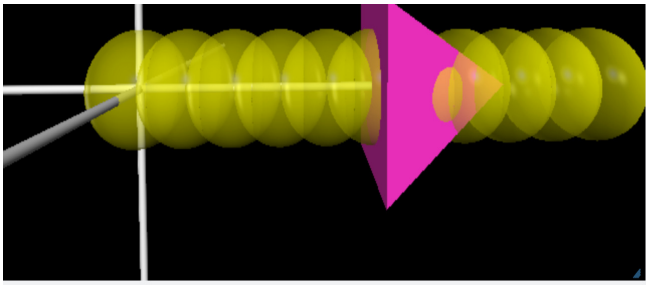
The goal of this assignment is for you to learn how to use Python to compute AND visualize center of mass problems.

Task 1: Determine center of mass for a <u>uniform</u> rod

It is always best to start with a simple problem for which we already know the answer. The first code (written for you on the next page) shows how to create a uniform rod of length L from point masses (spheres) The rod has its left end on the origin and extends parallel to the positive x-axis.



After typing the code, try modifying the following:

1. Change the number of point masses to 41
2. Change the length of the rod to 7.3
3. Change the color & opacity of the balls *after* the code computes the COM

```
for ball in rod:
    rate(8)
    ball.opacity = 0.5
    ball.color = vec(1,1,0)
```

4. Add code to compute the percent difference between x-coordinates of the exact solution and the output.
5. Determine the minimum number of balls required to make the percent difference less than 1%.
6. How does your answer to the above problem change if you modify the while loop as shown below?

```
while (i <= N):
```

7. A useful trick is to introduce a parameter for x in the code just before the while loop as shown below. If you modify the code as shown below, does it still work? Is the percent difference affected?

```
x = 0
while (x <= left_end.x + L):
    …
    …
    …
    x += dx
```

8. Now change the location of the left end of the rod to `vec(5,2,0)`. Is your % difference calculation still valid? If not, how could you generalize your % difference calculation to accommodate the variable?

Bullet 3 is used to emphasize the power of using lists, the `append` function, and `for` statements.

Bullets 5 & 6 above are supposed to show the code *rapidly* converges on the correct answer if the rod uses $N + 1$ balls (using `while (i <= N):` for the loop). In contrast, the code requires many balls to converge on the correct answer if one used N balls (using `while (i < N):` for the loop).

Bullet 7 is a useful style. If you plan to do a lot more coding, you will appreciate knowing it. By using this style, the limits of your loop relate to the limits of an object in the code (instead of some random increment limit).

Bullet 8 is to remind you to generalize codes whenever practical or easy.

Starter Code for uniform rod

```
GlowScript 2.7 VPython
from visual import *

#draw a set of coordinate axes
line_x=cylinder(pos=vec(-5, 0, 0), axis=vec(10, 0, 0), radius=0.05)
line_y=cylinder(pos=vec(0, -5, 0), axis=vec(0, 10, 0), radius=0.05)
line_z=cylinder(pos=vec(0, 0, -5), axis=vec(0, 0, 10), radius=0.05)

lambda = 1.0                #LINEAR mass density of rod
L = 5.1                     #length of rod
N = 21                      #number of slices for rod, must be >1
left_end = vec(0, 0, 0)     #specify location of left end of rod
dx = L/N                    #compute size of slice

numer = vec(0,0,0)          #initialize numerator of center of mass formula
denom = 0                   #initialize denominator of center of mass formula

rod = []                    #initialize list for organizational purposes
i = 0                       #initialize increment parameter

while (i < N):
    ball = sphere(pos=left_end+vec(i*dx,0,0), radius=dx)
    ball.dm = lambda*dx        #define & compute mass of slice
    rod.append(ball)
    numer += rod[i].dm*rod[i].pos
    denom += rod[i].dm
    i += 1

COM = numer/denom
print('Center of mass is at'+COM+'.')
COM_indicator = pyramid(pos=COM,
                        size=3*vec(dx,dx,dx),
                        color=vec(1, 0.2, 0.8))
```

Task 2: Determine center of mass for rod with <u>non-uniform</u> linear mass density.

