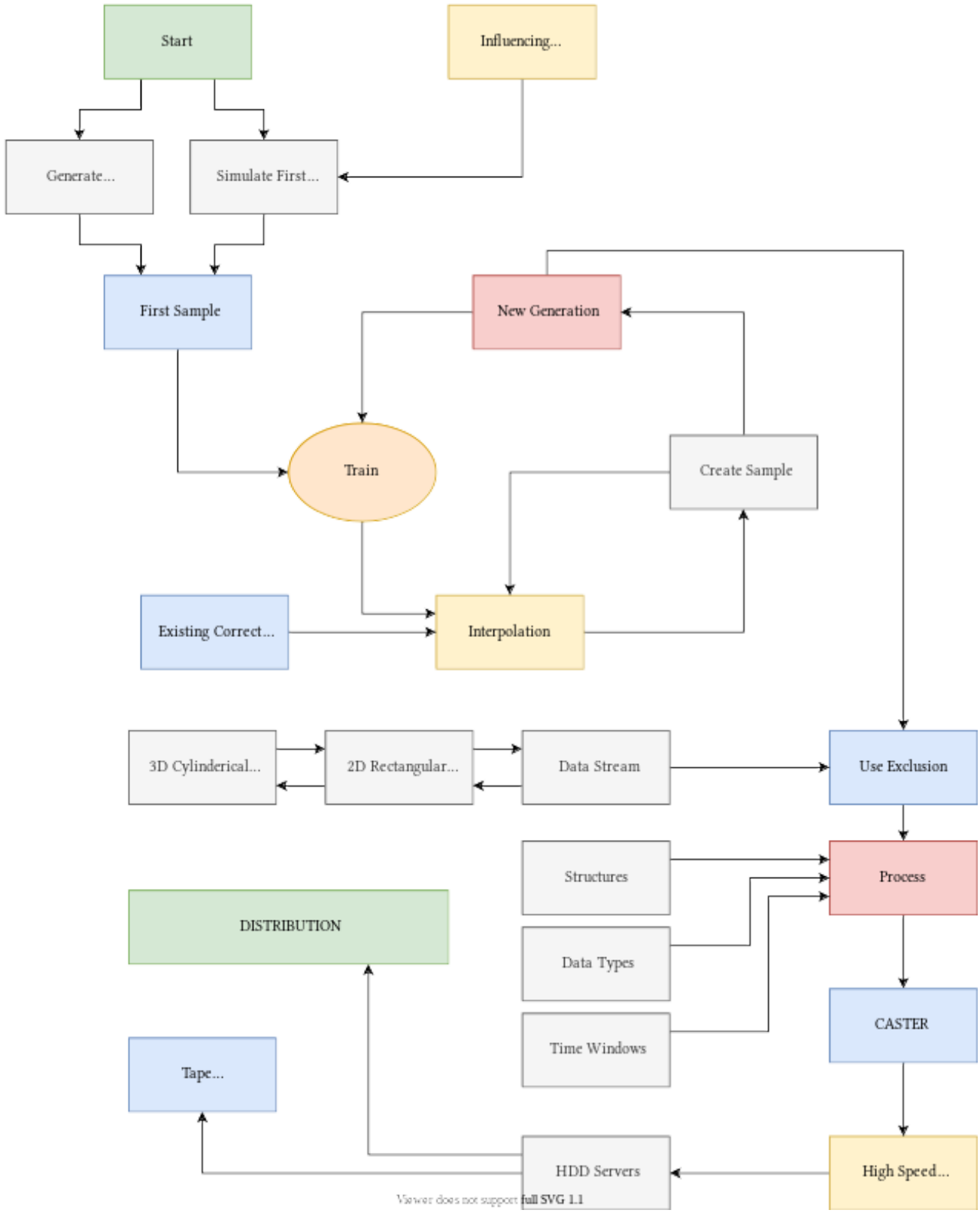

Figure 10: Neural Network System for this Project

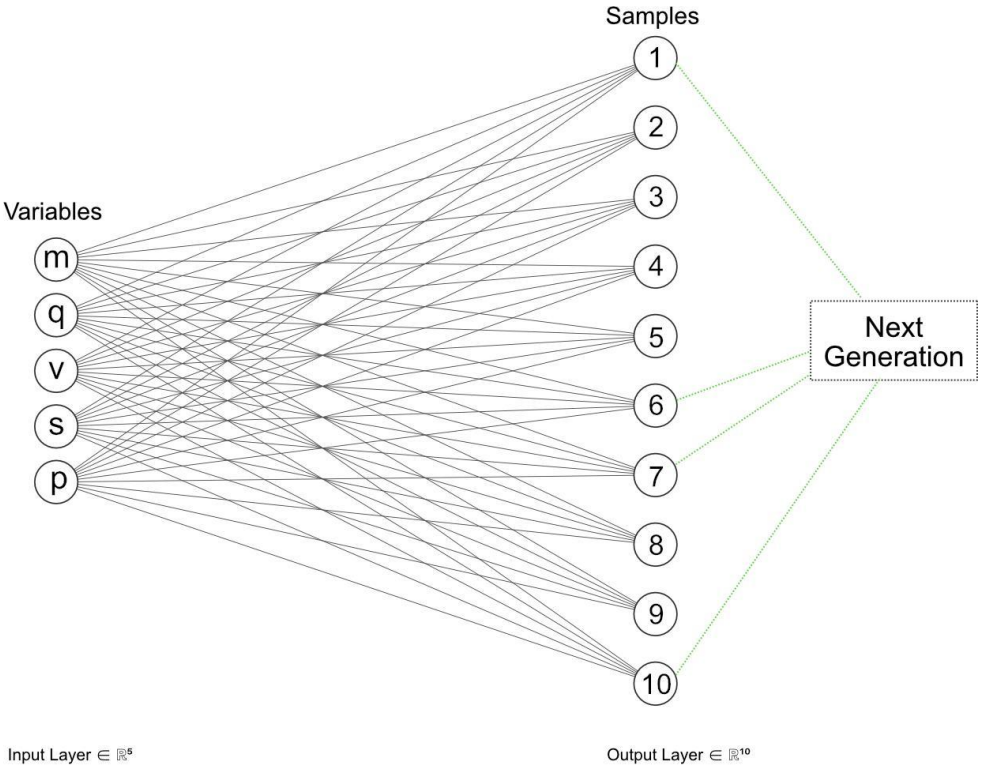

Figure 11: Neural Network System for this Project

The model in *Figure 10* shows a neural network. Variables discussed in the "Metrics" section of the project each contribute to producing samples each generation. The samples in a generation are subjected to either being selected for further reproduction or deletion on the basis of how well they come close to experimental data. One of the defects of this approach is that it takes time and processing power to train each generation.

Optimization Of Structures

The System CERN and the LHC uses is based on C++. This choice is because rather than interpreting languages, C++ allows for faster execution and compilation. The objects are linked and are treated as the same thing since the compiler doesn't see them as files at all. The C++ code is converted into intermediate code like .asm and then to binary code through a native compiler. These binaries run extremely fast rather because they aren't being run on a virtualized environment like in java or python.

