Royal Holloway University of London

PH4100: Major Project

BDSIM & pyg4ometry

Ben Shellswell

Abstract

Supervised by
Prof. S Boogert
January 16, 2020

Contents

1 Introduction **1**
1.1 BDSIM . . . 1
1.2 pyg4ometry . . . 1
1.3 Geant4 . . . 1
1.4 Project Aims . . . 1

2 Primitive Meshing **1**
2.1 Co-ordinate Systems . . . 1
2.1.1 Cylindrical Co-ordinate System . . . 1
2.1.2 Spherical Co-ordinate System . . . 2
2.1.3 Toroidal Co-ordinate System . . . 3
2.2 Plane Direction . . . 3
2.3 Meshing performance testing . . . 3
2.4 Curved primitive solids . . . 3
2.4.1 Cons . . . 4
2.4.2 CutTubs . . . 5
2.4.3 Ellipsoid . . . 6
2.4.4 EllipticalCone . . . 7
2.4.5 EllipticalTube . . . 8
2.4.6 Hyperboloid . . . 9
2.4.7 Orb . . . 10
2.4.8 Paraboloid . . . 11
2.4.9 Polycone . . . 12
2.4.10 Sphere . . . 13
2.4.11 Torus . . . 14
2.4.12 Tubs . . . 15
2.5 Performance tests . . . 16

3 BDSIM **16**
3.1 Particle collisions with meshed solids . . . 16

4 CAD **16**

5 Appendix (Python scripts) **17**

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 angles swept through to generate that solid in a given coordinate system.

2.1 Co-ordinate Systems

The various primitive solids were all constructed by using the predefined paramters 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

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

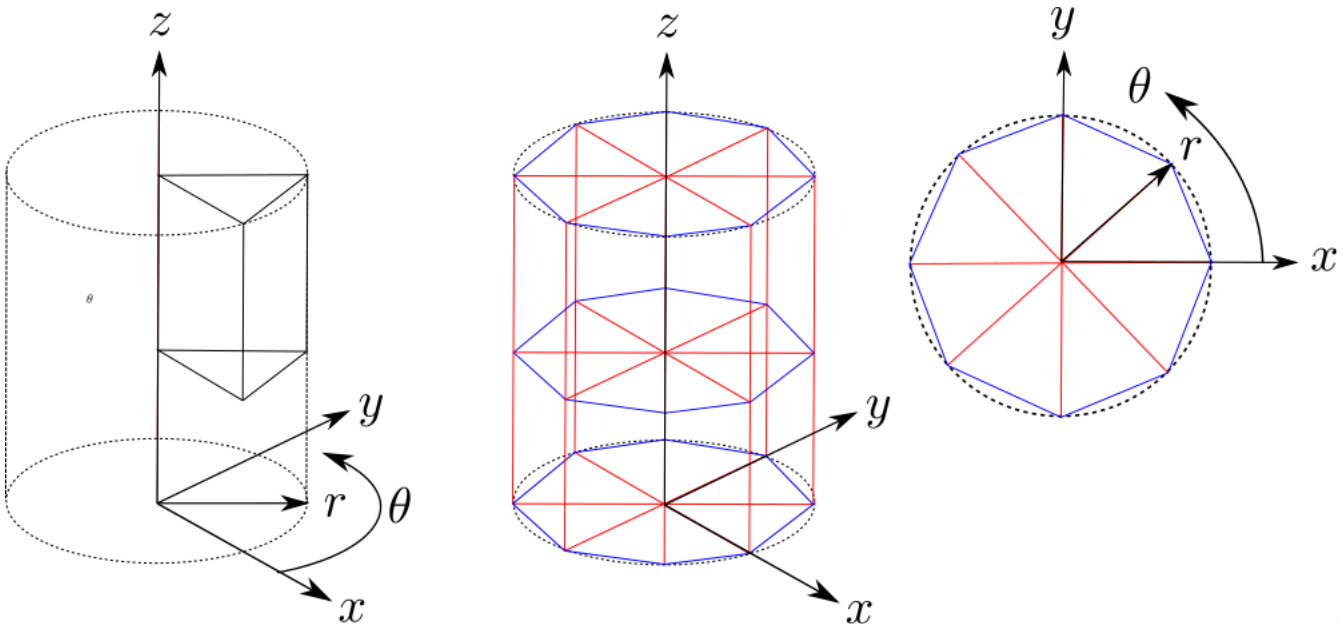

Figure 1: Diagram showing the meshing method for a cylindrical coordinate system
Red = Slices (8)
Blue = Stack (2)

