Assignment 2: Lampshade Lattice

MECE 4606 Digital Manufacturing

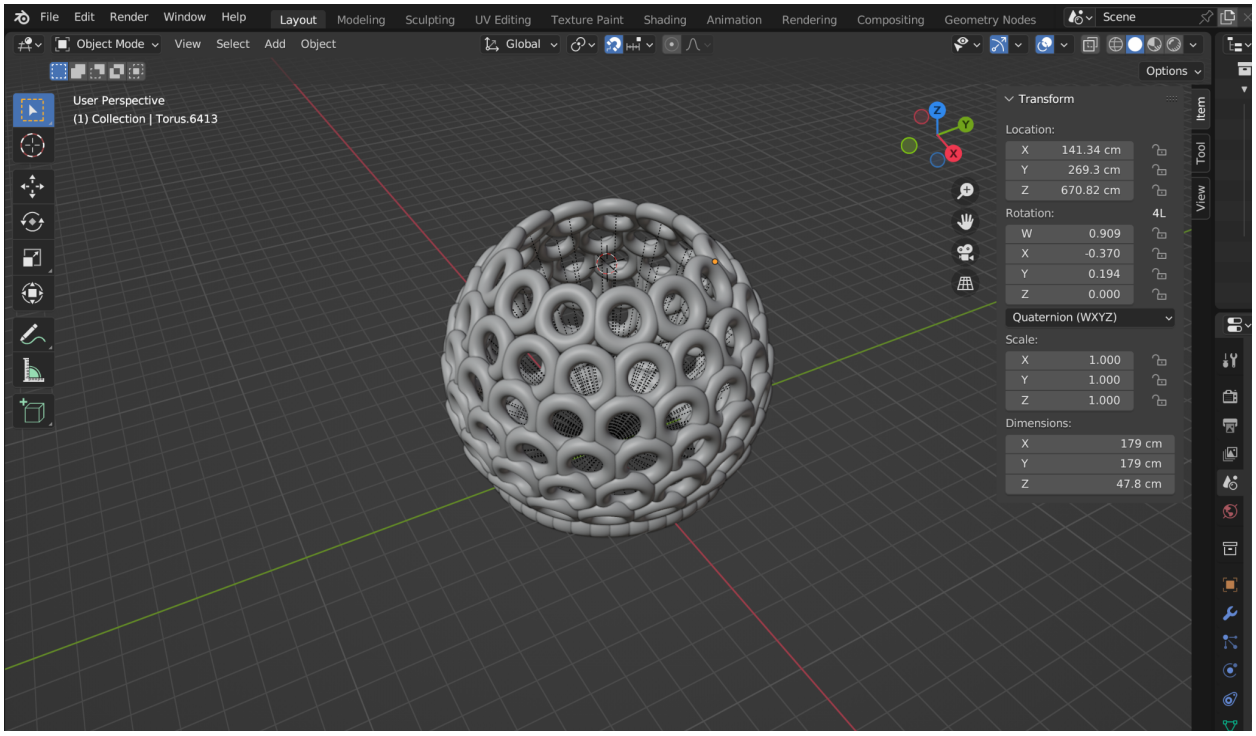
Nico Aldana (na2851) & Silvester Nava (sn2818)

