ROYAL HOLLOWAY UNIVERSITY OF LONDON

PH4100: MAJOR PROJECT

Meshing of Primitive Solids in pyg4ometry & BDSIM

Ben Shellswell

Abstract

Supervised by
Prof. S BOOGERT
January 17, 2020

Contents

1 Introduction

1.1 BDSIM

BDSIM (or Beam Delivery SIMulation) is a software package written by the John Adams Institute for accelerator science (JAI), 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) and concepts magnets for MRI medical scanners.

1.2 pyg4ometry

pyg4ometry is a python packaged also generated by JAI, its purpose it to convert 3D CAD models between different representations to allow compatibility with BDSIM for the testing of new concepts. The '4' in 'pyg4ometry' comes from the consistencey the package has with Geant4 1.3.

1.3 Geant4

Geant4 (or GEometry ANd Tracking) is a software developed for the simulation and tracking of particles traveling through matter.

1.4 Project Aims

The aims of this project are to optimize the pyg4ometry package to improve and performance test the results. The main areas for improvement and where most of the computational energy in wasted is in the meshing of the primitive Geant4 solids.

2 Primitive Meshing

This section will describe the work done to optimize the python scripts that generate the three dimensional meshing for the primitive solids. All the solids used are constructed such that they are compatible with Geant4's solids. It was orginally thought that it would be best to use triangles meshes to construct the 3D solids, however it has been realised that the computation of triangles compared with polygons is much more intensive and ineffecient, in most cases. In particular with the curved solids, i.e circular and elliptical based solids.

All the python meshing scripts 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 coordinate system. The density of the meshing is defined by a user inputted number of slices and stacks, demonstrated in Figure **??**

One of the major reasons that the code has been improved is the computation of cut up primitive solids. The meshing of hollow or sliced solids were previously computed by Boolean subtractions and additions, which involved creating two separate solids and acting upon both of them. Which resulted in a very computationally heavy and less aestetic outcome. The the mesh lines ('slice and stack'), were not in radial directions.

2.1 Co-ordinate Systems

The various primitive solids are all constructed by using the predefined parameters used by Geant4, to be consistent with Geant4's own solids. The parameters would be properties of a 3D solid such as height or radius. Which are then used as a way to define the points of the object via basic trigonometry.

2.1.1 Cylindrical Co-ordinate System

The meshing for the primitive solids in cylindrical polar coordinate systems are constructed by looping though the number of slices and stacks which the cylinder is being cut into. The Loop then creates the coordinates for 3 or 4 points at a time, which can then be defined as a triangular or polygonal face. The only cases where the mesh produces triangles is at the top and bottom faces of the cylinder, provided it does has a minimum radius equal to zero (creating a tube or cone).

