Royal Holloway University of London

PH4100: Major Project

Meshing of Primitive Solids in Pyg4ometry & BDSIM

Ben Shellswell

Abstract

The testing and analysis of radiation travelling through geometries of devices, such as a medical magnets, spacecraft or new particle accelerators, is often a very expensive and time consuming process. The open-source software packages Pyg4ometry & BDSIM are designed to enable scientists and people within in the industry to virtually simulate these tests, with accurate physics concepts. This project looks at improving the 3D simulation of the events and devices, by the remeshing of the basic primitive solids.

Supervised by
Prof. S Boogert
February 25, 2020

Contents

1 Introduction

1.1 Project Aims

The aims of this project are to contribute towards the optimization of the Pyg4ometry package 2.1 (and subsequently BDSIM 2.3), by improving parts of the code and conducting performance test to produce results that can be analysed. The main areas for improvement and where most of the computational energy in wasted, is in the meshing of the primitive Geant4 2.2 compatible solids. The inefficient computation is due to the unessesscary use of boolean operations 3.3.2.

1.2 Report Structure

The subsequent sections are constructed in the following way, the software packages that are used and referenced through out this report are discussed in Section 2, the concepts and details of the primitive meshing used in Pyg4ometry (Section 3). The interactions of meshed solids and objects in BDSIM (Section 4) and a conclusion and summary of the results of the report (Section 5). Followed by an Appendix A.1, which lists a variety of content produced in the Project, but is not included in the report its self, due to repetition.

2 Software Packages

This section goes through each package of software related to and used throughout the duration of the project. It outlines the key details of each package, describing its function and link to the project. At the time of the project a lot of the prerequisite packages were only compatible with linux systems. Due to owning a machine that operated on windows, a lot of time was initially spent setting up virtual machines running Cent0s 7 (standard free linux used by CERN [7]). However despite getting it setup, the packages were not performing nearly as well as they were on the macs. Therefore for the main duration of project a loaner linux laptop (2011 Apple MacBookPro running OS El Capitan) was used from the particle physics department of Royal Holloway.

2.1 Pyg4ometry

Pyg4ometry is an open source python package also generated by JAI, its purpose is to convert 3D CAD (Computer Aided Design) models between different representations to allow compatibility with BDSIM for the testing of new concepts. The "4" in "Pyg4ometry" comes from the consistency the package has with Geant4 2.2. The package is a key tool for allowing multiple file formats to become compatible with BDSIM, which increases the number of people who can utilise the package.

Most of the development in this project is conducted in Pyg4ometry and managed using an online git in combination with Sourctree (A git management GUI). The package is currently written in and only supports Python 2.7, however as of January 2020 Python 2 is no longer being backed as the newer version Python 3 is taking over. The transition to Python 3 means adjusting the syntax of several functions and files in Pyg4ometry, this has began but a full transition will take sometime as it is not an immediate priority.

2.2 Geant4

Geant4 (or GEometry ANd Tracking) is a software developed in C++ for the simulation and tracking of particles traveling through matter. The package is used by many particle physicists and is one of the more popular packages for handling the geometry within interactions. Geant4 has its own preset solids that are used for simulating particle interactions. For ease of conversion between file formats Pyg4ometry uses the same conventions when meshing its primitive solids.

The materials used in Pyg4ometry and throughout this project are also from the Geant4 database

[5]. In this report the three main materials used are $G4\_Fe$, $G4\_Ti$ and $G4\_Galactic$ (Iron,Titanium and Vacuum). The use of $G4\_Galactic$ is arguably the most important as it is used to set the material of the world environment in with other objects are placed into. By default the world material in BDSIM is set to be air, which means the particles would interact with the air before passing into the object being tested, this was avoided by using the option $worldMaterial$ = "$G4\_Galactic$" withtin the GMAD files.

2.3 BDSIM

BDSIM (or Beam Delivery SIMulation) is an open-source software package written by the John Adams Institute (JAI) [6], for the use of modelling particle beam interactions. BDSIM has many applications, such as modelling complex particle accelerators, for example the Large Hadron Collider (LHC) or concepts magnets for medical scanners used to treat tumours. The package allows a user to specify the physics being used for a particular particle of a set energy colliding with a provided object. The scattering of the particle trajectories and decays are computed using Monte Carlo simulations, to make the results as consistent with experimental results as possible. The software outputs a full analysis of each run, and can even allow multiple runs to run at once (batch mode).

2.4 ROOT

Suprisingly ROOT is not an acronym and is a name made around the idea that it is a system for other system to grow off of, much like the roots of a tree that has many branches. ROOT is adopted by many physics communities such as CERN [7], where it was first written. Naturally making it a popular format for particle physicists in particular. ROOT files are the default output for analysis by BDSIM, as it has been heavily used by particle physicists since its release. ROOT has its own GUI for file browsing due to the nature of its formatting. In this report no ROOT files or plots are shown, as all the data has been extracted from the root files and replotted using Python and Matplotlib.

2.5 Freecad (Libraries)

In this report the libraries from Freecad 0.18 **??** are used directly within Pyg4ometry 2.1, without the use of the Freecad GUI. The libraries were used to import STEP files and to convert them into a tessellated meshed solid, that can then be written out as a GDML file (a format compatible with BDSIM).

2.6 Visulisation

Most of the packages mentioned above al have their own GUI's to visualise the meshed objects. However the only two used in this project are the ones connected with Pyg4ometry and BDSIM. Pyg4ometry uses VTK (as seen in Figure 1) to create its GUI's and BDSIM uses Geant4's GUI's under the hood (as seen in Figure 2).

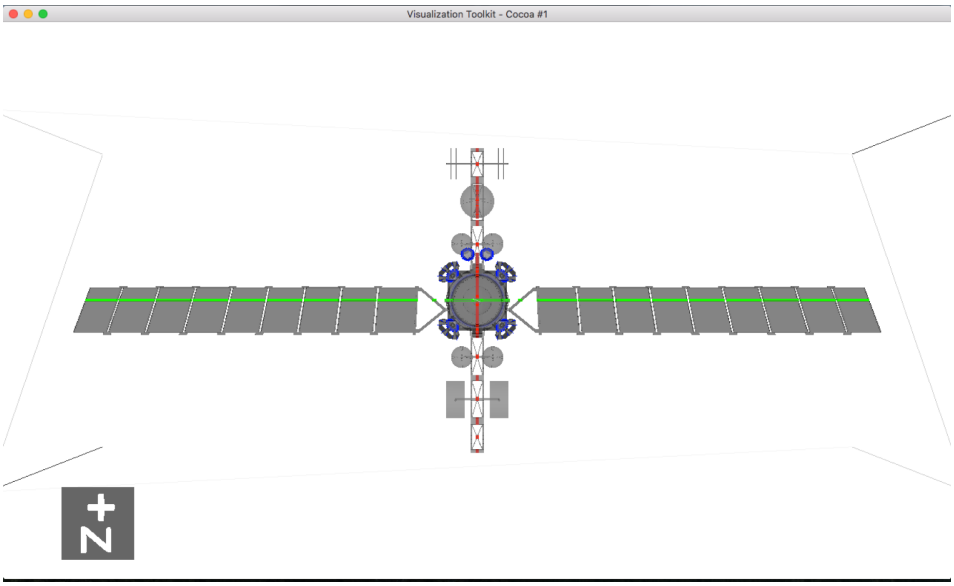

Figure 1: STEP file of a space satellite imported into Pyg4ometry and viewed in VTK