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

Listing 1: Python example

2.1.2 Spherical Co-ordinate System

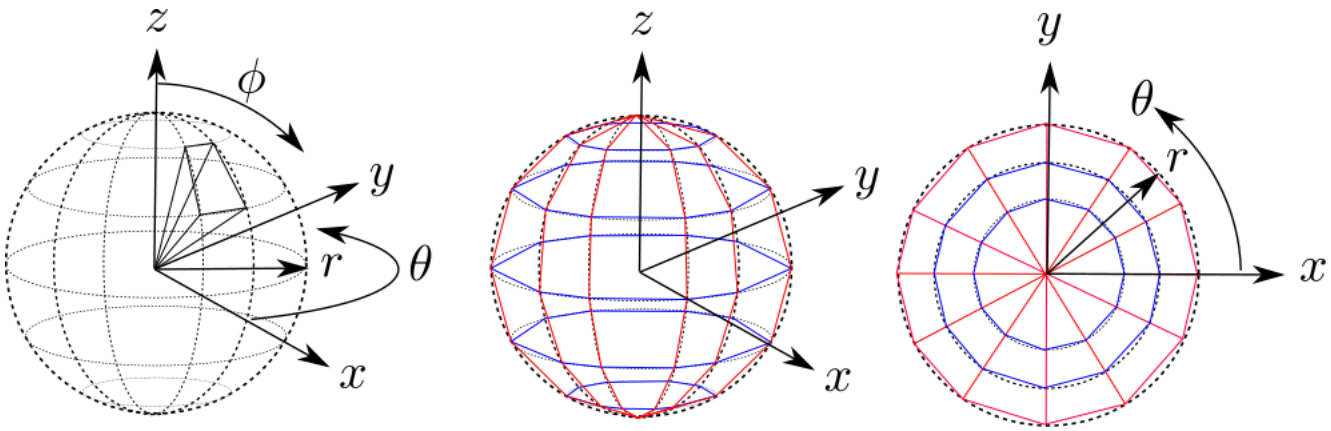

Figure 2: Diagram showing the meshing method for a spherical coordinate system
Red = Slices (12)
Blue = Stack (6)

