PRÁCTICA 4 – GEOMETRÍA COMPUTACIONAL – 2023

PABLO FERNÁNDEZ DEL AMO

1. INTRODUCCIÓN:

La siguiente práctica se enfoca en la transformación isométrica afín correspondiente a una rotación y una translación aplicada sobre varios sistemas con distinto número de variables de estado..

2. MATERIAL USADO:

Para resolver el problema se utilizaron conceptos de geometría como la transformación isométrica afín, la métrica euclídea, el cálculo del diámetro y centroide de un conjunto de puntos.

Se utilizará la métrica euclídea y se generará una animación que reproduzca la transformación simultánea de una rotación en torno al centroide del sistema y una translación sobre la figura en 3D.

Para realizar esta animación de la evolución de la transformación utilizaremos la matriz de rotación M(θ·t) y el vector de translación v(d·t), dando valores a t en el intervalo [0,1], pudiendo así representar está evolución – siendo d el diámetro mayor del sistema, calculado a partir de la envolvente convexa por optimización de la eficiencia del código.

Además, se aplicará la misma transformación otro sistema representado por la imagen digital 'arbol.png', donde se calculará el centroide y el diámetro mayor de la hoja verde del árbol.

El objetivo de esta práctica es aplicar los conceptos de transformaciones isométricas afines para visualizar y manipular objetos en tres dimensiones.

En Python, se utilizaron las siguientes bibliotecas y módulos: NumPy: para el manejo de matrices y vectores. Matplotlib: para la creación de gráficas, visualización de datos y animaciones (FuncAnimation), Scikit-image: para cargar la imagen 'arbol.png' y realizar operaciones de procesamiento de imagen, Scipy-Spatial: Para calcular la envolvente convexa y el diámetro del conjunto de puntos, y mpl_toolkits.mplot3d: Para la generación de una figura 3Dinicial. Las ideas giran en torno al código proporcionado por Robert en la plantilla ‘GCOM2023-Practica4_plantilla’.

3. RESULTADOS:

i) Genera una figura en 3 dimensiones (puedes utilizar la figura 1 de la plantilla) y realiza una animación de una familia paramétrica continua que reproduzca desde la identidad hasta la transformación simultánea de una rotación de θ = 3π y una translación con v = (0,0,d), donde d es el diámetro mayor de S.

La *Ilustración 1* muestra la figura en 3 dimensiones pedida en su instante inicial, es decir, cuándo aún no se ha llevado a cabo la transformación simultánea pedida.

En cambio, en la *Ilustración 2* se puede ver claramente como esta transformación ha sido ejecutada, representando la figura inicial después de ser sometida a una rotación de θ = 3π sobre el plano OXY y una translación con v = (0,0,d), siendo este diámetro mayor.

La animación de esta transformación se encuentra en el archivo ‘animacion_apartado1.gif’, adjunto a esta memoria.

El centroide no es costoso de calcular al tratarse del punto medio a todos los puntos del sistema. En cambio, para calcular el diámetro mayor hemos tenido que obtener primero la envolvente convexa y obtener el diámetro mayor del conjunto formado por los vértices, siendo este el mismo que el buscado y reduciendo enormemente la cantidad de puntos involucrados y, por tanto, la complejidad computacional.

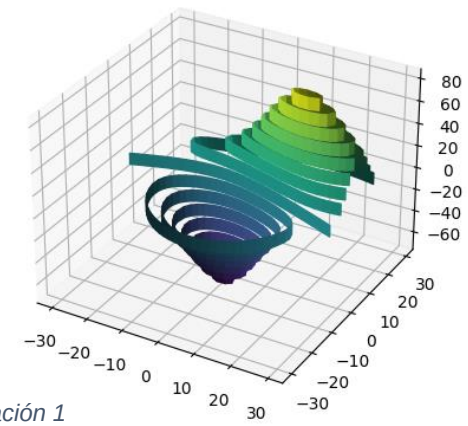

Ilustración 1

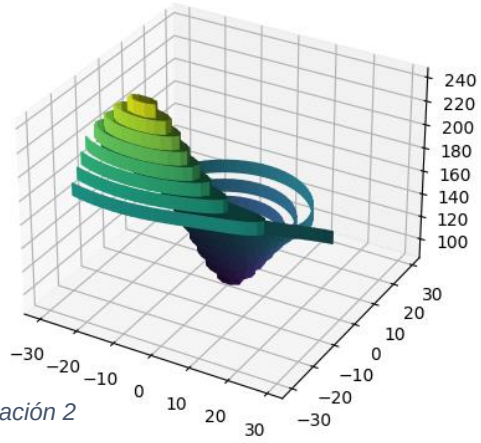

Ilustración 2

ii) Dado el sistema representado por la imagen digital 'arbol.png', considera el subsistema σ dado por el segundo color (verde) cuando verde < 240. ¿Dónde se sitúa el centroide? Realiza la misma transformación que en el apartado anterior con θ = 3π y v = (d,d,0), donde d es el diámetro mayor de σ.