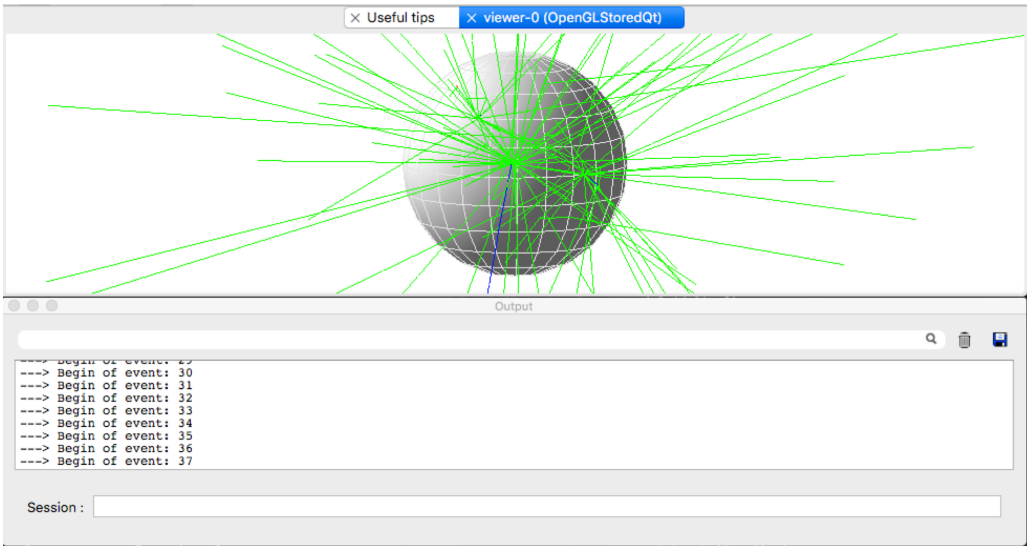

Figure 2: BDSIM GUI screenshot of particle interaction and output window, using a Geant4 sphere with 10,000 1.3 GeV neutrons.

3 Primitive Meshing

This section will describe the work done to optimize the Python scripts that generate the three dimensional meshing for the primitive solids within the Pyg4ometry package 2.1. All the primitive solids used are constructed such that they are compatible with Geant4's solids. It was orginally thought that it would be best to use triangular based meshes in combination with boolean operations 3.3.2 to construct the 3D solids. However it had been realised that the computation of triangles and boolean operations in most cases compared with polygons and adapted trigonometry is much more intensive and inefficient. In particular with the curved solids, i.e circular and elliptical based solids, due to the boolean operations generating more complicated meshes.

One of the major improvements to the Pyg4ometry 2.1 code is the computation of cut up primitive solids. The meshing of hollow or sliced solids were previously computed by boolean subtractions and unions, which involved creating two separate solids and acting upon both of them. Discussed more in Section 3.3.2. Which resulted in a very computationally heavy and less aesthetic outcome, where the mesh lines ("slice and stack"), were not meshed in radial directions.

3.1 Co-ordinate Systems

The various primitive solids are all constructed by using the predefined parameters used by Geant4 2.2, to be consistent with Geant4's own solids. The parameters of a 3D solid are properties relating to the coordinate system it is constructed in, such as height or radius. The parameters are then used to define the points of the object via basic trigonometry. The in which the points are appended is also very important and is discussed in Section 3.2.

The Python meshing scripts for all co-ordinate systems follow a similar structure, of first defining an empty list of faces (polygons). Then running the associated trigonometric equations through a number of loops to generate and append polygons to that list. The number of loops is associated with the number of sections a surface of a solid is being split up into in a given co-ordinate system. The density of the meshing is defined by a user inputted number of slices and stacks, demonstrated in Figure 3.

3.1.1 Cylindrical Co-ordinate System

The meshing for the primitive solids in cylindrical polar co-ordinate systems are constructed by looping through user defined number of slices and stacks (as shown in Figure 3) which the cylinder is being cut into (Listing 1). The Loop then creates the co-ordinates for 3 or 4 points at a time using an adaption to the trigonometry in Equations 1, which can then be defined as a triangular or polygonal face. The only cases where the new meshing produces triangles is at the top and bottom faces of the cylinder, provided it does not have a minimum radius equal to zero (creating a tube or cone). The same logic for the polygons also applied to triangles, just using 3 vertex points to make a face.

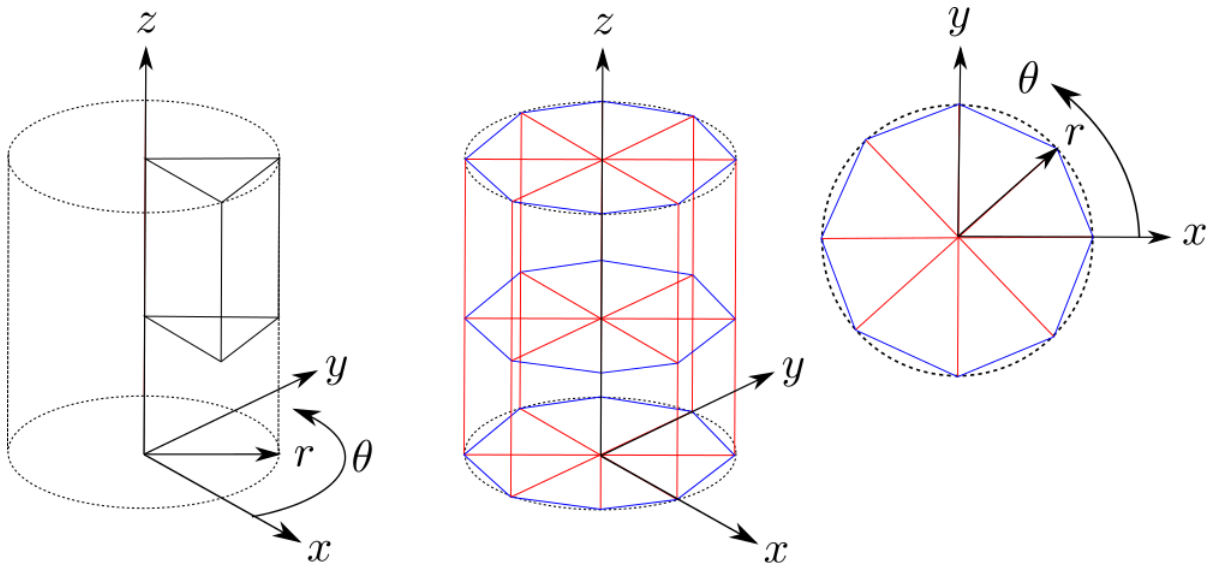

Figure 3: Diagram showing the meshing method for a cylindrical co-ordinate system
Red = Slices (8)
Blue = Stacks (2)

The trigonometry that converts the points from cylindrical polar co-ordinates to cartesian, are:

$$\begin{aligned} x &= r\cos\theta \\ y &= r\sin\theta \\ z &= z \end{aligned} \tag{1}$$

```
polygons = []

for j0 in range(nslice):
    j1 = j0
    j2 = j0 + 1

    vertices = []

    for i0 in range(nstack):
            i1 = i0
            i2 = i0 + 1
```

Listing 1: Basic method structure for Pyg4ometry primitive meshing of solids

The code in Listing 1 generates counters so that you can choses from two slices and two stacks, in order to gain the four points surrounding a desired face. These points are then used to define a polygon.

The only time a stack is needed in the cylindrical co-ordinate system is when the solid has a non linear function in the r-z plane. For example a paraboloid (Figure 37) would need a stack, but a linear cone (Figure 23) would not. This is due to the fact how that a plane can not represent a curved surface with a singular face.

3.1.2 Spherical Co-ordinate System

The meshing for the primitive solids in spherical co-ordinate systems are constructed by similar means the that of the cylindrical 3.1.1. Just with different trigonometric equations (Equations 2) as a result of two angle parameters ϕ and θ. The stack (blue) and slice (red) for solids in the spherical co-ordinate system works, like the longitude and latitude on a globe, as shown in Figure 4.

