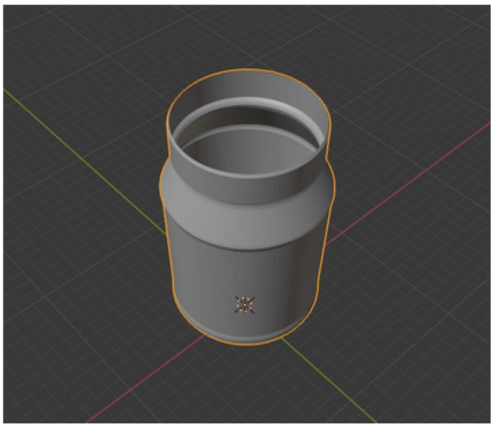
LA VERSIÓN EN HTML CON LOS MODELOS SE ENVIÓ POR CORREO, PORQUE CANVAS NO ACEPTA .ZIP

Modelo 3D



El modelo 3d fue creado con el siguiente script para el API de blender.

En `assets/model_script.txt` se encuentra una copia del código, que se puede cargar a blender. De cualquier otra manera, en el mismo folder se encuentran un archivo `.stl` y `.glb`, que se pueden visualizar con software especializado.

```
import bpy
import numpy as np

def f(x):
    '''
    Represents the piecewise function of the bottle
    '''
    if x==0:
        return 0*x
    if 0<x<0.44:
        return 4.0307*x+1
    elif 0.44<=x<0.8:
        return 1.1275*x+2.2774
    elif 0.8<=x<1.12:
        return 0.6264*x+2.6783
    elif 1.12<=x<1.6:
        return 1.6045*x+1.5829
    elif 1.6<=x<1.95:
```

```
    elif 1.6<=x<1.95:
        return 1.11203*x + 2.37076
    elif 1.95<=x<2.52:
        return 0.55421*x + 3.4585
    elif 2.52<=x<3:
        return 0.04246*x + 4.74811
    elif 3<=x<3.2:
        return -0.37745*x + 6.00784
    elif 3.2<=x<11.88:
        return 0.02057*x + 4.73419
    elif 11.88<=x<12.11:
        return 0.44992*x - 0.36651
    elif 12.11<=x<12.2634:
        return -0.21464*x + 7.68128
    elif 12.2634<=x<14.236:
        return -0.46361*x + 10.73447
    elif 14.236<=x<14.4845:
        return -0.29342*x + 8.31168
    elif 14.4845<=x<14.7:
        return -0.01277*x + 4.24658
    elif 14.7<=x<14.89:
        return 1.52057*x - 18.29346
    elif 14.89<=x<16.83:
        return 0.00851*x + 4.22108
    elif 16.83<=x<=17:
        return -0.67978*x + 15.80506
    else:
        return 0.0

#Im still not exactly sure about what vectorize does, but it kind of turns the function into a numpy array(?), b
ut its necessary for numpy
vf = np.vectorize(f)

#define the ranges of numbers to use
ll, ul = 0,17
w = np.linspace(ll,ul,100)
#print(w)
y = np.linspace(0, 2*np.pi,100)
#print(y)
```

```
#print("\n\n")

#manual mesh creation
X=np.outer(vf(w), np.sin(y))
Y=np.outer(vf(w), np.cos(y))
Z=np.zeros_like(X)

#z axis meesh
for i in range(len(w)):
    Z[i:i+1,:] = np.full_like(Z[0,:], w[i])

# This code creates arrays of tuples based on the mesh to be used with the blender api to do the 3d model
pointArray = []
facesArray = []

for i in range(len(w)):
    for j in range(len(w)):
        pointArray += (X[i][j], Y[i][j], Z[i][j]),

for i in range(len(w)-1):
    for j in range(len(w)-1):
        facesArray += ((i*100)+j, (i*100)+j+1, (i*100)+101+j,  (i*100)+100+j),

faces = facesArray
vertices = pointsArray
edges = []

# make mesh
new_mesh = bpy.data.meshes.new('new_mesh')
new_mesh.from_pydata(vertices, edges, faces)
new_mesh.update()

# make object from mesh
new_object = bpy.data.objects.new('new_object', new_mesh)

# make collection
new_collection = bpy.data.collections.new('new_collection')
bpy.context.scene.collection.children.link(new_collection)
```

```
    # add object to scene collection
    new_collection.objects.link(new_object)
```

Volumen

El volumen de nuestra función se puede determinar a través de la fórmula que ya hemos usado para los sólidos de revolución.

Todas las integrales y evaluaciones fueron generadas utilizando el modulo Sympy para python, en la parte del código viene cómo replicarlas, pero ocupaban mucho espacio, entonces fueron computadas por aparte

$$v = \pi \int_a^b [F(x)]^2 \, dx$$

Primero, tenemos que definir las integrales a usar para cada función de nuestra función original

$$\int_0^0 0\, dx + \int_0^{0.44} 16.24654249(x + 0.248095864241943)^2\, dx + \int_{0.44}^{0.8} 5.18655076(0.495082111179415x + 1)^2\, dx + \int_{0.8}^{1.12} 7.17329089(0.233879699809581x + 1)^2\, dx + \int_{1.12}^{1.6} 2.57442025(x + 0.986537862262387)^2\, dx + \int_{1.6}^{1.95} 5.6205029776(0.469060554421367x + 1)^2\, dx + \int_{1.95}^{2.52} 11.96122225(0.160245771288131x + 1)^2\, dx$$