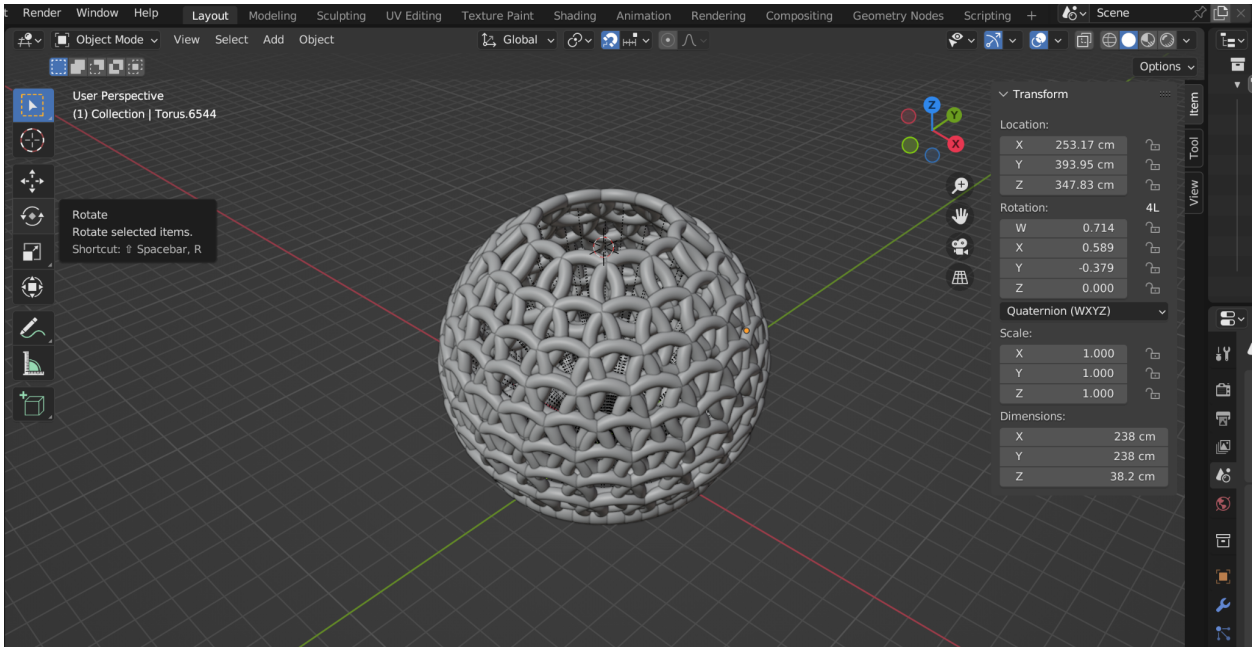
Submitted: Feb 19th, 1:00pm

Late Hours Remaining: 81.6 (+179 for submitting A2 early)

