Ray Tracer Simulation of SBC

David W. Barker

Overview

The purpose of this notebook is to create a simulation through Ray Tracer that will emulate the intensity of the scintillating light coming off of nucleation events and the other sources of energy deposits within the Argon chamber (especially as it relates to the Argon 41 decay).

Status

The geometry has been completed for the SBC and is listed below. However, there is a significant bug that I could not get to the bottom of. There seems to be an issue with any rays that hit the outer surface of a shape. When this occurs, it seems to ignore the inbounds function. If it hits the inner surface it's fine, and so the issue seems to only be related to rays that intersect the outer surface. More detail will be provided below.

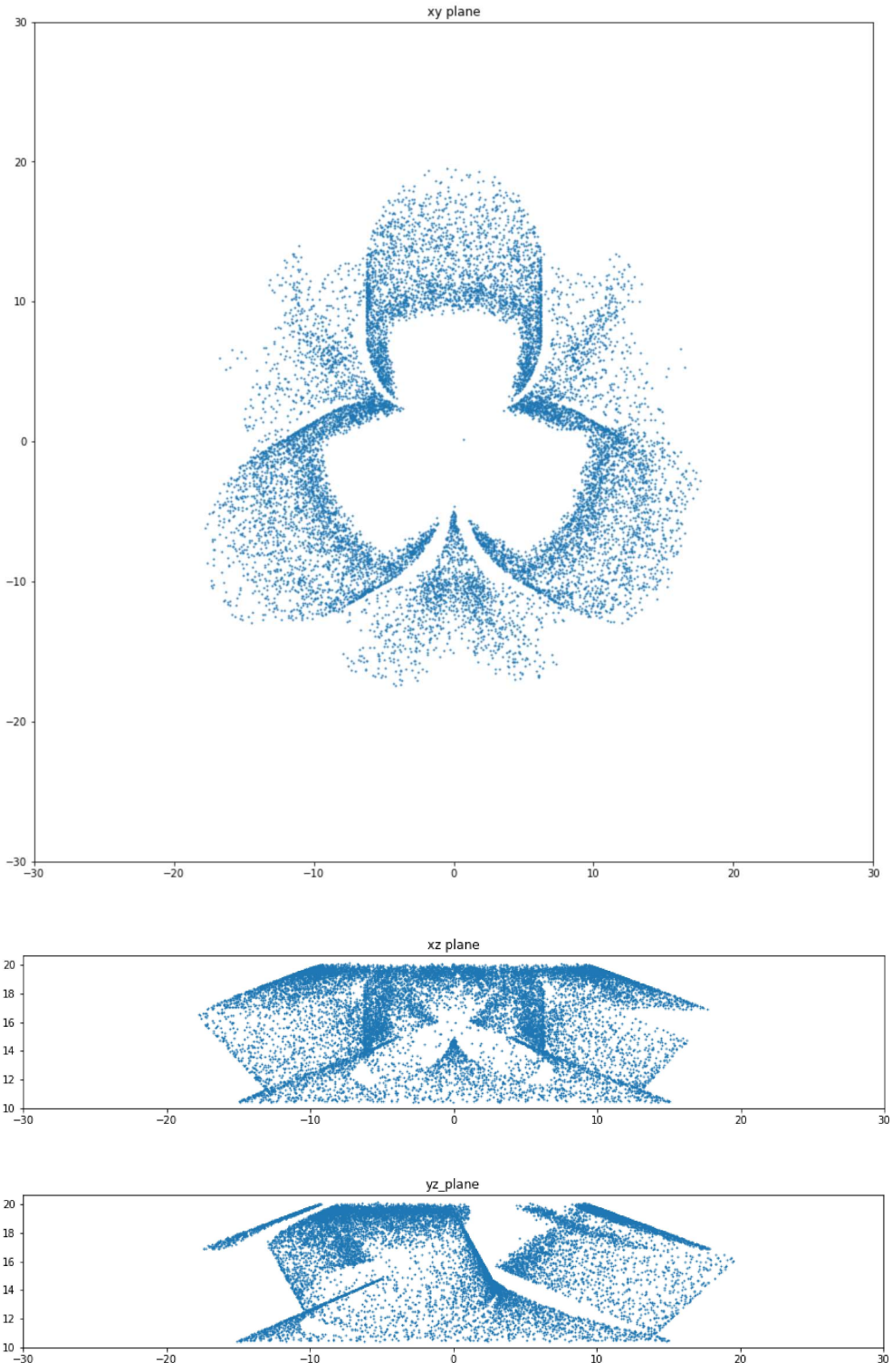
The Geometry

I've created a function that plots the valid intersections to get a sense of the shape created from the surface. Using this function, I plotted the surfaces I've been working on. I'll also use it to show you the bug I have been unable to fix.

Let's start at the top and work our way down.

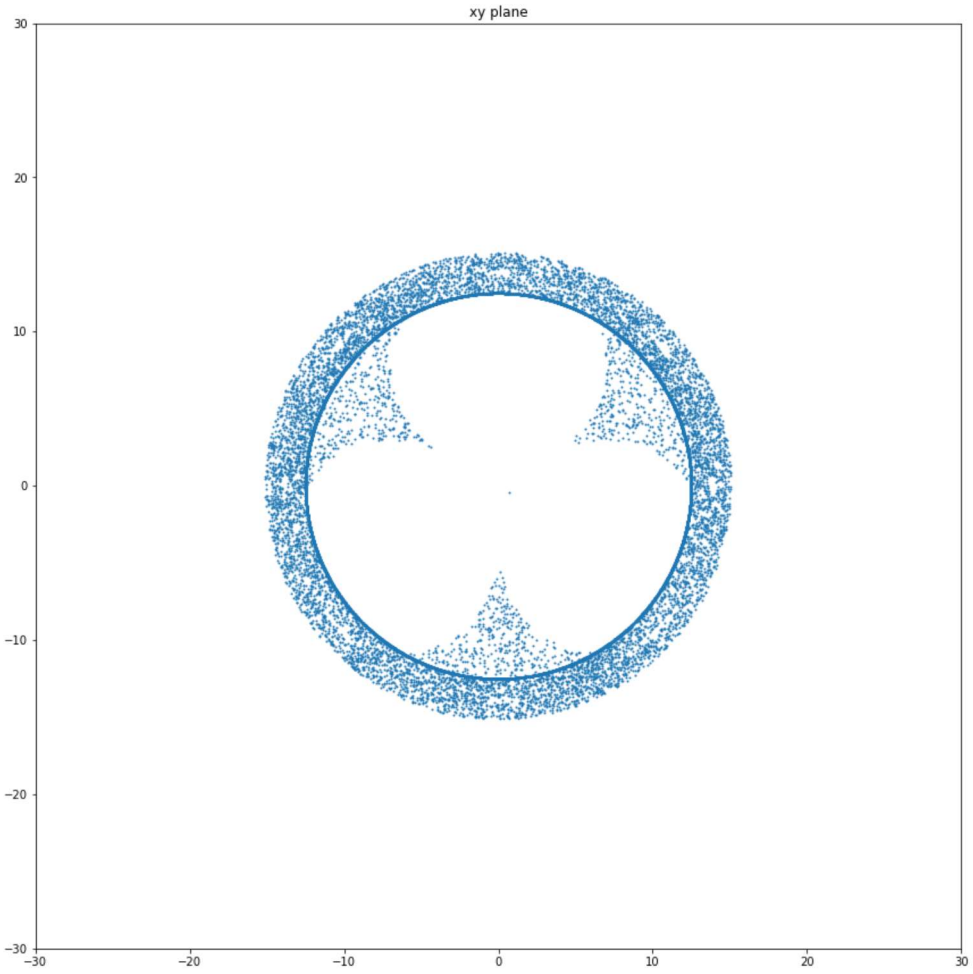
Camera housing and top refelective cone (where the camera viewports are mounted)

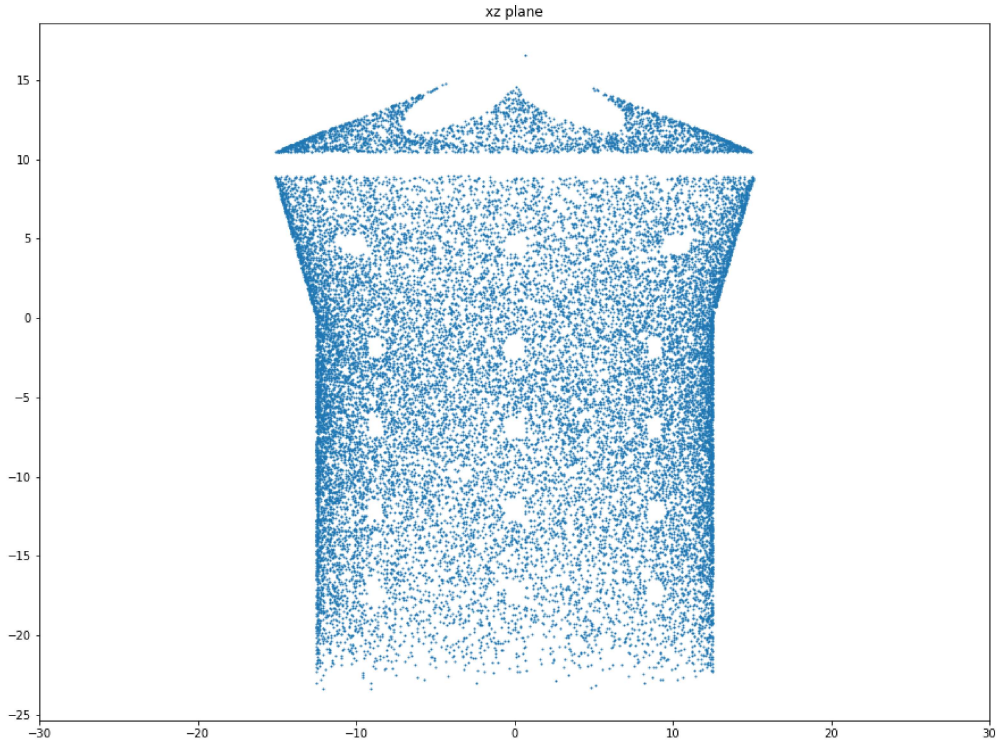
This sections shows the top cone where the camera viewports would be and the camera cans. This one is a bit hard to see from my plotting function because of some of the angles, but you can see that the cans exist and are at angles that make sense. There is also a cone that caps off the ends of the cans to reflect those rays back. I should note that I've listed the height of these camera cans as 7cm, but I'm not sure that is correct. That was the only value I did not have an exact figure for.

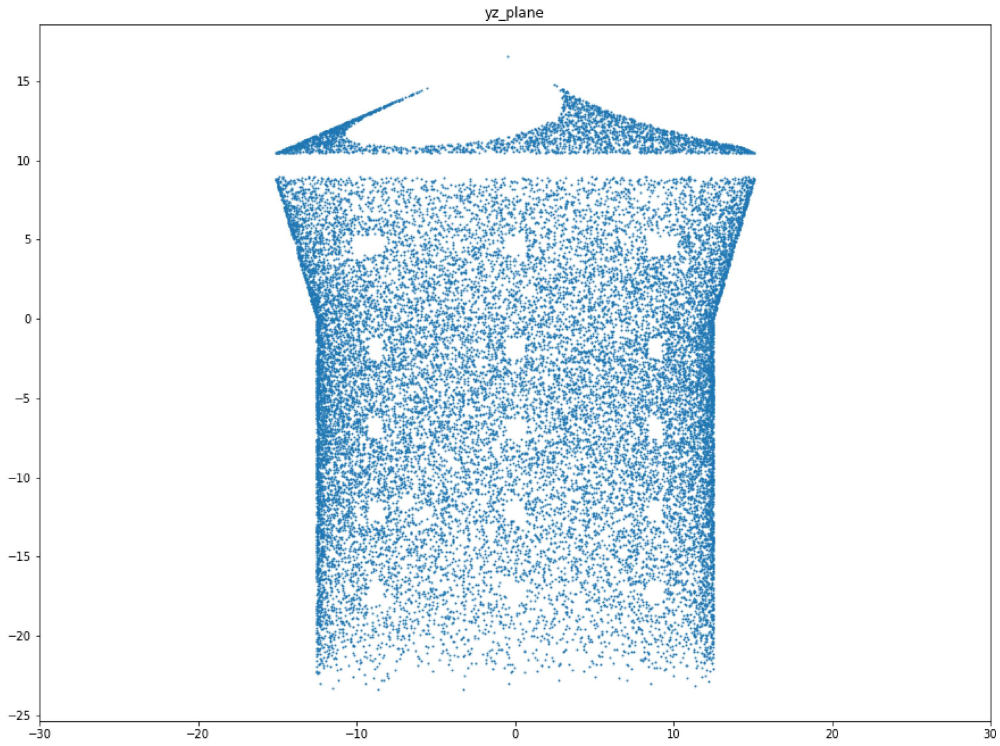


All reflective surfaces below the top cone

Includes the reflective teflon outer cylinder (where the side SiPMs are mounted), the reflective upward facing cone (where the upward facing SiPMs are mounted), and the top cone that was already shone in the section above, but this time without the camera housing and viewports.

