Íñigo Iriondo Delgado
Geometría Computacional

PRÁCTICA 4

(1) Introducción

El objetivo de la práctica es, primero, generar una figura tridimensional con una animación paramétrica continua que hace una rotación de 3π rad alrededor del centroide y una traslación a lo largo del eje z. Después, analizamos el subsistema del color verde de la imagen arbol.png y efectuamos una transformación como la primera.

(2) Material usado

Utilizamos las librerías math, os, numpy, matplotlib.pyplot, mpl_toolkits.mplot3d, axes3d, animation matplotlib e io de skimage.

Apartado i

- transf1D: transformación isométrica afín en una dimensión.
- Se crea la figura en tres dimensiones con plt.axes y un gráfico de contorno con ax.contour.
- animate1: hace la animación de la transformación de la rotación de 3π rad y traslación a lo largo del eje z.
- FuncAnimation: crea la animación y se guarda en "p7a.gif".

Apartado ii

- Se hace un contorno en la imagen en el canal de color verde de la imagen con contourf.
- Diam2: calcula el diámetro mayor.
- centroide: calcula el centroide.
- transf2D: realiza una transformación en dos dimensiones similar a la del primer apartado en el subsistema de la imagen.
- Se visualiza un gráfico de contorno y de dispersión con plt.contourf y plt.scatter.
- animate2: hace la animación la transformación.
- FuncAnimation: crea la animación y se guarda en "p7b.gif".
- Calculamos el centroide filtrado.

(3) Resultados

Obtenemos ambas animaciones descritas y la ubicación del centroide filtrado: [173.48190476 204.15631103].

Observación: si no se quiere ajustar los ejes para que no se salga la figura al moverse se tiene que quitar la línea:
ax.autoscale(enable=True) # Ajuste automático de los ejes

(4) Conclusión

Hemos aplicado transformaciones isométricas afines a una figura 3D y al canal verde de una imagen que visualizamos con las animaciones en tiempo real. También hemos calculado el centroide de los puntos filtrados en la imagen, lo cual es útil para entender la distribución de los datos.

(5) Anexo con el script

Código compartido con David Diez Roshan.

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

# Apartado 1

def transf1D(x, y, z, M, v=np.array([0, 0, 0])):
    xt = x * 0
    yt = y * 0
    zt = z * 0
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            q = np.array([x[i, j], y[i, j], z[i, j]])
            xt[i, j], yt[i, j], zt[i, j] = np.matmul(M, q) + v

    return xt, yt, zt

fig = plt.figure()
ax = plt.axes(projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
cset = ax.contour(X, Y, Z, 16, extend3d=True, cmap=plt.cm.get_cmap('viridis'))
ax.clabel(cset, fontsize=9, inline=1)
plt.show()
```

```
def animate1(t):
    d = np.max([X.max()-X.min(), Y.max()-Y.min(), Z.max()-Z.min()])
    theta = 3 * math.pi
    M = np.array([[np.cos(theta), -np.sin(theta), 0],
                  [np.sin(theta), np.cos(theta), 0],
                  [0, 0, 1]])
    v = np.array([0, 0, d * t])
    ax = plt.axes(xlim=(X.min() - d, X.max() + d), ylim=(Y.min() - d, Y.max() + d),
zlim=(Z.min(), Z.max() + d), projection='3d')

    XYZ = transf1D(X, Y, Z, M, v)
    ax.contour(XYZ[0], XYZ[1], XYZ[2], 16, extend3d=True,
cmap=plt.cm.get_cmap('viridis'))

    return ax

def init1():
    return animate1(0)

fig = plt.figure(figsize=(6, 6))
ani = animation.FuncAnimation(fig, animate1, frames=np.arange(0, 1, 0.025),
init_func=init1,
                              interval=20)
ani.save("p7a.gif", fps=10)

# Apartado 2

img = io.imread('arbol.png')
fig2 = plt.figure(figsize=(5, 5))

p = plt.contourf(img[:, :, 0], cmap=plt.cm.get_cmap('viridis'), levels=np.arange(0, 240, 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[:, :, 0]
zz = np.asarray(z).reshape(-1)
```

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

def diam2(x, y):
    lista = []
    for i in range(0, len(x), 10):
        p = np.array([x[i], y[i]])
        for j in range(0, len(y), 10):
            d = math.sqrt((p[0] - x[j])**2 + (p[1] - y[j])**2)
            lista.append(d)
    return max(lista)

def centroide(x, y):
    x1 = sum(x0) / len(x0)
    y1 = sum(y0) / len(y0)

    return [x1, y1]

def transf2D(x, y, z, M, v):
    xt = x0 * 0
    yt = y0 * 0
    zt = z0 * 0
    centroid = centroide(x, y)
    for i in range(len(x)):
        q = np.array([x[i] - centroid[0], y[i] - centroid[1], z[i]])
        xt[i], yt[i], zt[i] = np.matmul(M, q) + v

    return xt, yt, zt


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 = fig2.add_subplot(1, 2, 2)
plt.scatter(x0, y0, c=col, s=0.1)
plt.show()

def animate2(t):
    d = diam2(x, y)  # Diámetro de la segunda imagen
    theta = 3 * 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([d, d, 0]) * t
    ax = plt.axes(xlim=(0, 400), ylim=(0, 400), projection='3d')
    XYZ = transf2D(x0, y0, z0, M, v)
    col = plt.get_cmap("viridis")(np.array(0.1 + XYZ[2]))
    ax.scatter(XYZ[0], XYZ[1], c=col, s=0.1, animated=True)
    ax.autoscale(enable=True)  # Ajuste automático de los ejes
    return ax,

def init2():
    return animate2(0),

animate2(np.arange(0.1, 1, 0.1)[5])
plt.show()

fig = plt.figure(figsize=(6, 6))
ani2 = animation.FuncAnimation(fig, animate2, frames=np.arange(0, 1, 0.025),
init_func=init2,
                               interval=20)

ani2.save("p7b.gif", fps=10)

# Calcular el centroide para los puntos filtrados
centroid_filtered = np.array([np.mean(x0), np.mean(y0)])
print("Ubicación del centroide filtrado: ", centroid_filtered)
```