Using pyMGeo to Create Geometry for MCNP Input Files
By Elizabeth Skiba

This document assumes a basic knowledge of object-oriented programming and a familiarity with pyMGeo.

Introduction

pyMGeo is a library that users can use to write Python programs to generate geometry for MCNP input files. pyMGeo simplifies MCNP input file creation by providing functions for generating MCNP cards. Since pyMGeo is a library, users write programs that use pyMGeo to describe the specific geometry of a model. Using a programming approach to generating MCNP input files has many advantages that will be described throughout this document.

As a running example, we will use a model for an activated carbon experiment. This model consists of a group of cells that simulate an activated carbon disk emitting gamma rays and one or more gamma ray detectors. It is based on an original created by hand by Dr. Sharon Stephenson of Gettysburg College. Fig. 1 shows the original geometry drawn by IViPP.

(Fig. 1)

The central concept of pyMGeo is to add objects to a model. In pyMGeo the MCNPModel class is a top level definition for a model that pulls together all the information needed to write an MCNP input file. The class MCNPObject and its user-defined subclasses represent a single cell or group of cells to be added to an instance of MCNPModel. These single cells or groups of cells can be thought of as subgroups of the entire model. It is helpful to define a main, top level MCNPObject subclass that does not define any cells itself, only instantiates sub-groups of the model. Fig. 2 shows a diagram of this idea applied to the activated carbon demo. ‘Activated Carbon Demo’ represents a top level definition of the geometry. Two subgroups in ‘Activated Carbon Demo’ are ‘Carbon Disk Setup’ and ‘Gamma Detector’. The ‘Carbon Disk Setup’ contains cells ‘Sodium Disk’, ‘Plastic Disk’, ‘Holder’, ‘Top Carbon Disk’, and ‘Bottom Carbon Disk.’ The sub-group ‘Gamma Detector’ contains a single cell, the ‘Detector.’
Based on the diagram in Fig. 2, the constructor of an ‘Activated Carbon Demo’ class instantiates ‘Carbon Disk Setup’ and ‘Gamma Detector’ objects. Fig. 3 shows a sample of code for an ‘Activated Carbon Demo’ class.

```python
1 from MCNPObject import MCNPObject
2 from MCNPModel import MCNPModel
3 from CarbonDiskSetup import CarbonDiskSetup
4 from GammaDetector import GammaDetector
5
class ActivatedCarbonDemo( MCNPObject ):
    def __init__( self, model ):
        MCNPObject.__init__( self, model )

    topDetectorTransformation = Composite( Translate( y = 1.199 ), RotateX( 90 ) )
    topDetector = GammaDetector( model, topDetectorTransformation )

    bottomDetectorTransformation = Composite( Translate( y = -5.219 ), RotateX( 90 ) )
    bottomDetector = GammaDetector( model, bottomDetectorTransformation )

carbonDisk = CarbonDiskSetup( model, topDetector, bottomDetector )
```

Lines 1 and 2 of Fig. 3 import classes `MCNPObject` and `MCNPModel` from `pyMGeo`. Lines 3 and 4 import two classes, `CarbonDiskSetup` and `GammaDetector`, which create and define corresponding portions of the activated carbon model. Line 6 defines the class `ActivatedCarbonDemo` as a subclass of `MCNPObject`. The majority of defining and adding geometry occurs in the constructor of subclasses of `MCNPObject`. Line 7 signals the beginning of the constructor of class `DetectorDemo` and line 8 calls the constructor of the superclass, `MCNPObject`. Lines 10 and 12 create transformations that are used on subsequent lines in placing the gamma detectors. Line 15 defines and creates the geometry for the carbon disk setup.