El diámetro mayor del sistema representado por la imagen digital 'arbol.png' - *Ilustración 3* - es 357.4144 unidades de distancia.

Por su parte, el centroide se encuentra, considerando solo las variables de posición (es decir: X, Y y Z) en el punto (173.4819, 204.1563, 0). La componente espacial Z es 0, tanto desde el inicio de la transformación - identidad : *Ilustración 4* - hasta el final - rotación y translación simultánea : *Ilustración 5*; ya que 'árbol.png' es una imagen 2D, y la transformación se realiza únicamente sobre el plano OXY.

Cabe resaltar que para la realización de esta animación se ha utilizado como base la plantilla proporcionada por Robert, 'GCOM2023-Practica4_plantilla'; utilizando las mimas ideas para restringir el sistema de la imagen 2D al subsistema σ pedido con verde < 240.

La animación de esta transformación 'animacion_apartado2.gif' se encuentra junto a esta memoria.

Ilustración 3

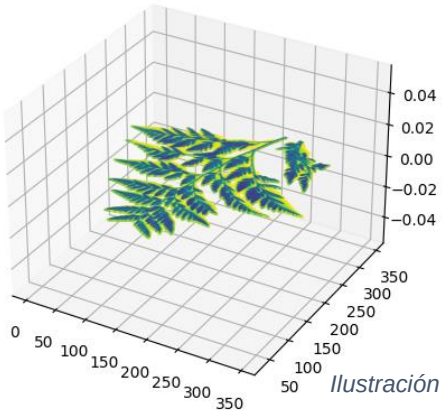

Ilustración 4

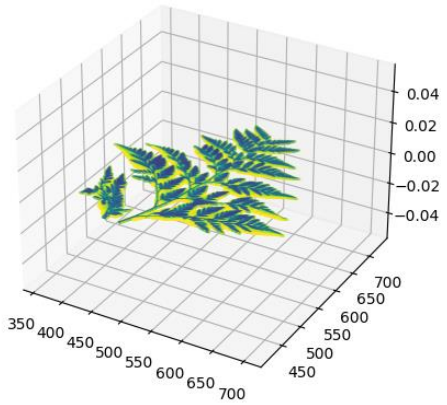

Ilustración 5

4. CONCLUSIÓN:

En conclusión, en esta práctica se ha trabajado con transformaciones geométricas isométricas afines en un espacio euclídeo de tres dimensiones.

En ambos apartados, se ha utilizado una translación y rotación simultánea para mover los elementos del sistema manteniendo sus distancias, ángulos y áreas. Este tipo de transformación es especialmente útil en aplicaciones de procesamiento de imágenes, como en el caso del análisis de la forma de hojas, para reducir las diferencias en la posición y escala de los elementos del sistema.

En general, se ha podido comprobar que las transformaciones afines isométricas son un conjunto de herramientas poderosas para el análisis y procesamiento de sistemas en un espacio euclídeo de tres dimensiones, permitiendo la creación de animaciones y visualizaciones útiles para la comprensión de los sistemas en estudio.