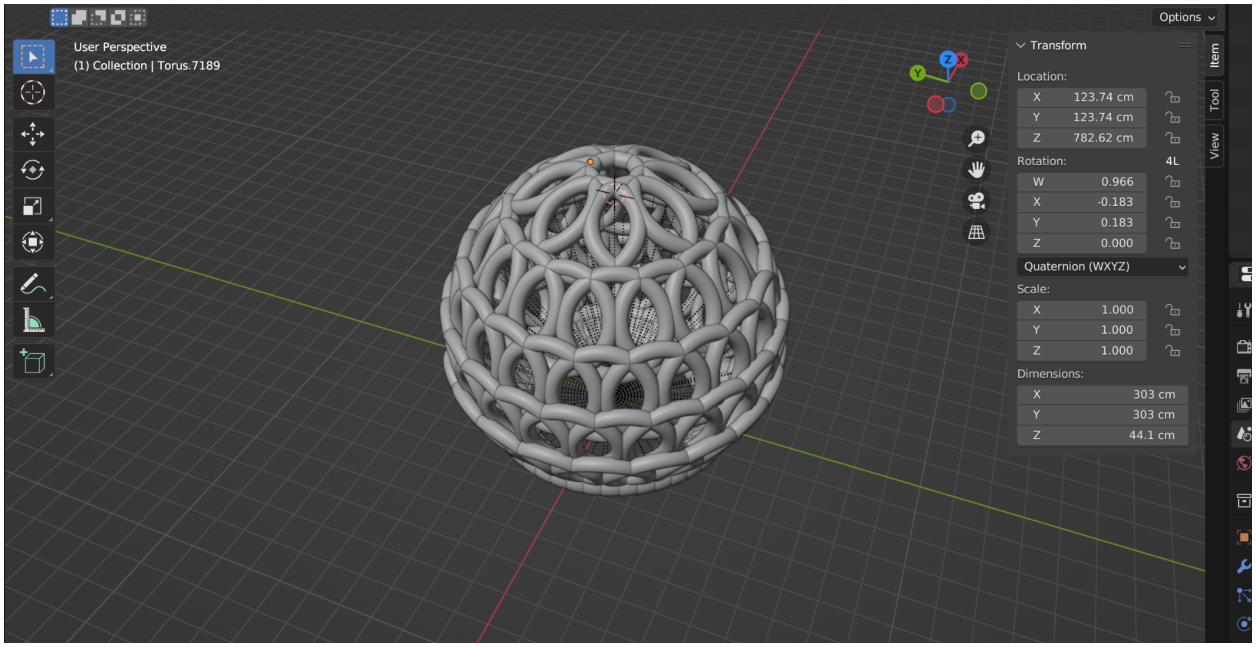
CAD Models

Here are a few example lampshades generated by our program.






Object Mode
View
Select
Add
Object
Global
Options
User Perspective
(1) Collection | Torus.6634
Transform
Location:
X 338.07 cm
Y 90.587 cm
Z 0 cm
Rotation:
4L
W 0.924
X 0.099
Y -0.370
Z 0.000
Quaternion (WXYZ)
Scale:
X 1.000
Y 1.000
Z 1.000
Dimensions:
X 292 cm
Y 292 cm
Z 180 cm
Item
Tool
View


Options
User Perspective
(1) Collection | Torus.7189
Transform
Location:
X 123.74 cm
Y 123.74 cm
Z 782.62 cm
Rotation:
4L
W 0.966
X -0.183
Y 0.183
Z 0.000
Quaternion (WXYZ)
Scale:
X 1.000
Y 1.000
Z 1.000
Dimensions:
X 303 cm
Y 303 cm
Z 44.1 cm
Item
Tool
View

Approach & Algorithm

We chose to make our lampshade lattice out of torus (donut) shapes. Instead of trying to create solid objects and cut through-holes, having torus shapes would automatically make the lampshade have holes for light to shine through.

We decided to build the lampshade shape layer by layer, where each layer consists of a ring of interlocking tori. In order to make the layers converge into a sphere, we needed a method to control the following values: the number of tori per layer (n), the z-height of the layer (z), the radius (r) and thickness (t) of each torus, the distance between the z-axis and the center of each torus (RP), the angle at which to tilt each torus away from the origin (phi), and an angle offset to rotate each layer clockwise about the z-axis (theta). All of these parameters are passed into a function that generates a layer of tori.

Python

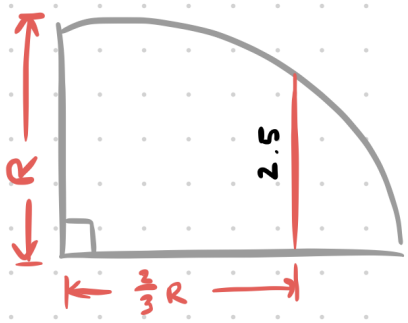
```
        def create_tori(n, RP, r, theta, z, t, phi): # Function that creates the ring of
tori for each layer
            # Set the origin of the scene to (0, 0, z)
            bpy.context.scene.cursor.location = (0, 0, z)
            bpy.ops.object.empty_add(type='PLAIN_AXES', location=(0, 0, 0))

            # Determine subdivision count to smooth the model
            subdivisions = 2

            for i in range(n):
                # Calculate the angle for the current torus
                angle = 2 * math.pi * i / n
```

```
                # Calculate the x and y coordinates for the current torus along a circle
of radius RP
                x = RP * math.cos(angle + math.pi/2)
                y = RP * math.sin(angle + math.pi/2)

                # Calculate the normal vector to the tangent plane for the current torus
                normal = mathutils.Vector((x, y, z)).normalized()

                s = 0.85
                # Scale the major and minor radii of the torus to create an oblong shape
                major_radius = r * s
                minor_radius = t / s

                # Create the torus object relative to the origin
                #bpy.ops.mesh.primitive_torus_add(location=(x, y, 0), rotation=(0, 0,
angle + math.pi/2), major_radius=r, minor_radius=t)
                bpy.ops.mesh.primitive_torus_add(location=(x, y, 0), rotation=(0, 0, angle
+ math.pi/2), major_radius=major_radius, minor_radius=minor_radius)
                torus = bpy.context.object

                # Move the torus to the correct z position
                torus.location.z = z

                # Parent the torus to the empty object at the origin
                torus.parent = bpy.context.scene.objects["Empty"]

                # Rotate the torus outward along the z-axis
                torus.rotation_euler[2] = theta
```

```
                # Calculate the rotation axis for the tilt of the torus
                axis = mathutils.Vector((x, y, z)).normalized().cross(mathutils.Vector((0,
0, 1)))
                # If the axis vector is close to zero, set it to a default direction
                if axis.length < 1e-5:
                    axis = mathutils.Vector((1, 0, 0))

                # Calculate the angle for the tilt of the torus
                angle = math.radians(phi)

                # Create the rotation quaternion for the tilt of the torus
                rot_quat = mathutils.Quaternion(axis, angle)

                # Rotate the torus to perform the tilt
                torus.rotation_mode = 'QUATERNION'
                torus.rotation_quaternion = rot_quat

                # Add a subdivision surface modifier to the torus
                mod = torus.modifiers.new("Subdivision", 'SUBSURF')
                mod.levels = subdivisions
```