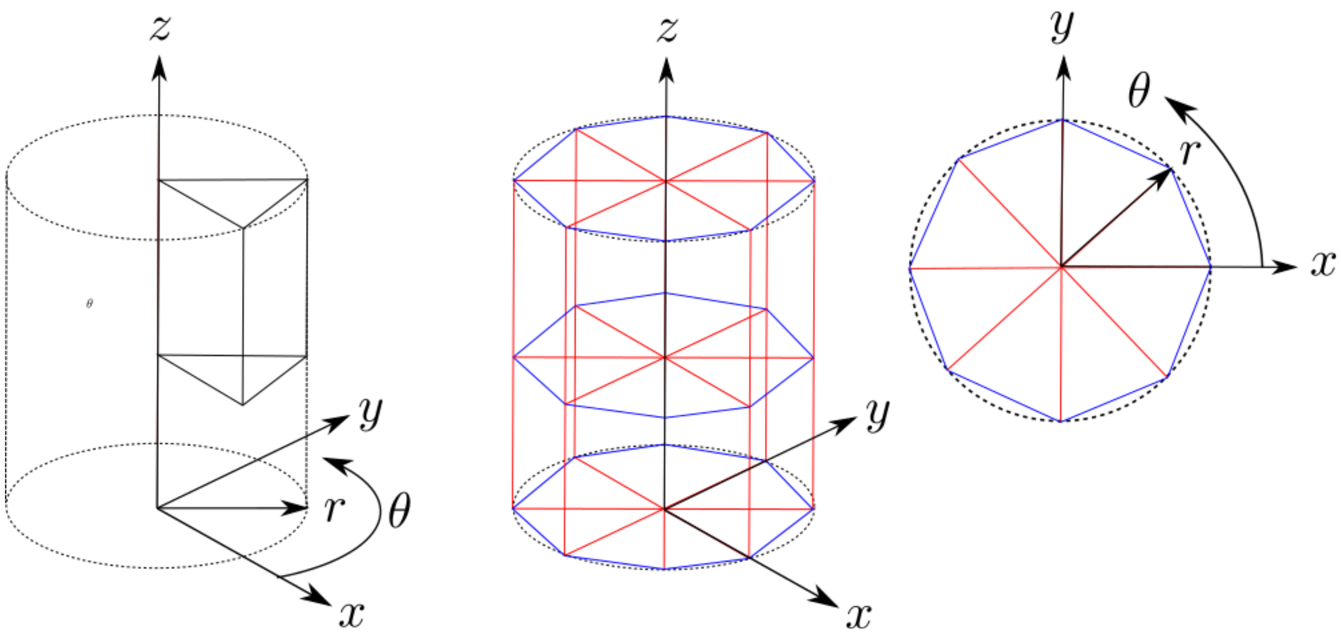

Figure 1: Diagram showing the meshing method for a cylindrical coordinate system
Red = Slices (8)
Blue = Stack (2)

The trigonometry that converts the points from cylindrical polar coordinates to cartesian, are:

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

```python
polygons = []

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

    vertices = []

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

Listing 1: Python example

The code in Listing 2.1.1 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. Same logic applies for the triangles, just using 3 points.

The only time stack is needed in the cylindrical coordinate system is when the solid has a non linear function in the r-z plane. For example a paraboild (Figure 18) would need a stack but a linear cone (Figure 5) would not.

2.1.2 Spherical Co-ordinate System

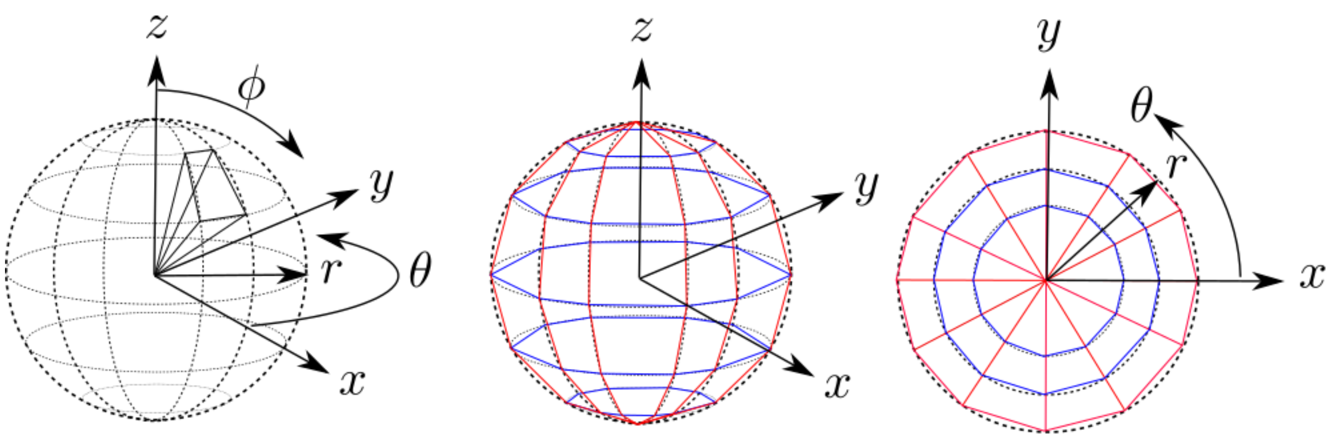

Figure 2: Diagram showing the meshing method for a spherical coordinate system
Red = Slices (12)
Blue = Stack (6)