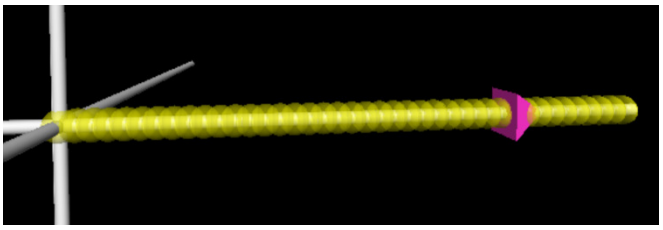
I modified my code using the following lines in the beginning:

```
c = 1.0                 #constant for LINEAR mass density = c * x**k
k = 2                   #exponent for LINEAR mass density = c * x**k
```

I also used the following line in the loop:

```
            ball.dm = c*ball.pos.x**k*dx      #define & compute mass of slice
```

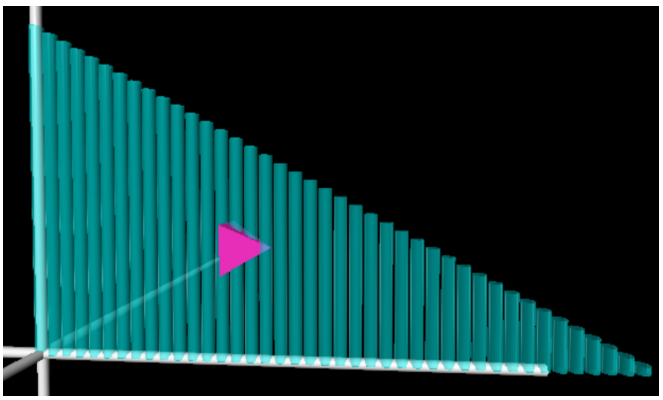
Using the input parameters $c = 1, k = 2, L = 5, \& \ N = 41$ the output of my code ended up looking like the one shown at right.



Think: we did non-uniform mass distribution problems in class. Problems **9.33**, **9.34**, & **9.38** all have exact solutions already done for you. Compare your *approximate* solutions (form the code) to the *analytic* results (from the workbook).

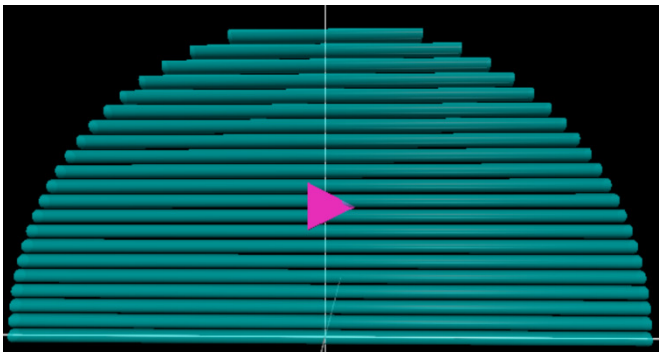
Task 3: Determine C.O.M. for a uniform triangular plate

- Use my code shown on the next page.
- The output of my code is shown at right.
- We know the *exact* result for right triangular plate (uniform) is 1/3 from the fat end (both horizontally & vertically).
- Add code to compute the percent difference between your *approximate* solution (code output) and the *exact* result.
- Determine the minimum number of chunks required to get the percent difference under 1%.



Task 4: Determine C.O.M. of a uniform, semi-circular plate.

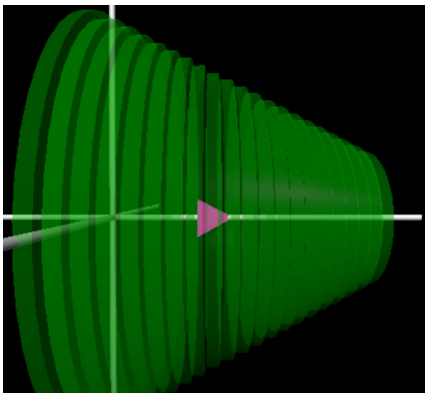
- Modify the code from task 3 to compute the desired computation.
- A screen shot of my code output is shown at right.
- The analytic result was determined in problem **9.30**.
- Add code to compute the percent difference between your *approximate* solution (code output) and the *analytic* result.
- Determine the minimum number of chunks required to get the percent difference under 1%.



