UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

CARRERA PROFESIONAL DE INGENIERIA INFORMATICA Y DE SISTEMAS

COMPUTACION GRAFICA 2

DOCENTE: M.SC. HECTOR E. UGARTE R.

HOJA DE EJERCICIOS 7: MALLAS Y MAPEO UV

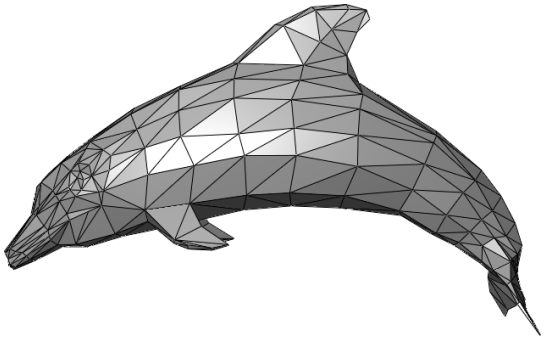
1. COMPETENCIAS

- El estudiante conoce el concepto mallas 3D.
- El estudiante entiende cómo aplicar mapeo de texturas UV a objetos 3D.

2. MARCO TEORICO

Mallas

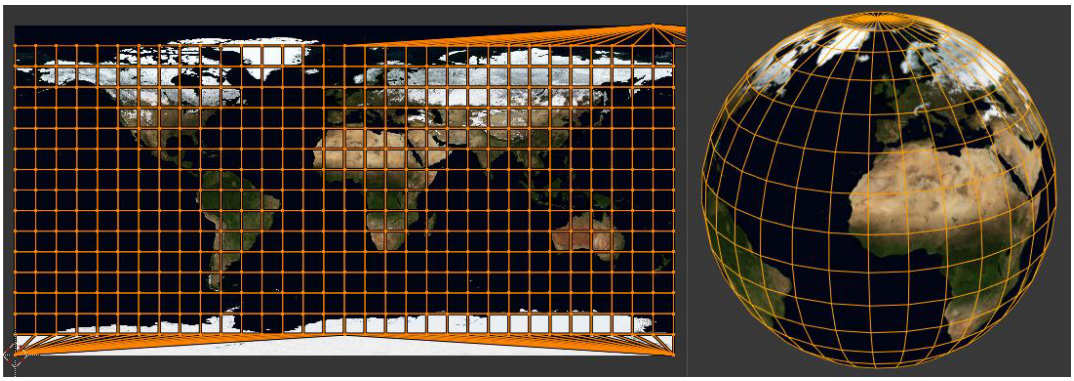
En gráficos por computadora y computación científica, la malla es una disposición de puntos, líneas y superficies que delinean la forma y la estructura de un objeto o superficie 3D. Hacer mallas es un proceso crítico en varias industrias, como el modelado 3D, la simulación, la visualización y los juegos. Exploraremos algunas bibliotecas y técnicas para crear mallas en Python.



Mapeado de texturas

El mapeo de texturas transforma patrones o imágenes realistas en modelos de computadora. El mapeo UV proyecta esta imagen 2D en la superficie de un modelo 3D. El proceso asigna píxeles de imagen a áreas de superficie en el modelo.

El mapeo UV es el proceso de modelado 3D que consiste en proyectar la superficie de un modelo 3D en una imagen 2D para el mapeo de texturas.



Mapeo UV en una esfera

Para cualquier punto P sobre la esfera, calcular d, que es el vector unitario de P al origen de la esfera.

Suponiendo que los polos de la esfera están alineados con el eje Y, las coordenadas UV en el rango [0,1], se puede calcular de la siguiente manera:

$$u = 0.5 + \frac{\arctan2(d_z, d_x)}{2\pi},$$
$$v = 0.5 + \frac{\arcsin(d_y)}{\pi}.$$

Librerías a utilizar:

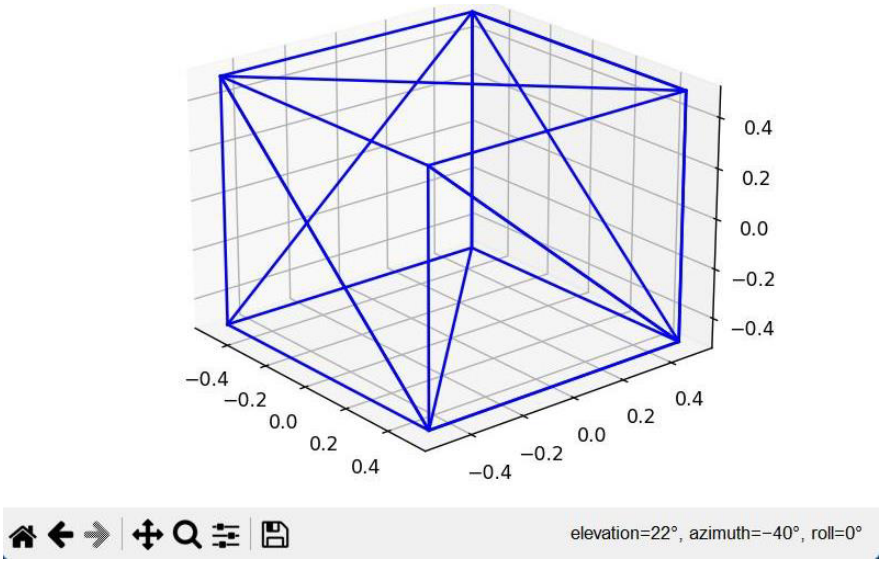
- Numpy: pip install numpy
- Matplotlib: pip install matplotlib
- Pyvista: pip install pyvista