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

```
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

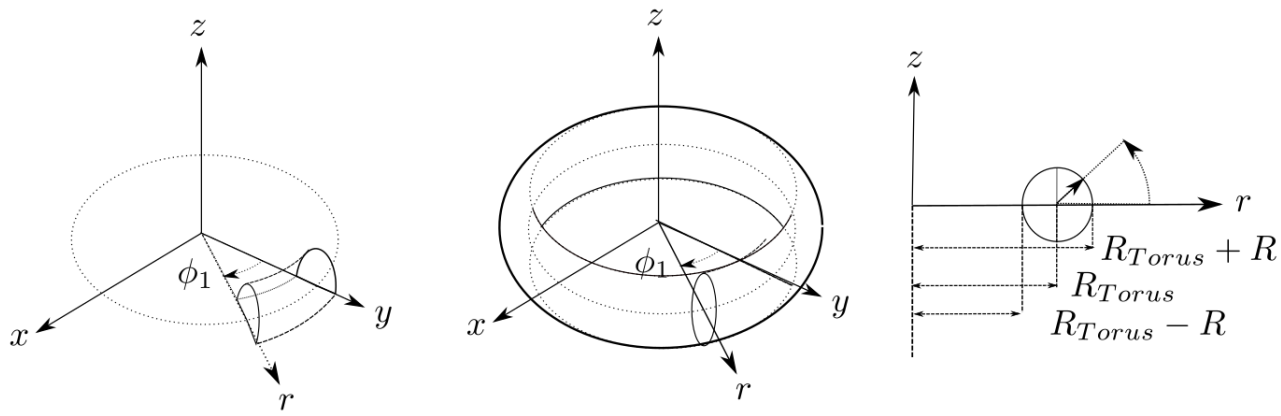

Figure 3: Diagram showing the meshing method for a toroidal coordinate system

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

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.

2.3 Meshing performance testing

2.4 Curved primitive solids

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

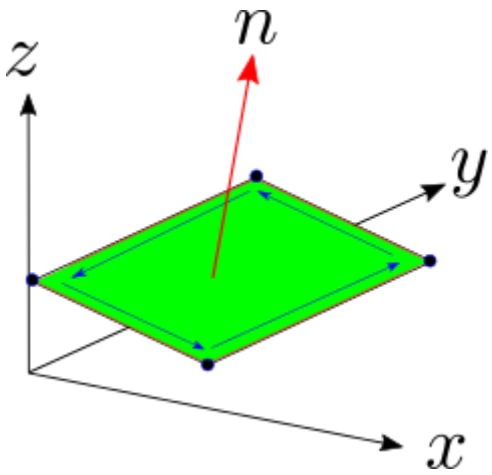