The meshing for the primitive solids in spherical coordinate systems are constructed by similar means the that of the spherical just with different trigonometric equations (2) as a result of two angle parameters ϕ and θ.

The trigonometry that converts the points from spherical coordinates to cartesian, are:

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

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

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

Listing 2: Python example

2.1.3 Toroidal Co-ordinate System

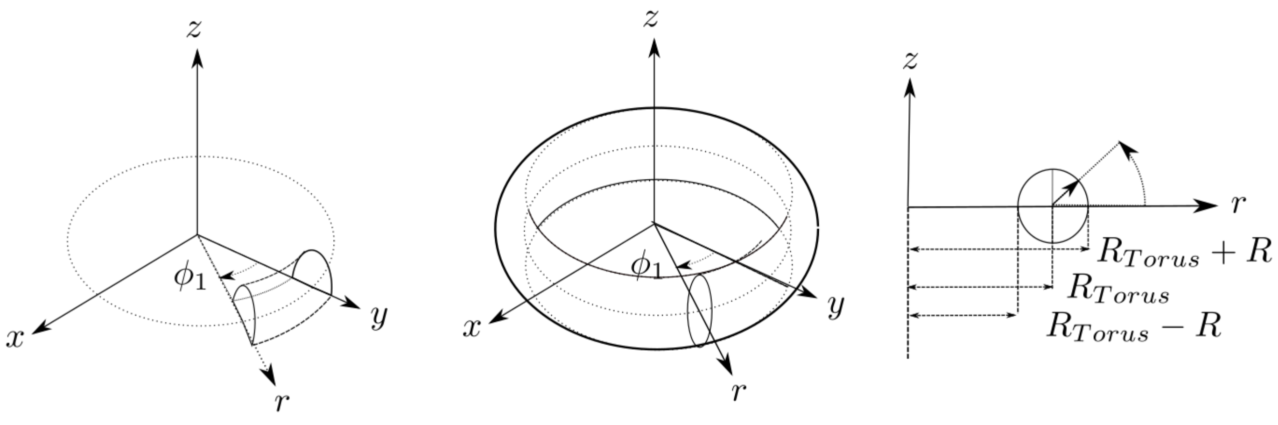

Figure 3: Diagram showing the meshing method for a toroidal coordinate system

The torus is much harder to visualise for a stack and slice, a toroidal slice is an R_{Torus} radial cut taken out of the angle ϕ, shown in Figure 3. The toroidal stack is a R radial cut out of the angle θ.