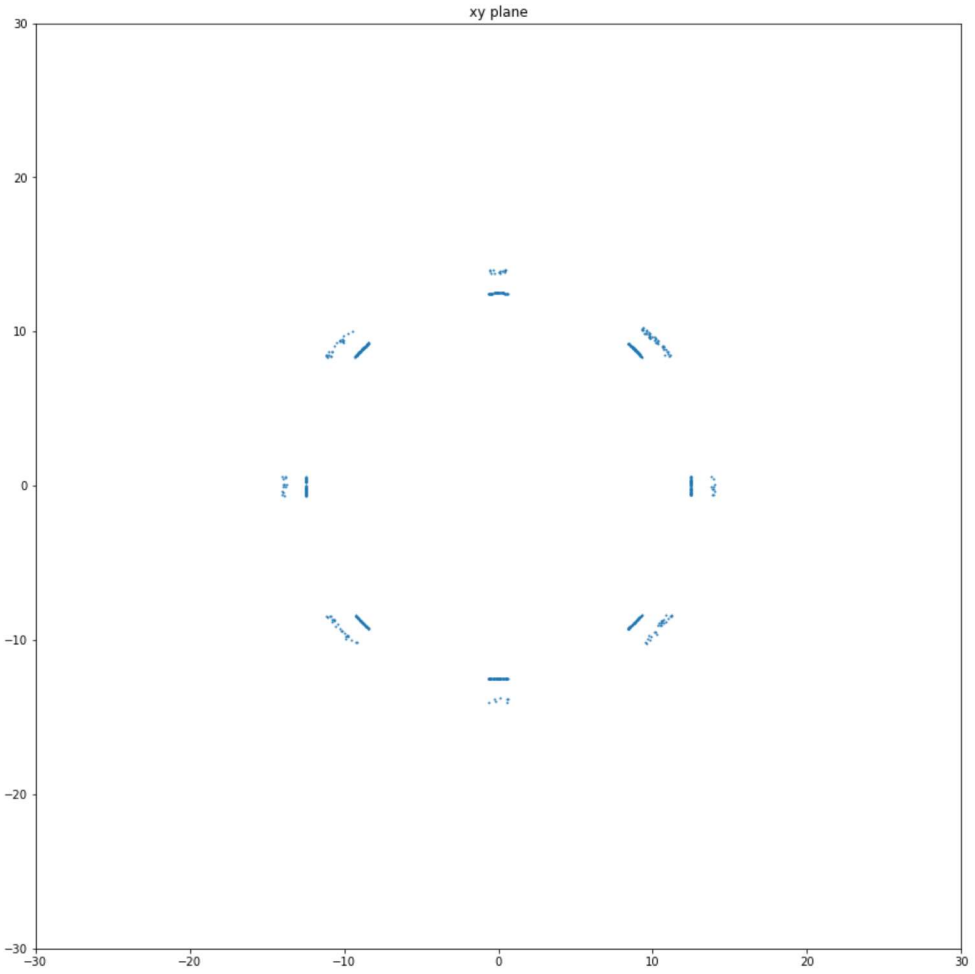

xy plane
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30

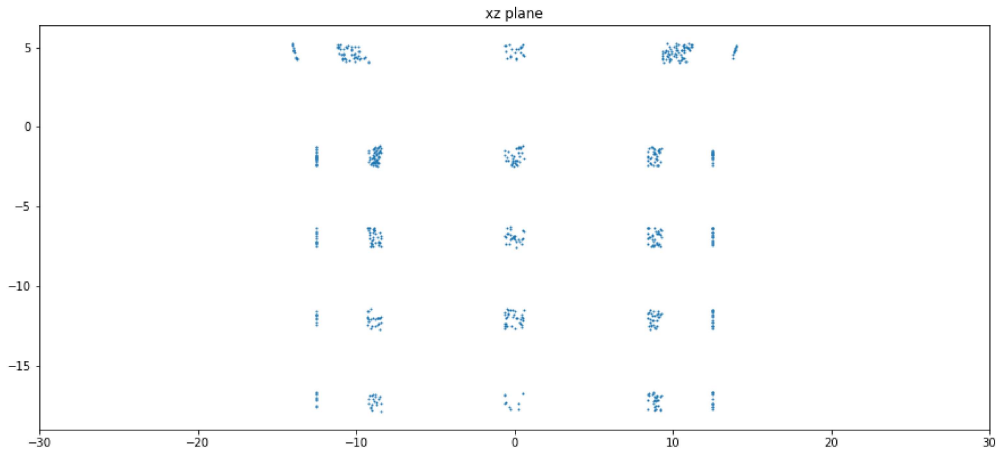


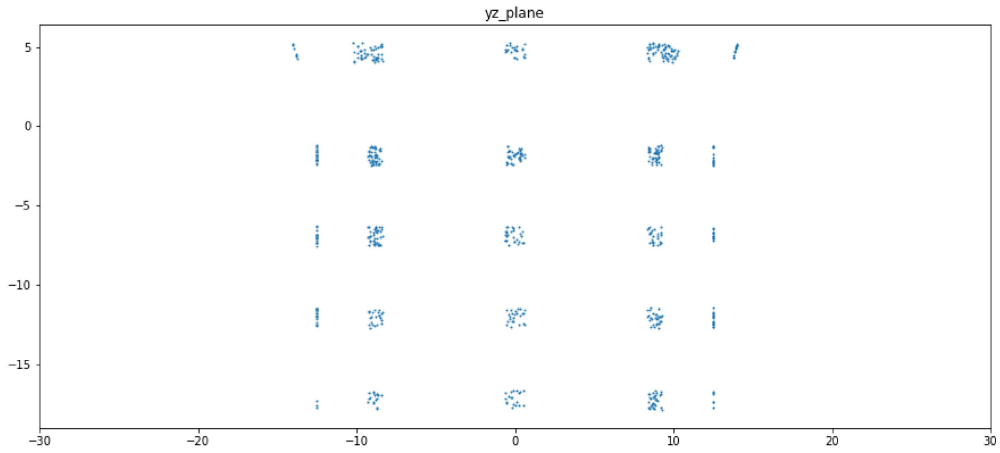


SiPMs

You can see their absence in the section above, but here is just the SiPMs.


xy plane
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30


xz plane
5
0
−5
−10
−15
−30
−20
−10
0
10
20
30



Some Notes on the Geometry

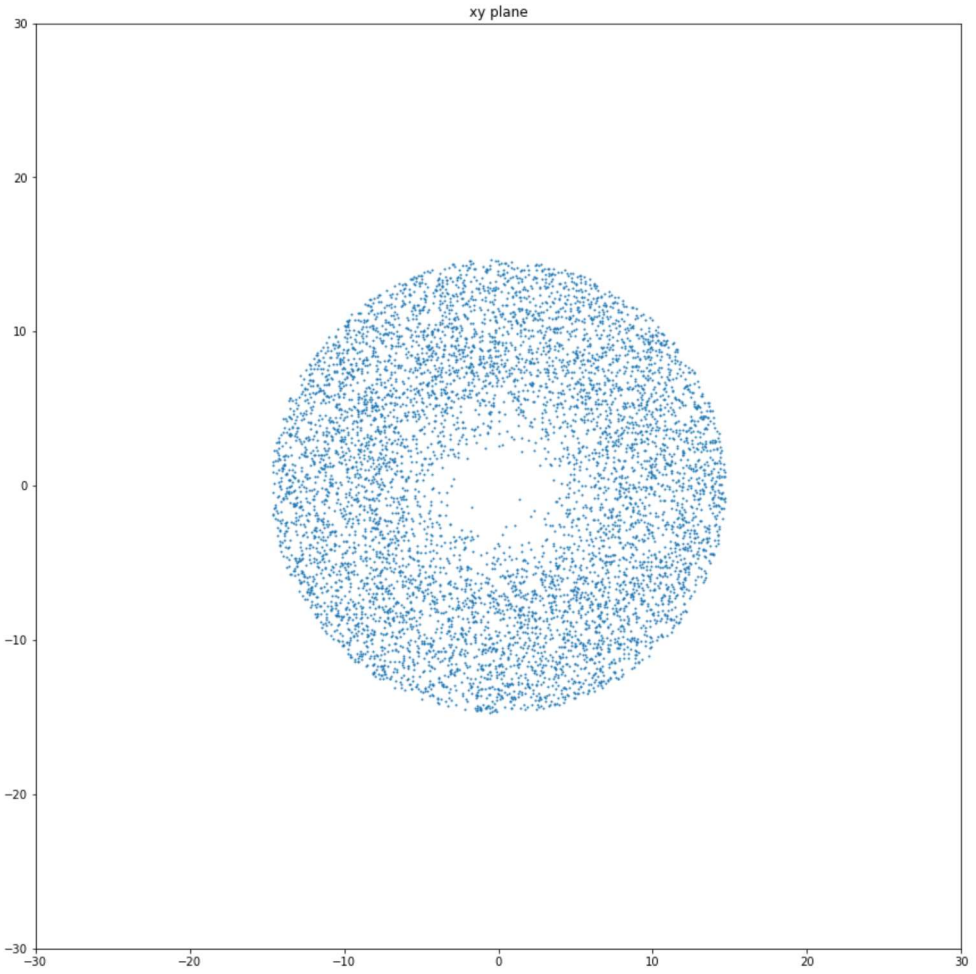
I did not play with n_outside and n_inside material much, so I'm not sure that those values properly reflect the actual geometry. I think all of them are set to hydraulic fluid, which might be accurate, but again, didn't play around with that much.

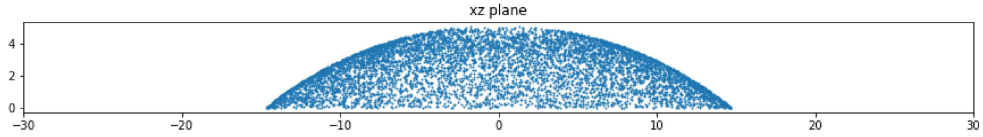
I also didn't adjust the absorption number much as well. I set it to 0 for glass and 1 for everything else which is likely not entirely accurate. Also, the inner jar should probably be set to absorption 1 since just outside of that jar is a teflon reflector.

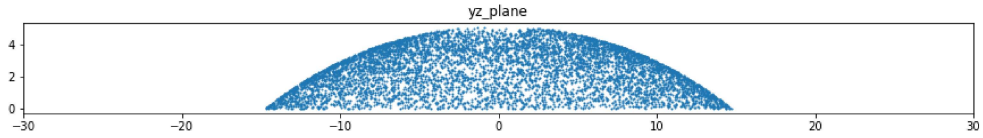
Troubleshooting

As I mentioned above, there is an issue with rays that intersect the outer surface of a shape. Let me give you two good examples:

These next few images are of the same shape, the outer jar's inner dome: As you can see, there is no problem with this shape, since all of the rays originate within the sphere, so that all rays will interact only with the inner surface.



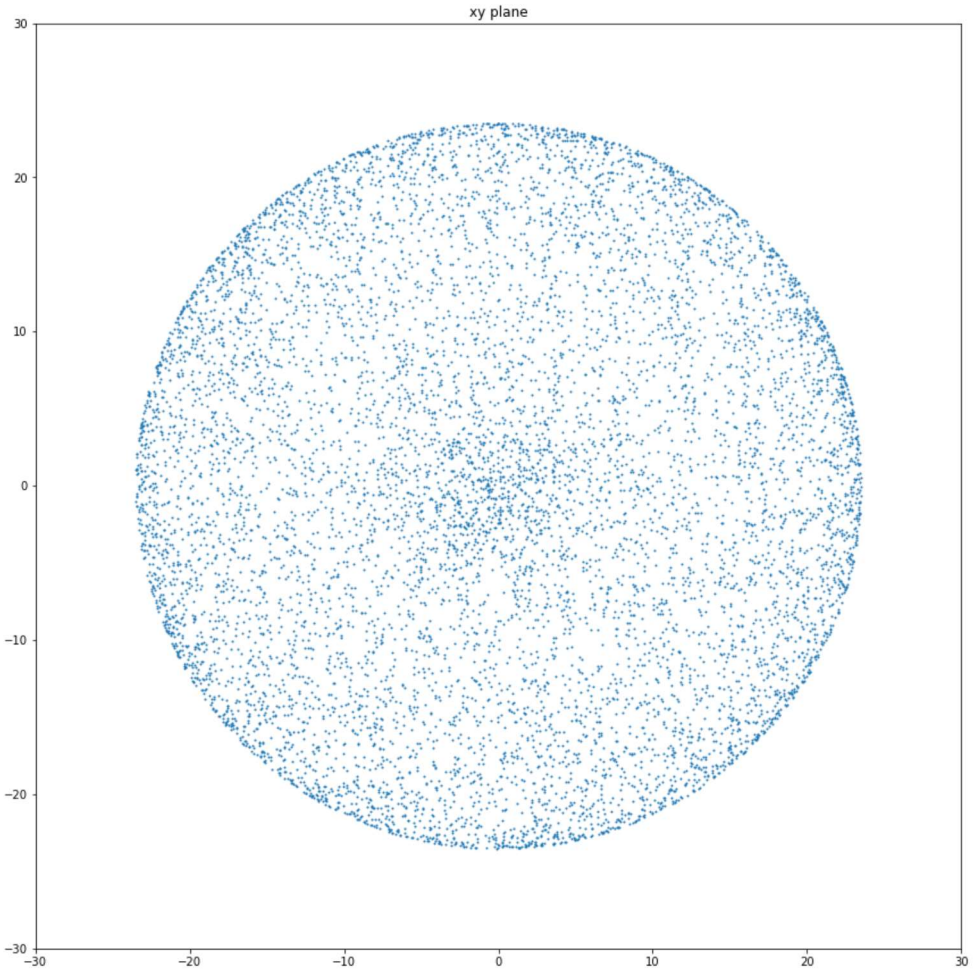


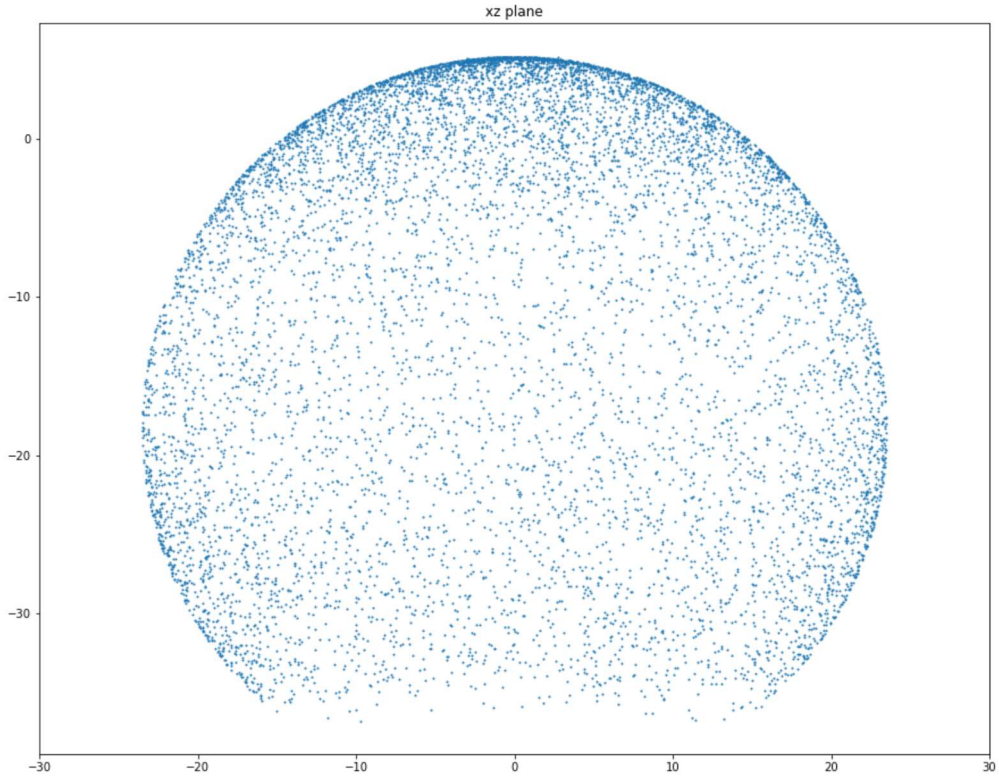


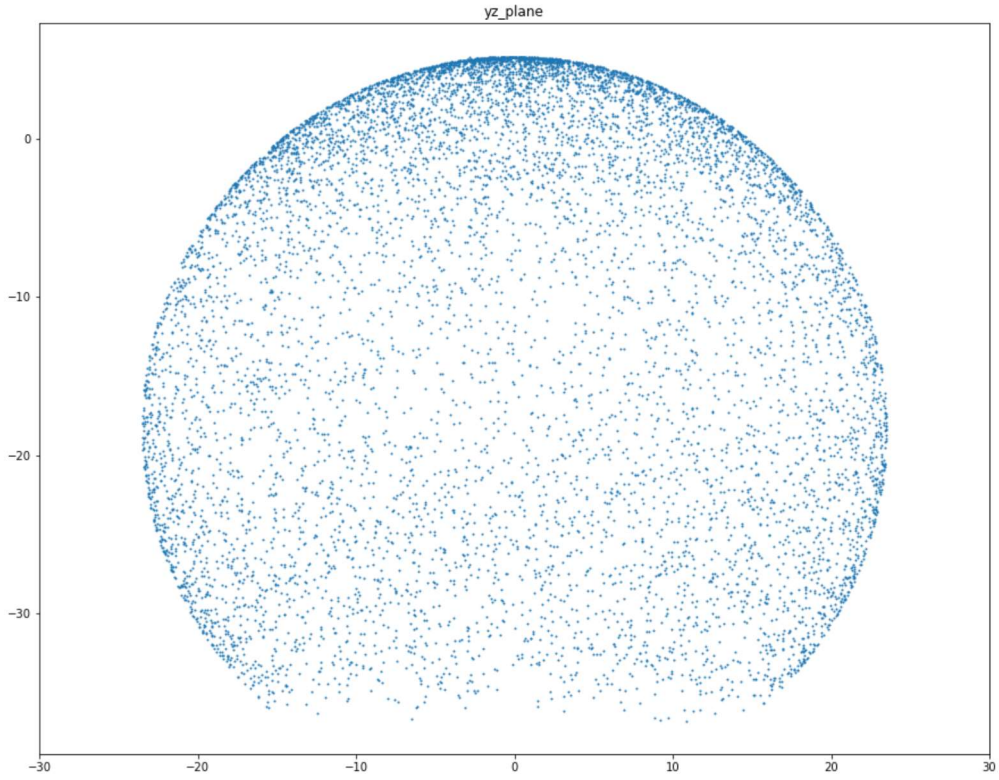
When it goes wrong:

But now let's see what happens when some of the rays are allowed to originate above the dome (and hit the outer surface):
It is as if the inbounds function is ignored when the rays hit the outer surface.
You also may be thinking: "who cares, no rays will originate from above the upper dome of that

jar". Well, the issue occurs with the dome of the inner jar as well, and in that case the rays will interact exclusively with the outer surface making that dome's surface extend much further out than it should, which is a problem. This is not just a problem with spheres either.