Figure 4: Diagram showing the order convention of appending points to define the normal to a plane

2.4.1 Cons

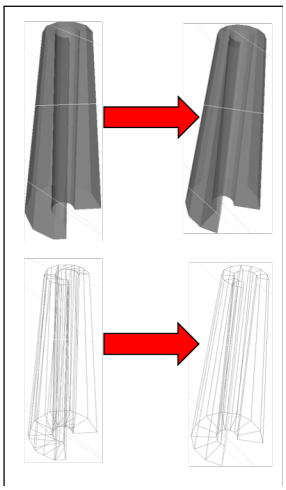

Figure 5: Toroidal Coordinate System

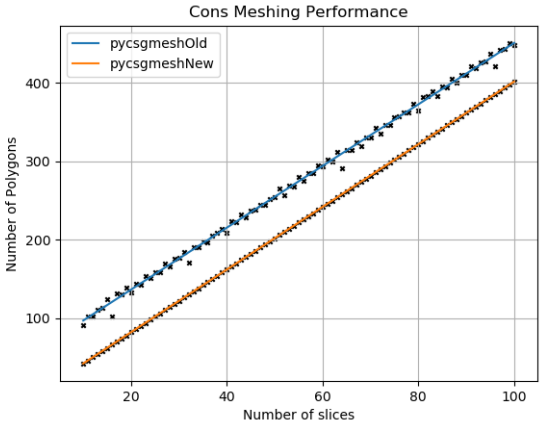

Figure 6: Spherical Coordinate System

2.4.2 CutTubs

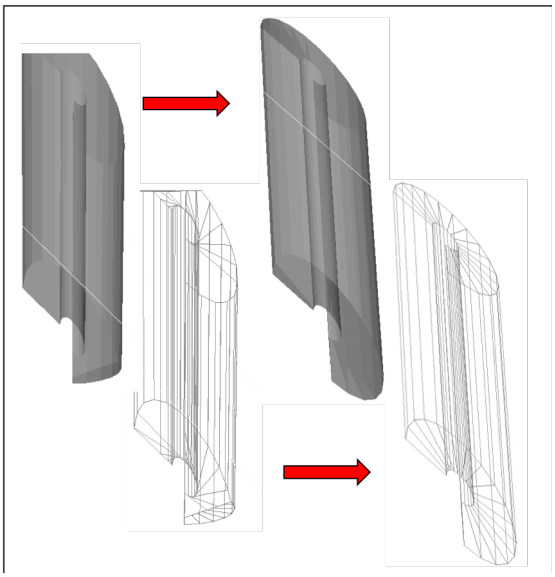

Figure 7: Toroidal Coordinate System

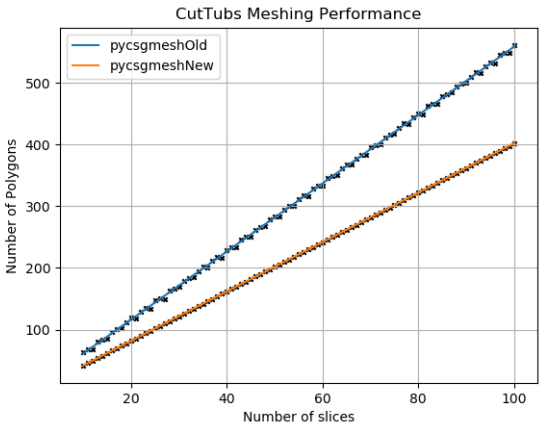

Figure 8: Spherical Coordinate System

2.4.3 Ellipsoid

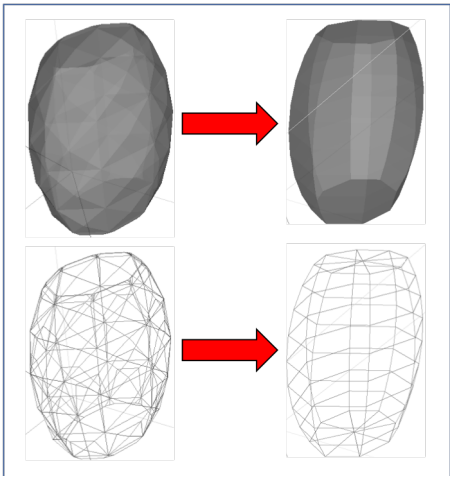

Figure 9: Toroidal Coordinate System

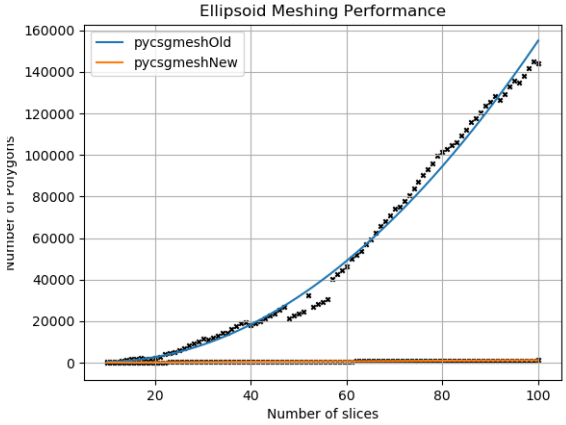