The trigonometry that converts the points from toroidal coordinates to cartesian, are:

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

2.2 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 incorrect, will lead to missing faces, when the meshing is made. The concept is demonstrated in Figure 4.

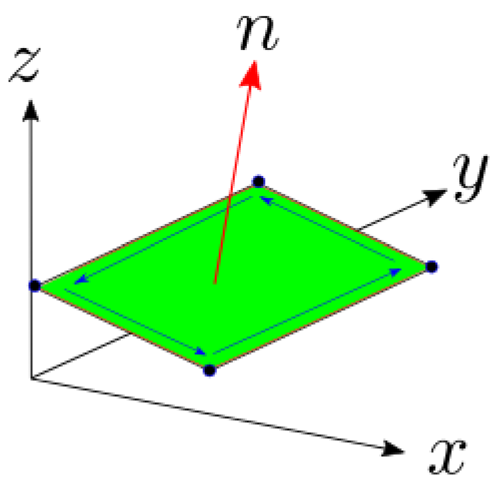

Figure 4: Diagram showing the order convention of appending points to define the normal to a plane

2.3 Degenerate points

Multiple points occupying the same area

2.4 Meshing performance testing

2.5 Curved primitive solids

for each shape
stack and slice ?
radially mesh ?
which coord system

2.5.1 Cons

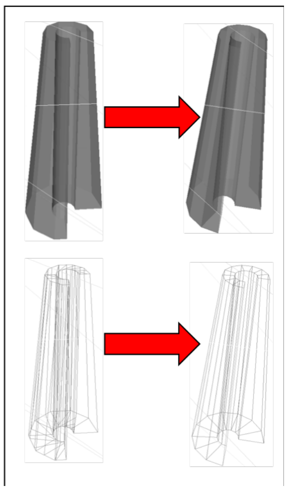

Figure 5: Toroidal Coordinate System

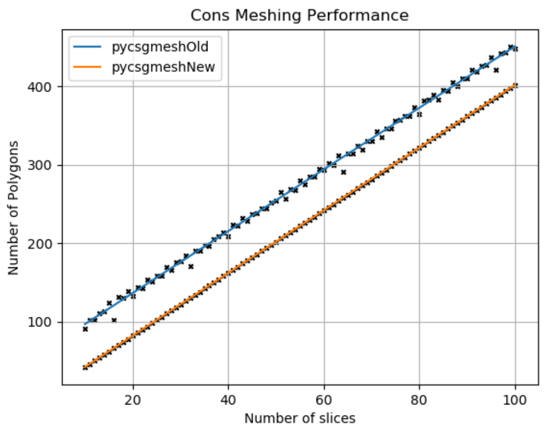