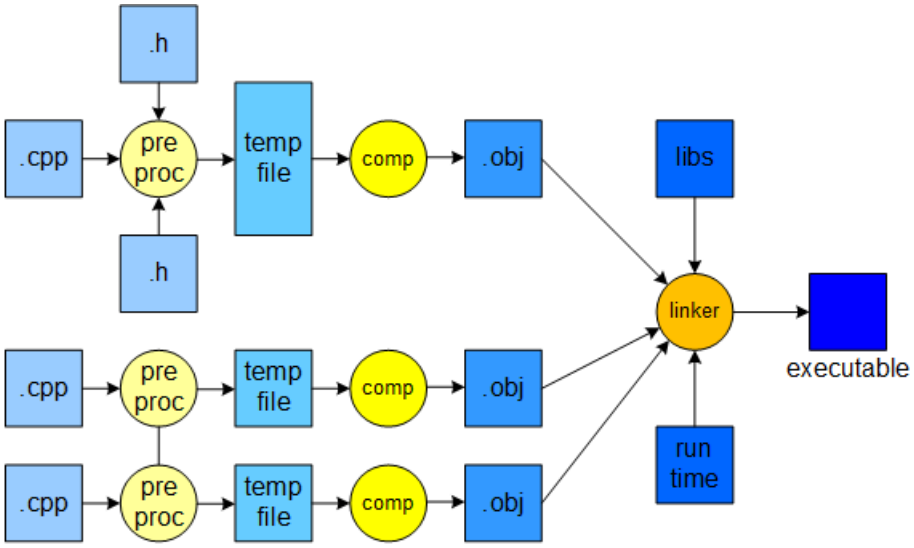

Figure 12: How C++ Compiler Works

The reason why classes or linked lists are not used in the construction of the custom data types for the LHC is that structs and arrays are more primitive and have less overhead in terms of memory. Our main concern in the project is to conserve storage and possibly reduce processing as well. So, preallocated static arrays give us index access and no processing on the runtime at all. The System only needs to place the raw data into these structures.

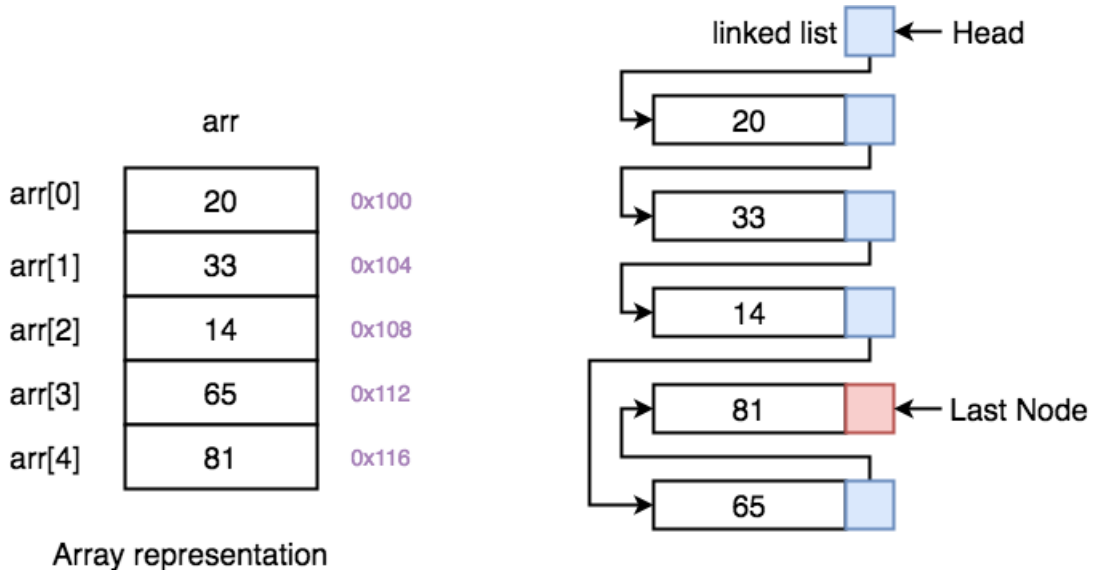

Figure 13: Linked List vs. Arrays

There is a need for the overhead of Constructors and Destructors for memory cleanups since we are aware of how many particles are expected. This means that we can accommodate space for them beforehand and even leave some space for unexpected subatomic particles as margins.

The final mechanism that could help the trajectory prediction algorithm is “timed windows” or “triggers”. The principle is very simple. Referring to *Figure 1.* , the LHC has 8 detectors that also have very accurate timers. So, they can be used to create exact properly sized windows in which actual notable events take place.

Let, T_n be the start time of the recording window,

t_n be the time at a specific detector

x be the time between detectors

So,

$T_n = (t_{n-1} + x) - (\text{tolerance window})$

The other method requiring triggers works by recording data of higher than set fluctuations as shown.