The trigonometry that converts the points from spherical co-ordinates to cartesian, are:

$$\begin{aligned} x &= r\cos\theta\sin\phi \\ y &= r\sin\theta\sin\phi \\ z &= z \end{aligned} \tag{2}$$

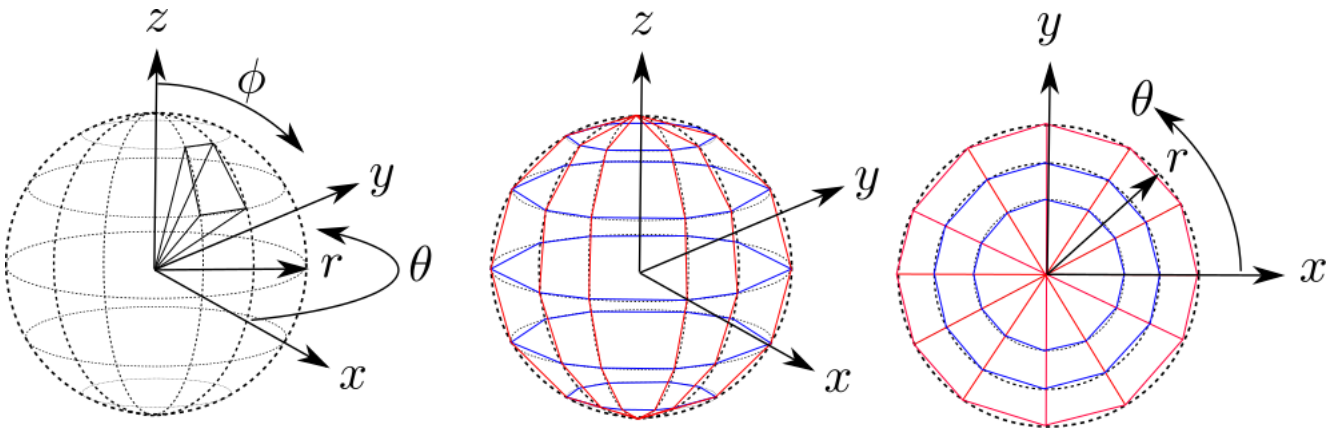

Figure 4: Diagram showing the meshing method for a spherical co-ordinate system
Red = Slices (12)
Blue = Stacks (6)

The structure of the code for a spherical system is the same as used in Listing 1.

The only time triangles are constructed in the spherical co-ordinate system is if the solid has a complete pole at the top or bottom of the solid. The solids constructed in the spherical always have both a stack and a slice.

3.1.3 Toroidal Co-ordinate System

The toroidal co-ordinate system is a special case and is only needed, for toroidal based solids. A toroidal shape is much harder to visualise a stack and slice, due to the fact it is a rotating co-ordinate system. A toroidal slice is an R_{Torus} radial cut taken out of the angle ϕ, as shown in Figure 5. The toroidal stack is a R radial cut out of the angle θ.

The trigonometry that converts the points from toroidal co-ordinates to cartesian, are:

$$\begin{aligned} x &= R_{Torus} + R\cos\theta\cos\phi \\ y &= R_{Torus} + R\cos\theta\sin\phi \\ z &= R\sin\theta \end{aligned} \tag{3}$$

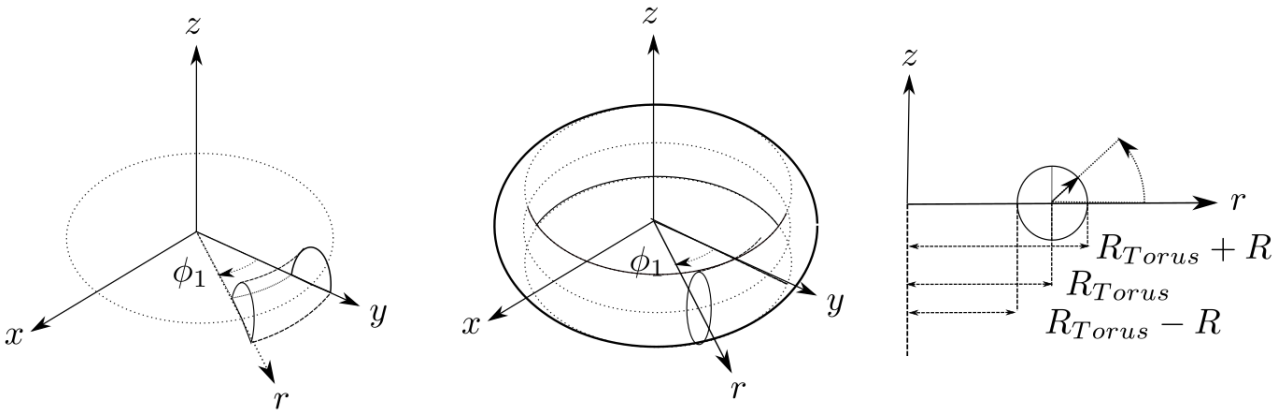

Figure 5: Diagram showing the meshing method for a toroidal co-ordinate system

3.2 Plane Direction

One key thing to be taken into account is the convention being used in the code for the order in which points are appended to make a plane, i.e to define a face on a solid. This is important as the direction the normal of the plane points in, dictates wether a face is considered an inside or outside face on the given solid. Getting this order incorrect, will lead to missing faces, when the meshing is made. The concept is demonstrated in Figure 6.

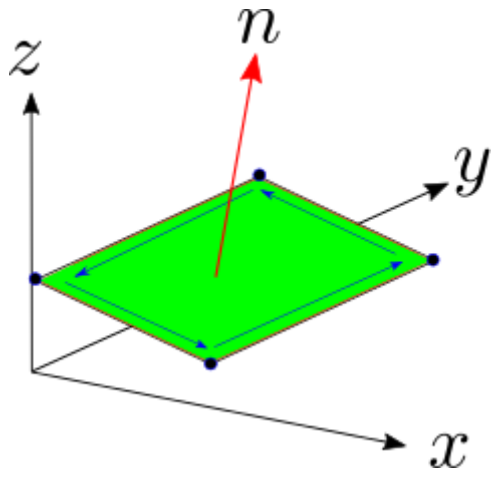

Figure 6: Diagram showing the order convention of appending points to define the normal to a plane

The simplest way to test this is by performing boolean operations with a box, as the boolean operation will only work nicely if all the planes are correct on both shapes. This test only works for the visualiser

in BDSIM as it uses Geant4, which displays face directions. VTK inside Pyg4ometry is much more lenient and will display solids even if their normals are incorrect, giving you a false idea of what is being displayed.

3.3 New Meshing of Curved Primitive Solids

In total there are 12 curved primitive solids, of which many of the examples and concepts are very similar. Therefore only a few solids will be discussed in this section, but their development can all be viewed in Appendix B. The naming convention of the solids being in this project and within this report are the ones used by Geant4.

One of the curved solids is the polycone, the old and new meshing development of the polycone is shown in Figure 7. It can be seen that in this example that the meshing at the top and bottom faces becomes more radially uniform. This is due to the replacement of boolean subtractions with simple trigonometry within the new meshing algorithm for the polycone.