Note that the orientation is calculated using quaternion modifiers. Using normalized vectors ensures each individual torus was tilted away from the z-axis, creating radial symmetry. The last two lines of code are to smooth out the resultant mesh to avoid low-polygon models. The *s* parameter is to modify the shape of each torus to make them slightly oblong.

The only hard-coded parameters are the radius and thickness of each torus. The rest are derived from several formulas to ensure the curvature of the lampshade follows a perfect sphere. We first drew out our approach and derived formulas on paper:



$f(x) = \sqrt{R^2 - x^2}$

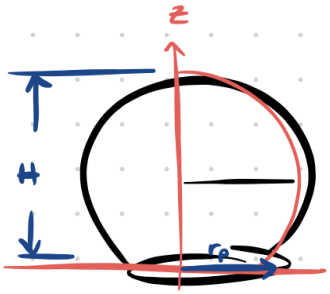
$f(\frac{2}{3}R) = 2.5$

$\sqrt{R^2 - (\frac{2}{3}R)^2} = r$

$\sqrt{R^2 - \frac{4}{9}R^2} = r$

$\frac{5}{9}R^2 = r^2$

$R = \sqrt{\frac{9}{5}r^2}$

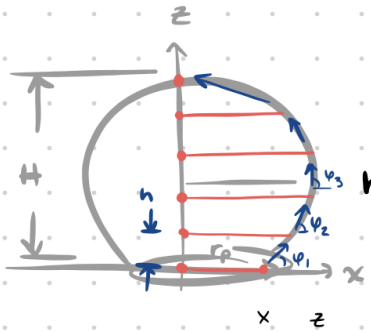


$r_p = 2.5$

$s(z) = \sqrt{R^2 - (z - \frac{2}{3}R)^2}$

$R = \sqrt{\frac{9}{5}r_p^2}$

$H = \frac{5}{3}R$



$h = \frac{1}{n}H$

$P_1 = (r_p, 0)$

$P_2 = (s(h), h)$

$P_3 = (s(2h), 2h) \ldots$

$\varphi_n = \arctan\left(\frac{P_{n+1_z} - P_{n_z}}{P_{n+1_x} - P_{n_x}}\right)$

Then, we began coding in these parameters in Python. We started with a function defining the governing equation of the curve.

```python
def curve(z): # Define the equation following the curve of the spherical lampshade
    return math.sqrt(R**2 - (z - (2/3)*R)**2)
```