Figure 6: Spherical Coordinate System

2.5.2 CutTubs

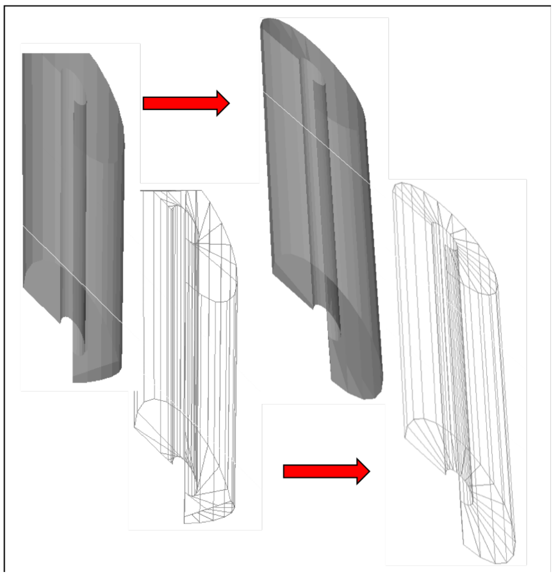

Figure 7: Toroidal Coordinate System

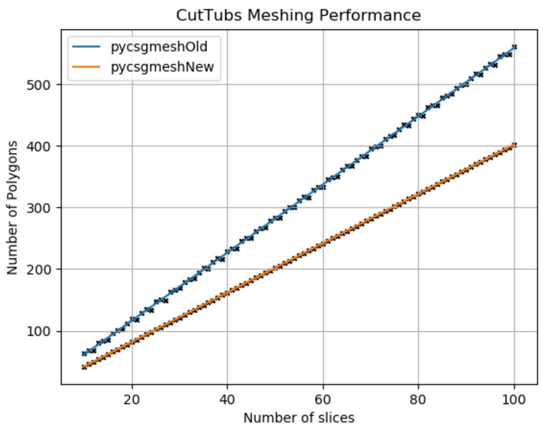

Figure 8: Spherical Coordinate System

2.5.3 Ellipsoid

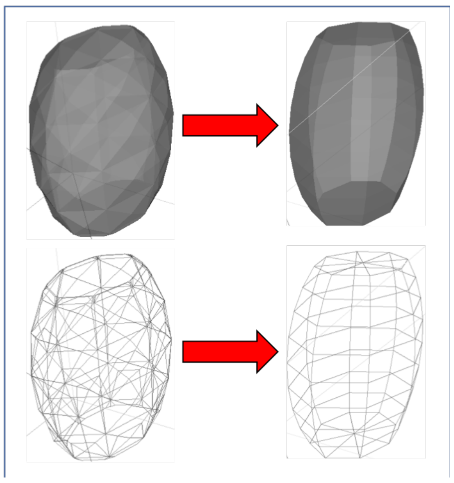

Figure 9: Toroidal Coordinate System

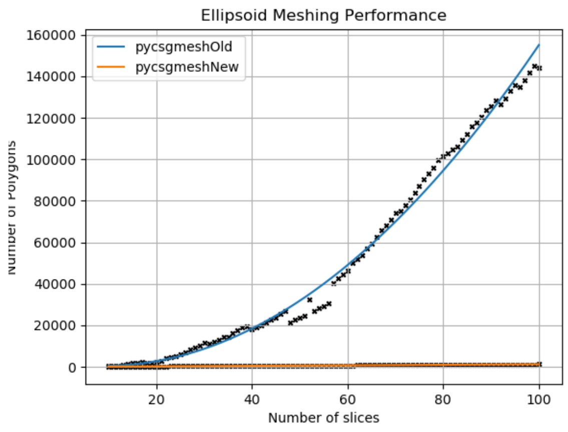

Figure 10: Spherical Coordinate System

2.5.4 EllipticalCone

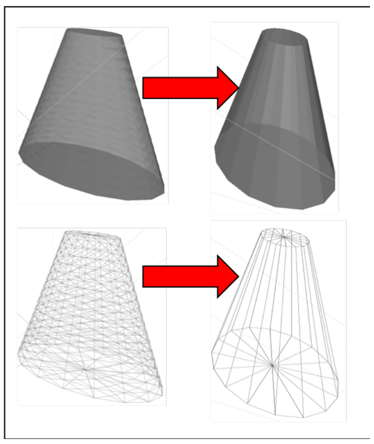

Figure 11: Toroidal Coordinate System

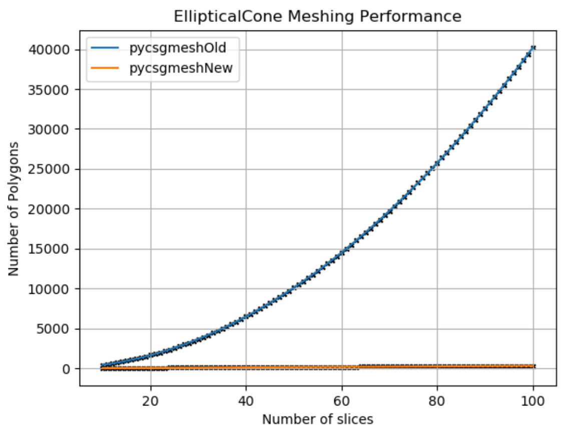

Figure 12: Spherical Coordinate System

2.5.5 EllipticalTube

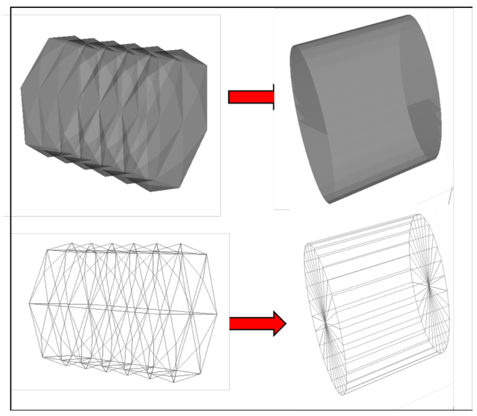

Figure 13: Toroidal Coordinate System

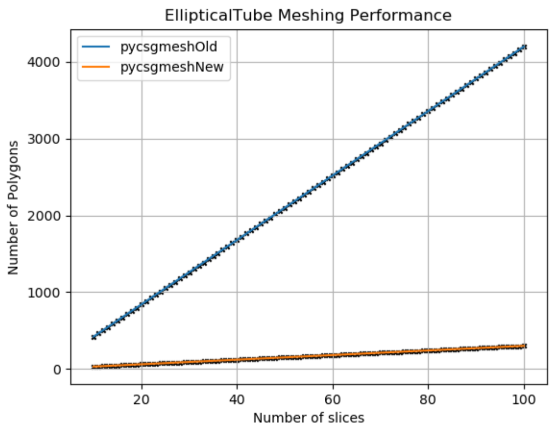

Figure 14: Spherical Coordinate System

2.5.6 Hyperboloid