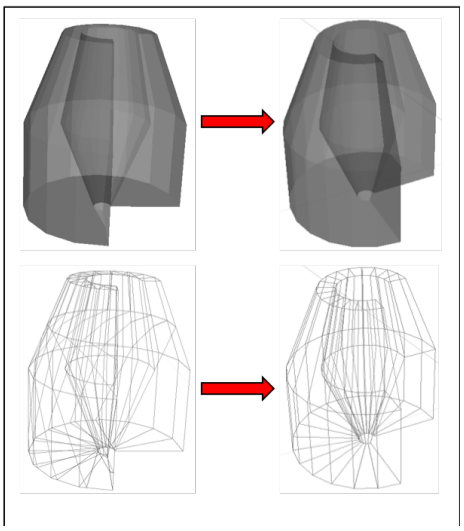

Figure 7: Meshing Development for polycone (Solid & Mesh View)

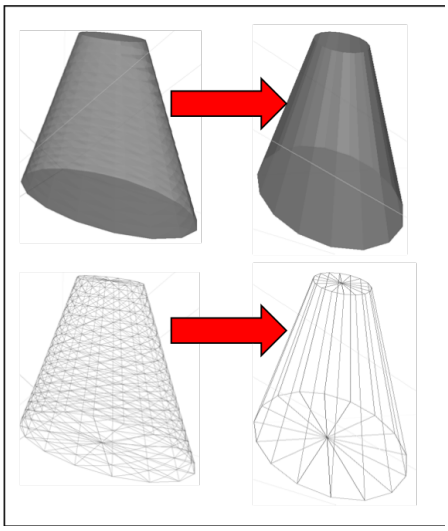

Figure 8: Meshing Development for ellipticalcone (Solid & Mesh View)

Another curved solid that was remeshed is the ellipticalcone, as seen in Figure 8. Despite having no boolean operation to generate this solid, the meshing is still improved by replacing all the unessessary triangular faces with quadrilateral ones. The stack was also removed as for the cone the faces in the r-z plane (in cylindrical co-ordinates) is a linear function.

3.3.1 Degenerate Points

Multiple meshing points occupying the same area can spring a few errors, sometimes without entirely crashing the code, making it a hard error to identify. It is typically given away when a *DivisionByZero* error occours, within the pycsg meshing of a solid. You can identify wether this is the error by debugging each polygon and triangle in a mesh, looking out for a face that has two or more vertices with the same (x, y, z) coordinates. This typically happens when you mean to mesh a triangle, but are still using the format for meshing a quadrilateral face. Or it can be due to the incorrect choice of an incorrect stack and slice iteration when constructing a face, as mentioned before in Section 3.1.

3.3.2 Boolean Operations

One of the largest improvements to the performance of the new meshing methods compared with the previous methods, is the discarding of boolean operations in order to create hollow or cut-up primitive solids. The idea can be clearly seen in Figure 9, where you conduct basic operations on simple solids,

resulting in a complex solid being made.

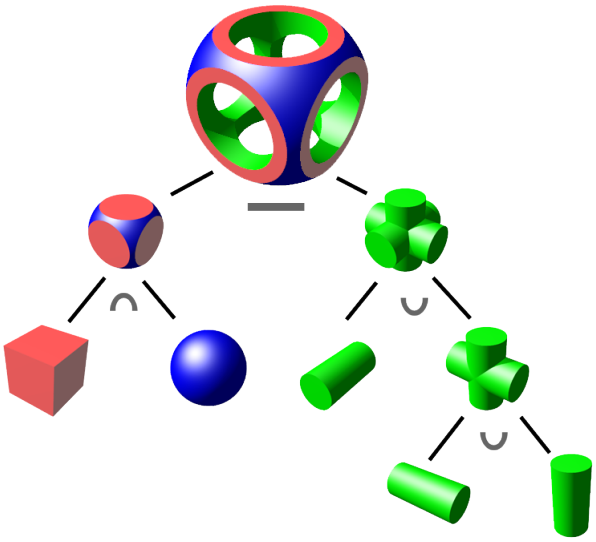

Figure 9: A diagram showing the basic method of constructing a more complicated 3D solid out of boolean operations with simpler primitive solids.
- = Boolean Subtraction
n = Boolean Intersection
u = Boolean Unions

The Figures 10 & 11 are of the meshed boolean union and subtraction of a box with a hollow sphere (in solid view). The coloured lines are representing the perpendicular planes in which the final object is placed in. These were made in the process of checking the normals before passing them into BDSIM to undergo interactions.

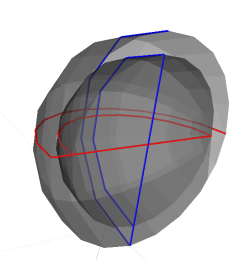

Figure 10: Example screenshot of a Boolean Union produced in Pyg4ometry & viewed in VTK.

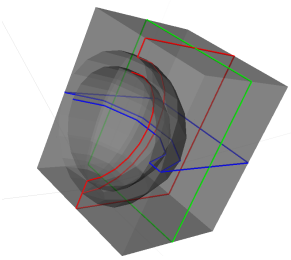

Figure 11: Example screenshot of a Boolean Subtraction produced in Pyg4ometry & viewed in VTK.

The old meshing algorithms would heavily rely on these operations. The old meshing algorithms used intersections to slice solids, as seen in Figure 25, and subtractions to make solids hollow of cut-up, as seen in Figure 35. For example Tubs is made from two cylinders being subtracted in order to create a tube as seen in Figure 22. The boolean operations worked, however are very computationally heavy compared with that of the adapted trigonometry applied in the new method. This is the case especially when one or more of the orginal solids is curved in structure. Another thing the boolean operations affected was the appearance of the mesh its self, the boolean operations worked by trying to identify common mesh points then remesh. This created a lot of non radially uniform mesh sections as seen in solids such as the polycone and Tubs , which do not appear in the new meshing algorithms.

3.4 Meshing Performance Testing

3.4.1 Polygon Count

One way in which the meshing performance of the Pyg4ometry primitive solids is conducted, is by counting the number of polygons produced by both the old and new meshings. I.e counting the number of triangular and quadrangular faces in order to make a comparison. A plot for each primitive solid counting its number of generated polygons was produced. They were generated by varying the user inputted number of slices across a range. Most were put through the range of (10-100) slice whilst keeping the number of stacks at a constant 10. However a few old solid meshings took so long to produce at higher mesh densities, they were only measured across shorter ranges e.g the old Hyperboloid meshing was left to run for over a couple hours and only collected data up to a slice of 50.

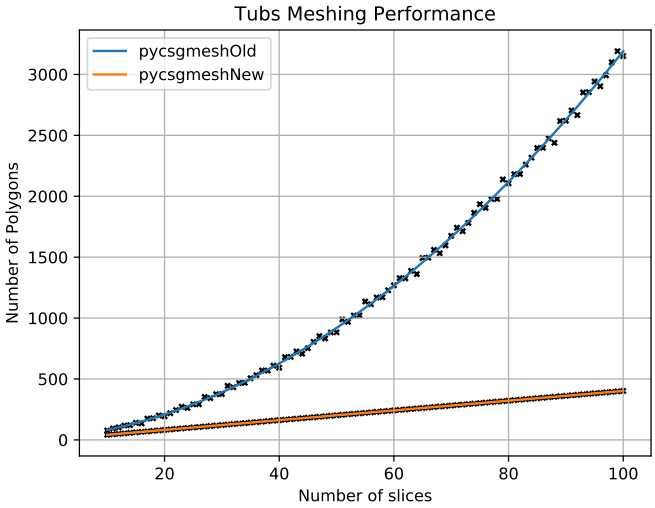

Figure 12: A plot showing the comparison of the number of polygons (and triangles) generated by the new and old meshing methods, across a range of slices 10-100.

Figure 12 is one of many plots, which are all displayed in the Appendix B. As you can see in Figure 12 the new meshing function is much more computationally reduced compared with the old meshing function. The number of polygons increases more uniformly and linearly for the new pycsgmesh function. The old function exhibits a more scatters and quadratic relationship. If you extrapolated both graphs it is clear than the new mesh is not only improving the structural apperance of the meshed solids, but also saving large amounts of computational power and time. The quadratic parameters describing all theses fits are also tabled in the Appendix 26.
The polygon count plots for the remaining primitive solids can be found in the Appendix B and a table of parameters for the quadratic fits can also be found in the Appendix 1.

4 BDSIM Interactions

The second aim of this project was to measure how the physics is altered, between the new and old meshings as the mesh density is varied. As mentioned before Geant4 already has its own set of primitive solids, which can be used as a guideline for comparison, due to no slice and stack dependence. The physics being measured will be features such as the trajectories and energy deposition, through solids.