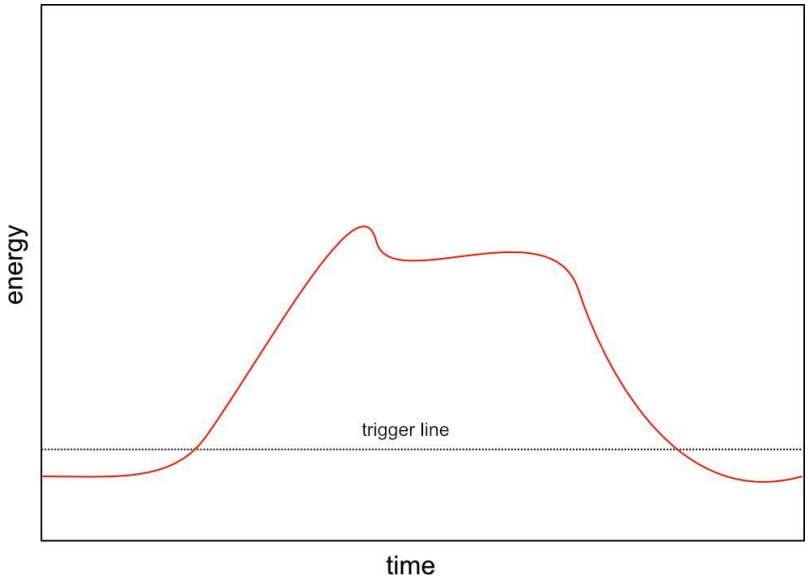

Figure 14: Timed Triggers

Results

The trajectory prediction algorithm isn't the sole component of the project to help preserve storage space. Introduction of newer ideas interfacing with each other add up to a bigger change. Quantifying this change is a challenge but approximating it can be done.

The file size of each reading at CERN is approximately 8Gb. Out of which 1 GB is for RAW readings and the rest is extrapolated data and miscellaneous.

1 GB = 1000 Mb = 1000000 KB = 1000000000 bytes

or, $8 * 10^9$ bytes.

Using our byte stream method, for each 64-bit precision floating point value for indexing, we need

memory per proton = n * 64-bits

memory per proton = 3 * 64

memory per proton = 192 bits

Additionally, there are values for the indexes (unsigned int) So,

total memory per proton = 192 + 4 = 196

≈ 196 bytes

196 bits / 8 = 25 bytes (per proton)

There are 30-40 collisions per cycle, So conservatively,

25 * 40 = 1000 bytes or 1KB

For the amount of data CERN does each reading, we can do,

1000000 KB / 1 KB = 1000000

or again, conservatively,

1000000 or 10^5 loop readings

Conclusions

The results of the project branch off to new techniques, algorithms and structures that can theoretically aid the LHC at CERN to save more storage and perform experiments more storage space and in return run more frequent tests. This will create more data for scientists and physicists around the world to view and learn from. The extent of what we know about particle physics is in its initial stages and there are still so much to learn.

Personally, I had to go through enough documentation and resources that I've lost count. I've had to shorten and summarize the full content of the project to stay on topic. I still wanted to discuss:

- Collisions of other particles
- Momentum Calculations **[6]**
- BMP Files and they are generated
- Neural Networks in Detail
- Completing a full application in time

References

[1] Institute of Physics CERN, Particle Detection, Espl. des Particules 1, 1211 Meyrin, Switzerland, April 2020

[2] Esra Ozcesmeci, Large Hadron Collider: pushing computing to the limits, Meryin, Switzerland, March 1 2019

[6] CERN, Momentum; Taking a closer look at LHC, Switzerland, 2019

[3] CERN, MoEDAL; Taking a closer look at LHC, Switzerland, 2019

[4] SOL-PROG, Perlin_Noise Github, 2012

[5] Paul Macklin, Easy BMP; C++ lib, SourceForge, February 20 2011

Other Supporting Sources:

CERN Document Server, https://home.cern/resources, http://castor.web.cern.ch/