Geometría Computacional – Práctica 4

Miguel Manzano Rodríguez

1. Introducción (motivación y objetivo)

En la presente práctica de Geometría Computacional, nos enfocaremos en la realización de una transformación isométrica afín, realizando animaciones de familias paramétricas continuas que, a partir de figuras tridimensionales, realicen desplazamientos y rotaciones. El desplazamiento se determina mediante el diámetro mayor de la figura, lo que requiere una preparación previa de los datos para su posterior cálculo. Para ello, en ambos apartados se hará uso de la envoltura convexa. De esta manera, se explora la discretización de sistemas dinámicos en geometría diferencial para su análisis computacional, reconociendo la necesidad de transformaciones continuas en el procesamiento de información.

2. Material usado (método y datos)

Método

En este apartado, hemos empleado diversas librerías de Python para llevar a cabo las transformaciones isométricas afines y la generación de gráficos y animaciones. Específicamente, hemos utilizado las siguientes:

- **numpy**: Para manipulación de matrices y cálculos numéricos.
- **matplotlib.pyplot** y **matplotlib.animation**: Para la visualización de gráficos y la creación de animaciones.
- **mpl_toolkits.mplot3d.axes3d**: Para generar gráficos tridimensionales.
- **scipy.spatial.ConvexHull**: Para calcular la envoltura convexa de conjuntos de puntos.

Además, se ha empleado la librería **skimage** de scikit-image para la manipulación de imágenes digitales.

Datos

Los datos utilizados en esta práctica incluyen la figura tridimensional generada para la animación, así como la imagen digital 'hurricane-isabel.png' que se utiliza en el segundo apartado. También se han utilizado parámetros específicos proporcionados en el enunciado, como los ángulos de rotación (θ) y los vectores de translación (v) aplicados en las transformaciones isométricas.

3. Resultados

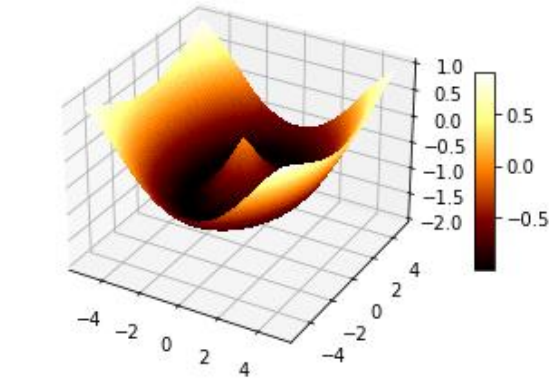
Apartado i) En el primer apartado, utilizamos la figura proporcionada, y generamos un GIF que representa la familia paramétrica continua. Se proporcionan por salida el diámetro y centroide calculados para realizar las rotaciones y traslaciones, como observamos en la figura 3-1. Asimismo, se puede observar en la figura 3-2 una representación de la figura a la que realizaremos la rotación y traslación en la dirección vertical.

```
 Centroide:  [-0.05       -0.05       -0.45735031]
Diametro:  9.9
```

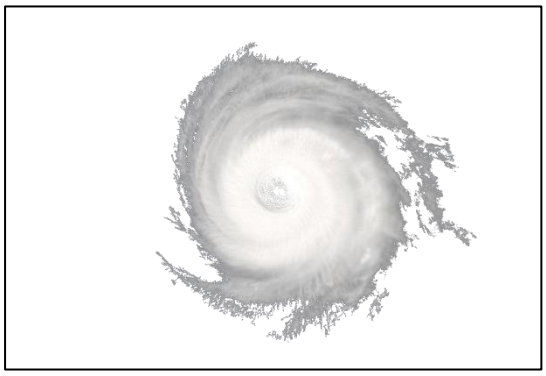
Ilustración 3-1: Centroide y diámetro apartado i)

```
(580, 860, 4)
Centroide:  [291.18396558 466.96062855   0.78153434]
Diametro:  522.3609866600563
```

3-3: Centroide y diámetro apartado ii)



3-2: Figura apartado i)



3-4: Imagen apartado ii)

Apartado ii) En este segundo apartado, se toma el subsistema para el color azul ≥ 100. En las ilustraciones superiores, podemos observar la imagen, así como, de nuevo, el diámetro y centroide de la misma. En ambos apartados, obtenemos como resultado los archivos GIF generados, que se adjuntan junto al código .py y esta memoria.