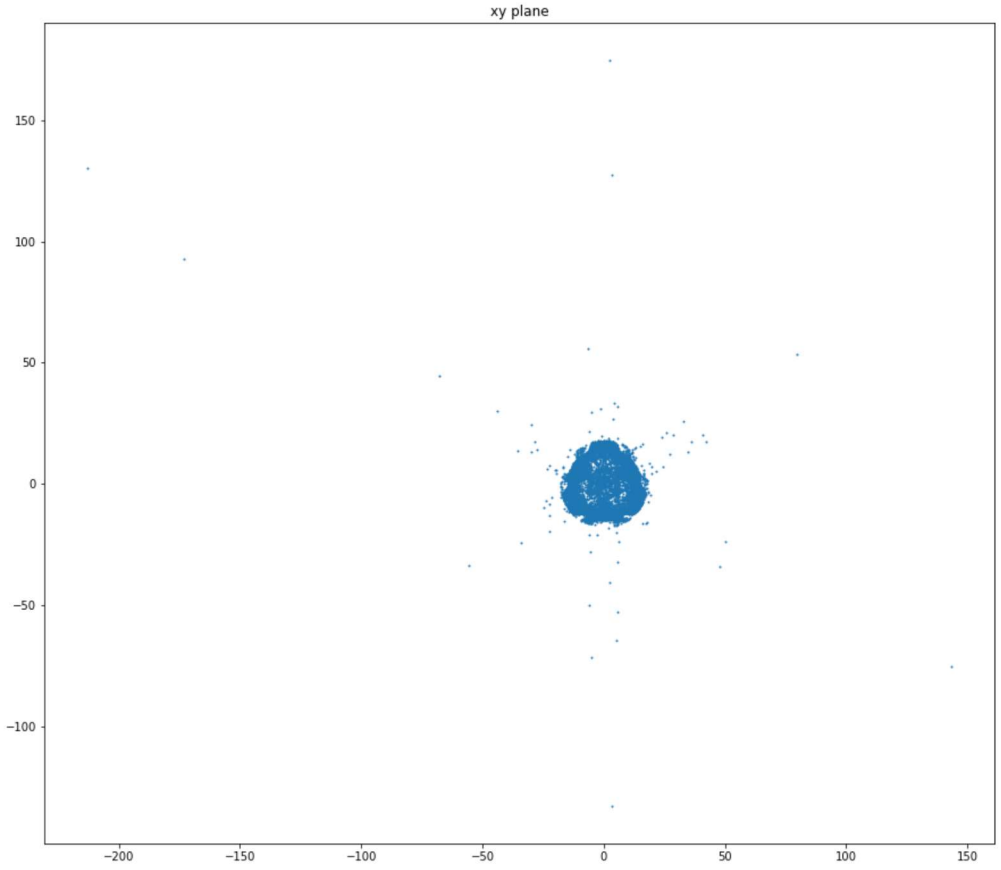

xz plane
0
−10
−20
−30
−30
−20
−10
0
10
20
30

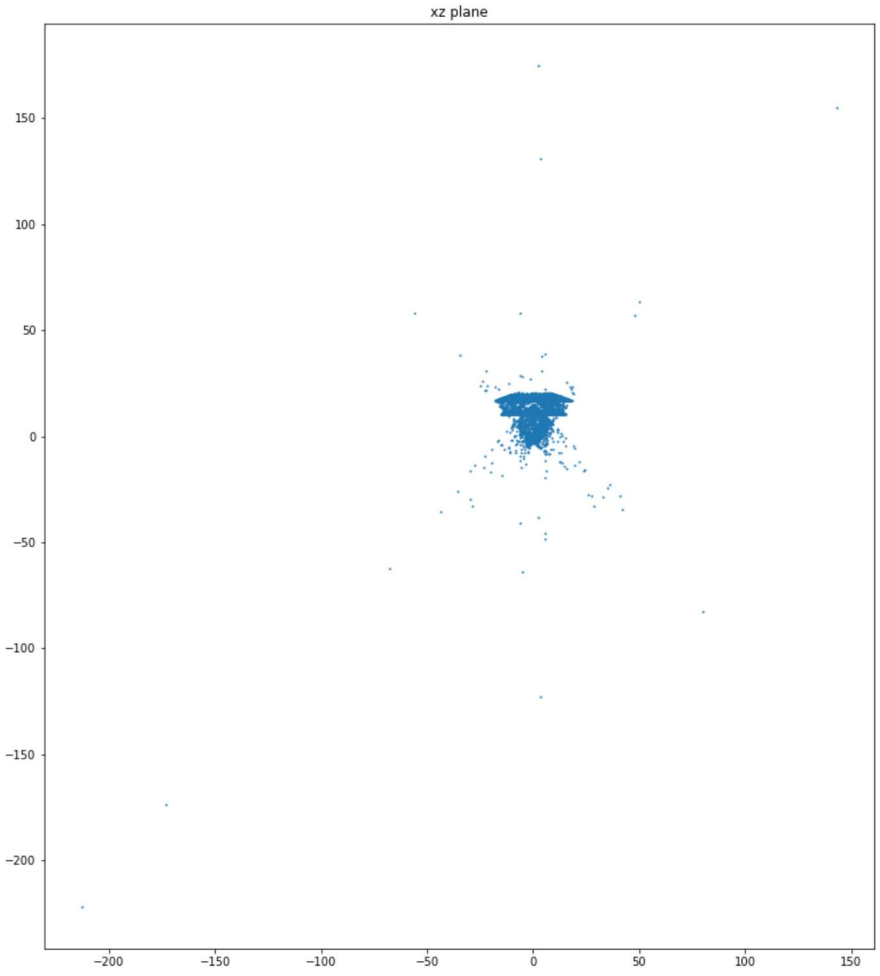


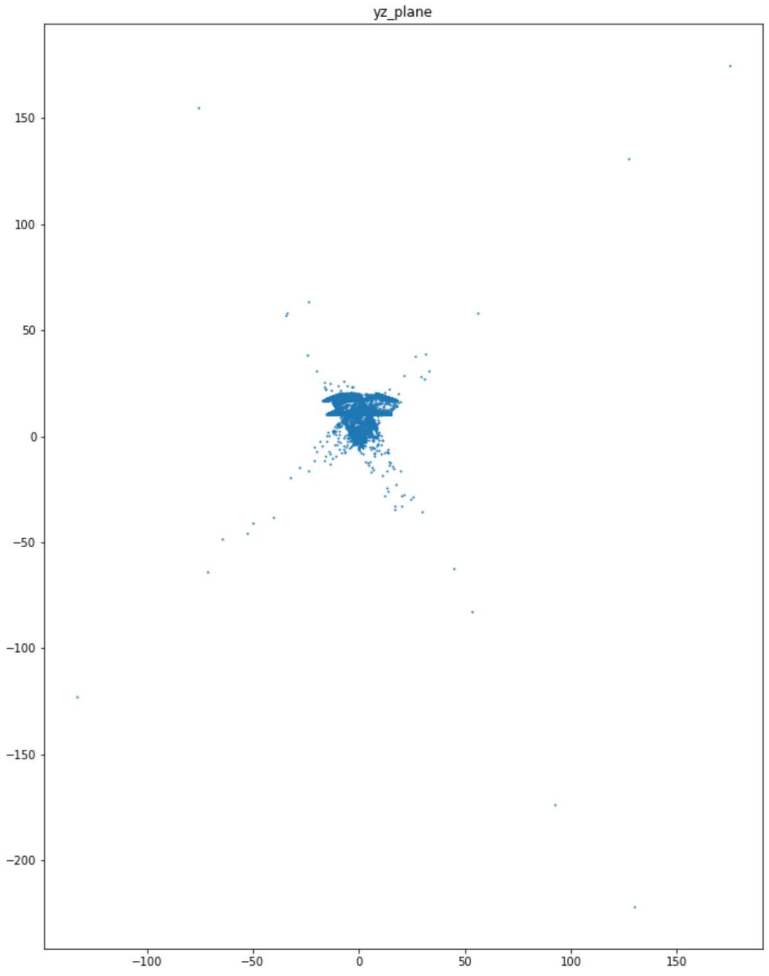
Second Example:

If you look back to the geometry plots for the camera housing (those tilted cylinders coming off of the top cone), I had to create that plot with very specific origin points for the sample rays. Let's see what happens when I allow the rays to have orgins outside of the cylinders that map the camera housings:

In this example you can see that it thinks there are valid intersections well outside of the inbounds function. It only seems to do this when the rays interact with the outer surface of a shape.


xy plane
150
100
50
0
−50
−100
−200
−150
−100
−50
0
50
100
150


xz plane
150
100
50
0
−50
−100
−150
−200
−200
−150
−100
−50
0
50
100
150



Conclusion on Troubleshooting

I wish I had a more helpful comment, but after combing through the ray tracer code, I'm not sure what would make the program map valid intersections well outside of the inbouds function, but only for outward pointing normal vectors (outer surfaces). If I had more time I'd investigate more, but unfortunately I do not. Hope this helps some. Also, I checked my plotting function, and since it's pretty simple and just calls the valid intersections from the ray tracer output, I don't think the issue is in the plotting function I've made. In fact, I double checked the raw numbers without using my plotting function at all, and the ray tracer output will include far larger values than it should for the x and y coordinates when it interacts with an outer surface.

CODE

Below you will see the code for the SBC geometry and the plotting function I used to visually display the surfaces.

In [1]:
```
import RayTracer2_Display
import numpy as np
#import SBCGeometry
import RayTracer2
import surface

%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as img
```