5. ANEXO:

Código de Python utilizado:

```
import os
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import animation

from scipy.spatial import ConvexHull
from scipy.spatial import distance
from skimage import io, color

"""
PRÁCTICA 4: GEOMETRÍA COMPUTACIONAL

PABLO FERNÁNDEZ DEL AMO   04231435X
"""

# define functions to be used later

# calculate centroid of a set of points
def calcularCentroide(S):
    return np.mean(S,axis=0)

# calculate diameter of a set of points
# this function takes long to execute when considering large number of points, so we obtain
the diameter from its convex hull (the same one)
def calcularDiametro(S):
    diameter = 0
    hull = ConvexHull(S)
    for i in range(len(hull.vertices)):
        for j in range(i+1,len(hull.vertices)):
            distance_max = distance.euclidean(S[hull.vertices[i]], S[hull.vertices[j]])
            if distance_max > diameter:
                diameter = distance_max
    return diameter

# apply transformation on a set of points
# M is a matrix representing rotation and scaling
# v is a vector representing translation
# centroid is the center of the set of points
def transformation(S, M, centroid, v):
    return centroid + (np.matmul(M, (S - centroid).T)).T + v

# apply rotation and translation on a set of points
# theta is the angle of rotation
# num determines the type of translation
# S is the set of points
# centroid is the center of the set of points
# diameter is the diameter of the set of points
def rotation_translation(S, centroid, diameter, theta, num):
    M =  np.array([[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], \
                                [0, 0, 1]])
```

```
    if num == 1:  # for problem I
        v = np.array([0, 0, diameter])
    else:  # for problem
        v = np.array([diameter, diameter, 0])

    return transformation(S, M, centroid, v)

# animate first set of points (APARTADO I)
def animate1(t, S, centroid, diameter, theta, shape):
    St = rotation_translation(S, centroid, t * diameter, t * theta, num = 1)

    ax = plt.axes(projection='3d')
    ax.set_xlim(-30,30)
    ax.set_ylim(-30,30)
    ax.set_zlim(-60,240)
    cset = ax.contour(St[:, 0].reshape(shape), St[:, 1].reshape(shape), St[:,
2].reshape(shape),\
                      16, extend3d=True,cmap = plt.cm.get_cmap('viridis'))
    ax.clabel(cset, fontsize=9, inline=1)



    return ax,

# animate second set of points (APARTADO II)
def animate2(t, S, centroid, diameter, theta):
    St = rotation_translation(S, centroid, t * diameter, t * theta, num = 2)

    ax = plt.axes(projection='3d')
    ax.set_xlim(0,700)
    ax.set_ylim(0,700)
    ax.set_zlim(-0.04,0.04)
    col = plt.get_cmap("viridis")(np.array(0.1+S[:,2]))

    ax.scatter(St[:, 0], St[:, 1],c=col,s=0.1,animated=True)
    return ax,

"""
APARTADO 1
"""
def apartado1():
    print ('APARTADO I:')
    theta = 3 * np.pi
    X, Y, Z = axes3d.get_test_data(0.05)
    shape = X.shape
    S = np.column_stack([X.flatten(),Y.flatten(),Z.flatten()])
    diameter = calcularDiametro(S)
    centroid = calcularCentroide(S)

    # Obtenemos la figura en inicio:

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    cset = ax.contour(X, Y, Z, 16, extend3d=True,cmap = plt.cm.get_cmap('viridis'))
    ax.clabel(cset, fontsize=9, inline=1)
```

```
    fig.savefig('Inicio.png',format='png')
    plt.show()

    St = rotation_translation(S, centroid, diameter, theta, num = 1)

    # Y la final

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    cset = ax.contour(St[:, 0].reshape(shape), St[:, 1].reshape(shape), St[:,
2].reshape(shape),\
                      16, extend3d=True,cmap = plt.cm.get_cmap('viridis'))
    ax.clabel(cset, fontsize=9, inline=1)
    fig.savefig('Final.png',format='png')
    plt.show()

    # Creamos la animación
    fig = plt.figure(figsize=(6,6))
    ani = animation.FuncAnimation(fig, animate1,frames=np.arange(0,1.01,0.05),
                                  fargs = (S,centroid, diameter, theta, shape), interval=20)
    ani.save('animacion_limited1.gif',fps=5)
    plt.show()



"""
Apartado II
"""
def apartado2():
    print ('APARTADO II:')
    os.getcwd()
    img = io.imread('arbol.png')
    #dimensions = color.guess_spatial_dimensions(img)
    #print(dimensions)
    #io.show()
    #io.imsave('arbol2.png',img)


    fig = plt.figure(figsize=(5,5))
    p = plt.contourf(img[:,:,0],cmap = plt.cm.get_cmap('viridis'), levels=np.arange(0,240,2))
    fig.savefig('Hoja1.png',format='png')
    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[:,:,0]
    zz = np.asarray(z).reshape(-1)


    #Consideraremos sólo los elementos con zz < 240
```

```
    #Variables de estado coordenadas
    x0 = xx[zz<240]
    y0 = yy[zz<240]
    z0 = zz[zz<240]/256.


    #Variable de estado: color
    col = plt.get_cmap("viridis")(np.array(0.1+z0))

    fig = plt.figure(figsize=(5,5))
    ax = fig.add_subplot(1, 2, 1)
    plt.contourf(x,y,z,cmap = plt.cm.get_cmap('viridis'), levels=np.arange(0,240,2))
    ax = fig.add_subplot(1, 2, 2)
    plt.scatter(x0,y0,c=col,s=0.1)
    fig.savefig('Hoja2.png',format='png')
    plt.show()


    #Calculamos el diametro de la hoja para hacer la transformación
    S = np.column_stack([x0, y0, z0])
    diameter = calcularDiametro(S)
    print(f'Diámetro: {diameter} ')

    centroid = calcularCentroide(S)
    #Calculamos el centroide de la hoja
    print(f"centroide: {centroid} ")

    theta = 3 * np.pi

    #Generamos la animación
    fig = plt.figure(figsize=(6,6))
    ani = animation.FuncAnimation(fig, animate2, frames=np.arange(0,1,0.025), fargs =
(S,centroid, diameter, theta), \
                                  interval=20)
    #os.chdir(vuestra_ruta)
    ani.save("Hoja_arbol_limited1.gif", fps = 10)
    os.getcwd()


apartado1()
apartado2()
```