`CarbonDiskSetup` is also a subclass of the `MCNPObject` class. Since there are a lot of dimension parameters for these cells it is helpful to define them once at the beginning of the class definition. Generally, properties such as dimensions or materials of a group of cells are inherent aspects of that group and should be declared as member variables. Our programming approach allows these inherent properties to be referred to by name and only requires each property to be declared once. Fig. 4 shows a sample of code defining the member variables for class `CarbonDiskSetup`. Lines 9 through 16 define height and radii for the cylindrical surfaces in the carbon disk setup. Lines 18 through 21 define
elements and materials used in the carbon disk setup. Note that commonly used isotopes are defined in pyMGeo's class Elements.

```python
import Surface, Elements
from Material import Material
from Cell import Cell
from MCNPObject import MCNPObject
from CSG import *
from Transform import *

class CarbonDiskSetup( MCNPObject ):
    carbonRadius = 3.81
    holderRadius = 4.1265
    plasticRadius = 1.216
    sodiumRadius = 0.31726
    topCarbonThickness = 1.792
    holderThickness = 0.56
    plasticThickness = 0.33
    sodiumThickness = 0.21

    sodium_23 = Elements.Isotope( 11, 23 )
    positron_source = Material( "Sodium23", sodium_23( 100 ) )
    carbon_12 = Material("Carbon12", Elements.carbon_12( 100 ) )
    plastic = Material( "Plastic", Elements.carbon( 60 ), Elements.hydrogen( 8.05 ), Elements.oxygen_16( 31.95 ) )

    def __init__( self, model, topDetector, bottomDetector, transformation=None ):
        MCNPObject.__init__( self, model )
        sodiumTransformation = Composite( RotateX( 90 ) , Translate( z = 0.558 ) )
        sodiumDisk = Surface.Cylinder( "Outer cylinder for Sodium Disk", self.sodiumThickness, self.sodiumRadius, sodiumTransformation )
        self.cells.append( Cell( "Sodium23 disk", positron_source, 0.971, sodiumDisk, ("P", 1), ("E",1) ) )

        plasticTransformation = Composite( RotateX( 90 ), Translate( z = 0.498 ) )
        plasticDiskSurface = Surface.Cylinder( "Outer cylinder for Plastic Disk", self.plasticThickness, self.plasticRadius, plasticTransformation )
        plasticDisk = Intersection( plasticDiskSurface, Compliment(sodiumDisk ) )
        self.cells.append( Cell( "Plastic Disk", plastic, 1.19, plasticDisk, ("P", 1), ("E",1) ) )

        holderTransformation = Composite( RotateX( 90 ), Translate( z = 0.383 ) )
        self.holderSurface = Surface.Cylinder( "Outer cylinder for Holder", self.holderThickness, self.holderRadius, holderTransformation )
        holder = Intersection( self.holderSurface, Compliment( plasticDisk ) )
        self.cells.append( Cell("Holder", plastic, 1.19, holder, ("P", 1), ("E",1)) )
```

The class CarbonDiskSetup directly defines MCNP cells, not subgroups of cells (as seen in Fig. 2). Fig. 5 shows a sample of code defining materials, surfaces, and cells for the CarbonDiskSetup.

Line 22 shows the constructor for class CarbonDiskSetup (note that the two gamma detectors are passed as parameters). Lines 24 through 26 define a transformation, define a surface, and add a cell for the sodium disk to the CarbonDiskSetup object. An MCNPObject contains a list of cells. Every cell contained in an object must be added to this list using the append method. Lines 26, 31, and 36 append cells for the sodium disk, the plastic disk, and the holder.

As stated in the pyMGeo documentation, Cell objects can be created using a Surface object or a CSGOperator object. A CSGOperator object is a collection of surfaces combined using operators such as union, intersection, or complement. Line 30 defines a CSGOperator object to represent the
The plastic disk takes the area inside the cylindrical surface defined in line 29 and the area outside the sodium disk surface. Generally, CSGOperator objects should be used when a Cell is defined by multiple surfaces.

Similarly, lines 33 through 35 define a transformation, define a surface, and define a CSGOperator object. Note that the CSGOperator takes the complement of the plastic surface, not the Cell itself. Taking the complement of a Cell produces a lot of extra surfaces in the MCNP input file.

MCNP requires that two adjacent cells share a common surface. Programmers using pyMGeo should be aware of this constraint. One way to ensure that two adjacent cells share a common surface is to use a “buffer technique”. This technique extends one surface of two adjacent cells into an adjacent surface and uses the Intersection CSG operator to create the cell. Fig. 6 provides a visual example. Let cells be represented by A and B and let outer surfaces be represented by a, b. Let C be a cell adjacent to both A and B and defined by surface c. By the buffer technique surface c extends into cells A and B. Cell C is then defined as

\[ c \cap -a \cap -b \]

where \(-a\) and \(-b\) represent the area outside surfaces a and b.