4. Conclusión

En conclusión, en esta práctica de Geometría Computacional, hemos podido explorar de manera práctica la realización de transformaciones isométricas afines, centrándonos en particular en la rotación de figuras en el espacio tridimensional. Una lección adquirida es que la elección de cómo están almacenados los datos puede facilitar significativamente la tarea, como en el caso de trabajar con matrices para representar coordenadas tridimensionales. Esto permite realizar la rotación de la figura con mayor eficacia.

En definitiva, la práctica ha sido de utilidad para ampliar conocimientos en cuanto al procesamiento y aplicación de conceptos de Geometría Computacional utilizando Python.

5. Anexo: Script/código utilizado

```
# -*- coding: utf-8 -*-
"""

@author: Miguel Manzano Rodríguez
"""

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import animation
from matplotlib import cm
from skimage import io, color
import math
# Importante: El módulo skimage requiere tener instalado:
# la librería scikit-image (por ejemplo con Anaconda o pip)

from scipy.spatial import ConvexHull

vuestra_ruta = ""

# FIGURA 1 DE EJEMPLO

fig = plt.figure(figsize=plt.figaspect(0.5))

ax = fig.add_subplot(1, 1, 1, projection='3d')

X = np.arange(-5, 5, 0.1)
Y = np.arange(-5, 5, 0.1)
X, Y = np.meshgrid(X, Y)
R = -np.sqrt(X**2/2 + Y**2/4)
Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.afmhot,
                       linewidth=0, antialiased=False)
ax.set_zlim(-2.01, 1.01)
fig.colorbar(surf, shrink=0.5, aspect=10)

plt.show()


"""
######################################################################
APARTADO 1: Familia paramétrica continua: rotación y traslación.
######################################################################
"""


xf = X.flatten()
yf = Y.flatten()
zf = Z.flatten()


centroide = np.array([np.mean(X), np.mean(Y), np.mean(Z)])

K = np.array([[X[i][j], Y[i][j], Z[i][j]] for j in range(len(X[0])) for i in
range(len(X))])

hull = ConvexHull(K)

distances = np.linalg.norm([(X[i] - X[j], Y[i] - Y[j], Z[i] - Z[j])
                            for i in range(len(X)) for j in range(len(X)) if i != j],
axis=1)
diametro = distances.max()
```

```
61.
62.
63. print("Centroide: ", centroide, "\nDiametro: ", round(diametro, 3));
64.
65. fig = plt.figure(figsize=(6,6));
66.
67. m0 = len(X[0])
68. n0 = len(X)
69.
70. def transformacionIso(x,y,z, M, v):
71.
72.     xt = x * 0
73.     yt = y * 0
74.     zt = z * 0
75.
76.     for i in range(len(x)):
77.         q = np.array([x[i], y[i], z[i]])
78.         xt[i], yt[i], zt[i] = np.matmul(M, q) + v
79.
80.     return xt, yt, zt
81.
82.
83.
84. def animate(t):
85.
86.     theta = 3*math.pi*t
87.
88.     M = np.array([[np.cos(theta), -np.sin(theta), 0],
89.                   [np.sin(theta), np.cos(theta), 0], [0, 0, 1]])
90.
91.     v = np.array([0, 0, diametro * t])
92.
93.     (xt, yt, zt) = transformacionIso(xf-centroide[0], yf-centroide[1],
94.                                      zf-centroide[2], M, v)
95.
96.     X = np.array([[xt[m0 * i + j] for j in range(m0)] for i in range(n0)])