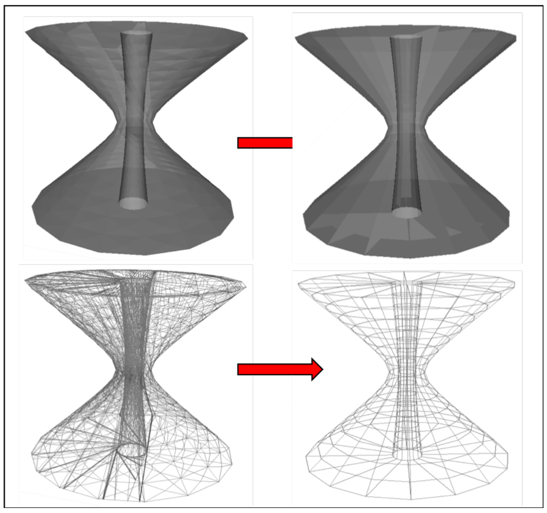

Figure 15: Toroidal Coordinate System

2.5.7 Orb

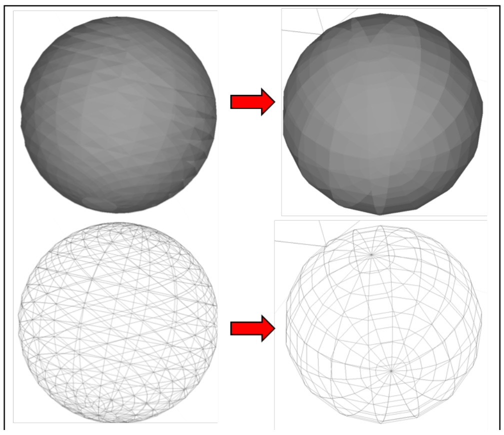

Figure 16: Toroidal Coordinate System

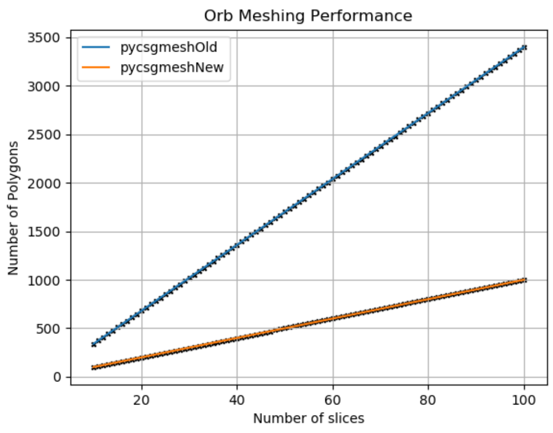

Figure 17: Spherical Coordinate System

2.5.8 Paraboloid

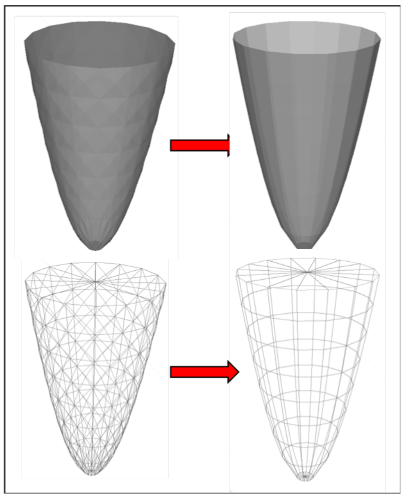

Figure 18: Paraboid Meshing Development

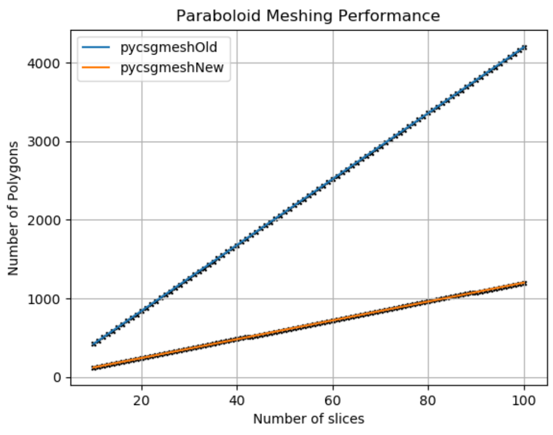

Figure 19: Spherical Coordinate System

2.5.9 Polycone

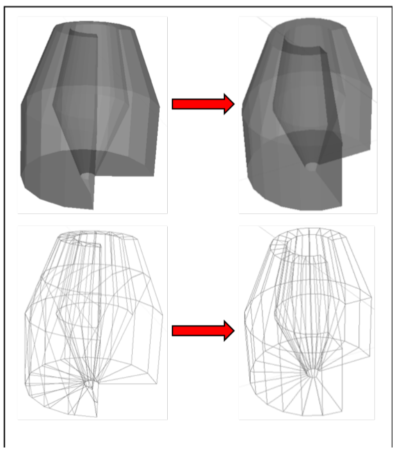

Figure 20: Toroidal Coordinate System

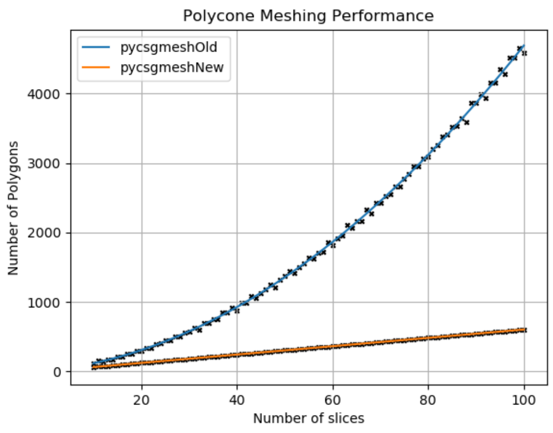

Figure 21: Spherical Coordinate System

2.5.10 Sphere

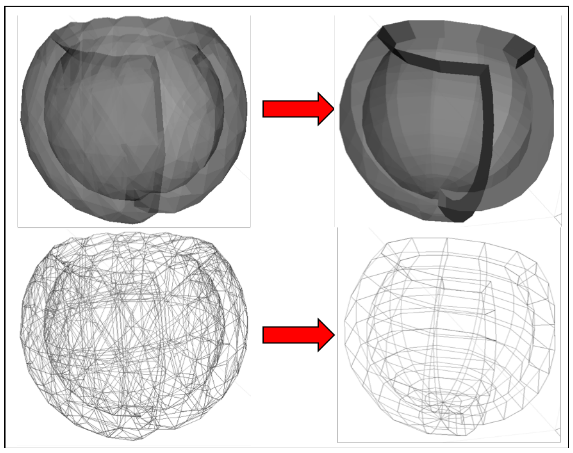

Figure 22: Toroidal Coordinate System

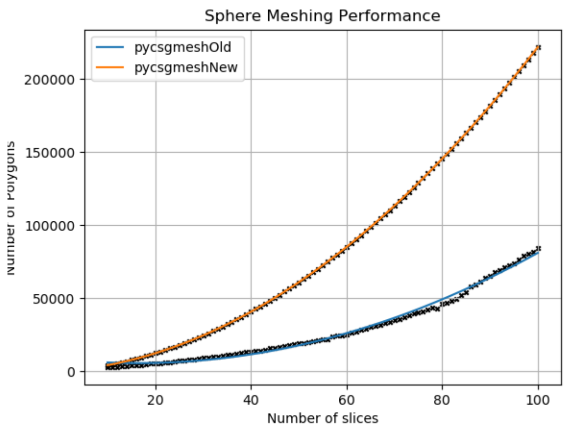

Figure 23: Spherical Coordinate System

2.5.11 Torus

