Task 1 – Global Thresholding

Introduction and Objective

The first task of the project consists on writing a program that implements an iterative thresholding. The program will be used to segment a given image. The algorithm method is defined in the next points:

1. Select an initial estimate for T
2. Segment the image using T
3. Compute averages of G1 and G2
4. Compute the new threshold value Tnew = 0.5*(m1+m2)
5. Repeat steps 2 thru 4 until the change of T is sufficiently small

Results

First, we will see how the histogram of the image looks like:

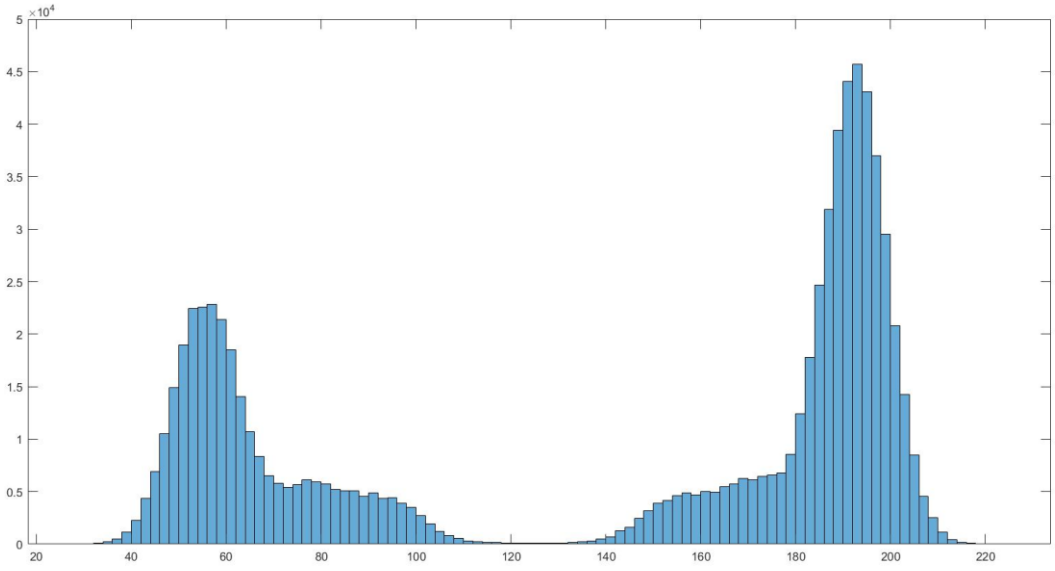

Figure 1 Histogram of first image

It has two main high areas with a valley inside. Moreover, the values are spread into all of the grey levels so performing segmentation of the image should be quite easy.

The program plots the original image and the segmented image:

Figure 2 Global Thresholding. Raw image on the right and segmented image on the left

The grey background is removed, and a binary image of a fingerprint is shown after applying the auto-threshold method, which is a clearer image without so much noise.

Source code

```
function [g,T] = AutoThresholding(f)
    To = 1e-18; % Min change of T
    Tnew = 255*rand(1); % Randomly pick the starting T
    dT = 1000; % A high value

    hist=imhist(f); % Compute the histogram
    N=sum(hist);
    for i=1:256
        P(i)=hist(i)/N; % Compute the cumulative sums
    end

    while dT>To
        Told = Tnew;

        mean1 = sum([0:Told-1] .* P(1:Told));
        mean2 = sum([Told:255] .* P(Told+1:256));

        Tnew = 0.5*(mean1+mean2);
        dT = abs(Told-Tnew);
    end
    T = Tnew;

    g = zeros(size(f));
    g(find(f>=T)) = 255;
    g(find(f<T)) = 0;
```

```
end
```

Task 2 – Otsu's Thresholding

Introduction and Objective

The objective of the second task of the project is to implement Otsu's optimum thresholding algorithm from scratch.

Otsu's algorithm consists on following this procedure:

1. Compute the normalized histogram of the input image
2. Compute the cumulative sums P(k)
3. Compute the cumulative means m(k)
4. Compute the global intensity mean mg
5. Compute the between-class variance sigma(k)
6. Obtain the Otsu threshold Kopt as the value of k with maximum sigma.