Fig. 7 applies the buffer technique in defining the two carbon disks that are adjacent to the holder.

```
37  buffer = 0.1
38  topCarbonDiskTransformation = Composite( RotateX( 90 ), Translate( z = 0.943 - buffer ) )
39  self.topCarbonDiskSurface = Surface.Cylinder( "Outer cylinder for top carbon disk",
              self.topCarbonThickness + 2*buffer, self.carbonRadius, topCarbonDiskTransformation )
40  topCarbonDisk = Intersection( self.topCarbonDiskSurface, Compliment( self.holderSurface ), Compliment( self.topDetector.detectorSurface ) )
41  self.cells.append( Cell( "Top carbon disk", carbon_12, 2.25, topCarbonDisk, ("P", 1), ("E", 1) ) )
42
43  bottomCarbonDiskTransformation = Composite( RotateX( 90 ), Translate( z = -1.409 - buffer ) )
44  self.bottomCarbonDiskSurface = Surface.Cylinder( "Outer cylinder for bottom carbon disk",
              self.bottomCarbonThickness + 2*buffer, self.carbonRadius, bottomCarbonDiskTransformation )
45  bottomCarbonDisk = Intersection( self.bottomCarbonDiskSurface, Compliment( self.holderSurface ), Compliment( self.bottomDetector.detectorSurface ) )
46  self.cells.append( Cell( "Bottom carbon disk", carbon_12, 2.25, bottomCarbonDisk, ("P", 1), ("E", 1) ) )
47
48  self.addToModel()
```

(Fig. 7)

Line 38 of Fig. 7 defines a transformation for the top carbon disk surface and accounts for the buffer. In our example this involves translating the cylindrical surface by an additional 0.1 centimeters in the –z direction. Since most of the surfaces pyMGeo works with are macro bodies, using this style is encouraged. On Line 39 the buffer is then added to the height of the cylinder. Lines 40 and 45 define CSGOperator objects for the two carbon disks similar to the example shown in Fig. 6 (recall that the detectors were passed in as parameters). Line 48 adds the cells in the CarbonDiskSetup object to
the entire model. In order for cells to be added to the entire model, the `self.addToModel()` method must be called at the end of the constructor.

Fig. 8 shows sample code for the class `GammaDetector`. As opposed to the `CarbonDiskSetup`, a `GammaDetector` only contains one cell. Again, inherent properties such as the dimensions and materials are defined as member variables.

```python
1 import Elements, Surface
2 from MCNPObject import MCNPObject
3 from Material import Material
4 from CSG import *
5 from Transform import *
6
7 class GammaDetector( MCNPObject ):
8     sodiumIodideThickness = 3.81
9     sodiumIodideRadius = 3.81
10
11     sodium = Elements.Isotope( 11, 23 )
12     iodide = Elements.Isotope( 53, 127 )
13     sodium_iodide = Material( "Sodium Iodide", sodium( 50 ), iodide( 50 ) )
14
15     def __init__( self, model, transformation=None ) :
16         MCNPObject.__init__( self, model )
18
19         self.cells.append( Cell( "Sodium Iodide Detector", self.sodium_iodide, 3.67, self.detectorSurface, ("P", 1), ("E", 1) )
20
21         self.addToModel()
```

(Fig. 8)

Notice that a surface and a cell are defined in the constructor, but a transformation is not. The transformation will be supplied as a parameter to a `GammaDetector` and then applied to the surface. Recall that transformations for the gamma detectors were defined in the constructor for class `DetectorDemo`. Parameterized classes such as `GammaDetector` are another benefit gained from using a programming approach to generating MCNP input files. For example, Fig. 9 demonstrates a hypothetical modification a user might make to the activated carbon experiment. The code in Fig. 9 defines transformations for four additional detectors placed 9.5 centimeters from the origin at 90° angles from each other in the yz plane. Given a list of angles or transformations to iterate over, numerous detectors could easily be added to the model this way.

```python
1 numDetectors = 4
2 for i in range numDetectors:
3     transformation = Composite( Translate( y = 9.5 ), RotateX( i * 90 ) )
4     detector = GammaDetector( model, transformation )
```

(Fig. 9)

Finally, the MCNP input file will be generated in a main method. Fig. 7 shows a sample of code for the main method.

```python
1 if __name__ == '__main__':
2     model = MCNPModel( "Activated Carbon Demo" )
3     ActivatedCarbonDemo(model)
4     file = model.write( "ActivatedCarbonDemoOutput" )
```
Line 5 of Fig. 10 creates an ActivatedCarbonDemo object, telling it what MCNPModel object it is a part of. Line 7 writes the MCNP model to the desired file (see write method in MCNPModel). After the geometry information is written to the MCNP input file, users can use Python’s ‘write’ method for files to add any additional information needed as illustrated in lines 9 through 29.

