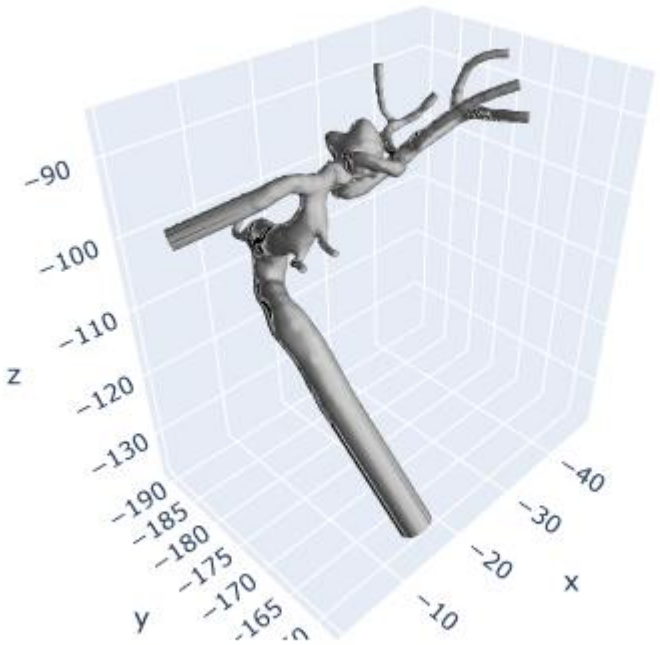
```
# Phong Shading
import numpy as np
import open3d as o3d
import plotly.graph_objects as go

mesh = o3d.io.read_triangle_mesh("Aneurism.obj")
if mesh.is_empty(): exit()

if not mesh.has_vertex_normals(): mesh.compute_vertex_normals()
if not mesh.has_triangle_normals(): mesh.compute_triangle_normals()

triangles = np.asarray(mesh.triangles)
vertices = np.asarray(mesh.vertices)
# Calculate normal vectors
normals = np.zeros(vertices.shape)
for triangle in triangles:
    v1 = vertices[triangle[0]]
    v2 = vertices[triangle[1]]
    v3 = vertices[triangle[2]]
    normal = np.cross(v2 - v1, v3 - v1)
    normals[triangle[0]] += normal
    normals[triangle[1]] += normal
    normals[triangle[2]] += normal
normals = normals / np.linalg.norm(normals, axis=1)[:, None]

# Define the reflect function
def reflect(vector, normal):
    return vector - 2.0 * np.dot(vector, normal) * normal

# Apply Phong shading
colors3 = np.ones(vertices.shape)
ambient_color = np.array([0.1, 0.1, 0.1])
diffuse_color = np.array([0.7, 0.7, 0.7])
specular_color = np.array([1.0, 1.0, 1.0])
light_position = np.array([0.0, 0.0, 1.0])
view_position = np.array([0.0, 0.0, 0.0])
shininess = 32.0
for i, vertex in enumerate(vertices):
    normal = normals[i]
    light_direction = light_position - vertex
    light_direction = light_direction / np.linalg.norm(light_direction)
    view_direction = view_position - vertex
    view_direction = view_direction / np.linalg.norm(view_direction)
    ambient_term = ambient_color
    diffuse_term = diffuse_color * np.maximum(np.dot(normal,
light_direction), 0.0)
    specular_term = specular_color *
np.power(np.maximum(np.dot(reflect(light_direction, normal),
view_direction), 0.0), shininess)
    colors3[i] = ambient_term + diffuse_term + specular_term
```

```
def reflect(vector, normal):
    return vector - 2.0 * np.dot(vector, normal) * normal

# Draw the mesh with the callback function
fig = go.Figure(data=[go.Mesh3d(x=vertices[:, 0], y=vertices[:, 1],
z=vertices[:, 2], i=triangles[:, 0], j=triangles[:, 1], k=triangles[:,
2], vertexcolor=colors3)])
fig.show()
```



```
# Surface Normals
import cv2
import numpy as np

dcm_slice = dcmread("brain_010.dcm")
sampling_freq = 15

img = np.array(dcm_slice.pixel_array)
smoothed_img = ndimage.gaussian_filter(img, sigma=3)
downsampled_img = smoothed_img[::sampling_freq,::sampling_freq]


################################################
# Calculate the normals of the down-samled image
################################################

#Calculate the surface normals
sobelx = cv2.Sobel(downsampled_img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(downsampled_img,cv2.CV_64F,0,1,ksize=5)
normal = np.dstack((-sobelx, -sobely, np.ones_like(downsampled_img)))
normal = normal / np.sqrt(np.sum(normal ** 2, axis=2, keepdims=True))

# Create a figure (window)
fig = plt.figure(figsize=(20,10))

ax1 = fig.add_subplot(1, 3, 1)
ax1.set_title('Downsampled MRI Slice')
ax1.imshow(downsampled_img, cmap='gray')

ax2 = fig.add_subplot(1, 3, 2)
ax2.set_title('Downsampled MRI Slice with normals')
ax2.imshow(downsampled_img, cmap='gray')
# add normals to the plot here, e.g. by using plt.quiver()
plt.quiver(normal[:,:,0], normal[:,:,1], normal[:,:,2])

plt.show()
```