Step 1: Geometry

Surface Definition

In [2]:
```
# SBC Geometry from the module itself plus some added shapes

import numpy as np
import math
import random
import RayTracer2
import surface
import matplotlib.pyplot as plt

void = np.inf
n_target = 1.17 # n=1.224 for Ar @ 940nm, 90K
n_jar = 1.4512 # SiO2 @ 940nm
n_hydraulic = 1.22 # 1.21 @ CF4; 940nm, 146K ;  1.237 @ 7eV, 146K, another said 1
n_pressurewindow = 1.7569 # Al2O3 @ 940nm
n_pressurewall = void
n_air = 1.00



## Useful equations
# n0 = index of refraction at known density
# r = rho/rho0 (rho = density now, rho0 = density corresponding to n0)
# returns index of refraction at density rho
# clausius_mossotti = @(n0, r)(sqrt(((1 + 2*r).*n0.*n0 + 2 - 2*r)./((1 - r).*n0.*
clausius_mossotti = lambda n0, r: math.sqrt(((1 + 2*r) * n0**2 + 2 - 2*r) / ((1 -

## Dimensions in cm
ojar_thick = .5 # thickness of cylinder wall
ojar_cylrad = 12 # outer radius of cylinder
ojar_axrad = 24 # outer radius of sphere (along cylinder axis)
ojar_knucklerad = 4
ojar_cyllength = 30
ojar_elevation = 0

ijar_thick = .5 # thickness of cylinder wall
ijar_cylrad = 10.5 # outer radius of cylinder
ijar_axrad = 21 # outer radius of sphere (along cylinder axis)
ijar_knucklerad = 3
ijar_cyllength = 10.0
ijar_elevation = -2.54*7.74

z0 = 2.54*(20.73+7.74) # +distance from seal to z=0
vp_focuselev = 65.40 - z0
vp_focuslen = 29.23 #25;%33.5; % distance from convergence point to CF seal
vp_phi = np.pi/3

### the below variables (vp_{variable}) are for an outdated viewport ###
### Updated viewport variables are listed with the viewport surface code ###

vp_s = 10.0 # radial position of air-side center of viewport
vp_elev = 60.0 # vertical position of air-side center of viewport
vp_win_rad = 1.375*.5*2.54 # radius of glass (black on circumference)
vp_air_rad = 1.25*.5*2.54 # radius of air-side can (black on circumference)
vp_can_rad = 2.54
vp_can_wall = .0625*2.54
vp_flange_rad = .5*5.5*2.54
```

```
vp_nip_rad = 2.54*4.75*.5 # radius of hydraulic-side nipple (black on circumferer
vp_win_thick = .2*2.54 #%mpf is .23*2.54;
vp_nip_top = -.254*2.54 #% mpf is .08*2.54%
vp_theta = .3737 #20*pi/180;%21.8*pi/180;
vp_can_OAL = 6.43*2.54
vp_flange_thick = np.array([2.88, .69, .625, .625, .625])* 2.54


rdcone_gap = 1.5 # this is used to estimate the amount of light that may get thro
                 # a decent seperation between those general shapes.  This is a
                  # much of an effect it has.
rd_rad = 12.5 # reflector - diffuser radius
rd_top = 0
rd_bot = -25
rdcone_top = 8.96
rdcone_toprad = 15.13
rdtopcone_rad = rdcone_toprad
rdtopcone_bot = rdcone_top + rdcone_gap
rdtopcone_apex = rdtopcone_bot+6.5
rdbotcone_apex = -15.2-10
rdbotcone_rad = 10.5
rdbotcone_bot = -20.0-10


pv_top = vp_focuselev + vp_focuslen*np.cos(vp_theta)-2.54*(5.49-1.5)
pv_bot = pv_top - 2.54*31.17
pv_rad = 8*2.54
pv_thick = .375*2.54
pv_axrad = (3.07)*2.54

## Derived dimensions
t_o = np.array([0, ojar_thick])
t_i = np.array([0, ijar_thick])

r1 = np.concatenate((ojar_cylrad - t_o, ijar_cylrad - t_i))
r2 = np.concatenate((ojar_knucklerad - t_o, ijar_knucklerad - t_i))
r3 = np.concatenate((ojar_axrad - t_o, ijar_axrad - t_i))

s = r3 * (r1-r2) / (r3-r2) # axis to knuckle-dome transition

z = r2 * np.sqrt(1 - (s / r3) ** 2) #  equator to knuckle-dome transition

d = r3 * z * ((1 / r3) - (1 / r2)) # equator to dome sphere center

vp_axis = np.array([0, -math.sin(vp_theta), math.cos(vp_theta)])
vp_center = np.array([0, -vp_s, vp_elev])


head_out_Q = np.vstack(([pv_rad ** (-2), 0, 0], [0, pv_rad ** (-2), 0],  [0, 0, p
head_in_Q = np.vstack(([pv_rad-pv_thick ** (-2), 0, 0], [0, pv_rad-pv_thick ** (-
head_out_P = np.array([0, 0, ((-2)*pv_top) * (pv_axrad ** (-2))]) # .* is element
head_in_P = np.array([0, 0, ((-2)*pv_top) * (pv_axrad-pv_thick ** (-2))])
head_out_R = (pv_top / pv_axrad)**2 - 1
head_in_R = (pv_top / (pv_axrad - pv_thick))**2 - 1

rd_cone_b = (rdcone_toprad - rd_rad) / (rdcone_top - rd_top)
rd_cone_z0 = rd_top - (rd_rad / rd_cone_b)
```

```
rd_cone_Q = np.vstack(([1, 0, 0], [0, 1, 0], [0, 0, -rd_cone_b**2])) # - (rd_cone
rd_cone_P = np.array([0, 0, 2* rd_cone_b**2 * rd_cone_z0])
rd_cone_R = -(rd_cone_b * rd_cone_z0)**2

rd_topcone_b = rdtopcone_rad / (rdtopcone_apex - rdtopcone_bot)
rd_topcone_Q = np.vstack(([1, 0, 0], [0, 1, 0], [0, 0, -rd_topcone_b**2]))
rd_topcone_P = np.array([0, 0, 2 * rd_topcone_b**2 * rdtopcone_apex])
rd_topcone_R = -(rd_topcone_b * rdtopcone_apex)**2

rd_botcone_b = rdbotcone_rad / (rdbotcone_apex - rdbotcone_bot)
rd_botcone_Q = np.vstack(([1, 0, 0], [0, 1, 0], [0, 0, -rd_botcone_b**2]))
rd_botcone_P = np.array([0, 0, 2 * rd_botcone_b**2 * rdbotcone_apex])
rd_botcone_R = -(rd_botcone_b * rdbotcone_apex)**2

## The surfaces themselves

## Useful equations
# n0 = index of refraction at known density
# r = rho/rho0 (rho = density now, rho0 = density corresponding to n0)
# returns index of refraction at density rho
# clausius_mossotti = @(n0, r)(sqrt(((1 + 2*r).*n0.*n0 + 2 - 2*r).//((1 - r).*n0.*
clausius_mossotti = lambda n0, r: math.sqrt(((1 + 2*r) * n0**2 + 2 - 2*r) / ((1 -

surface_list = []

## Inner Jar
#inner_jar_density = -2.203

inner_jar_inner_cyl = surface.surface()
inner_jar_inner_cyl.description = 'inside surface of inner quartz jar cylinder'
inner_jar_inner_cyl.shape = 'cylinder'
inner_jar_inner_cyl.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), r1[3]
inner_jar_inner_cyl.inbounds_function = lambda p: np.reshape((p[:, 2, :] >= (ijar
inner_jar_inner_cyl.n_outside = n_jar
inner_jar_inner_cyl.n_inside = n_hydraulic
inner_jar_inner_cyl.surface_type = 'normal'
inner_jar_inner_cyl.absorption = 0
surface_list.append(inner_jar_inner_cyl)

inner_jar_outer_cyl = surface.surface()
inner_jar_outer_cyl.description = 'outside surface of inner jar cylinder'
inner_jar_outer_cyl.shape = 'cylinder'
inner_jar_outer_cyl.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), r1[2]
inner_jar_outer_cyl.inbounds_function = lambda p: np.reshape((p[:,2,:] >= (ijar_e
inner_jar_outer_cyl.n_outside = n_target
inner_jar_outer_cyl.n_inside = n_jar
inner_jar_outer_cyl.surface_type = 'normal'
inner_jar_outer_cyl.absorption = 0
surface_list.append(inner_jar_outer_cyl)

# will have to split up above into different sections with insulation rather thar

inner_jar_inner_dome = surface.surface()
inner_jar_inner_dome.description = 'inside surface of inner jar dome'
inner_jar_inner_dome.shape = 'sphere'
inner_jar_inner_dome.param_list = [np.array([0, 0, ijar_elevation + d[3]]), r3[3]
inner_jar_inner_dome.inbounds_function = lambda p: np.reshape((p[:,2,:] > (z[3] +
```

```
inner_jar_inner_dome.n_outside = n_jar
inner_jar_inner_dome.n_inside = n_hydraulic
inner_jar_inner_dome.surface_type = 'normal'
inner_jar_inner_dome.absorption = 0
surface_list.append(inner_jar_inner_dome)

inner_jar_outer_dome = surface.surface()
inner_jar_outer_dome.description = 'outside surface of inner jar dome'
inner_jar_outer_dome.shape = 'sphere'
inner_jar_outer_dome.param_list = [np.array([0, 0, ijar_elevation + d[2]]), r3[2]
inner_jar_outer_dome.inbounds_function = lambda p: np.reshape((p[:,2,:] > (z[2] +
inner_jar_outer_dome.n_outside = n_target
inner_jar_outer_dome.n_inside = n_jar
inner_jar_outer_dome.surface_type = 'normal'
inner_jar_outer_dome.absorption = 0
surface_list.append(inner_jar_outer_dome)


## Outer Jar
outer_jar_inner_cyl = surface.surface()
outer_jar_inner_cyl.description = 'inner surface of outer jar cylinder'
outer_jar_inner_cyl.shape = 'cylinder'
outer_jar_inner_cyl.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), r1[1]
outer_jar_inner_cyl.inbounds_function = lambda p: np.reshape((p[:, 2, :] >= (ojar
outer_jar_inner_cyl.n_outside = n_jar
outer_jar_inner_cyl.n_inside = n_target
outer_jar_inner_cyl.surface_type = 'normal'
outer_jar_inner_cyl.absorption = 0
surface_list.append(outer_jar_inner_cyl)

outer_jar_outer_cyl = surface.surface()
outer_jar_outer_cyl.description = 'outside surface of outer jar cylinder'
outer_jar_outer_cyl.shape = 'cylinder'
outer_jar_outer_cyl.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), r1[0]
outer_jar_outer_cyl.inbounds_function = lambda p: np.reshape((p[:, 2, :] >= (ojar
outer_jar_outer_cyl.n_outside = n_hydraulic
outer_jar_outer_cyl.n_inside = n_jar
outer_jar_outer_cyl.surface_type = 'normal'
outer_jar_outer_cyl.absorption = 0
surface_list.append(outer_jar_outer_cyl)

outer_jar_inner_dome = surface.surface()
outer_jar_inner_dome.description = 'inner surface of outer jar dome'
outer_jar_inner_dome.shape = 'sphere'
outer_jar_inner_dome.param_list = [np.array([0, 0, ojar_elevation + d[1]]), r3[1]
outer_jar_inner_dome.inbounds_function = lambda p: np.reshape((p[:,2,:] > 0), (p.
outer_jar_inner_dome.n_outside = n_jar
outer_jar_inner_dome.n_inside = n_target
outer_jar_inner_dome.surface_type = 'normal'
outer_jar_inner_dome.absorption = 0
surface_list.append(outer_jar_inner_dome)

outer_jar_outer_dome = surface.surface()
outer_jar_outer_dome.description = 'outer surface of outer jar domee'
outer_jar_outer_dome.shape = 'sphere'
outer_jar_outer_dome.param_list = [np.array([0, 0, ojar_elevation + d[0]]), r3[1]
outer_jar_outer_dome.inbounds_function = lambda p: np.reshape((p[:,2,:] > (z[0] +
```

```
outer_jar_outer_dome.n_outside = n_hydraulic
outer_jar_outer_dome.n_inside = n_jar
outer_jar_outer_dome.surface_type = 'normal'
outer_jar_outer_dome.absorption = 0
surface_list.append(outer_jar_outer_dome)

## Knuckles
iiknuckle = surface.surface()
iiknuckle.description = 'inner surface of inner knuckle'
iiknuckle.shape = 'torus'
iiknuckle.param_list = [np.array([0, 0, ijar_elevation]), np.array([0, 0, 1]), r1
iiknuckle.inbounds_function = lambda p: np.reshape((p[:,2,:] > ijar_elevation) *
iiknuckle.n_outside = n_jar
iiknuckle.n_inside = n_hydraulic
iiknuckle.surface_type = 'normal'
iiknuckle.absorption = 0
surface_list.append(iiknuckle)

oiknuckle = surface.surface()
oiknuckle.description = 'outer surface of inner knuckle'
oiknuckle.shape = 'torus'
oiknuckle.param_list = [np.array([0, 0, ijar_elevation]), np.array([0, 0, 1]), r1
oiknuckle.inbounds_function = lambda p: np.reshape((p[:,2,:] > ijar_elevation) *
                                                   * ((p[:,0,:]**2 + p[:,1,:]**2)
oiknuckle.n_outside = n_hydraulic
oiknuckle.n_inside = n_jar
oiknuckle.surface_type = 'normal'
oiknuckle.absorption = 0
surface_list.append(oiknuckle)

ioknuckle = surface.surface()
ioknuckle.description = 'inner surface of outer knuckle'
ioknuckle.shape = 'torus'
ioknuckle.param_list = [np.array([0, 0, ojar_elevation]), np.array([0, 0, 1]), r1
ioknuckle.inbounds_function = lambda p: np.reshape((p[:,2,:] > ojar_elevation) *
                                                   * ((p[:,0,:]**2 + p[:,1,:]**2)
ioknuckle.n_outside = n_jar
ioknuckle.n_inside = n_target
ioknuckle.surface_type = 'normal'
ioknuckle.absorption = 0
surface_list.append(ioknuckle)

ooknuckle = surface.surface()
ooknuckle.description = 'outer surface of outer knuckle'
ooknuckle.shape = 'torus'
ooknuckle.param_list = [np.array([0, 0, ojar_elevation]), np.array([0, 0, 1]), r1
ooknuckle.inbounds_function = lambda p: np.reshape((p[:,2,:] > ojar_elevation) *
                                                   * ((p[:,0,:]**2 + p[:,1,:]**2)
ooknuckle.n_outside = n_hydraulic
ooknuckle.n_inside = n_jar
ooknuckle.surface_type = 'normal'
ooknuckle.absorption = 0
surface_list.append(ooknuckle)

## other black surfaces to trap rays
rd = surface.surface()
rd.description = 'reflector/diffuser'
```

```
rd.shape = 'cylinder'
rd.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
rd.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_bot) * (p[:, 2, :] <
& ~(SiPM_1.inbounds_function(p))& ~(SiPM_2.inbounds_function(p))& ~(SiPM_3.inbour
& ~(SiPM_4.inbounds_function(p))& ~(SiPM_5.inbounds_function(p))& ~(SiPM_6.inbour
& ~(SiPM_7.inbounds_function(p))& ~(SiPM_8.inbounds_function(p))& ~(SiPM_9.inbour
& ~(SiPM_10.inbounds_function(p))& ~(SiPM_11.inbounds_function(p))& ~(SiPM_12.inb
& ~(SiPM_13.inbounds_function(p))& ~(SiPM_14.inbounds_function(p))& ~(SiPM_15.inb
& ~(SiPM_16.inbounds_function(p))& ~(SiPM_17.inbounds_function(p))& ~(SiPM_18.inb
& ~(SiPM_19.inbounds_function(p))& ~(SiPM_20.inbounds_function(p))& ~(SiPM_21.inb
& ~(SiPM_22.inbounds_function(p))& ~(SiPM_23.inbounds_function(p))& ~(SiPM_24.inb
& ~(SiPM_25.inbounds_function(p))& ~(SiPM_26.inbounds_function(p))& ~(SiPM_27.inb
& ~(SiPM_28.inbounds_function(p))& ~(SiPM_29.inbounds_function(p))& ~(SiPM_30.inb
& ~(SiPM_31.inbounds_function(p))& ~(SiPM_32.inbounds_function(p))
rd.n_outside = n_hydraulic
rd.n_inside = n_hydraulic
rd.surface_type = 'normal'
rd.absorption = 1
surface_list.append(rd)

rd_cone = surface.surface()
rd_cone.description = 'reflector/diffuser cone'
rd_cone.shape = 'quadsurface'                          # QuadSurface needs to be tes
rd_cone.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
rd_cone.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
& ~(SiPM_33.inbounds_function(p))& ~(SiPM_34.inbounds_function(p))& ~(SiPM_35.inb
& ~(SiPM_36.inbounds_function(p))& ~(SiPM_37.inbounds_function(p))& ~(SiPM_38.inb
& ~(SiPM_39.inbounds_function(p))& ~(SiPM_40.inbounds_function(p))
rd_cone.n_outside = n_hydraulic
rd_cone.n_inside = n_hydraulic
rd_cone.surface_type = 'normal'
rd_cone.absorption = 1
surface_list.append(rd_cone)

rd_topcone = surface.surface()
rd_topcone.description = 'reflector/diffuser topcone'
rd_topcone.shape = 'quadsurface'                          # QuadSurface needs to be
rd_topcone.param_list = [rd_topcone_Q, rd_topcone_P, rd_topcone_R]
rd_topcone.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rdtopcone_bot)
& ~(viewport_1.inbounds_function(p)) & ~(viewport_2.inbounds_function(p))\
& ~(viewport_3.inbounds_function(p)) & ~(viewport_flange_1.inbounds_function(p))\
& ~(viewport_flange_2.inbounds_function(p)) & ~(viewport_flange_3.inbounds_functi
rd_topcone.n_outside = n_hydraulic
rd_topcone.n_inside = n_hydraulic
rd_topcone.surface_type = 'normal'
rd_topcone.absorption = 1
surface_list.append(rd_topcone)

### The viewports for the cameras ###
## Both the viewports' and their flanges are just inbounds functions that pick ou
#useful viewport variables:
vp_radius = 6.25                # viewport radius
vp_flange_radius = 7            # viewport flange radius (annulus bordering the v
topcone_slope = (rdtopcone_apex - rdtopcone_bot)/(rdcone_toprad)   # the slope of
topcone_theta = np.arctan(topcone_slope) #the angle made between the xy plane and

viewport_1 = surface.surface()
```

```
viewport_1.description = 'camera viewport on top cone on +y side'
viewport_1.shape = 'quadsurface'                        # QuadSurface needs to be
viewport_1.param_list = [rd_topcone_Q, rd_topcone_P, rd_topcone_R]
viewport_1.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rdtopcone_bot)
& ((p[:,0,:]**2+((p[:,1,:]-0.51*rdcone_toprad)/np.cos(topcone_theta))**2)<vp_radi
viewport_1.n_outside = n_hydraulic
viewport_1.n_inside = n_hydraulic
viewport_1.surface_type = 'normal'
viewport_1.absorption = 0
surface_list.append(viewport_1)

viewport_2 = surface.surface()
viewport_2.description = 'camera viewport on top cone on -y, +x side'
viewport_2.shape = 'quadsurface'                        # QuadSurface needs to be
viewport_2.param_list = [rd_topcone_Q, rd_topcone_P, rd_topcone_R]
viewport_2.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rdtopcone_bot)