To see all of the source code for the activated carbon demo please see Appendix 1. The source code includes an extra class for defining air around the model.
Appendix 1 – Source code for the Activated Carbon Demo using pyMGeo

... 
Created on May 16, 2011 

Authors: Elizabeth Skiba 

Copyright @ 2011, Research Foundation of the State of New York 

... 

File: DetectorDemo 

This is the main module for generating our activated carbon model. This is where entire sections of the model such as the carbon disks and detectors are added to the model. After everything is is added to the model the MCNP input file is generated and physics information is added.

Note: The model was defined in centimeters but if the user wishes to define the model in inches a conversion factor may be passed to the write method for MCNPModel and the output will be converted.

...

#imports from pyMGeo
from MCNPModel import MCNPModel
from MCNPObject import MCNPObject
from CSG import *
from Transform import *

#imports from DetectorDemo
from CarbonDiskSetup import CarbonDiskSetup
from GammaDetector import GammaDetector
from DemoAir import DemoAir

...

Class: ActivatedCarbonDemo

This class represents the entire model. It extends MCNPObject and takes the model as a parameter. The carbon disk setup, the two detectors, and the air surrounding the model are then added.

...

class ActivatedCarbonDemo(MCNPObject):
    def __init__(self, model):
        MCNPObject.__init__(self, model)

        #Define the two detectors that are adjacent to the carbon disk setup
        #by first defining a desired transformation and then passing that transformation
        #to the GammaDetector.__init__
        topDetectorTransformation = Composite(Translate(y = 1.199), RotateX(90))
        self.topDetector = GammaDetector(model, topDetectorTransformation)
        bottomDetectorTransformation = Composite(Translate(y = -5.219), RotateX(90))
        self.bottomDetector = GammaDetector(model, bottomDetectorTransformation)

        #Define Carbon Disk Setup
        carbonDisk = CarbonDiskSetup(model, self.topDetector, self.bottomDetector)

        #Add Air to Model (includes void cells)
        air = DemoAir(model, carbonDisk, self.topDetector, self.bottomDetector)
The following shows an example of how additional detectors could be added to the model. Note that the definition for the air and the void cell for the model would need to change, this is just hypothetical.

```python
for i in range(0,5):
    transformation = Composite(Translate(y = 9.5), RotateX(i*90))
    detector = GammaDetector(model, transformation)
```

Main method:

This is where the cells, surfaces, and materials are added to the model using MCNPModel's write method. The necessary physics information is then added to the file by the user.

```python
if __name__ == '__main__':
    model = MCNPModel("Activated Carbon Demo")

    ActivatedCarbonDemo(model)

    file = model.write("C:/Users/eks6/Desktop/demo")
    file = open('C:/Users/eks6/Desktop/demo.txt', 'a')

    file.write('mode p e\n')
    file.write('nps 1000\n')
    file.write('phys:p .546 1.0\n')
    file.write('phys:p .511009 1.0\n')
    file.write('sdef par=-e erg=D1 x=0 Y=0 Z=.663\n')
    file.write('SI1 .120 9i .545\n')
    file.write('SP1 -4 .330 .215\n')
    file.write('f11:p 2.3\n')
    file.write('e11 0 .3 .5 .51 .52 .6 1.0\n')
    file.write('f21:p 2.2\n')
    file.write('e21 0 .3 .5 .51 .52 .6 1.0\n')
    file.write('f31:p 1.2\n')
    file.write('e31 0 .3 .5 .51 .52 .6 1.0\n')
    file.write('f41:p 1.3\n')
    file.write('e41 0 .3 .5 .51 .52 .6 1.0\n')
    file.write('f4:p 1\n')
    file.write('e4 0 .3 .5 .51 .52 .6 1.0\n')
    file.write('f14:p 2\n')
    file.write('e14 0 .3 .5 .51 .52 .6 1.0\n')
    file.write('print\n')
    file.write('PTRAC Filter=.511008,erg EVENT=sur FILE=asc TYPE=p CELL=7 WRITE=all')
    file.close()
```