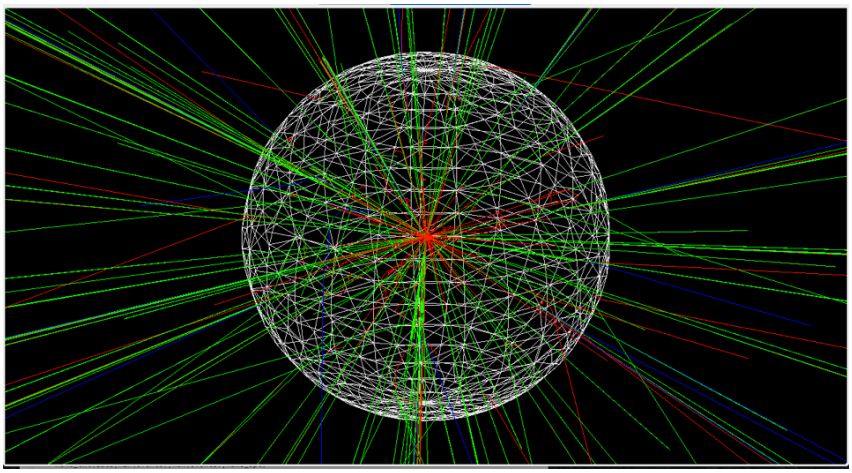

Figure 13: Titanium Sphere generated with new meshing interacting with 1.3 GeV electrons in BDSIM.

Within BDSIM the charge of a particle is represented by its colour, green = neutral, blue = positive and red = negative, as seen in Figure 13, where electrons and secondaries are being radiated from the centre of a meshed sphere. A secondary particle is anything that is produced as a result of the initial particles interacting with the solid, therefore a secondary itself can produce anther particle which is considered to also be a secondary particle. The probability of certain events and the scattering of a particles trajectories is generated by a Monte Carlo simulation. Each seed that produces a different outcome in the Monte Carlo simulations has a unique seed number. This allows a particular event to be repeated with the same physics, which is key to this project when it comes to comparing the interactions of two different objects under the same conditions. The other properties of the particle and physics processes that may occur can also be user defined to tweak the experiment. Properties such as the particles initial energy, wether secondaries get produced and much more. The seeds used in this project could not be logically chosen due to the fact they are generated via the Monte Carlo simulation. Therefore they were selected by running a lot of test runs in BDSIM to see which seeds generate a decent amount of secondaries.

BDSIM works by running a GMAD file and then outputs a ROOT file with the option of a GUI. The GUI allows the user to visualise the interactions in 3D. The GMAD contains the basic information needed to produce an output, i.e a particle and a target file (Typically of the format GDML). The option "physicsList" defines the physics which is used for the interaction, for the project the option was set to"g4FTFP_BERT". The ROOT file contains a detailed analysis of the interaction. One of the elements analysed within this project is the energy loss across the Z-axis.

Due to Geant4 already having its own primitive solids, you cannot directly pass a meshed primitive solid into BDSIM without it being replaced with a Geant4 one. The way around this is to pass the mesh in as a Geant4 tessellated solid, which intakes the properties of a meshed solid without overriding them with a Geant4 alternative. This proved to be more difficult than first intended due to many of the solids having a mixture of quadrangular and triangular faces to be unpacked. It orginaly sprung a few problems such as missing faces or faces of the wrong shape. Relitorating the importance of the normlas needing to be correct.

One of the first shapes to be tested in BDSIM was the sphere, which is made of a minimum and

maximum radius, making it a hollow solid. As seen in Figure 19 the sphere was chosen to be orientented such that the particle beam is fire from the centre of the sphere outwards. The conversion from a Pyg4ometry mesh into a Geant4 tessellated solid, results in only triangular faces. This is why all the quadrilateral faces are cut into two in Figure 19.

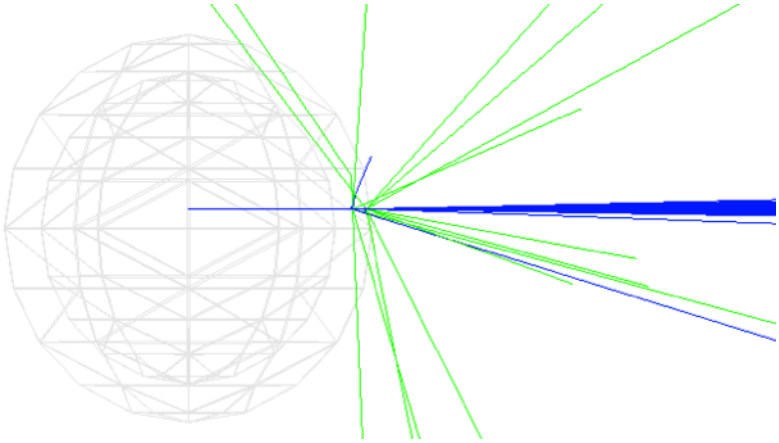

Figure 14: BDSIM screenshot of 200 1.3 Gev protons interacting with a Iron sphere, generated with the new meshing using a stack & slice of 10 & 10. (Shown in mesh view)

One way that was though to analyse the energy deposition of the interactions, was to look at the CPU duration times as the mesh densities are increased. In this scenario the Geant4 solid was used as a guideline as it does not depend on a user defined stack & slice.

4.1 Iron Sphere Interactions

rmin = 8mm
rmax = 10mm
1.3Gev protons
10,000

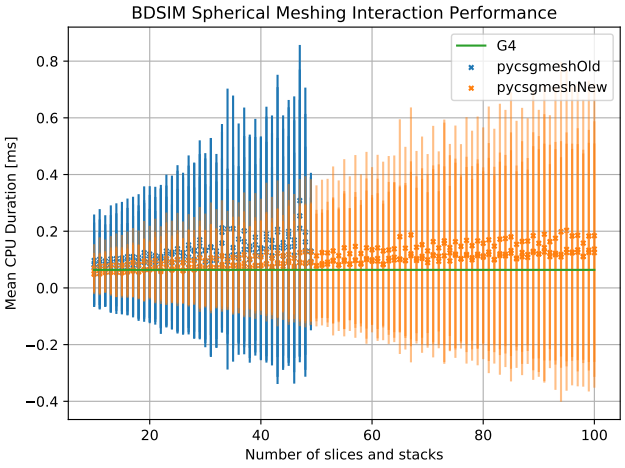

Figure 15: A plot showing mean CPU duration of 10,000 protons interacting with a Iron sphere of various meshed forms.

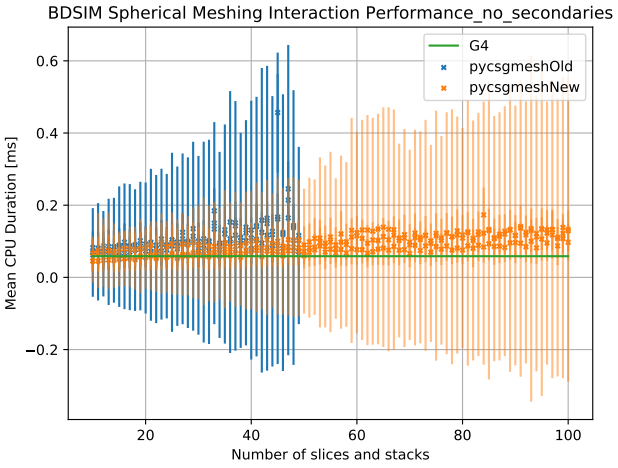

Figure 16: A plot showing mean CPU duration of 10,000 protons interacting with a Iron sphere of various meshed forms, with secondary particles disabled.