& ((((p[:,0,:]-0.51*rdcone_toprad*np.cos(7*np.pi/6))*np.cos(4*np.pi/6)+(p[:,1,:]-
*np.sin(4*np.pi/6))**2+((((p[:,0,:]-0.51*rdcone_toprad*np.cos(7*np.pi/6))*np.sin(
*np.sin(7*np.pi/6))*np.cos(4*np.pi/6))/np.cos(topcone_theta))**2)<vp_radius**2),(
viewport_2.n_outside = n_hydraulic
viewport_2.n_inside = n_hydraulic
viewport_2.surface_type = 'normal'
viewport_2.absorption = 0
surface_list.append(viewport_2)

viewport_3 = surface.surface()
viewport_3.description = 'camera viewport on top cone on -y, -x side'
viewport_3.shape = 'quadsurface'                        # QuadSurface needs to be
viewport_3.param_list = [rd_topcone_Q, rd_topcone_P, rd_topcone_R]
viewport_3.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rdtopcone_bot)
& ((((p[:,0,:]-0.51*rdcone_toprad*np.cos(-np.pi/6))*np.cos(-4*np.pi/6)+(p[:,1,:]-
*np.sin(-4*np.pi/6))**2+((((p[:,0,:]-0.51*rdcone_toprad*np.cos(-np.pi/6))*np.sin(
*np.sin(-np.pi/6))*np.cos(-4*np.pi/6))/np.cos(topcone_theta))**2)<vp_radius**2),(
viewport_3.n_outside = n_hydraulic
viewport_3.n_inside = n_hydraulic
viewport_3.surface_type = 'normal'
viewport_3.absorption = 0
surface_list.append(viewport_2)

#viewport flanges

#flange 1
viewport_flange_1 = surface.surface()
viewport_flange_1.description = 'flange for viewport 1 on top cone on +y side'
viewport_flange_1.shape = 'quadsurface'                        # QuadSurface needs
viewport_flange_1.param_list = [rd_topcone_Q, rd_topcone_P, rd_topcone_R]
viewport_flange_1.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rdtopcon
& ((p[:,0,:]**2+((p[:,1,:]-0.51*rdcone_toprad)/np.cos(topcone_theta))**2)<vp_flan
& ~(viewport_1.inbounds_function(p)),(p.shape[0], -1))
viewport_flange_1.n_outside = n_hydraulic
viewport_flange_1.n_inside = n_hydraulic
viewport_flange_1.surface_type = 'normal'
viewport_flange_1.absorption = 1
surface_list.append(viewport_flange_1)

#flange 2
viewport_flange_2 = surface.surface()
```

```
viewport_flange_2.description = 'flange for viewport 2 on top cone on -y, +x side
viewport_flange_2.shape = 'quadsurface'                          # QuadSurface needs
viewport_flange_2.param_list = [rd_topcone_Q, rd_topcone_P, rd_topcone_R]
viewport_flange_2.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rdtopcor
& ((((p[:,0,:]-0.51*rdcone_toprad*np.cos(7*np.pi/6))*np.cos(4*np.pi/6)+(p[:,1,:]-
*np.sin(4*np.pi/6))**2+((((p[:,0,:]-0.51*rdcone_toprad*np.cos(7*np.pi/6))*np.sin(
*np.sin(7*np.pi/6))*np.cos(4*np.pi/6))/np.cos(topcone_theta))**2)<vp_flange_radiu
& ~(viewport_2.inbounds_function(p)),(p.shape[0], -1))
viewport_flange_2.n_outside = n_hydraulic
viewport_flange_2.n_inside = n_hydraulic
viewport_flange_2.surface_type = 'normal'
viewport_flange_2.absorption = 1
surface_list.append(viewport_flange_2)

#flange 3
viewport_flange_3 = surface.surface()
viewport_flange_3.description = 'flange for viewport 3 on top cone on -y, -x side
viewport_flange_3.shape = 'quadsurface'                          # QuadSurface needs
viewport_flange_3.param_list = [rd_topcone_Q, rd_topcone_P, rd_topcone_R]
viewport_flange_3.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rdtopcor
& ((((p[:,0,:]-0.51*rdcone_toprad*np.cos(-np.pi/6))*np.cos(-4*np.pi/6)+(p[:,1,:]-
*np.sin(-4*np.pi/6))**2+((((p[:,0,:]-0.51*rdcone_toprad*np.cos(-np.pi/6))*np.sin(
*np.sin(-np.pi/6))*np.cos(-4*np.pi/6))/np.cos(topcone_theta))**2)<vp_flange_radiu
& ~(viewport_3.inbounds_function(p)),(p.shape[0], -1))
viewport_flange_3.n_outside = n_hydraulic
viewport_flange_3.n_inside = n_hydraulic
viewport_flange_3.surface_type = 'normal'
viewport_flange_3.absorption = 1
surface_list.append(viewport_flange_3)

#camera cans
# useful variables:

#this is the z value for all of the center points of the viewports.  This allows
zpoint = -topcone_slope*0.51*rdcone_toprad+rdtopcone_apex

camcan_h = 7  #height of the camera cans [cm]

# this is the upper z point if you follow the cylinder axis to the top end of the
zpoint_upper = -topcone_slope*0.51*rdcone_toprad+rdtopcone_apex+(camcan_h*np.cos(

camcan_1 = surface.surface()
camcan_1.description = 'can leading to camera on the +y side of the detector'
camcan_1.shape = 'cylinder'
camcan_1.param_list = [np.array([0, 0.51*rdcone_toprad, zpoint]), np.array([0, 1,
camcan_1.inbounds_function = lambda p: np.reshape((p[:, 2, :] >= (p[:,1,:]-0.51*r
& (p[:, 2, :] <= (p[:,1,:]-(0.51*rdcone_toprad+camcan_h*np.sin(topcone_theta)))*-
camcan_1.n_outside = n_hydraulic
camcan_1.n_inside = n_hydraulic
camcan_1.surface_type = 'normal'
camcan_1.absorption = 1
surface_list.append(camcan_1)

camcan_2 = surface.surface()
camcan_2.description = 'can leading to camera on the -x, -y side of the detector'
camcan_2.shape = 'cylinder'
camcan_2.param_list = [np.array([0.51*rdcone_toprad*np.cos(7*np.pi/6), 0.51*rdcor
```

```
                        np.array([np.cos(7*np.pi/6), np.sin(7*np.pi/6), np.cos(top
camcan_2.inbounds_function = lambda p: np.reshape((p[:, 2, :] >= (p[:,0,:]-0.51*r
*topcone_slope+zpoint) & (p[:, 2, :] <= (p[:,0,:]-(0.51*rdcone_toprad*np.cos(7*np
*topcone_slope+zpoint_upper), (p.shape[0], -1))
camcan_2.n_outside = n_hydraulic
camcan_2.n_inside = n_hydraulic
camcan_2.surface_type = 'normal'
camcan_2.absorption = 1
surface_list.append(camcan_2)

camcan_3 = surface.surface()
camcan_3.description = 'can leading to camera on the -x, -y side of the detector'
camcan_3.shape = 'cylinder'
camcan_3.param_list = [np.array([0.51*rdcone_toprad*np.cos(-np.pi/6), 0.51*rdcone
                        np.array([np.cos(-np.pi/6), np.sin(-np.pi/6), np.cos(topco
camcan_3.inbounds_function = lambda p: np.reshape((p[:, 2, :] >= (p[:,0,:]-0.51*r
*-topcone_slope+zpoint) & (p[:, 2, :] <= (p[:,0,:]-(0.51*rdcone_toprad*np.cos(-np
*-topcone_slope+zpoint_upper), (p.shape[0], -1))
camcan_3.n_outside = n_hydraulic
camcan_3.n_inside = n_hydraulic
camcan_3.surface_type = 'normal'
camcan_3.absorption = 1
surface_list.append(camcan_3)

# upper most cone (only using the part of the cone that caps off the camera cans)
#useful variables for the camera end caps:
camcancaps_rad = rdtopcone_rad+(camcan_h*np.sin(topcone_theta))
camcancaps_apex = rdtopcone_apex + (camcan_h*np.cos(topcone_theta))
cancamcaps_bot = camcancaps_apex - (rdtopcone_apex-rdtopcone_bot)

camcancaps_b = camcancaps_rad / (camcancaps_apex - cancamcaps_bot)
camcancaps_Q = np.vstack(([1, 0, 0], [0, 1, 0], [0, 0, -camcancaps_b**2]))
camcancaps_P = np.array([0, 0, 2 * camcancaps_b**2 * camcancaps_apex])
camcancaps_R = -(camcancaps_b * camcancaps_apex)**2

camcancaps = surface.surface()
camcancaps.description = 'end caps for the camera cans'
camcancaps.shape = 'quadsurface'                        # QuadSurface needs to be
camcancaps.param_list = [camcancaps_Q, camcancaps_P, camcancaps_R]
camcancaps.inbounds_function = lambda p: np.reshape((p[:, 2, :] > cancamcaps_bot)
                                , (p.shape[0], -1))
camcancaps.n_outside = n_hydraulic
camcancaps.n_inside = n_hydraulic
camcancaps.surface_type = 'normal'
camcancaps.absorption = 1
surface_list.append(camcancaps)


#SiPMs!!
# Cuts of the cylinder that represent the SiPMs

# Important NOTE: in SBCGeometry: rd_cone is the cone extending outward that has
# add these variables to the top of SBC Geometry:
# Assumes an octagonal setup of the SiPMs (unlikely to change that part of the SB

SiPM_d = 1.269 #dimensions of the square SiPM surface
```

```
SiPM_r1 = [-17.88,-16.61]  #SiPMs in the first (bottom most) row's z: bottom z at
SiPM_r2 = [-12.71,-11.44]  #SiPMs in the second row's z: bottom z at 0 index, top
SiPM_r3 = [-7.54,-6.27]  #SiPMs in the third row's z: bottom z at 0 index, top va
SiPM_r4 = [-2.47,-1.2]  #SiPMs in the fourth row's z: bottom z at 0 index, top va
#SiPM_r5 = [b,t]  #SiPMs in the fifth row's (on the rd_cone) z: bottom z at 0 ind

# The convention for SiPM columns is that right is towards positive x and left is
### NOTE: columns 1 and 5 use y values, not x. Makes it easier to calculate ###

# column 1 values use y bound
SiPM_c1y = [-1/2*SiPM_d,1/2*SiPM_d] #SiPMs in first column (right most) x: left x
#SiPMs in second column +x angled side x: left x at 0 index, right x at 1 index
SiPM_c2 = [(1/2*(2*rd_rad/(1+np.sqrt(2)))+2*rd_rad/(2*np.sqrt(2)*(1+np.sqrt(2))))
          ,(1/2*(2*rd_rad/(1+np.sqrt(2)))+2*rd_rad/(2*np.sqrt(2)*(1+np.sqrt(2))))
SiPM_c3 = [-1/2*SiPM_d,1/2*SiPM_d] #SiPMs in third column (right most) x: left x
 #SiPMs in fourth column -x angled side x: left x at 0 index, right x at 1 index
SiPM_c4 = [-(1/2*(2*rd_rad/(1+np.sqrt(2)))+2*rd_rad/(2*np.sqrt(2)*(1+np.sqrt(2))))
          ,-(1/2*(2*rd_rad/(1+np.sqrt(2)))+2*rd_rad/(2*np.sqrt(2)*(1+np.sqrt(2))))
# column 5 values use y bounds
SiPM_c5y = [-1/2*SiPM_d,1/2*SiPM_d]  #SiPMs in fifth column (left most) x: left x
# heavily based on rd which is the surface they are mounted on.

## Numbering convention for SiPMs:  Start at 1 with the bottom most SiPM in the p
## along that same z height row counterclockwise.  Once you get to SiPM 9 you wil

## SiPMs in First Row

# SiPM 1
SiPM_1 = surface.surface()
SiPM_1.description = 'SiPM bottom row at +x, y=0'
SiPM_1.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
SiPM_1.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_1.inbounds_function =  lambda p: np.reshape(((p[:,1,:]<=SiPM_c1y[1]) & (p[:,
                                                  (p[:,2,:]>=SiPM_r1[0]) & (p[:,0,
SiPM_1.n_outside = n_hydraulic
SiPM_1.n_inside = n_hydraulic
SiPM_1.surface_type = 'normal'
SiPM_1.absorption = 1
surface_list.append(SiPM_1)

#SiPM 2
SiPM_2 = surface.surface()
SiPM_2.description = 'SiPM bottom row at +x,+y'
SiPM_2.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
SiPM_2.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_2.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c2[1]) & (p[:,0
                                                  (p[:,2,:]>=SiPM_r1[0]) & (p[:,1,
SiPM_2.n_outside = n_hydraulic
SiPM_2.n_inside = n_hydraulic
SiPM_2.surface_type = 'normal'
SiPM_2.absorption = 1
surface_list.append(SiPM_2)

#SiPM 3
SiPM_3 = surface.surface()
SiPM_3.description = 'SiPM bottom row at x=0,+y'
SiPM_3.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
```

```
SiPM_3.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_3.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c3[1]) & (p[:,0
                                                  (p[:,2,:]>=SiPM_r1[0]) & (p[:,1,
SiPM_3.n_outside = n_hydraulic
SiPM_3.n_inside = n_hydraulic
SiPM_3.surface_type = 'normal'
SiPM_3.absorption = 1
surface_list.append(SiPM_3)

#SiPM 4
SiPM_4 = surface.surface()
SiPM_4.description = 'SiPM bottom row at -x,+y'
SiPM_4.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
SiPM_4.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_4.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c4[1]) & (p[:,0
                                                  (p[:,2,:]>=SiPM_r1[0]) & (p[:,1,
SiPM_4.n_outside = n_hydraulic
SiPM_4.n_inside = n_hydraulic
SiPM_4.surface_type = 'normal'
SiPM_4.absorption = 1
surface_list.append(SiPM_4)

# SiPM 5
SiPM_5 = surface.surface()
SiPM_5.description = 'SiPM bottom row at -x,y=0'
SiPM_5.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
SiPM_5.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_5.inbounds_function =  lambda p: np.reshape(((p[:,1,:]<=SiPM_c1y[1]) & (p[:,
                                                  (p[:,2,:]>=SiPM_r1[0]) & (p[:,0,
SiPM_5.n_outside = n_hydraulic
SiPM_5.n_inside = n_hydraulic
SiPM_5.surface_type = 'normal'
SiPM_5.absorption = 1
surface_list.append(SiPM_5)

#SiPM 6
SiPM_6 = surface.surface()
SiPM_6.description = 'SiPM bottom row at -x,-y'
SiPM_6.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
SiPM_6.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_6.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c4[1]) & (p[:,0
                                                  (p[:,2,:]>=SiPM_r1[0]) & (p[:,1,
SiPM_6.n_outside = n_hydraulic
SiPM_6.n_inside = n_hydraulic
SiPM_6.surface_type = 'normal'
SiPM_6.absorption = 1
surface_list.append(SiPM_6)

#SiPM 7
SiPM_7 = surface.surface()
SiPM_7.description = 'SiPM bottom row at x=0,-y'
SiPM_7.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
SiPM_7.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_7.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c3[1]) & (p[:,0
                                                  (p[:,2,:]>=SiPM_r1[0]) & (p[:,1,
SiPM_7.n_outside = n_hydraulic
SiPM_7.n_inside = n_hydraulic
```

```
SiPM_7.surface_type = 'normal'
SiPM_7.absorption = 1
surface_list.append(SiPM_7)

#SiPM 8
SiPM_8 = surface.surface()
SiPM_8.description = 'SiPM bottom row at +x,-y'
SiPM_8.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
SiPM_8.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_8.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c2[1]) & (p[:,0
                                                  (p[:,2,:]>=SiPM_r1[0]) & (p[:,1,
SiPM_8.n_outside = n_hydraulic
SiPM_8.n_inside = n_hydraulic
SiPM_8.surface_type = 'normal'
SiPM_8.absorption = 1
surface_list.append(SiPM_8)

## SiPMs in Second Row

# SiPM 9
SiPM_9 = surface.surface()
SiPM_9.description = 'SiPM second row at +x, y=0'
SiPM_9.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometry
SiPM_9.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_9.inbounds_function =  lambda p: np.reshape(((p[:,1,:]<=SiPM_c1y[1]) & (p[:,
                                                  (p[:,2,:]>=SiPM_r2[0]) & (p[:,0,
SiPM_9.n_outside = n_hydraulic
SiPM_9.n_inside = n_hydraulic
SiPM_9.surface_type = 'normal'
SiPM_9.absorption = 1
surface_list.append(SiPM_9)

#SiPM 10
SiPM_10 = surface.surface()
SiPM_10.description = 'SiPM second row at +x,+y'
SiPM_10.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_10.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_10.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c2[1]) & (p[:,
                                                  (p[:,2,:]>=SiPM_r2[0]) & (p[:,1,
SiPM_10.n_outside = n_hydraulic
SiPM_10.n_inside = n_hydraulic
SiPM_10.surface_type = 'normal'
SiPM_10.absorption = 1
surface_list.append(SiPM_10)

#SiPM 11
SiPM_11 = surface.surface()
SiPM_11.description = 'SiPM second row at x=0,+y'
SiPM_11.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_11.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_11.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c3[1]) & (p[:,
                                                  (p[:,2,:]>=SiPM_r2[0]) & (p[:,1,
SiPM_11.n_outside = n_hydraulic
SiPM_11.n_inside = n_hydraulic
SiPM_11.surface_type = 'normal'
SiPM_11.absorption = 1
surface_list.append(SiPM_11)
```

```
#SiPM 12
SiPM_12 = surface.surface()
SiPM_12.description = 'SiPM second row at -x,+y'
SiPM_12.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_12.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_12.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c4[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r2[0]) & (p[:,1,
SiPM_12.n_outside = n_hydraulic
SiPM_12.n_inside = n_hydraulic
SiPM_12.surface_type = 'normal'
SiPM_12.absorption = 1
surface_list.append(SiPM_12)

# SiPM 13
SiPM_13 = surface.surface()
SiPM_13.description = 'SiPM second row at -x,y=0'
SiPM_13.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_13.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_13.inbounds_function =  lambda p: np.reshape(((p[:,1,:]<=SiPM_c1y[1]) & (p[:
                                                    (p[:,2,:]>=SiPM_r2[0]) & (p[:,0,
SiPM_13.n_outside = n_hydraulic