Next, we defined all our initial parameters. Since the Blender console does not support user input, we have a section at the beginning of our code where a user can tune these values.

```python
# Define initial parameters (cm)
rp = 3.5 # Bottom radius so that we can fit the tea light
R = math.sqrt((9/5)*rp**2) # Radius of bounding sphere
H = (5/3)*R # Height of lampshade

n = 5 # Number of "levels" of the sphere
h = H/n # Layer height

torus_radius = 2.0
torus_thickness = 0.4
```

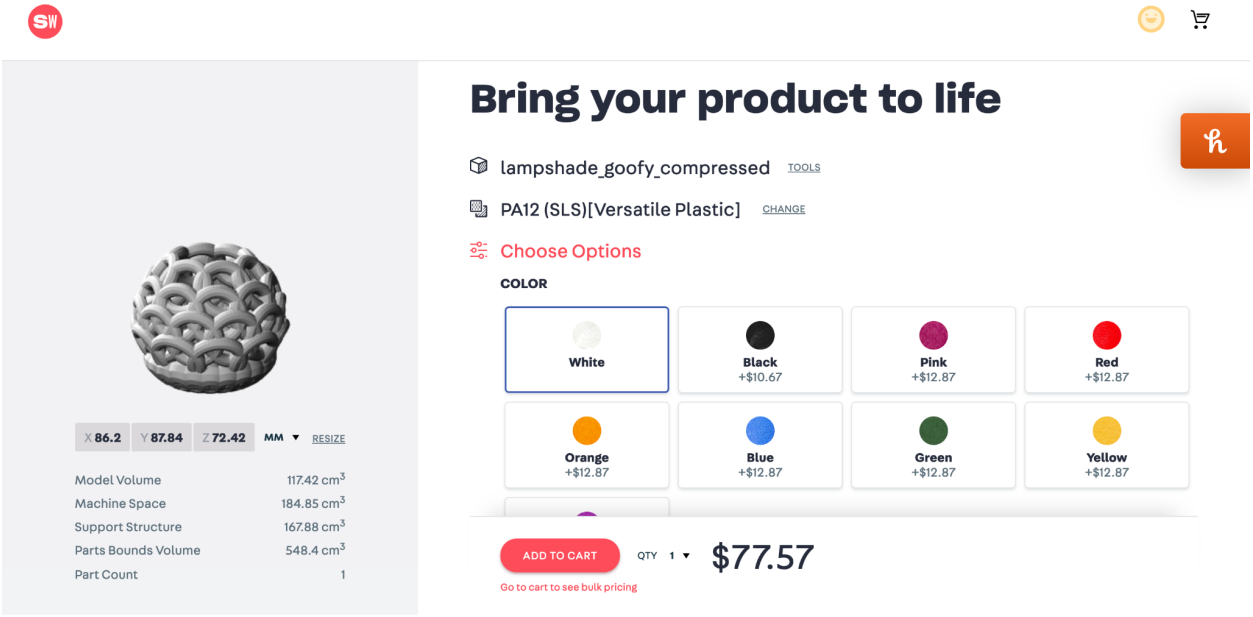
Finally, we define a function that calculates the appropriate parameters (coordinates for each torus, number of tori, and angle at which to tilt them) for each layer. The parameters are appended to lists, so each list index corresponds to the number layer of the sphere, starting from the bottom.