From FIgures 26 and 26, you can see that ...
It was orginaly thought that the standard deviations of the CPU durations was related to the number

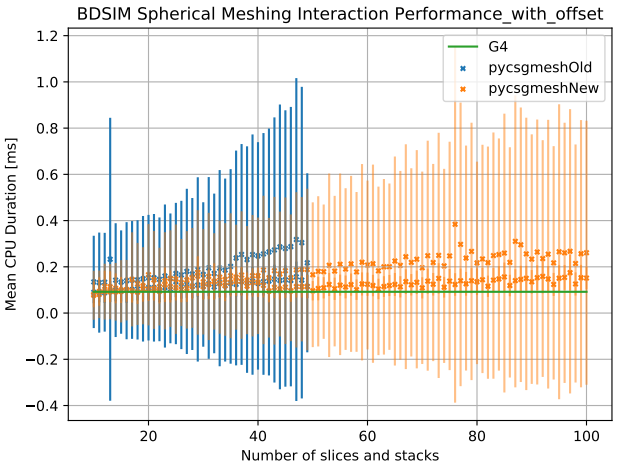

Figure 17: A plot showing mean CPU duration of 10,000 protons interacting with a Iron sphere of various meshed forms, with an 20 mm of set in the Z-axis.

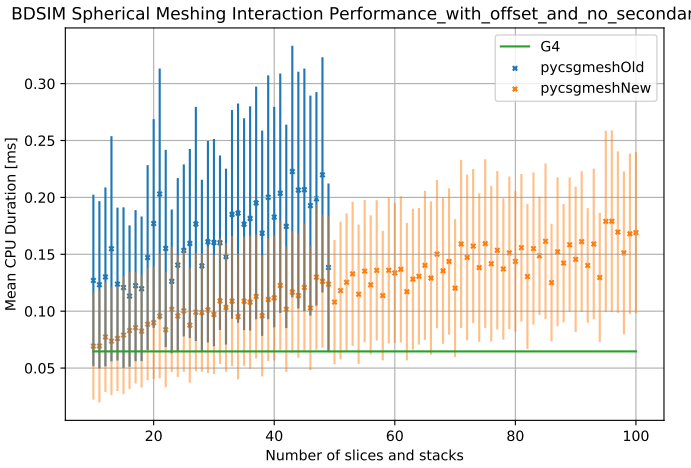

Figure 18: A plot showing mean CPU duration of 10,000 protons interacting with a Iron sphere of various meshed forms, with secondary particles disabled, with an 20 mm of set in the Z-axis and secondaries disabled.

of secondaries being produced. However this was disproved by Figure 26, where the same test was conducted with the BDSIM option that disables secondary particles from being produced implemented.

4.1.1 Error Reduction

To reduce the standard deviation of a data set the number of events needs to be increased, this proved by the relation ship shown in Equations 4. The relationship between the standard deviation and number of events is inversely square root proportional. Therefore the standard deviation should decrease with a increasing N.

$$\begin{aligned} \bar{x} &= \frac{1}{N}\sum x_i \\ \sigma &= \sqrt{\bar{x}^2} = \sqrt{\frac{1}{N}\sum (x_i - \bar{x})^2} \\ \sigma &\propto \frac{1}{\sqrt{N}} \end{aligned} \tag{4}$$

However Figure 26 shows...

The error bars are still very big so it was proposed to use the standard error of the main as an alternative. As well as as investigation for varying energy and the material of the target.

4.2 Choice of Materials

4.3 Choice of Materials

4.4 Titanium Sphere Interactions & Spherical Beam Distribution

rmin = 0.01mm
rmax = 10mm
beam dist = sphere
10 GeV electrons

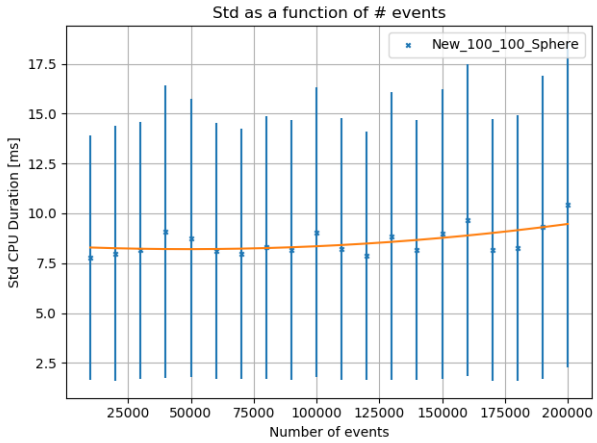

Figure 19: BDSIM screenshot of 200 1.3 Gev protons interacting with a Iron sphere, generated with the new meshing using a stack & slice of 10 & 10. (Shown in mesh view)

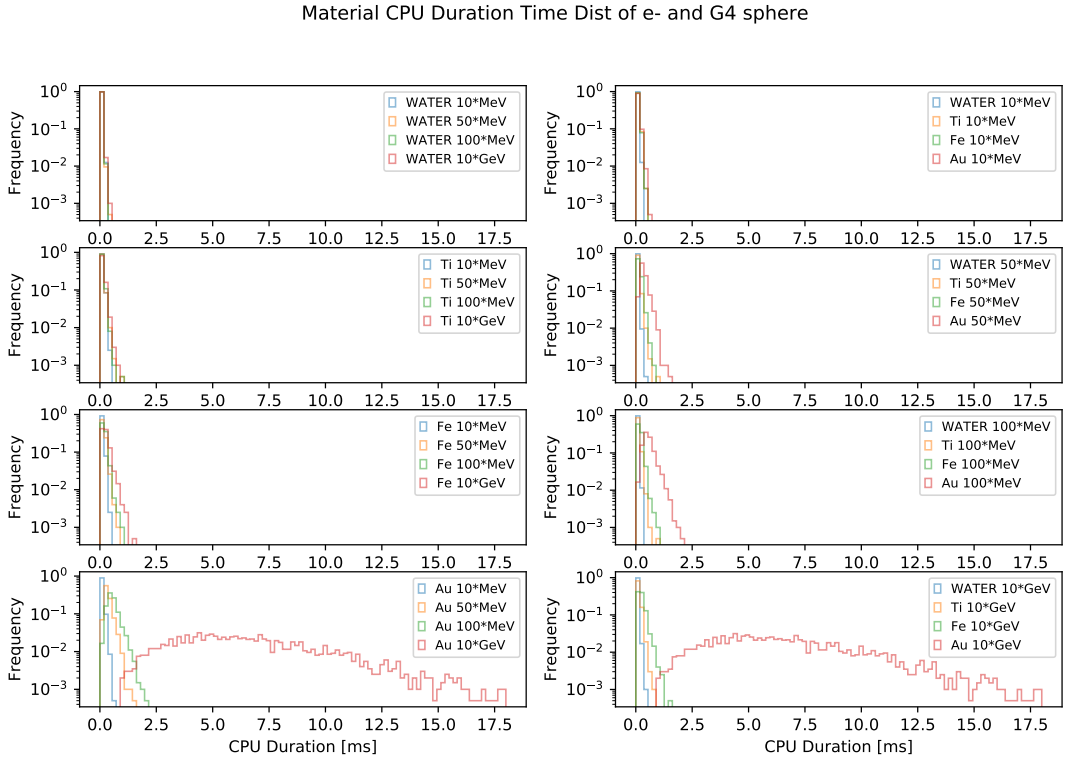

Figure 20: Meshing Development for Tubs (Solid & Mesh View)

N decided by reduction

4.5 Interaction with CAD Magnet

As a final goal to round off the project a compound boolean meshed solid of a real world magnet is analysed in the same way in which the meshed sphere was in the previous section. The CAD magnet is constructed out of basic primitive solids such that it can be compared between the old and new meshing methods.