Starter Code for Task3

```
GlowScript 2.7 VPython
from visual import *

#draw a set of coordinate axes
line_x=cylinder(pos=vec(-5, 0, 0), axis=vec(10, 0, 0), radius=0.05)
line_y=cylinder(pos=vec(0, -5, 0), axis=vec(0, 10, 0), radius=0.05)
line_z=cylinder(pos=vec(0, 0, -5), axis=vec(0, 0, 10), radius=0.05)

sigma = 1.0                 #AREA mass density of triangle
L = 6.0                     #length of triangle
H = 3.0                     #height of triangle
N = 41                      #number of slices for triangle, must be >1
left_end = vec(0, 0, 0)     #specify bottom left corner of triangle

#the following parameters relate to the triangle we are creating
#the word “slice” has a special meaning in python
#in my code I chose to use the word "chunk" instead of "slice"
dx = L/N
slope = -H/L
intercept = left_end + vec(0,H,0)

numer = vec(0,0,0)    #initialize numerator of center of mass formula
denom = 0             #initialize denominator of center of mass formula

plate = []            #initialize empty list for organizational purposes
i = 0                 #initialize increment parameter

while (i < N):
    chunk = cylinder(pos=left_end+vec(i*dx,0,0), radius=0.45*dx)
    chunk.axis = vec(0, slope*chunk.pos.x+intercept.y, 0)   #get chunk height
    chunk.dm = sigma*chunk.axis.y*dx                        #get chunk mass
    plate.append(chunk)
    numer += plate[i].dm*(plate[i].pos+0.5*plate[i].axis)
    denom += plate[i].dm
    i += 1

COM = numer/denom
print(COM)
COM_indicator = pyramid(pos=COM, size=3*vec(dx,dx,dx), color=vec(1,0.2,0.8))

#re-position the camera so it is centered on the COM
scene.camera.pos = COM + vec(0,0,5)
```

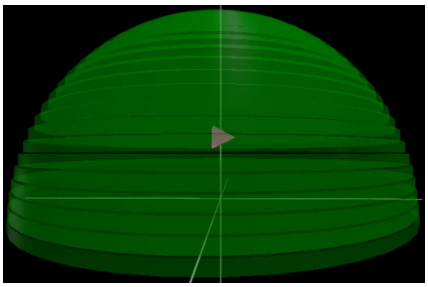
If you actually know what you are doing, you should now be able to use codes to determine the approximate center of mass for 3D objects.



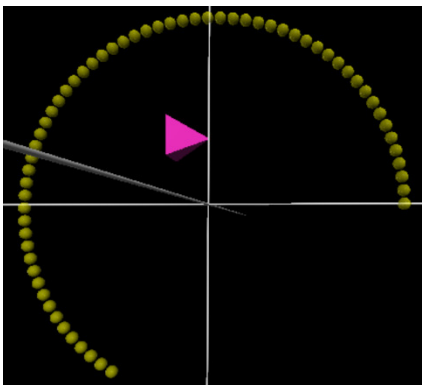
Task 5: Modify the triangle code (Task 3) to generate a frustum (a.k.a. a truncated cone).
A screenshot of my output is shown at right.
A nearly identical analytical result can be found in problem **9.40**.

Some bits of code I used are included below:

```
rho = 1.0                    #VOLUME mass density
L = 6.0                      #length of defining slope
H = 3.0                      #height of defining slope
x_max = 4.0                  #horizontal cut-off of frustum, use x_max < L
N = 31                       #number of slices for triangle, must be >1
left_end = vec(0, 0, 0)      #specify left end of frustum
…

x = left_end.x               #initialize horizontal coordinate
while (x <= x_max):
      …
      x += dx
```



Task 6: Use code to determine the approximate center of mass of a hemisphere
Output of my code is shown at right.
The analytic result can be found in problem **9.39**.



Task 7: Use code to determine the center of mass of an arc.
The output of my code is shown at right.
Ways to check your code:

1) Where *should* the C.O.M. be if the arc is a complete circle? Check it by inputting an appropriate value of θ and verifying the output is correct.
2) Modify your code to use a 45° starting angle for a ¼-circle arc. Compare to the exact result from problem **9.41**.

Some code fragments to help you out:

```
R = 2.0                      #radius of arc
theta_max_deg = 240          #max angle of arc IN DEGREES
theta_max = radians(theta_max_deg)  #max angle IN RADIANS
dtheta = theta_max/N                #angular increment IN RADIANS
ds = R*dtheta                       #arclength of a single chunk
…
while (theta <= theta_max):
      ball = sphere(pos=R*vec(cos(theta), sin(theta), 0))
      ball.dm = lambda*ds        #define & compute mass of slice
      …
      theta += dtheta
```