SiPM_13.n_inside = n_hydraulic
SiPM_13.surface_type = 'normal'
SiPM_13.absorption = 1
surface_list.append(SiPM_13)

#SiPM 14
SiPM_14 = surface.surface()
SiPM_14.description = 'SiPM second row at -x,-y'
SiPM_14.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_14.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_14.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c4[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r2[0]) & (p[:,1,
SiPM_14.n_outside = n_hydraulic
SiPM_14.n_inside = n_hydraulic
SiPM_14.surface_type = 'normal'
SiPM_14.absorption = 1
surface_list.append(SiPM_14)

#SiPM 15
SiPM_15 = surface.surface()
SiPM_15.description = 'SiPM second row at x=0,-y'
SiPM_15.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_15.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_15.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c3[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r2[0]) & (p[:,1,
SiPM_15.n_outside = n_hydraulic
SiPM_15.n_inside = n_hydraulic
SiPM_15.surface_type = 'normal'
SiPM_15.absorption = 1
surface_list.append(SiPM_15)

#SiPM 16
SiPM_16 = surface.surface()
SiPM_16.description = 'SiPM second row at +x,-y'
SiPM_16.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
```

```
SiPM_16.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_16.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c2[1]) & (p[:,
                                                   (p[:,2,:]>=SiPM_r2[0]) & (p[:,1,
SiPM_16.n_outside = n_hydraulic
SiPM_16.n_inside = n_hydraulic
SiPM_16.surface_type = 'normal'
SiPM_16.absorption = 1
surface_list.append(SiPM_16)

## SiPMs in Third row

# SiPM 17
SiPM_17 = surface.surface()
SiPM_17.description = 'SiPM third row at +x, y=0'
SiPM_17.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_17.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_17.inbounds_function =  lambda p: np.reshape(((p[:,1,:]<=SiPM_c1y[1]) & (p[:
                                                   (p[:,2,:]>=SiPM_r3[0]) & (p[:,0,
SiPM_17.n_outside = n_hydraulic
SiPM_17.n_inside = n_hydraulic
SiPM_17.surface_type = 'normal'
SiPM_17.absorption = 1
surface_list.append(SiPM_17)

#SiPM 18
SiPM_18 = surface.surface()
SiPM_18.description = 'SiPM third row at +x,+y'
SiPM_18.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_18.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_18.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c2[1]) & (p[:,
                                                   (p[:,2,:]>=SiPM_r3[0]) & (p[:,1,
SiPM_18.n_outside = n_hydraulic
SiPM_18.n_inside = n_hydraulic
SiPM_18.surface_type = 'normal'
SiPM_18.absorption = 1
surface_list.append(SiPM_18)

#SiPM 19
SiPM_19 = surface.surface()
SiPM_19.description = 'SiPM third row at x=0,+y'
SiPM_19.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_19.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_19.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c3[1]) & (p[:,
                                                   (p[:,2,:]>=SiPM_r3[0]) & (p[:,1,
SiPM_19.n_outside = n_hydraulic
SiPM_19.n_inside = n_hydraulic
SiPM_19.surface_type = 'normal'
SiPM_19.absorption = 1
surface_list.append(SiPM_19)

#SiPM 20
SiPM_20 = surface.surface()
SiPM_20.description = 'SiPM third row at -x,+y'
SiPM_20.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_20.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_20.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c4[1]) & (p[:,
                                                   (p[:,2,:]>=SiPM_r3[0]) & (p[:,1,
```

```
SiPM_20.n_outside = n_hydraulic
SiPM_20.n_inside = n_hydraulic
SiPM_20.surface_type = 'normal'
SiPM_20.absorption = 1
surface_list.append(SiPM_20)

# SiPM 21
SiPM_21 = surface.surface()
SiPM_21.description = 'SiPM third row at -x,y=0'
SiPM_21.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_21.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_21.inbounds_function =  lambda p: np.reshape(((p[:,1,:]<=SiPM_c1y[1]) & (p[:
                                                    (p[:,2,:]>=SiPM_r3[0]) & (p[:,0,
SiPM_21.n_outside = n_hydraulic
SiPM_21.n_inside = n_hydraulic
SiPM_21.surface_type = 'normal'
SiPM_21.absorption = 1
surface_list.append(SiPM_21)

#SiPM 22
SiPM_22 = surface.surface()
SiPM_22.description = 'SiPM third row at -x,-y'
SiPM_22.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_22.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_22.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c4[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r3[0]) & (p[:,1,
SiPM_22.n_outside = n_hydraulic
SiPM_22.n_inside = n_hydraulic
SiPM_22.surface_type = 'normal'
SiPM_22.absorption = 1
surface_list.append(SiPM_22)

#SiPM 23
SiPM_23 = surface.surface()
SiPM_23.description = 'SiPM third row at x=0,-y'
SiPM_23.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_23.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_23.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c3[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r3[0]) & (p[:,1,
SiPM_23.n_outside = n_hydraulic
SiPM_23.n_inside = n_hydraulic
SiPM_23.surface_type = 'normal'
SiPM_23.absorption = 1
surface_list.append(SiPM_23)

#SiPM 24
SiPM_24 = surface.surface()
SiPM_24.description = 'SiPM third row at +x,-y'
SiPM_24.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_24.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_24.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c2[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r3[0]) & (p[:,1,
SiPM_24.n_outside = n_hydraulic
SiPM_24.n_inside = n_hydraulic
SiPM_24.surface_type = 'normal'
SiPM_24.absorption = 1
surface_list.append(SiPM_24)
```

```
## SiPMs in Fourth row

# SiPM 25
SiPM_25 = surface.surface()
SiPM_25.description = 'SiPM fourth row at +x, y=0'
SiPM_25.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_25.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_25.inbounds_function =  lambda p: np.reshape(((p[:,1,:]<=SiPM_c1y[1]) & (p[:
                                                    (p[:,2,:]>=SiPM_r4[0]) & (p[:,0,
SiPM_25.n_outside = n_hydraulic
SiPM_25.n_inside = n_hydraulic
SiPM_25.surface_type = 'normal'
SiPM_25.absorption = 1
surface_list.append(SiPM_25)

#SiPM 26
SiPM_26 = surface.surface()
SiPM_26.description = 'SiPM fourth row at +x,+y'
SiPM_26.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_26.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_26.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c2[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r4[0]) & (p[:,1,
SiPM_26.n_outside = n_hydraulic
SiPM_26.n_inside = n_hydraulic
SiPM_26.surface_type = 'normal'
SiPM_26.absorption = 1
surface_list.append(SiPM_26)

#SiPM 27
SiPM_27 = surface.surface()
SiPM_27.description = 'SiPM fourth row at x=0,+y'
SiPM_27.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_27.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_27.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c3[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r4[0]) & (p[:,1,
SiPM_27.n_outside = n_hydraulic
SiPM_27.n_inside = n_hydraulic
SiPM_27.surface_type = 'normal'
SiPM_27.absorption = 1
surface_list.append(SiPM_27)

#SiPM 28
SiPM_28 = surface.surface()
SiPM_28.description = 'SiPM fourth row at -x,+y'
SiPM_28.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_28.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_28.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c4[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r4[0]) & (p[:,1,
SiPM_28.n_outside = n_hydraulic
SiPM_28.n_inside = n_hydraulic
SiPM_28.surface_type = 'normal'
SiPM_28.absorption = 1
surface_list.append(SiPM_28)

# SiPM 29
SiPM_29 = surface.surface()
```

```
SiPM_29.description = 'SiPM fourth row at -x,y=0'
SiPM_29.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_29.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_29.inbounds_function =  lambda p: np.reshape(((p[:,1,:]<=SiPM_c1y[1]) & (p[:
                                                    (p[:,2,:]>=SiPM_r4[0]) & (p[:,0,
SiPM_29.n_outside = n_hydraulic
SiPM_29.n_inside = n_hydraulic
SiPM_29.surface_type = 'normal'
SiPM_29.absorption = 1
surface_list.append(SiPM_29)

#SiPM 30
SiPM_30 = surface.surface()
SiPM_30.description = 'SiPM fourth row at -x,-y'
SiPM_30.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_30.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_30.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c4[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r4[0]) & (p[:,1,
SiPM_30.n_outside = n_hydraulic
SiPM_30.n_inside = n_hydraulic
SiPM_30.surface_type = 'normal'
SiPM_30.absorption = 1
surface_list.append(SiPM_30)

#SiPM 23
SiPM_31 = surface.surface()
SiPM_31.description = 'SiPM fourth row at x=0,-y'
SiPM_31.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_31.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_31.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c3[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r4[0]) & (p[:,1,
SiPM_31.n_outside = n_hydraulic
SiPM_31.n_inside = n_hydraulic
SiPM_31.surface_type = 'normal'
SiPM_31.absorption = 1
surface_list.append(SiPM_31)

#SiPM 32
SiPM_32 = surface.surface()
SiPM_32.description = 'SiPM fourth row at +x,-y'
SiPM_32.shape = 'cylinder' #SiPMs are actually planar, but this makes the geometr
SiPM_32.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
SiPM_32.inbounds_function =  lambda p: np.reshape(((p[:,0,:]<=SiPM_c2[1]) & (p[:,
                                                    (p[:,2,:]>=SiPM_r4[0]) & (p[:,1,
SiPM_32.n_outside = n_hydraulic
SiPM_32.n_inside = n_hydraulic
SiPM_32.surface_type = 'normal'
SiPM_32.absorption = 1
surface_list.append(SiPM_32)

# The next 8 SiPMs are mounted on the cone right above the cylinder.
# some useful variables:
hsl_bottom = 2*rd_rad/(1+np.sqrt(2)) #horizontal side length of octagon SiPMs are
sh = np.sqrt((rdcone_top-rd_top)**2+(rdcone_toprad-rd_rad)**2) #vertical side len
cntheta = np.arctan((rdcone_top-rd_top)/(rdcone_toprad-rd_rad)) # angle made with
SiPM_d_angled = np.sin(cntheta)*SiPM_d   # the dimensions of the SiPMs projected
ch = rdcone_top-rd_top # height of the cone
```

```
cnslope = (rdcone_top-rd_top)/(rdcone_toprad-rd_rad) #the slope of the line in th
                                                     #on top of the rd cyclinder
#center of SiPM for +x, +y tilted segment on cone. Can change some signs to make
# format = [x-value,z-value]
SiPM_center = [ch/2*(1/cnslope)+rd_rad-np.sqrt(2)/4*hsl_bottom,ch/2]


#SiPM 33
SiPM_33 = surface.surface()
SiPM_33.description = 'SiPM upper cone at +x,y=0'
SiPM_33.shape = 'quadsurface'                          # QuadSurface needs to be tes
SiPM_33.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
SiPM_33.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
& (p[:,1,:]<=SiPM_c1y[1]) & (p[:,0,:]>0) & (p[:,2,:]>= ((sh/2)-(0.5*SiPM_d_angled
, (p.shape[0], -1))
SiPM_33.n_outside = n_hydraulic
SiPM_33.n_inside = n_hydraulic
SiPM_33.surface_type = 'normal'
SiPM_33.absorption = 1
surface_list.append(SiPM_33)

#SiPM 34
SiPM_34 = surface.surface()
SiPM_34.description = 'SiPM upper cone at +x,+y'
SiPM_34.shape = 'quadsurface'                          # QuadSurface needs to be tes
SiPM_34.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
SiPM_34.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
(1/cnslope)+(rd_rad-np.sqrt(2)/4*hsl_bottom-np.sin(np.pi/4)*SiPM_d))) & (p[:,0,:]
(1/cnslope)+(rd_rad-np.sqrt(2)/4*hsl_bottom+np.sin(np.pi/4)*SiPM_d))) & (p[:,2,:]
(p[:,2,:]<=((sh/2)+(0.5*SiPM_d_angled))) & (p[:,1,:]>0),(p.shape[0], -1))
SiPM_34.n_outside = n_hydraulic
SiPM_34.n_inside = n_hydraulic
SiPM_34.surface_type = 'normal'
SiPM_34.absorption = 1
surface_list.append(SiPM_34)

#SiPM 35
SiPM_35 = surface.surface()
SiPM_35.description = 'SiPM upper cone at x=0,+y'
SiPM_35.shape = 'quadsurface'                          # QuadSurface needs to be tes
SiPM_35.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
SiPM_35.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
& (p[:,0,:]<=SiPM_c3[1]) & (p[:,1,:]>0) & (p[:,2,:]>= ((sh/2)-(0.5*SiPM_d_angled)
, (p.shape[0], -1))
SiPM_35.n_outside = n_hydraulic
SiPM_35.n_inside = n_hydraulic
SiPM_35.surface_type = 'normal'
SiPM_35.absorption = 1
surface_list.append(SiPM_35)

#SiPM 36
SiPM_36 = surface.surface()
SiPM_36.description = 'SiPM upper cone at -x,+y'
SiPM_36.shape = 'quadsurface'                          # QuadSurface needs to be tes
SiPM_36.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
SiPM_36.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
(1/cnslope)+(rd_rad-np.sqrt(2)/4*hsl_bottom-np.sin(np.pi/4)*SiPM_d))) & (p[:,0,:]
```

```
(1/cnslope)+(rd_rad-np.sqrt(2)/4*hsl_bottom+np.sin(np.pi/4)*SiPM_d))) & (p[:,2,:]
(p[:,2,:]<=((sh/2)+(0.5*SiPM_d_angled))) & (p[:,1,:]>0),(p.shape[0], -1))
SiPM_36.n_outside = n_hydraulic
SiPM_36.n_inside = n_hydraulic
SiPM_36.surface_type = 'normal'
SiPM_36.absorption = 1
surface_list.append(SiPM_36)

#SiPM 37
SiPM_37 = surface.surface()
SiPM_37.description = 'SiPM upper cone at -x,y=0'
SiPM_37.shape = 'quadsurface'                          # QuadSurface needs to be tes
SiPM_37.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
SiPM_37.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
& (p[:,1,:]<=SiPM_c5y[1]) & (p[:,0,:]<0) & (p[:,2,:]>= ((sh/2)-(0.5*SiPM_d_angled
, (p.shape[0], -1))
SiPM_37.n_outside = n_hydraulic
SiPM_37.n_inside = n_hydraulic
SiPM_37.surface_type = 'normal'
SiPM_37.absorption = 1
surface_list.append(SiPM_37)

#SiPM 38
SiPM_38 = surface.surface()
SiPM_38.description = 'SiPM upper cone at -x,-y'
SiPM_38.shape = 'quadsurface'                          # QuadSurface needs to be tes
SiPM_38.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
SiPM_38.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
(1/cnslope)+(rd_rad-np.sqrt(2)/4*hsl_bottom-np.sin(np.pi/4)*SiPM_d))) & (p[:,0,:]
(1/cnslope)+(rd_rad-np.sqrt(2)/4*hsl_bottom+np.sin(np.pi/4)*SiPM_d))) & (p[:,2,:]
(p[:,2,:]<=((sh/2)+(0.5*SiPM_d_angled))) & (p[:,1,:]<0),(p.shape[0], -1))
SiPM_38.n_outside = n_hydraulic
SiPM_38.n_inside = n_hydraulic
SiPM_38.surface_type = 'normal'
SiPM_38.absorption = 1
surface_list.append(SiPM_38)

#SiPM 39
SiPM_39 = surface.surface()
SiPM_39.description = 'SiPM upper cone at x=0,-y'
SiPM_39.shape = 'quadsurface'                          # QuadSurface needs to be tes
SiPM_39.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
SiPM_39.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
& (p[:,0,:]<=SiPM_c3[1]) & (p[:,1,:]<0) & (p[:,2,:]>= ((sh/2)-(0.5*SiPM_d_angled)
, (p.shape[0], -1))
SiPM_39.n_outside = n_hydraulic
SiPM_39.n_inside = n_hydraulic
SiPM_39.surface_type = 'normal'
SiPM_39.absorption = 1
surface_list.append(SiPM_39)

#SiPM 40
SiPM_40 = surface.surface()
SiPM_40.description = 'SiPM upper cone at +x,-y'
SiPM_40.shape = 'quadsurface'                          # QuadSurface needs to be tes
SiPM_40.param_list = [rd_cone_Q, rd_cone_P, rd_cone_R]
SiPM_40.inbounds_function = lambda p: np.reshape((p[:, 2, :] > rd_top) * (p[:, 2,
```

```
(1/cnslope)+(rd_rad-np.sqrt(2)/4*hsl_bottom-np.sin(np.pi/4)*SiPM_d))) & (p[:,0,:]
(1/cnslope)+(rd_rad-np.sqrt(2)/4*hsl_bottom+np.sin(np.pi/4)*SiPM_d))) & (p[:,2,:]
(p[:,2,:]<=((sh/2)+(0.5*SiPM_d_angled))) & (p[:,1,:]<0),(p.shape[0], -1))
SiPM_40.n_outside = n_hydraulic
SiPM_40.n_inside = n_hydraulic
SiPM_40.surface_type = 'normal'
SiPM_40.absorption = 1
surface_list.append(SiPM_40)
```