97.     Y = np.array([[yt[m0 * i + j] for j in range(m0)] for i in range(n0)])
98.     Z = np.array([[zt[m0 * i + j] for j in range(m0)] for i in range(n0)])
99.     ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.afmhot,
100.                            linewidth=0, antialiased=False)
101.    return ax,
102.
103. def init():
104.    return animate(0),
105.
106.
107. fig = plt.figure(figsize=(10,10))
108. ax = fig.add_subplot(111, projection='3d')
109.
110. ani = animation.FuncAnimation(fig, animate, frames=np.arange(0,1,0.1),
111.                                init_func=init, interval=20)
112. ani.save("p4a.gif", fps = 10)
113.
114.
115. """
116. ###################################################################
117. APARTADO 2: Subsistema de azul y rotación y traslado de nuevo.
118. ###################################################################
119. """
120.
121. def transf1D(x,y,z,M, v=np.array([0,0,0])):
122.     xt = x*0
123.     yt = x*0
124.     zt = x*0
125.     for i in range(len(x)):
126.         q = np.array([x[i],y[i],z[i]])
127.         xt[i], yt[i], zt[i] = np.matmul(M, q) + v
128.     return xt, yt, zt
129.
130. """
```

```
Segundo apartado casi regalado

https://education.nationalgeographic.org/resource/coriolis-effect/
"""

img = io.imread('hurricane-isabel.png')

dimensions = img.data.shape
print(dimensions)
io.show()

fig = plt.figure(figsize=(5,5))
p = plt.contourf(img[:,:,2],cmap = plt.cm.get_cmap('afmhot'),
                 levels=np.arange(100,255,2))
plt.axis('off')

xyz = img.shape

x = np.arange(0,xyz[0],1)
y = np.arange(0,xyz[1],1)
xx,yy = np.meshgrid(x, y)
xx = np.asarray(xx).reshape(-1)
yy = np.asarray(yy).reshape(-1)
z  = img[:,:,2]
z = np.transpose(z)
zz = np.asarray(z).reshape(-1)

x0 = xx[zz>=100]
y0 = yy[zz>=100]
z0 = zz[zz>=100]/zz.max()

centroide = np.array([np.mean(x0), np.mean(y0), np.mean(z0)])

print(centroide)

X = np.array([x0,y0,z0]).T
hull = ConvexHull(X)

diametro = 0

for i in hull.vertices:
    for j in hull.vertices:
        if math.dist(X[i], X[j]) > diametro:
            diametro = math.dist(X[i], X[j])

print("Centroide: ", centroide, "\nDiametro: ", diametro);


def animate2(t):
    theta = 6 * math.pi * t;
    M = np.array([[np.cos(theta), -np.sin(theta), 0],
                  [np.sin(theta), np.cos(theta), 0],
                  [0, 0, 1]])
    v=np.array([diametro,diametro,0])*t

    print(t)

    ax = plt.axes(xlim=(0,1200), ylim=(0,1200))

    XYZ = transf1D(x0, y0, z0, np.array([[1,0,0], [0,1,0], [0,0,1]]),
                   v=centroide*(-1))
    XYZ = transf1D(XYZ[0], XYZ[1], XYZ[2], M=M, v = centroide)
    XYZ = transf1D(XYZ[0], XYZ[1], XYZ[2],np.array([[1,0,0], [0,1,0],
                                                    [0,0,1]]), v=v)

    col = plt.get_cmap("afmhot")(np.array(0.1+z0))
    ax.scatter(XYZ[0],XYZ[1],c=col,s=0.1,animated=True)
    return ax,

def init2():
```

```
    return animate2(0),

fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, animate2, frames=np.arange(0,1,0.1),
                              init_func=init2, interval=20)

ani.save("p4b.gif", fps = 10)
```