Práctica 7: Transformación isométrica afín

Guillermo Martín Sánchez

15/05/20

1 Introducción

Dado un sistema $S = \{a_j, \{x^j, y^j, ...\}\}_{j=1}^{N}$, el objetivo de esta práctica es la implementación de una rotación $R_\theta^{(xy)}$ aplicada entorno al centroide del sistema y con una translación $v = (v_1, v_2, ...)$

2 Material usado

2.1 Ejercicio 1

Modifica 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 $\theta = 3\pi$ y una translación con $v = (d, d, 0)$, donde d es el diámetro mayor de S.

Dado el sistema S lo podemos entender como un subconjunto del espacio afín donde hay N puntos que tienen coordenadas las variables de estado ($p_j = (x^j, y^j, ...)$). Así visto, calculamos el centroide como:

$$C = \frac{1}{N}\sum_{j=1}^{N} p_j = (\bar{x}, \bar{y}, ...) \tag{1}$$

Por otro lado para calcular el diámetro del conjunto usamos la fórmula:

$$d = \max_{i,j=1,...,N} ||p_i - p_j|| \tag{2}$$

El cálculo del diámetro es, usando esta fórmula, de orden $\mathcal{O}(n^2)$. Por ello para facilitarlo usamos que el diámetro de un conjunto S es igual al diámetro de su envolvente convexa (`https://people.scs.carleton.ca/~michiel/lecturenotes/ALGGEOM/diameter.pdf`). Para calcular la envolvente convexa usamos la clase *scipy.spatial.ConvexHull*.

Una vez tenemos el ángulo de rotación $\theta = 3\pi$, el centro de rotación (el centroide) (c_x, c_y, c_z) y el vector de translación $v = (d, d, 0)$, expresamos la rotación en el tiempo t como:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \mathcal{R}_\theta^{(x,y)}(t) \begin{pmatrix} x - c_x \\ y - c_y \\ z - c_z \end{pmatrix} + tv = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} R_\theta^{(x,y)}(t) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x - c_x \\ y - c_y \\ z - c_z \end{pmatrix} + t \begin{pmatrix} d \\ d \\ 0 \end{pmatrix} =$$
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} cos(t\theta) & -sin(t\theta) & 0 \\ sin(t\theta) & cos(t\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x - c_x \\ y - c_y \\ z - c_z \end{pmatrix} + t \begin{pmatrix} d \\ d \\ 0 \end{pmatrix} \tag{3}$$

En concreto nosotros hemos animado la Botella de Klein en 3 dimensiones, cuyas ecuaciones se representan en la siguiente figura:

$$v = 0 \rightarrow 2\pi$$
$$u = 0 \rightarrow 2\pi$$
$$r = 4\,(1 - \cos(u) / 2)$$
$$x = \begin{cases} 6\cos(u)\,(1 + \sin(u)) + r\cos(u)\cos(v) & 0 \le u < \pi \\ 6\cos(u)\,(1 + \sin(u)) + r\cos(v + \pi) & \pi < u \le 2\pi \end{cases}$$
$$y = \begin{cases} 16\sin(u) + r\sin(u)\cos(v) & 0 \le u < \pi \\ 16\sin(u) & \pi < u \le 2\pi \end{cases}$$
$$z = r\sin(v)$$

Ecuaciones de la Botella de Klein en 3 dimensiones (`http://paulbourke.net/geometry/toroidal/`)

2.2 Ejercicio 2

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

Nos hemos restringido al subsistema que tiene $rojo < 240$. Ahí hemos calculado el diámetro de dos maneras: considerando el sistema en 3 dimensiones (donde el color es una dimensión más) y considerando el sistema sólo en 2 dimensiones. La animación la hemos hecho usando la Eq 3 de la misma forma que la anterior con el diámetro en 2 dimensiones.

3 Resultados y discusión

3.1 Ejercicio 1

En la figura adjunta *p7a.gif* se puede ver el resultado de aplicar la transformación a la Botella de Klein desde $\theta = 0$ hasta $\theta = 3\pi$

3.2 Ejercicio 2

En la Figura 1 vemos la imagen entendida como un sistema en 3 dimensiones. El centroide en cyan y los puntos rojos son los puntos de la envolvente convexa. Hay que ver que efectivamente en este caso los puntos de la envolvente convexa se ponen "encima" de la imagen. Esto es porque hay que tener en cuenta que se encuentran en una dimensión extra. En este caso obtenemos un diámetro de ≈ 357.41.