Figure 10: Spherical Coordinate System

2.4.4 EllipticalCone

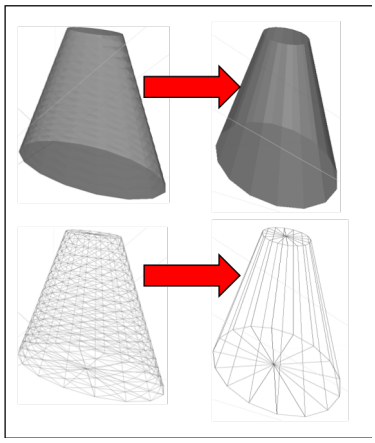

Figure 11: Toroidal Coordinate System

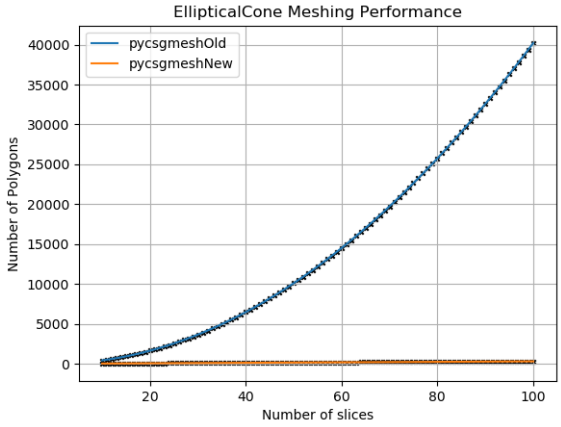

Figure 12: Spherical Coordinate System

2.4.5 EllipticalTube

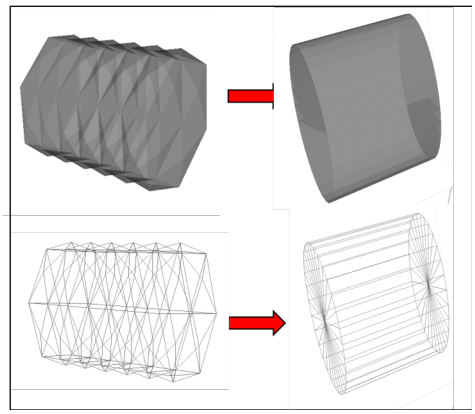

Figure 13: Toroidal Coordinate System

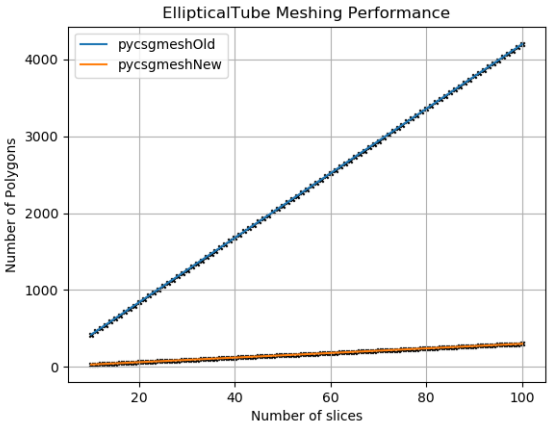

Figure 14: Spherical Coordinate System

2.4.6 Hyperboloid

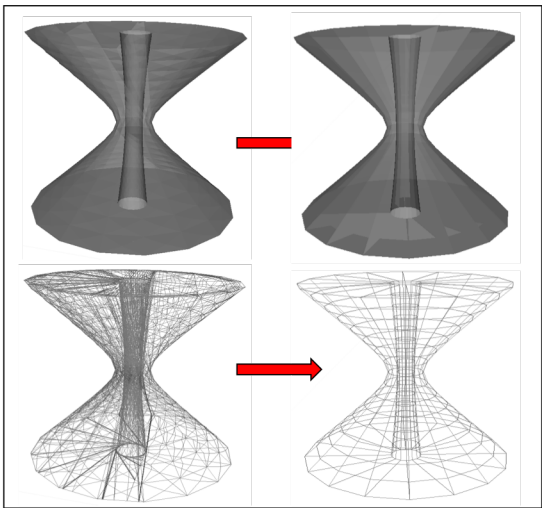

Figure 15: Toroidal Coordinate System

2.4.7 Orb

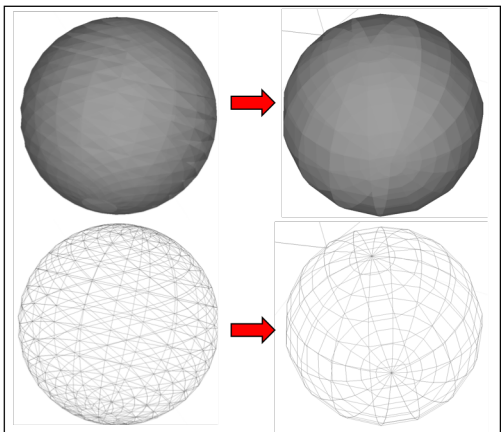