File: CarbonDiskSetup

```python
import Surface, Elements, Material
from Cell import *
from CSG import *
from Transform import *
from MCNPObject import MCNPObject
```
This class represents the carbon disk setup. The carbon disk setup contains a sodium23 disk surrounded by a plastic holder and two carbon disks on either side.

```python
class CarbonDiskSetup(MCNPObject):
    #Constant: carbonRadius
    #The radius of the carbon disk
    carbonRadius = 3.81

    #Constant: holderRadius
    #The radius of the holder
    holderRadius = 4.1265

    #Constant: plasticRadius
    #The radius of the plastic
    plasticRadius = 1.216

    #Constant: sodiumRadius
    #The radius of the sodium 23 disk
    sodiumRadius = 0.31726

    #Constant: topCarbonThickness
    #The thickness of the carbon disk on top of the holder
    topCarbonThickness = 0.256

    #Constant: bottomCarbonThickness
    #The thickness of the carbon disk on the bottom of the holder
    bottomCarbonThickness = 1.792

    #Constant: holderThickness
    #The radius of the holder
    holderThickness = 0.56

    #Constant: plasticThickness
    #The thickness of the plastic
    plasticThickness = 0.33

    #Constant: sodiumThickness
    #The radius of the sodium 23 disk
    sodiumThickness = 0.21

    #Define isotopes and materials used in the carbon disk setup
    sodium_23 = Elements.Isotope(11, 23)
    positron_source = Material.Material("Sodium23", sodium_23(100))
    carbon_12 = Material.Material("Carbon 12", Elements.carbon_12(100))
    plastic = Material.Material("Plastic", Elements.carbon(60), Elements.hydrogen(8.05), Elements.oxygen_16(31.95))
```

Constructor: __init__

Parameters:
- model: The model to which this carbon disk setup should be added.
- topDetector: The top detector adjacent to the carbon disk setup.
- bottomDetector: The bottom detector adjacent to the carbon disk setup.
Note: It is necessary for the carbon disk setup to know about the two detectors since
the detectors each share a common surface with the carbon disk setup.

```python
def __init__(self, model, topDetector, bottomDetector, transformation=None):
    MCNPObject.__init__(self, model)
    # Define cylindrical surface for the sodium 23 disk
    sodiumTransformation = Composite(RotateX(90), Translate(z=0.558))
    sodiumDisk = Surface.Cylinder("Outer cylinder for Sodium Disk",
                               self.sodiumThickness,
                               self.sodiumRadius,
                               sodiumTransformation)
    # Add sodium 23 disk to model
    self.cells.append(Cell.Cell("Sodium23 Disk", self.positron_source, .971, sodiumDisk,
                                ("P", 1), ("E",1)))
    # Define cylindrical surface for the plastic
    plasticTransformation = Composite(RotateX(90), Translate(z=0.498))
    plasticDiskSurface = Surface.Cylinder("Outer cylinder for Plastic Disk",
                                          self.plasticThickness,
                                          self.plasticRadius,
                                          plasticTransformation)
    # Add plastic cell to model, it is the space inside the cylinder outside the sodium 23
    plasticDisk = Intersection(plasticDiskSurface, Compliment(sodiumDisk))
    self.cells.append(Cell.Cell("Plastic Disk", self.plastic, 1.19, plasticDisk,
                                ("P", 1), ("E",1)))
    # Define cylindrical surface for the holder
    holderTransformation = Composite(RotateX(90), Translate(z=0.383))
    self.holderSurface = Surface.Cylinder("Outer cylinder for Holder",
                                          self.holderThickness,
                                          self.holderRadius,
                                          holderTransformation)
    # Add holder to cell model, it is the space inside the cylinder outside the plastic
    holder = Intersection(self.holderSurface, Compliment(plasticDisk))
    self.cells.append(Cell.Cell("Holder", self.plastic, 1.19, holder,
                                ("P", 1), ("E",1)))
    # This buffer is necessary since MCNP requires that all adjacent cells share a common
    # surface. We can take the difference of the surface for the holder and the surface
    # for the carbon disk to create a common surface between the two.
    buffer = 0.1
    # Define transformation and cylindrical surface for the top carbon disk
    topCarbonDiskTransformation = Composite(RotateX(90), Translate(z=0.943 - buffer))
    self.topCarbonDiskSurface = Surface.Cylinder("Outer cylinder for top carbon disk",
                                              self.topCarbonThickness + 2*buffer,
                                              self.carbonRadius,
                                              topCarbonDiskTransformation)
    # Create a cell for the top carbon disk
    topCarbonDisk = Intersection(self.topCarbonDiskSurface, Compliment(self.holderSurface),
                                  Compliment(topDetector.detectorSurface))
    self.cells.append(Cell.Cell("Top carbon disk", self.carbon_12, 2.25, topCarbonDisk,
                                ("P", 1), ("E", 1)))
    # Define transformations and cylindrical surface for the bottom carbon/detector
    bottomCarbonDiskTransformation = Composite(RotateX(90), Translate(z=-1.409 - buffer))
    self.bottomCarbonDiskSurface = Surface.Cylinder("Outer cylinder for bottom carbon disk",
                                           self.bottomCarbonThickness + 2*buffer,
                                           self.carbonRadius,
                                           bottomCarbonDiskTransformation)
```