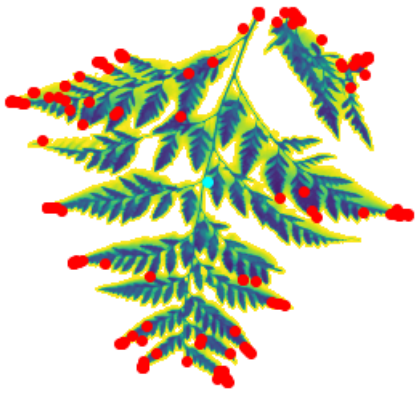

Figure 1: Sistema entendido en 3D. En cyan el centroide y en rojo los vértices de la envolvente convexa

En la Figura 2 vemos la imagen entendida como un sistema en 2 dimensiones. El centroide, como era de esperar, puesto que en el caso anterior lo proyectábamos en dos dimensiones, se encuentra en el mismo punto. Ahora los puntos rojos de los vértices de la envolvente convexa sí que son los intuitivamente esperados. Finalmente una cosa interesante es que el diámetro también es ≈ 357.41, luego probablemente los puntos más alejados entre sí se encuentran a la misma altura en la tercera dimensión, es decir, tienen el mismo color.

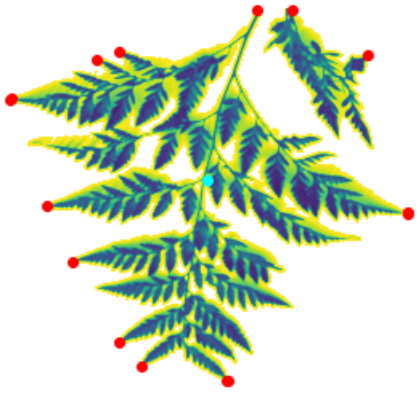

Figure 2: Sistema entendido en 2D. En cyan el centroide y en rojo los vértices de la envolvente convexa

Finalmente, en la figura adjunta *p7b.gif* vemos la animación de la rotación de la imagen desde $\theta = 0$ hasta $\theta = 3\pi$

4 Conclusión

Hemos estudiado varios métodos de geometría computacional: el cálculo del centroide, la envolvente convexa, y el diámetro de un conjunto de puntos. Además, hemos animado una transformación isométrica afín aplicada sobre dos sistemas diferentes.