Figure 16: Toroidal Coordinate System

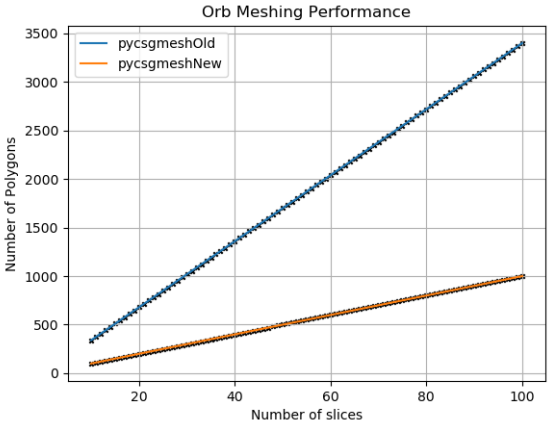

Figure 17: Spherical Coordinate System

2.4.8 Paraboloid

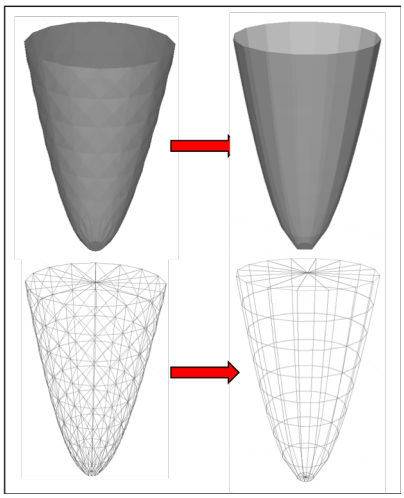

Figure 18: Toroidal Coordinate System

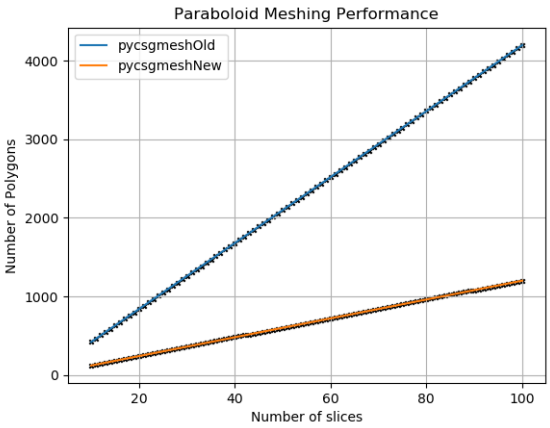

Figure 19: Spherical Coordinate System

2.4.9 Polycone

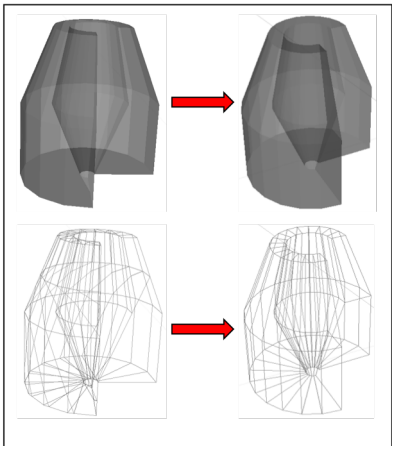

Figure 20: Toroidal Coordinate System

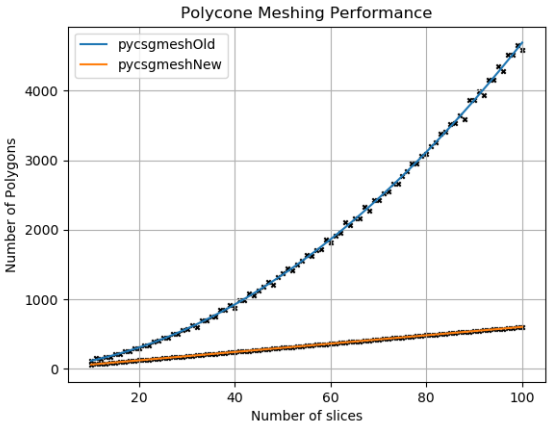

Figure 21: Spherical Coordinate System

2.4.10 Sphere

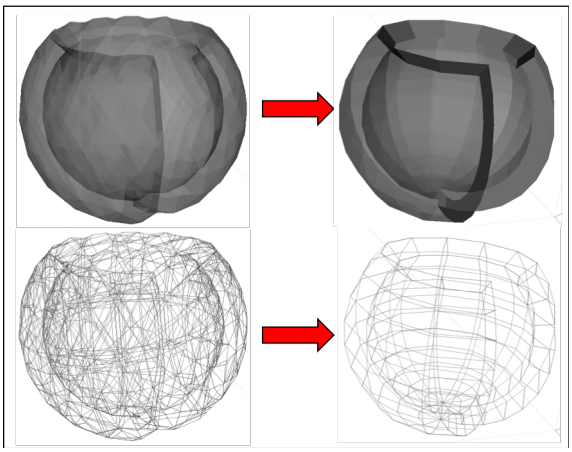