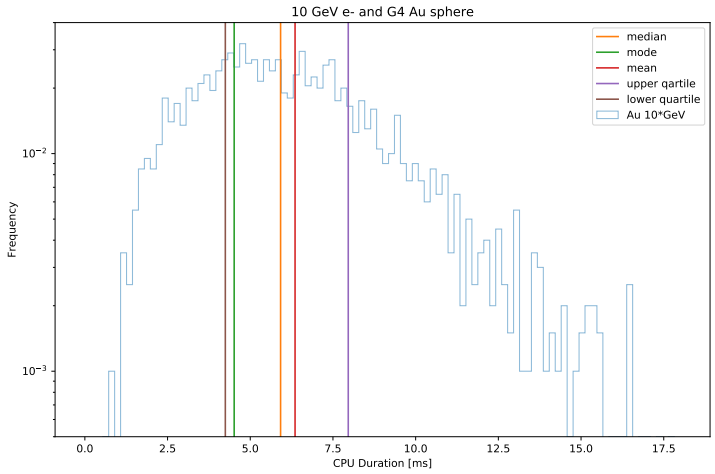

Figure 21: Meshing Development for Tubs (Solid & Mesh View)

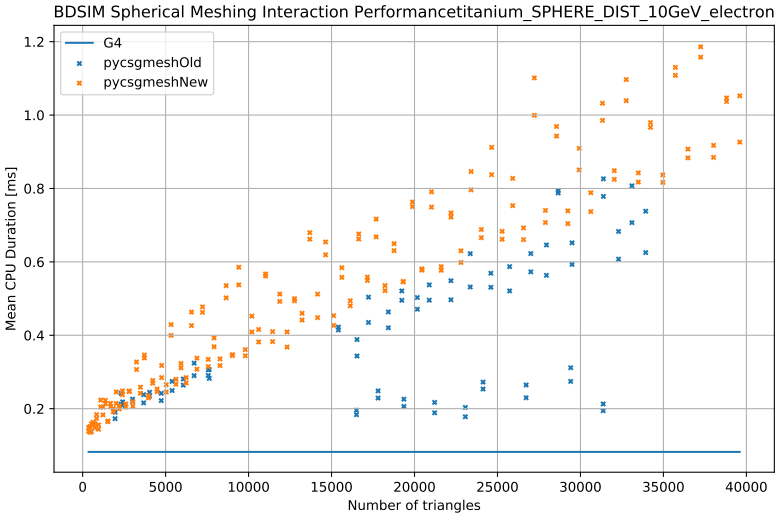

Figure 22: Meshing Development for Tubs (Solid & Mesh View)

5 Conclusion & Summary

5.1 Improvements

meshing is quicker
BDSIM interactions are quicker but still slower than G4 solid (as expected) improved coverage unit test speeds
meshing is neater and more uniform in structure
higher meshing density = closer to true solid as expected with bdsim interactions
quote graphs and sections

5.2 Applications

BDSIM and Pyg4ometry are both very powerful software packages that can be used to to aid not only the scientific research community of particle physicist, but also help everyday people by improving medical treatment. Thanks to the software being open source and its wide range of file compatibilities it can be used to simulate a growing number of projects. This report demonstrates this by CAD magnet

modelling.

References

[1] BDSIM Manual
http://www.pp.rhul.ac.uk/bdsim/manual/

[2] BDSIM Paper
https://doi.org/10.1016/j.cpc.2020.107200

[3] Pyg4ometry BitBucket
https://bitbucket.org/jairhul/Pyg4ometry/src/

[4] Geant4 Solids
http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/ForApplicationDeveloper/html/Detector/Geometry/geomSolids.html

[5] Geant4 Materials
http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/ForApplicationDeveloper/html/Appendix/materialNames.html

[6] JAI Homepage
https://www.adams-institute.ac.uk/

[7] CERN Homepage
https://home.cern/

[8] M. Pinto, P. Gonçalves
GUIMesh: A tool to import STEP geometries into Geant4 via GDML
https://doi.org/10.1016/j.cpc.2019.01.024

[9] M. Tanabashi et al
43. Monte Carlo Particle Numbering Scheme
http://pdg.lbl.gov/2019/reviews/rpp2019-rev-monte-carlo-numbering.pdf

A Appendix (Python scripts)

A.1 Sphere BDSIM Vary Mesh Test

```
from string import Template
import Target_Sphere as t
import numpy as np
import pybdsim

################################################################################

def run_gdml_spheres_same_slice_and_stack(min,max):

    """
        generate a .txt wth four lists of data, number of slices & runtimes,
        from a minimum and maxiumum number of slices and stacks
        """

    Meshing_ver = "New"

    for val in range(min,max+1):

        # create gdml
        t.Test(False,False,n_slice = val,n_stack = val)

        #make gmad
        f = open('Template.gmad','r')
        contents = f.read()
        f.close()
        template = Template(contents)
        d = {'value': str("gdml:../../GDMLs/"+Meshing_ver+"/Target_Sphere_"+str(val)+"_"+str(val)+".gdml")}
        rendered = template.substitute(d)
        gmadfilename = "GMADs/"+Meshing_ver+"/slice_"+str(val)+"_stack_"+str(val)+".gmad"
        f = open(gmadfilename, 'w')
        f.write(rendered)
        f.close()

        # use gmad and gdml to get root output
        pybdsim.Run.RunBdsim(gmadfilename,"root_outputs/"+Meshing_ver+"/"+str(val)+"_"+str(val))

        #load in root file to do analysis ...

        print "{}%".format(val-min+1//(max-min))

################################################################################

run_gdml_spheres_same_slice_and_stack(10,100)
```

Scripts//Run_New_Meshes.py

B All Meshed Solids and Polygon Count Plots

The following Figures are the meshing and polygon data for each primitive solid constructed in Pyg4ometry 2.1. The first Figure for each solid is a screenshot of the old meshing and new meshing visualized in VTK. They show the before and after of each primitive solid in both "solid view" and "mesh view". The second Figures Show the number of polygons and triangles produced by the solid as you increase the slice across a range of 10-100 (if there is a stack it is kept at a constant 10). The polygon data plots are generate using python 2.7. The naming convention is the one used by Geant4 2.2.