3

$$+\int_{2.52}^{3} 22.5445485721(0.00894250554431132x+1)^2\,dx$$

$$+\int_{3}^{3.2} 36.0941414656(1-0.0628262403792378x)^2\,dx$$

$$+\int_{3.2}^{11.88} 22.4125549561(0.00434498826620816x+1)^2\,dx$$

$$+\int_{11.88}^{12.11} 0.2024280064(x-0.814611486486487)^2\,dx$$

$$+\int_{12.11}^{12.2634} 59.0020624384(1-0.0279432594567572x)^2\,dx$$

$$+\int_{12.2634}^{14.236} 115.2288461809(1-0.0431889045290545x)^2\,dx$$

$$+\int_{14.236}^{14.4845} 69.0840244224(1-0.0353021290521291x)^2\,dx$$

$$+\int_{14.4845}^{14.7} 18.0334416964(1-0.00300712573412016x)^2\,dx$$

$$+\int_{14.7}^{14.89} 334.6506787716(0.0831209623548525x-1)^2\,dx$$

$$+\int_{14.89}^{16.83} 17.8175163664(0.00201607171624324x+1)^2\,dx$$

$$+\int_{16.83}^{17} 249.7999216036(1-0.043010276455768x)^2\,dx$$

Luego, se evalúan

0

$$\begin{aligned}
&0\\
&+1.68165867848939\\
&+3.1942743012\\
&+3.44301217604096\\
&+6.827906145216\\
&+6.61090080928133\\
&+12.5808245462685\\
&+11.3621639100052\\
&+4.68085031600668\\
&+207.519632780753\\
&+5.82006105273302\\
&+3.93619049595338\\
&+41.7285920860892\\
&+4.17351508714296\\
&+3.55263060805169\\
&+3.35830728600376\\
&+36.8118533107034\\
&+3.15311841568564\\
&=360.435492005624
\end{aligned}$$

Al final multiplicamos por π para obtener nuestro resultado

$$v = \pi \times 360.435492005624 = 1132.34149377789\, ml$$

El cual está a décimas del numero que habíamos obtenido previamente, lo cual, considerando que son mililitros, es un error despreciable

```
x = sp.Symbol("x")

func_array = [(0*x, (x,0,0)),
              (4.0307*x+1, (x, 0, 0.44)),
              (1.1275*x+2.2774, (x, 0.44, 0.8)),
              (0.6264*x+2.6783, (x, 0.8, 1.12)),
              (1.6045*x+1.5829, (x, 1.12, 1.6)),
              (1.11203*x + 2.37076, (x, 1.6, 1.95)),
              (0.55421*x + 3.4585, (x, 1.95, 2.52)),
              (0.04246*x + 4.74811, (x, 2.52, 3)),
              (-0.37745*x + 6.00784, (x, 3, 3.2)),
              (0.02057*x + 4.73419, (x, 3.2, 11.88)),
              (0.44992*x - 0.36651, (x, 11.88, 12.11)),
              (-0.21464*x + 7.68128, (x, 12.11, 12.2634)),
              (-0.46361*x + 10.73447, (x, 12.2634, 14.236)),
```

```python
            (4.0307*x+1, (x, 0, 0.44)),
            (1.1275*x+2.2774, (x, 0.44, 0.8)),
            (0.6264*x+2.6783, (x, 0.8, 1.12)),
            (1.6045*x+1.5829, (x, 1.12, 1.6)),
            (1.11203*x + 2.37076, (x, 1.6, 1.95)),
            (0.55421*x + 3.4585, (x, 1.95, 2.52)),
            (0.04246*x + 4.74811, (x, 2.52, 3)),
            (-0.37745*x + 6.00784, (x, 3, 3.2)),
            (0.02057*x + 4.73419, (x, 3.2, 11.88)),
            (0.44992*x - 0.36651, (x, 11.88, 12.11)),
            (-0.21464*x + 7.68128, (x, 12.11, 12.2634)),
            (-0.46361*x + 10.73447, (x, 12.2634, 14.236)),
            (-0.29342*x + 8.31168, (x, 14.236, 14.4845)),
            (-0.01277*x + 4.24658, (x, 14.4845, 14.7)),
            (1.52057*x - 18.29346, (x, 14.7, 14.89)),
            (0.00851*x + 4.22108, (x, 14.89, 16.83)),
            (-0.67978*x + 15.80506, (x, 16.83, 17))]

area_integral = 0
for function in func_array:
    # Para las integrales definidas con formato LaTex
    # print(sp.latex(sp.Integral(function[0]**2,function[1])))
    #
    # Para las evaluaciones
    # print(sp.integrate(function[0]**2,function[1]))
    area_integral += sp.integrate(function[0]**2,function[1])

print(area_integral*np.pi)
```

```
1132.34149377789
```

Bibliografía

calculo.cc. (2012). Volumen de un cuerpo de revolución. Obtenido de Calculo.cc: https://calculo.cc/temas/temas_bachillerato/segundo_ciencias/integral_defi/teoria/defi_volumen.html

Ruiz, E. (abril de 2018). Apuntes de Cálculo Aplicado. Obtenido de INSTITUTO POLITÉCNICO NACIONAL: https://www.escom.ipn.mx/docs/oferta/matDidacticoISC2009/CAplcd/Apuntes_CalAplicado.pdf

Martínez del Castillo, J. (2000). Aplicaciones integrales. Málaga, España: Departamento de matemática aplicada. doi: https://navarrof.orgfree.com/Docencia/MatematicasII/M2UT4/T5aplicacionesintegral.pdf

Sc., T. S. (1990). Cálculo diferencial e integral (3rd ed.). Madrid, España: Mcgraw-Hill Interamerican.