Figure 22: Toroidal Coordinate System

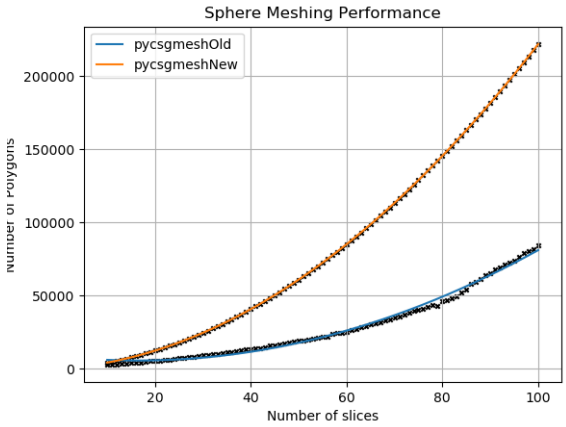

Figure 23: Spherical Coordinate System

2.4.11 Torus

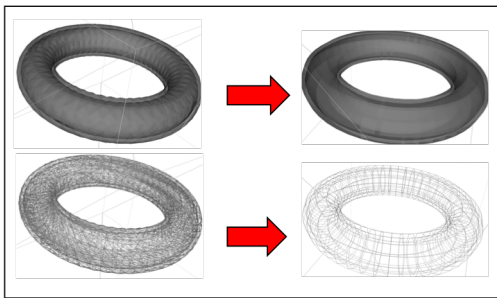

Figure 24: Toroidal Coordinate System

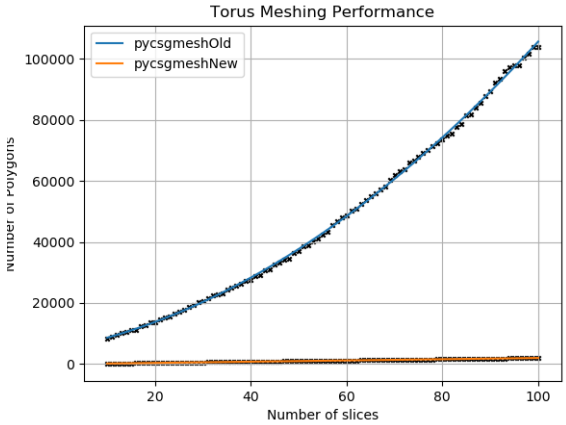

Figure 25: Spherical Coordinate System

2.4.12 Tubs

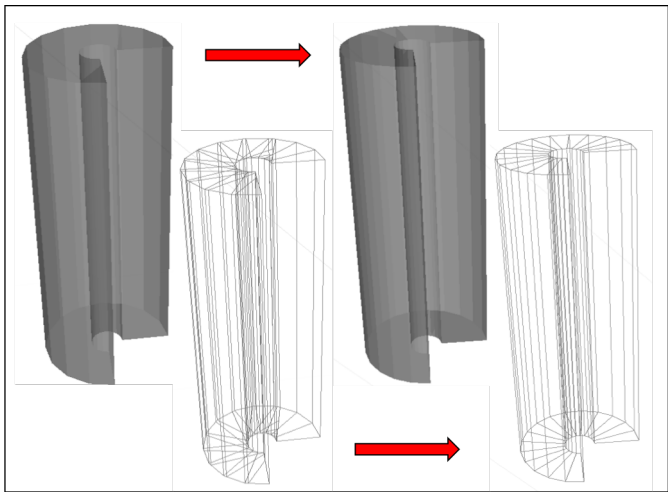

Figure 26: Toroidal Coordinate System

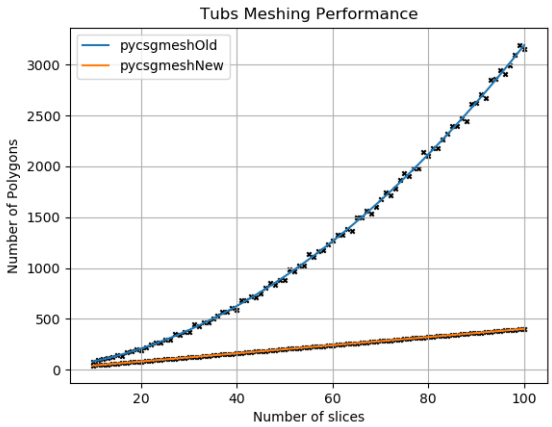

Figure 27: Spherical Coordinate System

2.5 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.