In [72]:
```
rd = surface.surface()
rd.description = 'reflector/diffuser'
rd.shape = 'cylinder'
rd.param_list = [np.array([0, 0, 0]), np.array([0, 0, 1]), rd_rad]
rd.inbounds_function = lambda p: ((p[:, 2, :] > rd_bot) * (p[:, 2, :] <= rd_top))
rd.n_outside = n_hydraulic
rd.n_inside = n_hydraulic
rd.surface_type = 'normal'
rd.absorption = 1
surface_list.append(rd)
rdtopcone_apex
zpoint_upper
camcan_h*np.cos(topcone_theta)
```

Out[72]:
```
6.431594126231089
```

In [ ]:
```

```

In [85]:
```
# I don't trust the surfaces just based on the numbers. Lets try to find a way to
import RayTracer2

def plot_surface (surface,plane,s=10,name='plot_surface'):
    '''Plots the surface within your surface list in a 2D plane of your choosing

        surface = the name of the specific surface object from your surface list
          plane = the plane you wish the surface to be plotted on (xy,xz,yz,all)
          s = size of the points plotted
          all = all planes are plotted
        name = the name you want the figures saved under'''
    n=30000
    #create random starting points:
    x_array = np.random.uniform(-3,3,n+1)
    y_array = np.random.uniform(-3,3,n+1)
    z_array = np.random.uniform(-5,10,n+1)
    starting_points = x_array
    starting_points = np.append([starting_points],[y_array,z_array],axis=0)
    starting_points = np.transpose(starting_points)


    # create some isotropically random rays
    random_rays_dir = np.array([[0,0,1]])
    random_rays_pol = np.array([[0,1,0]])
    random_rays_complete = np.zeros((n+1,10))
    #randomize the direction
    for i in range(n):
        rand1 = np.random.uniform(0,2*np.pi)
        rand2 = np.random.uniform(0,np.pi)
        random_ray_dir = np.array((np.cos(rand1),np.sin(rand1),np.cos(rand2)))
        random_rays_dir = np.append(random_rays_dir,[random_ray_dir],axis=0)
    #randomize the polarization
    for i in range(n):
        rand1 = np.random.uniform(0,2*np.pi)
        rand2 = np.random.uniform(0,np.pi)
        random_ray_pol = np.array((np.cos(rand1),np.sin(rand1),np.cos(rand2)))
        random_rays_pol = np.append(random_rays_pol,[random_ray_pol],axis=0)
    random_rays_complete[:,0:3]=random_rays_dir
    random_rays_complete[:,3:6]=random_rays_pol
    random_rays_complete[:,6] = 1
    if type(surface) == list :
        intersection_points = RayTracer2.RayTracer2(starting_points,random_rays_c
    else:
        intersection_points = RayTracer2.RayTracer2(starting_points,random_rays_c

    # creat the plot
    xlim = 30
    ylim = 30
    zlim = 40
    plt.figure(figsize=(20,20))
    if plane == 'xy' :
        plt.axes().set_aspect('equal')
        plt.scatter(intersection_points[:,0],intersection_points[:,1],s=s)
        plt.title('xy plane')
        plt.ylim(-ylim,ylim)
        plt.xlim(-xlim,xlim)
```

```
    if plane == 'xz' :
        plt.axes().set_aspect('equal')
        plt.scatter(intersection_points[:,0],intersection_points[:,2],s=s)
        plt.title('xz plane')
        plt.ylim(-ylim,ylim)
        plt.xlim(-xlim,xlim)
    if plane == 'yz' :
        plt.axes().set_aspect('equal')
        plt.scatter(intersection_points[:,1],intersection_points[:,2],s=s)
        plt.title('yz plane')
        plt.ylim(-ylim,ylim)
        plt.xlim(-xlim,xlim)
    if plane == 'all':

        plt.figure(figsize=(15,15))
        plt.axes().set_aspect('equal')
        plt.scatter(intersection_points[:,0],intersection_points[:,1],s=s)
        plt.title('xy plane')
        #plt.ylim(-ylim,ylim)
        #plt.xlim(-xlim,xlim)
        plt.savefig(f'{name}_xy_plane')
        plt.figure(figsize=(15,15))
        plt.axes().set_aspect('equal')
        plt.scatter(intersection_points[:,0],intersection_points[:,2],s=s)
        plt.title('xz plane')
        #plt.xlim(-xlim,xlim)
        #plt.ylim(-zlim,zlim)
        plt.savefig(f'{name}_xz_plane')
        plt.figure(figsize=(15,15))
        plt.axes().set_aspect('equal')
        plt.scatter(intersection_points[:,1],intersection_points[:,2],s=s)
        plt.title('yz_plane')
        #plt.xlim(-ylim,ylim)
        #plt.ylim(-zlim,zlim)
        plt.savefig(f'{name}_yz_plane')


        return starting_points
```

In [86]:
```
# these are all of the plot_surface functions I used for the figures above, comme

#plot_surface([camcan_3,camcan_2,camcan_1,rd_topcone,camcancaps],'all',s=1,name=
#plot_surface([rd,rd_topcone,rd_cone,],'all',s=1,name='rd')
#plot_surface([SiPM_1,SiPM_2,SiPM_3,SiPM_4,SiPM_5,SiPM_6,SiPM_7,SiPM_8,SiPM_9,SiP
#,SiPM_13,SiPM_14,SiPM_15,SiPM_16,SiPM_17,SiPM_18,SiPM_19,SiPM_20,SiPM_21,SiPM_22
#,SiPM_25,SiPM_26,SiPM_27,SiPM_28,SiPM_29,SiPM_30,SiPM_31,SiPM_32,SiPM_33,SiPM_34
#,SiPM_37,SiPM_38,SiPM_39,SiPM_40],'all',s=1,name='SiPMs')
#plot_surface(outer_jar_inner_dome,'all',s=1,name='ijid')
#plot_surface(outer_jar_inner_dome,'all',s=1,name='ojid_bad')
#plot_surface([camcan_3,camcan_2,camcan_1,rd_topcone,camcancaps],'all',s=1,name=
```

```
Intersections: [[[  0.19674819   0.19674819]
  [  2.89967091   2.89967091]
  [ 15.71140795   3.56873538]]

 [[ 25.7676977  -26.0747959 ]
  [  2.07460937  -0.43199314]
  [  5.85411706   8.79904961]]

 [[-20.8996269   16.64878114]
  [-16.76597881  20.45932794]
  [  5.44923675  -7.38107263]]

 ...

 [[ -5.99185825  11.91508869]
  [ 44.07542438 -41.64504725]
  [ -2.14941705  11.55887923]]

 [[ 14.1594616  -12.16057901]
  [ 21.09558384 -17.72294958]
  [ 27.87508859 -13.13125206]]

 [[-10.57287967   6.76708923]
  [-64.91901585  62.82041596]
  [ 45.21731921 -46.0601602 ]]]
Normals: [[[ 0.02672143  0.0027095 ]
  [ 0.39381984  0.03993262]
  [ 0.91879916  0.9991987 ]]

 [[ 0.39345219 -0.50793736]
  [ 0.03167763 -0.00841523]
  [ 0.91879916  0.8613529 ]]

 [[-0.30789587  0.12378685]
  [-0.24699846  0.15211899]
  [ 0.91879916  0.98057974]]

 ...

 [[-0.05317207  0.22793131]
  [ 0.39112764 -0.79665458]
  [ 0.91879916  0.55981139]]

 [[ 0.2199829  -0.07394713]
```

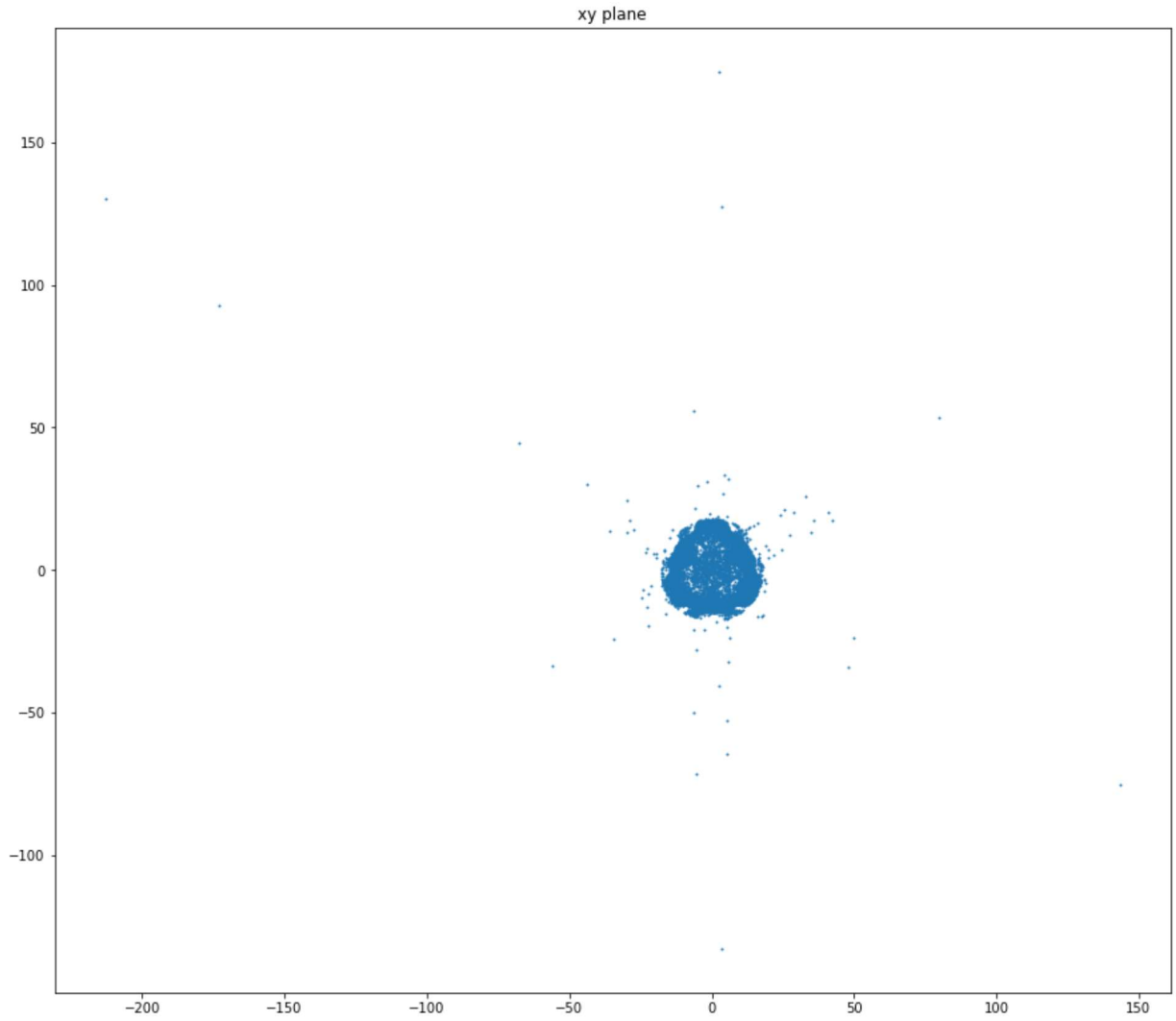
```
  [ 0.32774323 -0.10777129]
  [-0.91879916  0.99142179]]

 [[-0.06345002  0.01948769]
  [-0.38959235  0.1809086 ]
  [-0.91879916  0.98330682]]]
Normals after: [[[ 0.02672143  0.0027095 ]
  [ 0.39381984  0.03993262]
  [ 0.91879916  0.9991987 ]]

 [[ 0.39345219  0.50793736]
  [ 0.03167763  0.00841523]
  [ 0.91879916 -0.8613529 ]]

 [[-0.30789587  0.12378685]
  [-0.24699846  0.15211899]
  [ 0.91879916  0.98057974]]

 ...

 [[-0.05317207 -0.22793131]
  [ 0.39112764  0.79665458]
  [ 0.91879916 -0.55981139]]

 [[ 0.2199829   0.07394713]
  [ 0.32774323  0.10777129]
  [-0.91879916 -0.99142179]]

 [[-0.06345002 -0.01948769]
  [-0.38959235 -0.1809086 ]
  [-0.91879916 -0.98330682]]]
Intersections: [[[  0.19674819   0.19674819]
  [  2.89967091   2.89967091]
  [ 22.33581164  -3.05566831]]

 [[ 52.32143928 -52.62853747]
  [  3.35849196  -1.71587573]
  [  4.34572168  10.30744499]]

 [[-30.54191943  26.29107367]
  [-26.32530013  30.01864926]
  [  8.74401282 -10.6758487 ]]

 ...

 [[-15.29276103  21.21599147]
  [ 88.59880267 -86.16842555]
  [ -9.26953257  18.67899476]]

 [[ 19.37103238 -17.37214978]
  [ 28.78195247 -25.40931821]
  [ 35.9946599  -21.25082337]]

 [[-11.37938875   7.57359832]
  [-70.86037666  68.76177678]
  [ 49.46277758 -50.30561857]]]
Normals: [[[ 2.31140760e-02  9.81614530e-04]
```