B.0.1 Cons

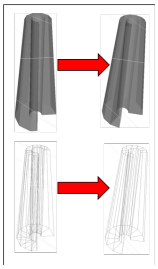

Figure 23

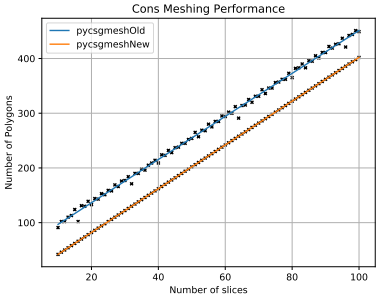

Figure 24

B.0.2 CutTubs

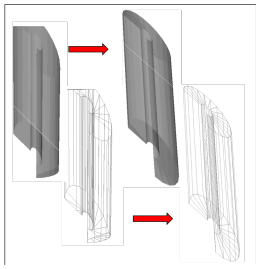

Figure 25

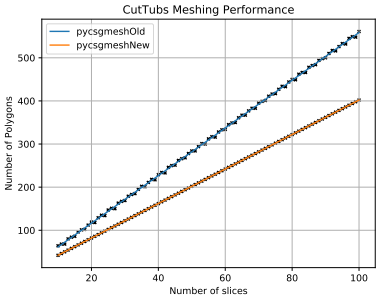

Figure 26

B.0.3 Ellipsoid

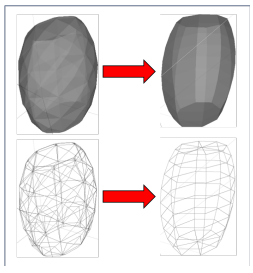

Figure 27

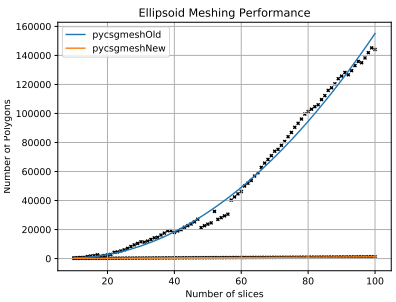

Figure 28

B.0.4 EllipticalCone

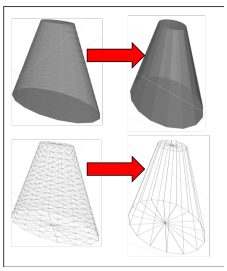

Figure 29

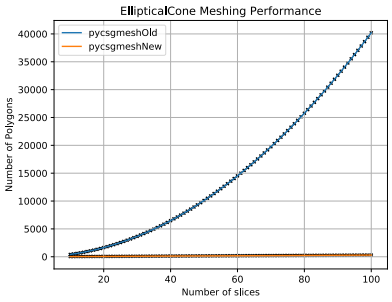

Figure 30

B.0.5 EllipticalTube

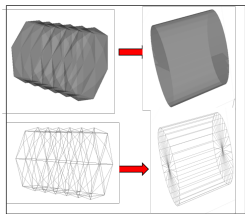

Figure 31

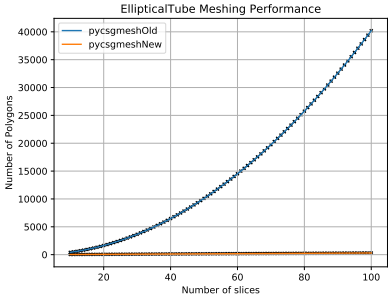

Figure 32

B.0.6 Hyperboloid

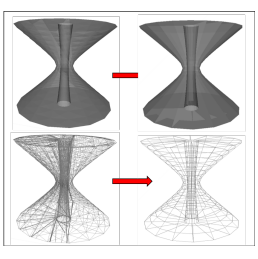

Figure 33

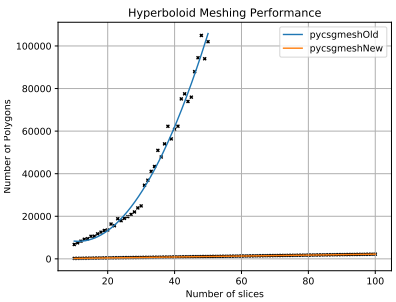

Figure 34

B.0.7 Orb

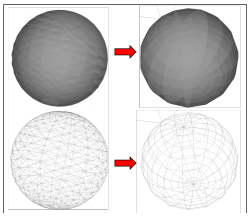

Figure 35

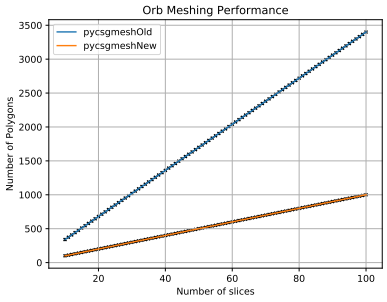

Figure 36

B.0.8 Paraboloid

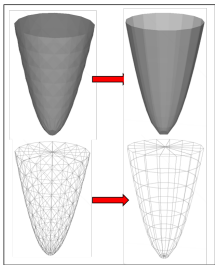

Figure 37

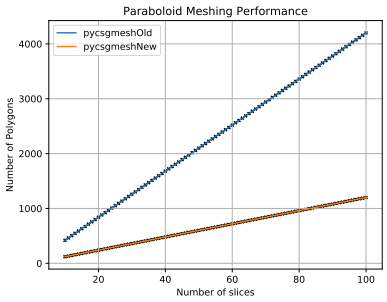

Figure 38

B.0.9 Polycone

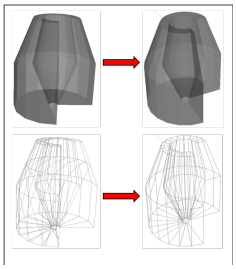

Figure 39

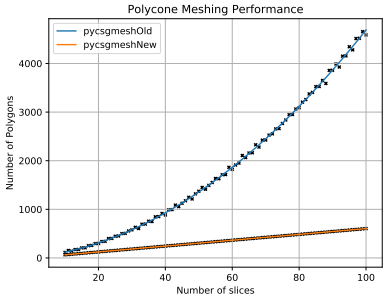

Figure 40

B.0.10 Sphere

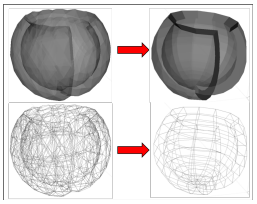

Figure 41

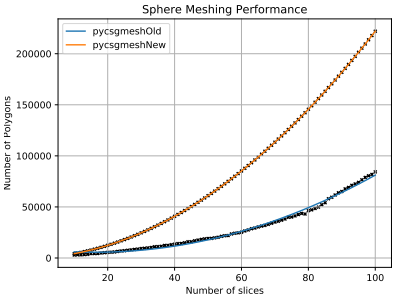

Figure 42

B.0.11 Torus

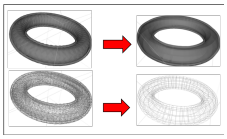

Figure 43

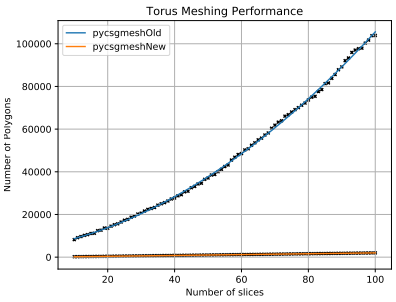

Figure 44

B.0.12 Tubs

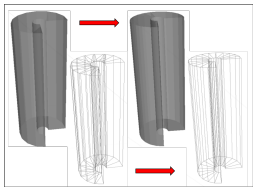

Figure 45

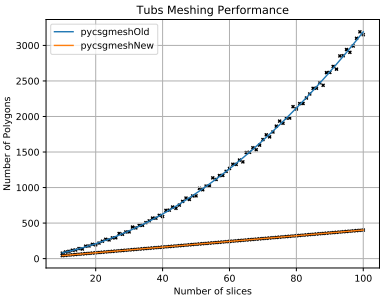

Figure 46

C Quadratic Parameters for polygon count plots

The quadratic fit where its parameters are of the form:

$$ax^2 + bx + c = 0 \tag{5}$$

Table 1: A Table showing the parameters to the quadratic fits for the polygon count plot.

Curved Primitive Solid	Old a	Old b	Old c	New a	New b	New c
Cons	-2.74E-04	3.97E+00	5.72E+01	-3.83E-17	4.00E+00	2.00E+00
CutTubs	2.19E-04	5.50E+00	6.30E+00	-3.83E-17	4.00E+00	2.00E+00
Ellipsoid	18.65090372	-333.5688153	1919.792901	1.61E-17	1.20E+01	0.00E+00
EllipticalCone	4.00E+00	2.00E+00	2.41E-12	4.03E-18	3.00E+00	0.00E+00
EllipticalTube	-1.94E-16	4.20E+01	-5.72E-13	4.03E-18	3.00E+00	0.00E+00
Hyperboloid	64.83177713	-1452.629624	16378.18198	4.84E-17	2.20E+01	-9.53E-14
Orb	-6.45E-17	3.40E+01	-1.91E-13	-8.06E-18	1.00E+01	-4.77E-14
Paraboloid	-1.94E-16	3.40E+01	-1.91E-13	1.61E-17	1.20E+01	0.00E+00
Polycone	0.39682347	7.13011557	7.42466252	-4.03E-17	6.00E+00	4.00E+00
Sphere	10.82154926	-356.6329124	8557.249195	2.00E+01	2.20E+02	1.98E-11
Torus	7.01776508	304.8746037	4899.196225	-1.61E-17	2.00E+01	-9.53E-14
Tubs	0.27241769	4.58191175	8.33675211	-3.83E-17	4.00E+00	2.00E+00