5 Código

```
# -*- coding: utf-8 -*-
"""
Coursework 7: Isometries
"""

import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull
from mpl_toolkits.mplot3d import axes3d
from matplotlib import animation
from matplotlib import cm
from skimage import io, color
```

```
# Compute the isometric transformation given by matrix M around center c
# with traslation v with coordenate vectors x,y,z
def transf1D(x,y,z,M, v=np.array([0,0,0]), c=np.array([0,0,0])):
    xt = np.empty(len(x))
    yt = np.empty(len(x))
    zt = np.empty(len(x))
    for i in range(len(x)):
        q = np.array([x[i],y[i],z[i]])
        xt[i], yt[i], zt[i] = np.matmul(M, q - c) + v + c
    return xt, yt, zt

# Compute the isometric transformation given by matrix M around center c
# with traslation v with coordenate matrices X,Y,Z
def transf2D(X,Y,Z,M, v=np.array([0,0,0]), c=np.array([0,0,0])):
    Xt = np.zeros_like(X)
    Yt = np.zeros_like(Y)
    Zt = np.zeros_like(Z)
    m = np.shape(X)[1]
    for i in np.arange(0,m):
        for j in np.arange(0,m):
            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 - c) + v + c
    return Xt, Yt, Zt

# Compute the maximum diameter of a set of points with coordinates x,y,z
def diameter(x,y,z):
    diam = 0
    m = len(x)
    for i in np.arange(0,m):
        for j in np.arange(0,m):
            newdiam = (x[i]-x[j])**2 + (y[i]-y[j])**2 + (z[i] - z[j])**2
            if diam < newdiam:
                diam = newdiam
    return np.sqrt(diam)

# Compute the centroid of a set of points with coordinates x,y,z
def centroid(x,y,z):
    xmean = np.mean(x)
    ymean = np.mean(y)
    zmean = np.mean(z)
    return xmean,ymean,zmean

#%%
"""
Exercise 1: Surface isometry
```

```
"""

vuestra_ruta = "C:/Users/guill/Documents/Carrera/GEOComp/Documentos/7-Isometries"

os.getcwd()
os.chdir(vuestra_ruta)


# Klein bottle parametrisation
cos = np.cos
sin = np.sin
sqrt = np.sqrt
pi = np.pi
def surf(u, v):
    """
    http://paulbourke.net/geometry/toroidal/
    """
    half = (0 <= u) & (u < pi)
    r = 4*(1 - cos(u)/2)
    x = 6*cos(u)*(1 + sin(u)) + r*cos(v + pi)
    x[half] = (
        (6*cos(u)*(1 + sin(u)) + r*cos(u)*cos(v))[half])
    y = 16 * sin(u)
    y[half] = (16*sin(u) + r*sin(u)*cos(v))[half]
    z = r * sin(v)
    return x, z, -y

u, v = np.linspace(0, 2*pi, 40), np.linspace(0, 2*pi, 40)
ux, vx =  np.meshgrid(u,v)
X, Y, Z = surf(ux, vx)


# Compute the diameter of the set via computing the convex hull
x = np.reshape(X,(-1,1))
y = np.reshape(Y,(-1,1))
z = np.reshape(Z,(-1,1))
hull = ConvexHull(np.concatenate((x,y,z),axis=1))
diam = diameter(x[hull.vertices],y[hull.vertices],z[hull.vertices])

# Compute the centroid
cx,cy,cz = centroid(x,y,z)


# For each t compute the image at time t
def animate(t):
    th = 3*np.pi
```

```
    M = np.array([[np.cos(t*th),-np.sin(t*th),0],[np.sin(t*th),np.cos(t*th),0],[0,0,1]])
    v=np.array([diam,diam,0])*t
    c = np.array([cx,cy,cz])

    ax = plt.axes(xlim=(-10,50), ylim=(-10,50), zlim=(-20,20), projection='3d')

    Xt,Yt,Zt = transf2D(X, Y, Z, M=M, v=v, c=c)
    ax.plot_surface(Xt, Yt, Zt, cmap=cm.coolwarm)
    return ax,

def init():
    return animate(0),


# Compute the animation of the transformation
fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, animate, frames=np.arange(0,1,0.025), init_func=init,
                              interval=20)
os.chdir(vuestra_ruta)
ani.save("p7a.gif", fps = 10)
os.getcwd()
#%%
"""
Exercise 2: Image isometry
"""

# Download the image
img = io.imread('arbol.png')
dimensions = color.guess_spatial_dimensions(img)
print(dimensions)
io.show()

xyz = img.shape

x = np.arange(0,xyz[0],1)
y = np.arange(0,xyz[1],1)
xx,yy = np.meshgrid(x, y)
zz  = img[:,:,0]



# Consider elements with zz < 240
x0 = xx[zz<240]
y0 = yy[zz<240]
z0 = zz[zz<240]/256.
```

```
x1 = np.reshape(x0,(-1,1))
y1 = np.reshape(y0,(-1,1))
z1 = np.reshape(z0,(-1,1))



# Compute the centroid
cx,cy,cz = centroid(x0,y0,z0)

# Compute the diameter in 2D and plot convex hull points
hull2D = ConvexHull(np.concatenate((x1,y1),axis=1))
fig = 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')
plt.scatter(x1[hull2D.vertices],y1[hull2D.vertices],color='r')
plt.scatter(cx,cy,color='cyan')
diam2D = diameter(x1[hull2D.vertices],y1[hull2D.vertices],
                  np.zeros_like(x1[hull2D.vertices]))

# Compute the diameter in 3D and plot convex hull points
hull3D = ConvexHull(np.concatenate((x1,y1,z1),axis=1))
fig = 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')
plt.scatter(x1[hull3D.vertices],y1[hull3D.vertices],color='r')
plt.scatter(cx,cy,color='cyan')
diam3D = diameter(x1[hull3D.vertices],y1[hull3D.vertices],z1[hull3D.vertices])

# Use diam in 2D or 3D
diam = diam2D

# For each t compute the image at time t
def animate(t):
    th = 3*np.pi
    M = np.array([[np.cos(t*th),-np.sin(t*th),0],[np.sin(t*th),np.cos(t*th),0],[0,0,1]])
    v = np.array([diam,diam,0])*t
    c = np.array([cx,cy,cz])

    ax = plt.axes(xlim=(100,600), ylim=(100,600), projection='3d')
    #ax.view_init(60, 30)

    XYZ = transf1D(x0, y0, z0, M=M, v=v, c=c)
    col = plt.get_cmap("viridis")(np.array(0.1+XYZ[2]))
    ax.scatter(XYZ[0],XYZ[1],c=col,s=0.1,animated=True)
    return ax,
```

```
def init():
    return animate(0),

# Compute the animation of the transformation
fig = plt.figure(figsize=(6,6))
ani = animation.FuncAnimation(fig, animate, frames=np.arange(0,1,0.025), init_func=init,
                              interval=20)
os.chdir(vuestra_ruta)
ani.save("p7b.gif", fps = 10)
os.getcwd()
```