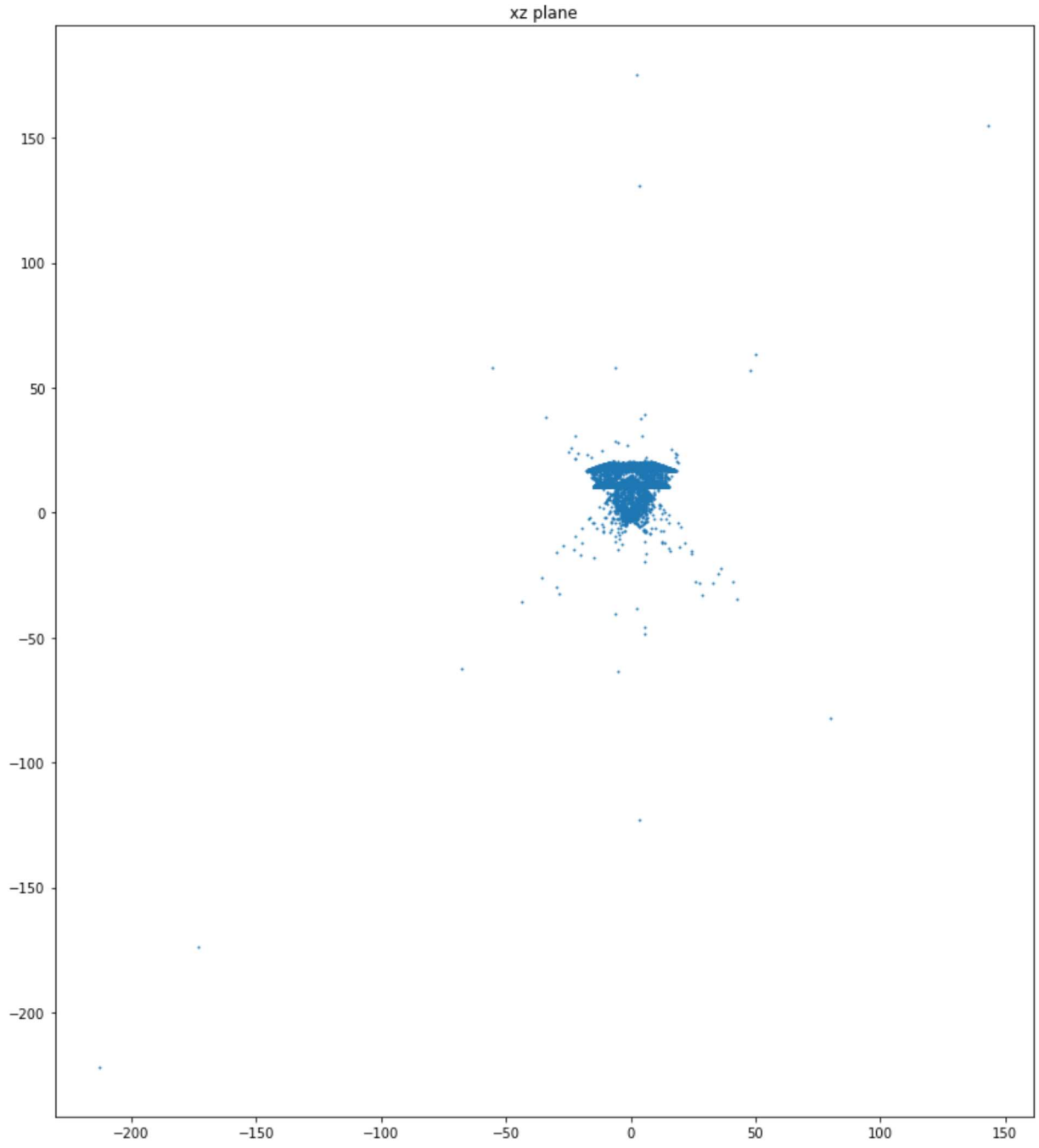
```
  [ 3.40654801e-01  1.44670158e-02]
  [ 9.39904275e-01  9.99894865e-01]]

 [[ 3.40736820e-01 -4.68791731e-01]
  [ 2.18717582e-02 -1.52842620e-02]
  [ 9.39904275e-01  8.83176485e-01]]

 [[-2.58625096e-01  1.00646375e-01]
  [-2.22919299e-01  1.14916122e-01]
  [ 9.39904275e-01  9.88263422e-01]]

 ...

 [[-5.80757724e-02  2.21790651e-01]
  [ 3.36462715e-01 -9.00799343e-01]
  [ 9.39904275e-01  3.73322180e-01]]

 [[ 1.90641224e-01 -5.11413504e-02]
  [ 2.83259382e-01 -7.48017293e-02]
  [-9.39904275e-01  9.95886170e-01]]

 [[-5.41375307e-02  1.34586408e-02]
  [-3.37118795e-01  1.22192915e-01]
  [-9.39904275e-01  9.92415113e-01]]]
Normals after: [[[ 2.31140760e-02  9.81614530e-04]
  [ 3.40654801e-01  1.44670158e-02]
  [ 9.39904275e-01  9.99894865e-01]]

 [[ 3.40736820e-01  4.68791731e-01]
  [ 2.18717582e-02  1.52842620e-02]
  [ 9.39904275e-01 -8.83176485e-01]]

 [[-2.58625096e-01  1.00646375e-01]
  [-2.22919299e-01  1.14916122e-01]
  [ 9.39904275e-01  9.88263422e-01]]

 ...

 [[-5.80757724e-02 -2.21790651e-01]
  [ 3.36462715e-01  9.00799343e-01]
  [ 9.39904275e-01 -3.73322180e-01]]

 [[ 1.90641224e-01  5.11413504e-02]
  [ 2.83259382e-01  7.48017293e-02]
  [-9.39904275e-01 -9.95886170e-01]]

 [[-5.41375307e-02 -1.34586408e-02]
  [-3.37118795e-01 -1.22192915e-01]
  [-9.39904275e-01 -9.92415113e-01]]]
incident angle: [0.67697447 0.59312786 0.90481158 ...        nan 0.80989315 0.5
8144313]
sin: [0.67697447 0.59312786 0.90481158 ...        nan 0.80989315 0.58144313]
n1: [1.22 1.22 1.22 ... 1.22 1.22 1.22]
n2: [1.22 1.22 1.22 ... 1.22 1.22 1.22]
cos: [0.7360065  +0.j 0.80510828 +0.j 0.42581218 +0.j ...        nan+nanj
 0.58657744 +0.j 0.81358705 +0.j]
all done!
```

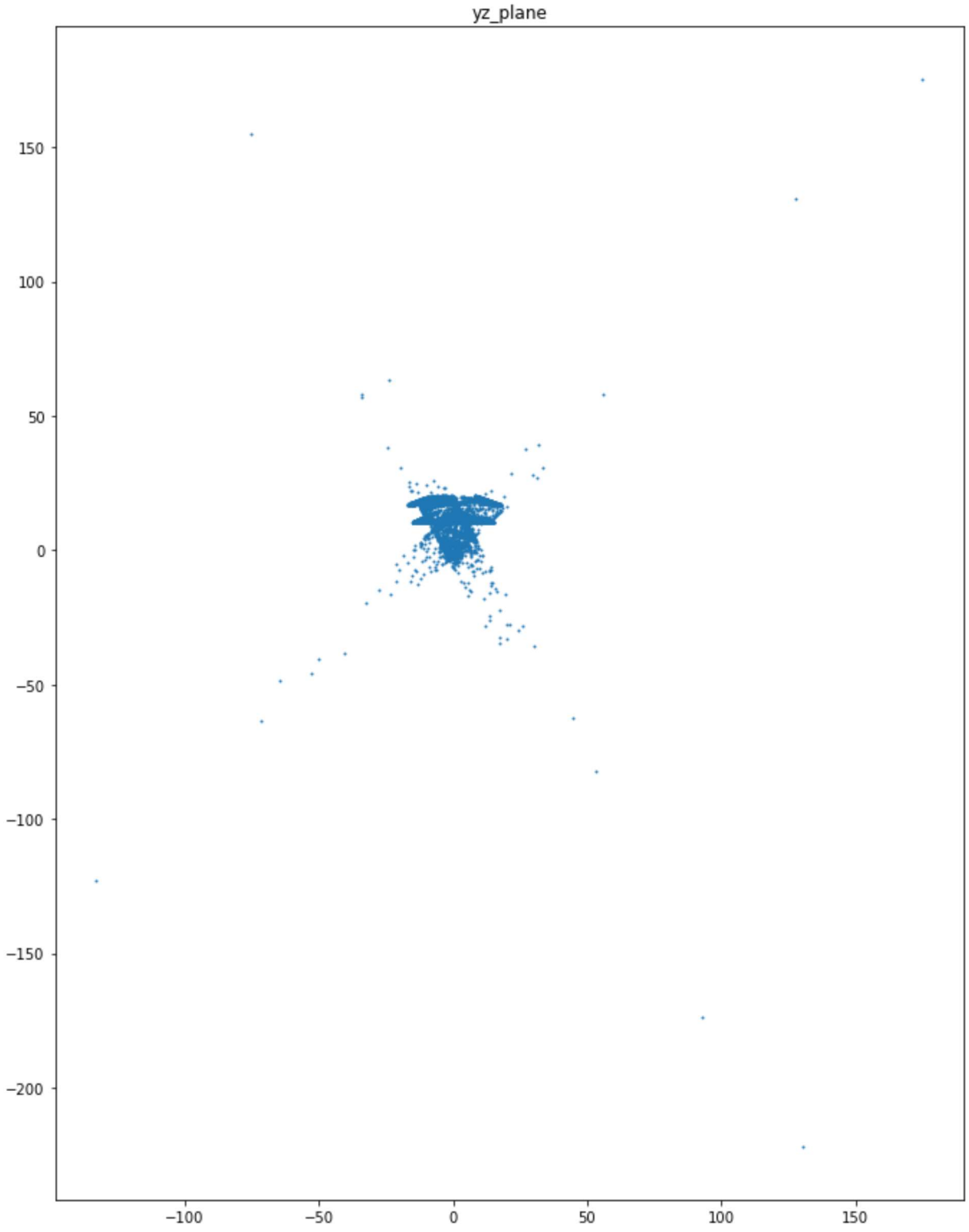
```
incoming: [[ 0.          0.          1.         ... -0.          0.
   0.        ]
 [ 0.28015368  0.81272971  0.51086625 ...  0.          0.
   0.        ]
 [-0.47540041  0.66723434  0.57340455 ...  0.          0.
   0.        ]
 ...
 [-0.05241062 -0.78040898  0.62306898 ...  0.          0.
   0.        ]
 [ 0.74558163  0.62931025  0.21926386 ...  0.          0.
   0.        ]
 [-0.51286971 -0.64391366  0.56774982 ... -0.          0.
   0.        ]]
reflected: [[-0.04633854  0.99543725 -0.08341114 ... -0.          0.
   0.        ]
 [-0.28015368 -0.81272971 -0.51086625 ...  0.          0.
   0.        ]
 [ 0.3079165   0.89333544  0.32732129 ... -0.          0.
   0.        ]
 ...
 [ 0.05241062  0.78040898 -0.62306898 ...  0.          0.
   0.        ]
 [-0.74558163 -0.62931025 -0.21926386 ...  0.          0.
   0.        ]
 [ 0.51286971  0.64391366 -0.56774982 ...  0.          0.
   0.        ]]
refracted: [[ 0.          0.          1.         ... -0.          0.
   0.        ]
 [ 0.28015368  0.81272971  0.51086625 ...  0.          0.
   0.        ]
 [-0.47540041  0.66723434  0.57340455 ... -0.          0.
   0.        ]
 ...
 [-0.05241062 -0.78040898  0.62306898 ...  0.          0.
   0.        ]
 [ 0.74558163  0.62931025  0.21926386 ...  0.          0.
   0.        ]
 [-0.51286971 -0.64391366  0.56774982 ... -0.          0.
   0.        ]]
```

Out[86]:
```
array([[ 0.19674819,  2.89967091,  9.64007166],
       [-0.1535491 ,  0.82130811,  7.32658333],
       [-2.12542288,  1.84667456, -0.96591794],
       ...,
       [ 2.96161522,  1.21518856,  4.70473109],
       [ 0.9994413 ,  1.68631713,  7.37191827],
       [-1.90289522, -1.04929994, -0.4214205 ]])
```

```
<Figure size 1440x1440 with 0 Axes>
```


xy plane
150
100
50
0
−50
−100
−200
−150
−100
−50
0
50
100
150


xz plane
150
100
50
0
−50
−100
−150
−200
−200
−150
−100
−50
0
50
100
150


yz_plane
150
100
50
0
−50
−100
−150
−200
−100
−50
0
50
100
150

In [ ]:

plt.show