# Create a cell for the bottom carbon disk
bottomCarbonDisk = Intersection(self.bottomCarbonDiskSurface, Compliment(self.holderSurface),
Compliment(bottomDetector.detectorSurface))
self.cells.append( Cell.Cell("Bottom carbon disk", self.carbon_12, 2.25, bottomCarbonDisk, ("P", 1), ("E", 1)))

# Add cells to model
self.addToModel()

File: GammaDetector

import Elements, Material, Surface
from CSG import *
from Transform import *
from MCNPObject import MCNPObject

Class: GammaDetector

This class represents the sodium iodide gamma ray detectors that are placed around the carbon disk in the model. Note that since these gamma detectors have their own class, it is easy to create multiple instances of them and position them around the carbon disk setup.

class GammaDetector(MCNPObject):
    # Constant: Height of sodium iodide disk
    sodiumIodideThickness = 3.81

    # Constant: Radius of sodium iodide disk
    sodiumIodideRadius = 3.81

    # Define material for the sodium iodide detector
    sodium = Elements.Isotope(11,23)
    iodide = Elements.Isotope(53,127)
    sodium_iodide = Material.Material("Sodium Iodide", sodium(50), iodide(50))

    # Constructor: __init__
    def __init__ (self, model, transformation=None):
        MCNPObject.__init__(self, model)

        # Define surface for sodium iodide detector
        self.detectorSurface = Surface.Cylinder("Outer surface for sodium iodide detector",
        self.sodiumIodideThickness,
        self.sodiumIodideRadius,
        transformation)

        # Create cell for sodium iodide detector
        self.cells.append( Cell.Cell("Sodium Iodide Detector", self.sodium_iodide, 3.67, self.detectorSurface, ("P", 1), ("E", 1)))

        # Add cells to model
self.addToModel()

---

File: DemoAir

---

import Cell, Material, Surface
from CSG import *
from Transform import *
from MCNPObject import MCNPObject

---

Class: DemoAir

This class represents the air surrounding the activated carbon model. It requires information about the carbon disk setup and the detectors.

---

class DemoAir(MCNPObject):

  ---

  Constructor: __init__

  Parameters:
  model - The model to which this air should be added.
  carbon - The carbon disk setup in the model
  detectors - The detectors in the model

  ---

def __init__(self, model, carbon, topDetector, bottomDetector, transformation=None):
  MCNPObject.__init__(self, model)

  #Define the cylindrical surface representing the air
  outerSurface = Surface.Cylinder("simulation area", 10.428, 4.2265, Composite(RotateX(90), Translate(z=-5.319)))

  #The air will be the area inside the outer surface but outside the surfaces for the carbon disks, the holder,
  #and the detectors. Note that these are *surfaces* and not cells. Performing the compliment operation on
  #cells makes cell definitions longer than necessary.
  air = Difference(outerSurface, carbon.holderSurface, carbon.topCarbonDiskSurface, carbon.bottomCarbonDiskSurface,
                   topDetector.detectorSurface, bottomDetector.detectorSurface)

  #Create the cell for the air in the model
  #Note that commonly used materials are defined in the Material module so defining a material is not necessary.
  self.cells.append(Cell.Cell("air in simulation area", Material.air, 0.001205, air, ("P", 1), ("E",1)))

  #Create the void area outside of the air
  self.cells.append(Cell.Void("Area outside simulation area", Compliment(air), ("P", 0), ("E",0)))

  #Add cells to model
  self.addToModel()