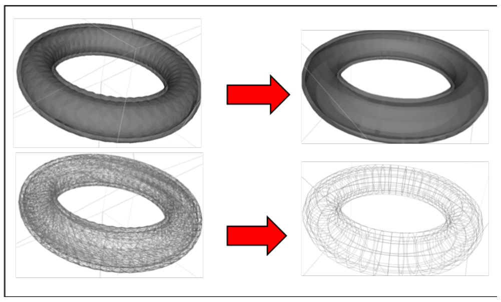

Figure 24: Toroidal Coordinate System

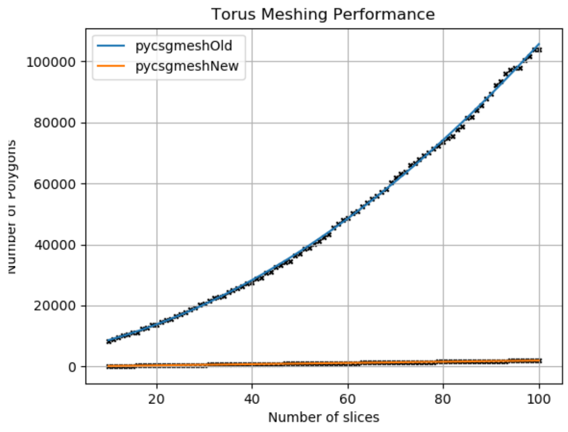

Figure 25: Spherical Coordinate System

2.5.12 Tubs

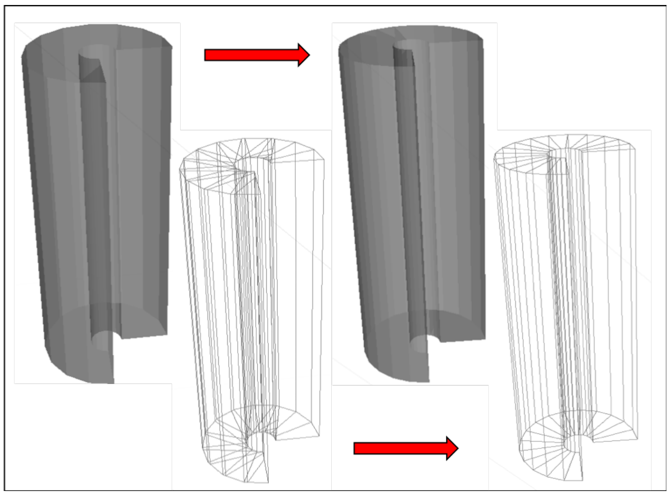
Figure 26: Toroidal Coordinate System

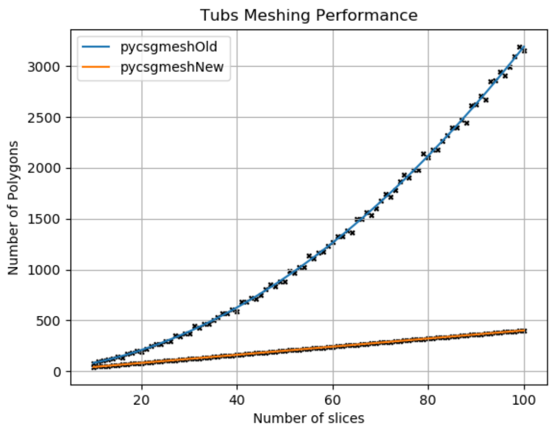

Figure 27: Spherical Coordinate System

2.6 Performance tests

table of perfrmance for each shape
plots

3 BDSIM

3.1 Particle collisions with meshed solids

4 CAD

5 Appendix (Python scripts)

This section lists the Python scripts used to generate some of the figures within this report. Data taken from Online NASA's confirmed exoplanet archive [?], downloaded as .cvs file and imported into Python 3.7. "path" is the path to your ".cvs" type file. May be required to delete unnecessary rows explaining column headers.