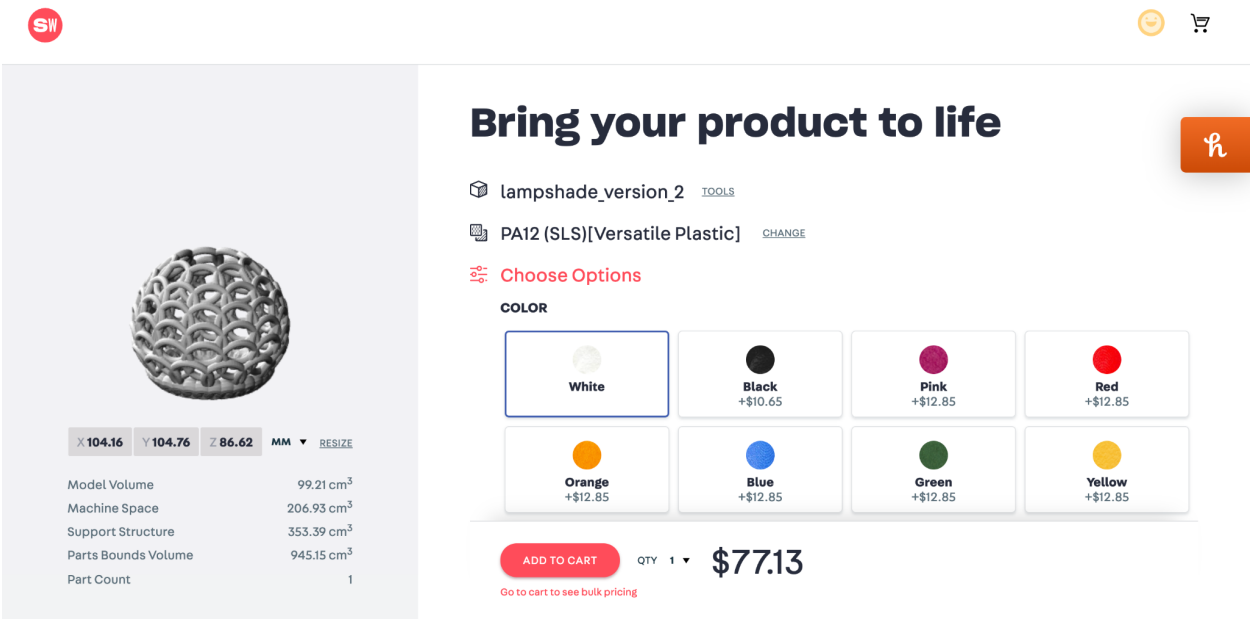
```python
def generate_lampshade():
    nodes = []
```

```
    phi = [45] # Angle of base layer
    quantities = []
    for i in range(0,n): # Calculate RP and z for each level
        nodes.append([curve(i*h),i*h]) # (RP, z)
    for i in range(0,n-1): # Calculate angles to tilt each level
        phi.append(math.degrees(math.atan((nodes[i+1][1] - nodes[i][1]) / (nodes[i+1][0] -
nodes[i][0]))))
    for i in range(0,n): # Calculate how many tori per level
        if(i == 0): # Have the base layer be slightly denser
        quantities.append(int((3/2)*math.pi*(nodes[i][0])/(torus_radius/3)))
        else:
         quantities.append(int((3/2)*math.pi*(nodes[i][0])/torus_radius))
    for i in range(0,n):
        if(i == 0): # Make sure the base layer tori are smaller and thicker for good
ground support
            create_tori(quantities[i], nodes[i][0], torus_radius/3, phi[i],
nodes[i][1],torus_thickness*2,phi[i])
        else:
            create_tori(quantities[i], nodes[i][0], torus_radius, phi[i],
nodes[i][1],torus_thickness,phi[i])
```

The script generates and displays the model in the Blender viewport. The scale, by default, is in meters, but the model parameters are intended to be in centimeters. This can be scaled manually in the slicer program when printing.

Shapeways Upload





Models generated by the program are under $100.

Physical Prints

Secondary Print:

Appendix

Project code is viewable at: https://github.com/reyluno/digital-manufacturing

Python

```
import bpy, bmesh
import math, mathutils, random

# Define initial parameters
rp = 3.5 # Bottom radius so that we can fit the tea light
R = math.sqrt((9/5)*rp**2) # Radius of bounding sphere
H = (5/3)*R # Height of lampshade

n = 5 # Number of "levels" of the sphere
h = H/n # Layer height

# Torus specifications
torus_radius = 1.0
torus_thickness = 0.2

def generate_lampshade():
    nodes = []
    phi = [45] # Angle of base layer
    quantities = []
    for i in range(0,n): # Calculate RP and z for each level of the
sphere
        nodes.append([curve(i*h),i*h]) # (RP, z)
    print(nodes)
    for i in range(0,n-1): # Calculate angles to tilt each level of the
sphere
        phi.append(math.degrees(math.atan((nodes[i+1][1] - nodes[i][1])
/ (nodes[i+1][0] - nodes[i][0]))))
    for i in range(0,n): # Calculate how many tori per level
        if(i == 0): # Have the base layer be slightly denser

quantities.append(int((3/2)*math.pi*(nodes[i][0])/(torus_radius/3)))
        else:

quantities.append(int((3/2)*math.pi*(nodes[i][0])/torus_radius))
    for i in range(0,n):
        if(i == 0): # Make sure the base layer tori are smaller and
thicker for good ground support
```

```
            create_tori(quantities[i], nodes[i][0], torus_radius/3,
phi[i], nodes[i][1],torus_thickness*2,phi[i])
        else:
            create_tori(quantities[i], nodes[i][0], torus_radius,
phi[i], nodes[i][1],torus_thickness,phi[i])
    # Generate lampshade ceiling (optional)
    create_tori(8, rp/2, torus_radius, 45, H, torus_thickness, -30)

def print(data): # This is just so that we can print to the Blender
console LOL
    for window in bpy.context.window_manager.windows:
        screen = window.screen
        for area in screen.areas:
            if area.type == 'CONSOLE':
                override = {'window': window, 'screen': screen, 'area':
area}
                bpy.ops.console.scrollback_append(override,
text=str(data), type="OUTPUT")

def curve(z): # Define the equation following the curve of the spherical
lampshade
    return math.sqrt(R**2 - (z - (2/3)*R)**2)

def create_tori(n, RP, r, theta, z, t, phi): # Function that creates the
ring of tori for each layer
    # Set the origin of the scene to (0, 0, z)
    bpy.context.scene.cursor.location = (0, 0, z)
    bpy.ops.object.empty_add(type='PLAIN_AXES', location=(0, 0, 0))

    # Determine subdivision count to smooth the model (Recommended: 2-3)
    subdivisions = 3

    for i in range(n):
        # Calculate the angle for the current torus
        angle = 2 * math.pi * i / n

        # Calculate the x and y coordinates for the current torus along
a circle of radius RP
        x = RP * math.cos(angle + math.pi/2)
        y = RP * math.sin(angle + math.pi/2)
```

```
        # Calculate the normal vector to the tangent plane for the
current torus
        normal = mathutils.Vector((x, y, z)).normalized()

        # Scale the major and minor radii of the torus to create an
oblong shape
        s = 1.3 # Recommended: 0.8 - 1.3
        major_radius = r * s
        minor_radius = t / s

        # Create the torus object relative to the origin
        bpy.ops.mesh.primitive_torus_add(location=(x, y, 0),
rotation=(0, 0, angle + math.pi/2), major_radius=major_radius,
minor_radius=minor_radius)
        torus = bpy.context.object

        # Move the torus to the correct z position
        torus.location.z = z

        # Parent the torus to the empty object at the origin
        torus.parent = bpy.context.scene.objects["Empty"]

        # Rotate the torus outward along the z-axis
        torus.rotation_euler[2] = theta

        # Calculate the rotation axis for the tilt of the torus
        axis = mathutils.Vector((x, y,
z)).normalized().cross(mathutils.Vector((0, 0, 1)))

        # If the axis vector is close to zero, set it to a default
direction
        if axis.length < 1e-5:
            axis = mathutils.Vector((1, 0, 0))

        # Calculate the angle for the tilt of the torus
        angle = math.radians(phi)

        # Create the rotation quaternion for the tilt of the torus
        rot_quat = mathutils.Quaternion(axis, angle)
```

```
        # Rotate the torus to perform the tilt
        torus.rotation_mode = 'QUATERNION'
        torus.rotation_quaternion = rot_quat

        # Add a subdivision surface modifier to the torus
        mod = torus.modifiers.new("Subdivision", 'SUBSURF')
        mod.levels = subdivisions

generate_lampshade() # Run the program
```