3. PRACTICA

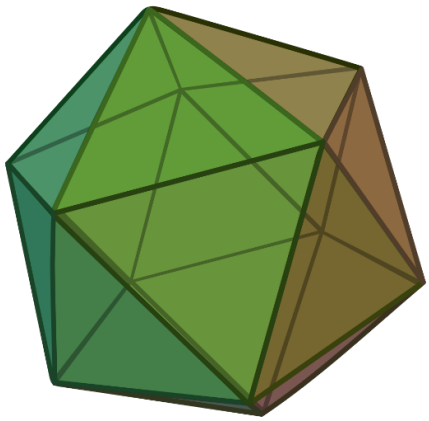
A. El siguiente código construye una malla para un cubo.

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

# Define vertices del cubo
vertices = np.array([
          [-0.5, -0.5, -0.5],
          [-0.5, -0.5, 0.5],
          [-0.5, 0.5, -0.5],
          [-0.5, 0.5, 0.5],
          [0.5, -0.5, -0.5],
          [0.5, -0.5, 0.5],
          [0.5, 0.5, -0.5],
          [0.5, 0.5, 0.5]
])

# Define indices (triangulos) del cubo
indices = np.array([
          [0, 1, 3],
          [0, 3, 2],
          [0, 2, 4],
          [2, 6, 4],
          [0, 4, 1],
          [1, 4, 5],
          [2, 3, 6],
          [3, 7, 6],
          [4, 6, 5],
          [5, 6, 7],
          [1, 5, 7],
          [1, 7, 3]
])

# Visualizamos la malla del cubo
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for triangulo in indices:
          ax.plot(vertices[triangulo, 0], vertices[triangulo, 1], vertices[triangulo, 2],
'b-')
plt.show()
```

Salida obtenida:



B. De manera similar implementa un icosaedro regular.



Notar que tiene 20 caras, 30 aristas y 12 vértices.

AYUDA:
VERTICES:
PHI = (1 + 5**0.5) / 2

[-1, PHI, 0],
[1, PHI, 0],
[-1, -PHI, 0],
[1, -PHI, 0],
[0, -1, PHI],
[0, 1, PHI],
[0, -1, -PHI],
[0, 1, -PHI],

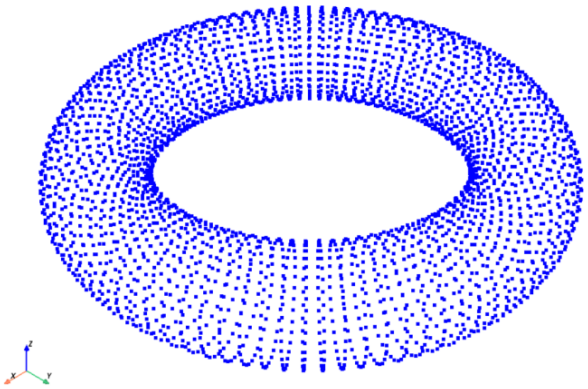
[PHI, 0, -1],
[PHI, 0, 1],
[-PHI, 0, -1],
[-PHI, 0, 1]

CARAS:
[0, 11, 5],
[0, 5, 1],
[0, 1, 7],
[0, 7, 10],
[0, 10, 11],
[1, 5, 9],
[5, 11, 4],
[11, 10, 2],
[10, 7, 6],
[7, 1, 8],
[3, 9, 4],
[3, 4, 2],
[3, 2, 6],
[3, 6, 8],
[3, 8, 9],
[4, 9, 5],
[2, 4, 11],
[6, 2, 10],
[8, 6, 7],
[9, 8, 1]

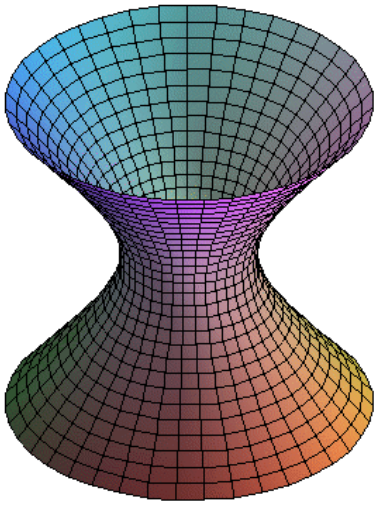
C. Ahora implementemos un esqueleto mas complejo. Un toroide:

```python
import pyvista as pv
import numpy as np

#Define los parametros del toro y sus nodos
radio = 2
n1 = 3
n2 = 7

#Define la funcion 3d para los nodos del toro
def nodos_toro(u, v):
    x = (radio + np.cos(n1 * u) * 0.5) * np.cos(n2 * v)
    y = (radio + np.cos(n1 * u) * 0.5) * np.sin(n2 * v)
    z = np.sin(n1 * u) * 0.5
    return x, y, z

#Crear una malla para la funcion 3d
u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, 2 * np.pi, 100)
x, y, z = nodos_toro(*np.meshgrid(u, v, indexing='ij'))
puntos = np.column_stack((x.ravel(), y.ravel(), z.ravel()))
mesh = pv.PolyData(puntos)
mesh.triangulate()
mesh = mesh.extract_surface()

# Visualizar la malla del toro
pv.plot(mesh, color='b', smooth_shading=True)
```

Salida obtenida:



D. De manera similar implementa un hiperboloide de una hoja similar a:

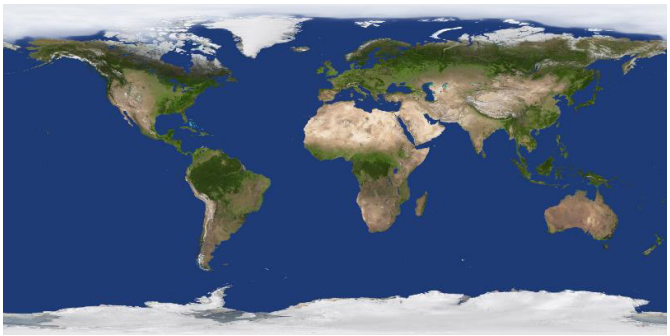


AYUDA: La ecuación de un hiperboloide es:

$$\begin{cases} x = a\ \cosh(\theta)\cos(\phi) \\ y = b\ \cosh(\theta)\,\mathrm{sen}(\phi) \\ z = c\ \sinh(\theta) \end{cases} \qquad \text{con } \theta \in \mathbb{R}, \quad \text{y} \quad 0 < \phi \leq 2\pi \ \text{(Hiperboloide de una hoja)}$$

E. Podemos implementar mapeo UV utilizando pyvista y las ecuaciones presentadas en el marco teórico. Probar el siguiente código, con la textura Tierra.png dada:

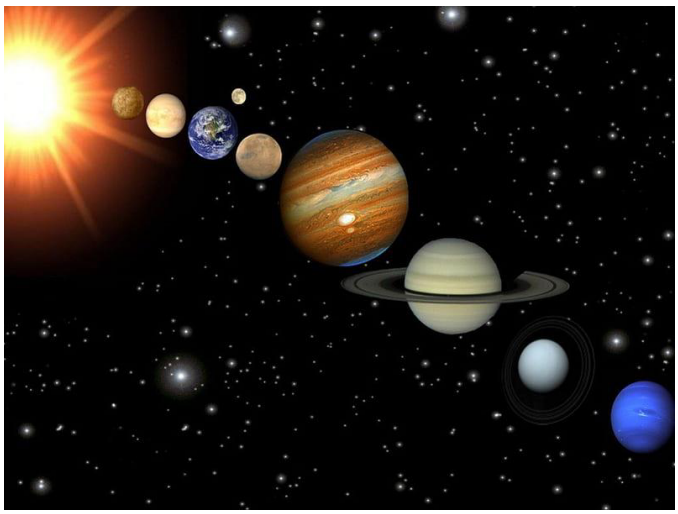
Tierra.png:



```
import pyvista
import math
import numpy as np
#creamos una esfera
esfera = pyvista.Sphere(radius=1, theta_resolution=120, phi_resolution=120,
                        start_theta=270.001, end_theta=270)
esfera.active_t_coords = np.zeros((esfera.points.shape[0], 2))
#definimos las ecuaciones UV para el mapeo de una esfera
esfera.active_t_coords[:, 0] = 0.5 + np.arctan2(-esfera.points[:, 0], esfera.points[:,
1])/(2 * math.pi)
esfera.active_t_coords[:, 1] = 0.5 + np.arcsin(esfera.points[:, 2]) / math.pi
#asignamos la textura y mostramos la imagen
tierra = pyvista.Texture("Tierra.png")
pl = pyvista.Plotter()
pl.add_mesh(esfera, texture=tierra, smooth_shading=False)
pl.show()
```

Salida obtenida:

F. Usando sus propias texturas cree un sistema solar similar a la imagen (10 esferas):



AYUDA:

Puede agregar una imagen de fondo con la funcion pl.add_background_image()