Results

As in the previous section, we will analyze the histogram of the objective image (polymersomes.tiff):

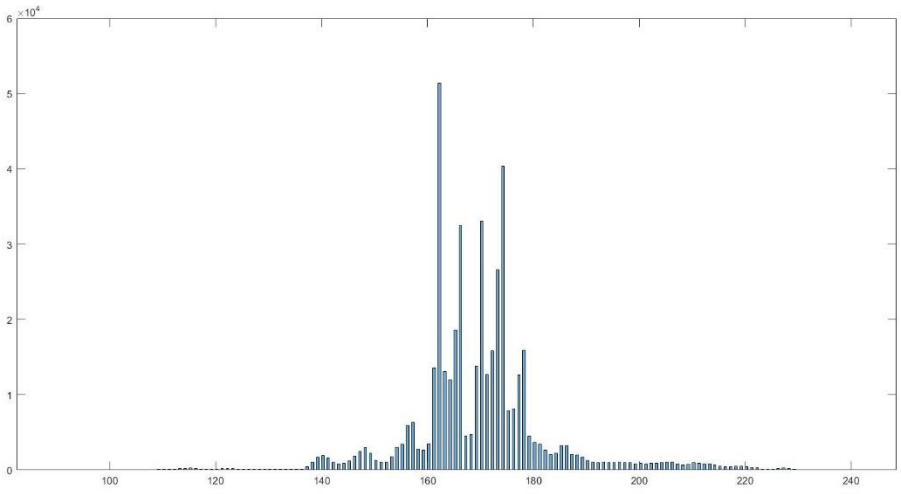

Figure 3 Histogram of the second image

This time the histogram is narrower and that means that image contrast is quite low. Most of the pixel intensities range from 160 to 180.

Next plots show the raw image next to the one segmented using Otsu's thresholding method:

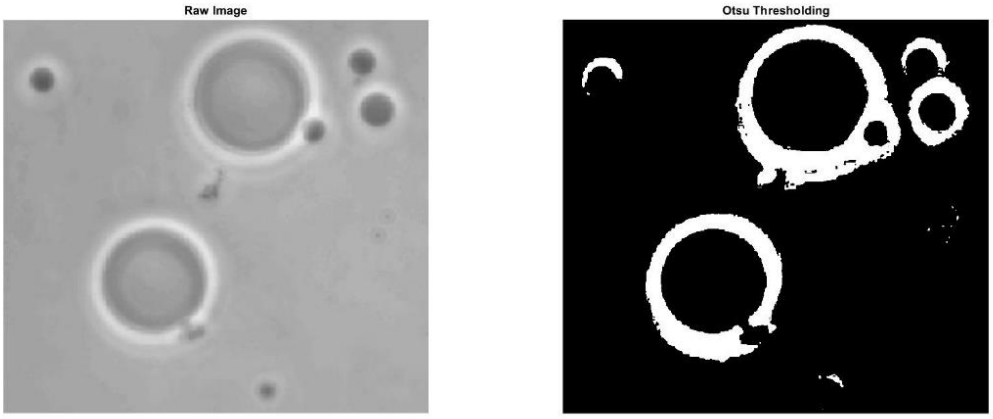

Figure 4 Otsu's Thresholding. Raw and thresholded images.

This method allows us to segment those circles, which will be not possible using general thresholding.

Source code

```
function [g,T]= Otsu_Thresholding(f)
num_bins = 256;
hist = imhist(f,num_bins); % compute histogram

p = hist / sum(hist);
omega = cumsum(p); % compute the cumulative
mu = cumsum(p .* (1:num_bins)');
mu_t = mu(end);

sigma = (mu_t * omega - mu).^2 ./ (omega .* (1 - omega));

% find the maximum value
maxval = max(sigma);
idx = mean(find(sigma == maxval));
T = (idx - 1) / (num_bins - 1);

g = zeros(size(f));
% create the binary image
g(find(f>=round(255*T))) = 255;
g(find(f<round(255*T))) = 0;
```

Task 3 – Chain Codes

Introduction and Objective

In this part, we are given an image of a circular stroke with specular noise. The objective of the problem is to obtain the Freeman chain code, the first difference of the outer boundary of the largest object, and the integer of minimum magnitude of the code. This is the image that will be used to implement the chain code:

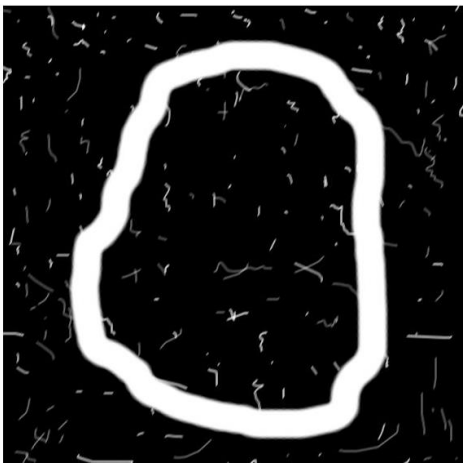

Figure 5 Noisy image

Results

First, we will apply a 9x9 averaging filter to smooth the image. Then, a binary image will be generated thresholding the smoothed image:

```
f = imread('circular_stroke.tif');
%% a)
h = ones(9,9)/81;
g = imfilter(f,h);
%% b)
[gB,~] = Otsu_Thresholding(g);
```

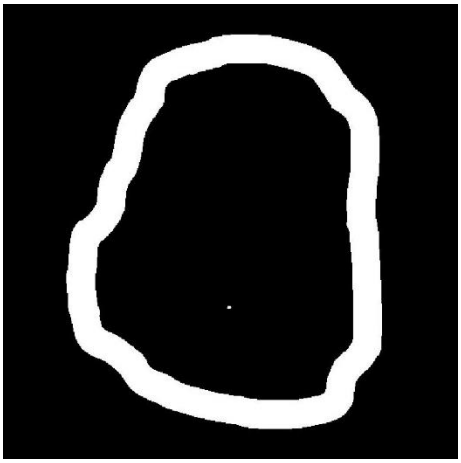

Figure 6 Binary smoothed image

After obtaining the binary image we will extract the outer boundary using bwboundaries function. This gives us all the possible boundaries in the image, however, only the first obtained boundary will be chosen, the outer one. This is what is obtained:

```
%% c)
B = bwboundaries(gB);
B1 = B{1}; % Main boundary is chosen
M = size(f,1);
N = size(f,2);

image = bound2im(B1,M,N);
figure(), imshow(image)
```

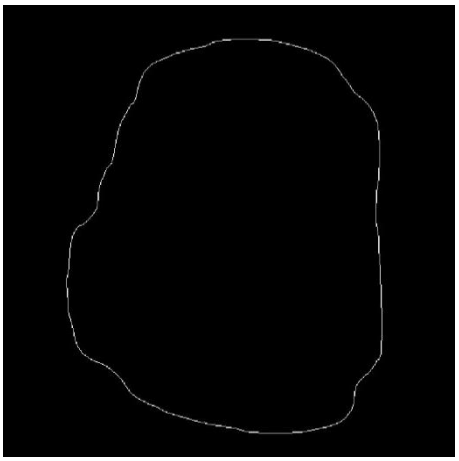

Figure 7 First Boundary

Then, the boundary will be subsampled into a grid whose lines are separated by 50 pixels:

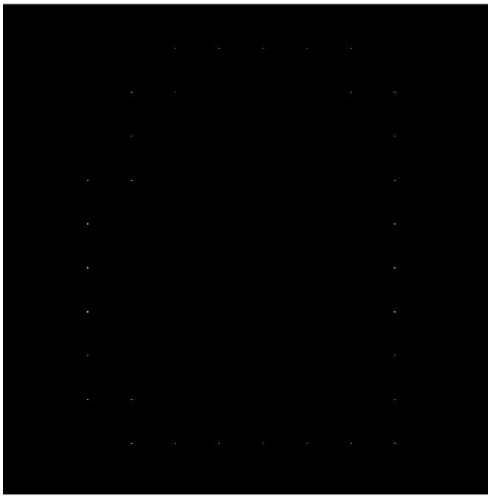

Figure 8 Points of the subsampled boundary

```
%% d)
gridsep=50;
[s, sUnit] = bsubsamp(B1, gridsep);
```

```
image2 = bound2im(s,M,N);
figure(), imshow(image2)


c = connectpoly(s(:,1), s(:,2));
image3 = bound2im(c,M,N);
figure(),imshow(image3)
%[x, y] = intline(x1, x2, y1, y2);
```

The previous image shows the points of the boundary and if we connect them with straight lines, we get this binary image:

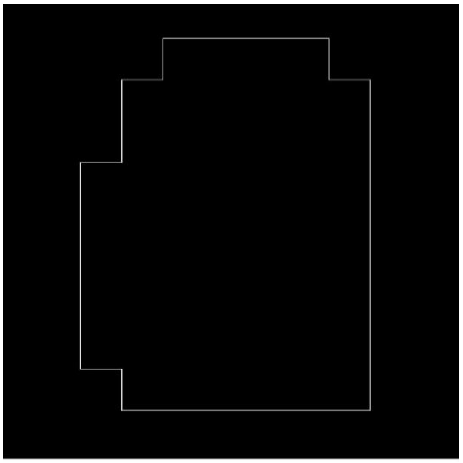

Figure 9 Subsampled boundary

Now that we have the boundary, Freeman chain code of it will be obtained. The function will include the option to choose between 4 or 8 – connected chain codes.

```
%% e)
CONN = 4;
c = FreemanChainCode(c, CONN);
```

For both cases, we need to map each movement with a unique index. For the case of 4 movements, idx=3*deltaX+deltaY+4 mapping function will be used where delta is the change in the direction and idx is the mapping index. That way, the mapping will be: cm([1 3 5 7])=[3 2 0 1];

```
  3  2  1
   \ | /
4 -- C -- 0
   / | \
  5  6  7
```

For the other case where there are 8 possible movements the mapping will follow this function: idx=3*deltaX+deltaY+5; That way, the mapping will be: cm([1 2 3 4 6 7 8 9])=[5 6 7 4 0 3 2 1];

The output of the function will be a struct with these values:

- Fcc. The chain code.
- Diff. First difference of Fcc:
- Mm. Integer of minimum magnitude from Fcc.
- Diffmm. First difference of code mm
- X0Y0. Coordinates of the starting point of the code

In our case the code will start in coordinates [358,103]. The chain code would be:

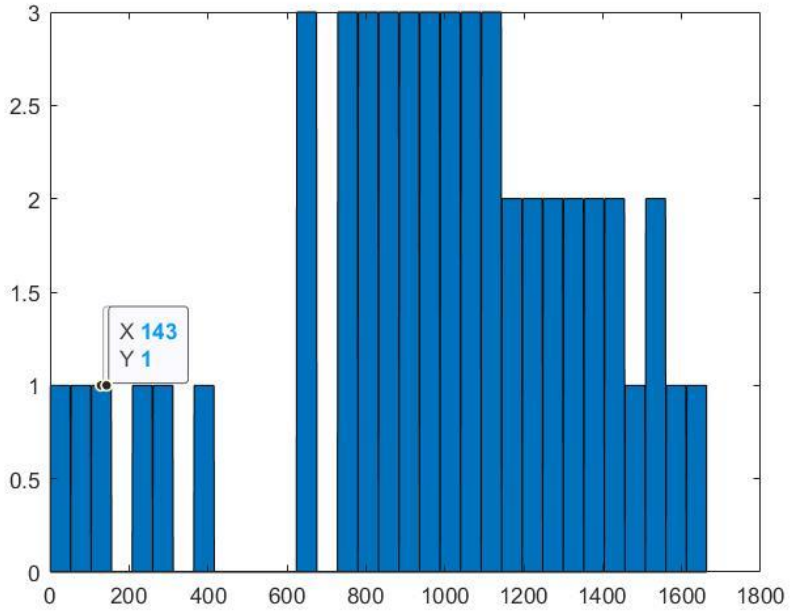

Figure 10 Chain Code

Numerically: 11101101000030333333333222222121100

And the first difference is described in the next graph:

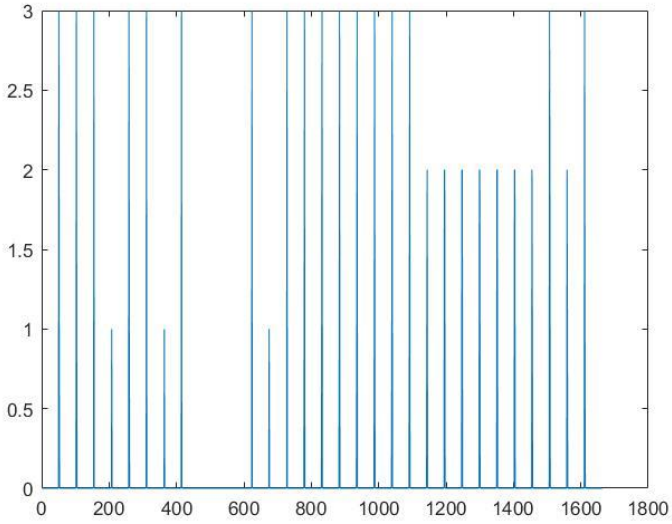

Figure 11 First difference

Finally, the integer of minimum magnitude is plotted in the next graph:

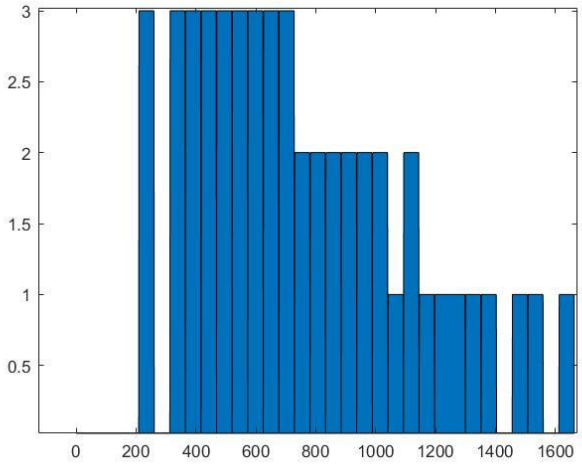

Figure 12 Integer of minimum magnitude

Source code

```
function c = FreemanChainCode(b, CONN)
% input b is a set of 2-D coordinate pairs for a boundary and
CONN can be
% 8 for an 8-connected chain code or 4 for a 4-connected chain
code.
% The output c is a structure with the following fields:
% c.fcc = chain code (1×?? where ?? is the number of boundary
pixels)
% c.diff = First difference of code c.fcc (1×??)
% c.mm = Integer of minimum magnitude from c.fcc (1×??)
```

```
% c.diffmm = First difference of code c.mm (1×??)
% c.x0y0 = Coordinates where the code starts (1×2)
if CONN == 4
    sb=circshift(b,[-1 0]);
    delta=sb-b;

    idx=3*delta(:,1)+delta(:,2)+4;
    cm([1 3 5 7])=[3 2 0 1];

    c.x0y0=[b(1,2),b(1,1)];
    fcc=(cm(idx))';
    c.fcc = fcc;
    c.diff = codediff(fcc,4);

    c.mm = MinimumMagnitude(fcc);
    c.diffmm = codediff(c.mm,4);

elseif CONN == 8
    sb=circshift(b,[-1 0]);
    delta=sb-b;

    idx=3*delta(:,1)+delta(:,2)+5;
    cm([1 2 3 4 6 7 8 9])=[5 6 7 4 0 3 2 1];

    c.x0y0=[b(1,2),b(1,1)];
    c.fcc=(cm(idx))';
    c.diff = codediff(c.fcc,8);
    c.mm = minmag(fcc);
    c.diffmm = codediff(c.mm,8);

else
    disp('Error, input CONN has a wrong value')
end

end


function z = minmag(c)
I = find(c == min(c));

J = 0;
A = zeros(length(I), length(c));
for k = I;
   J = J + 1;
   A(J, :) = circshift(c,[0 -(k-1)]);
end

[M, N] = size(A);
J = (1:M)';
for k = 2:N
   D(1:M, 1) = Inf;
   D(J, 1) = A(J, k);
   amin = min(A(J, k));
   J = find(D(:, 1) == amin);
   if length(J)==1
      z = A(J, :);
```

```
        return
    end
end
```

Task 4 – Fourier Descriptors

Introduction and Objective

The last part of this project is about the implementation of the Fourier descriptors. A binary image of a human chromosome will be used.

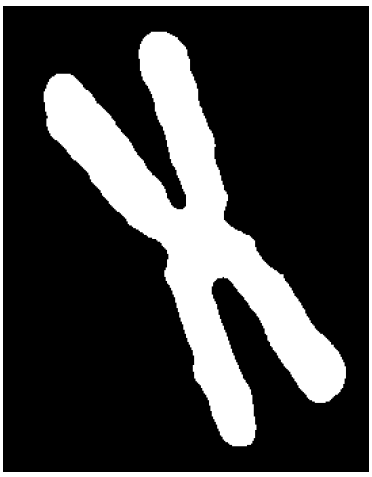

Figure 13 Binary image of human chromosome

After extracting the boundary of the chromosome, the program will compute the Fourier descriptors. Then, using the sequence of Fourier descriptors, the inverse of the Fourier transform will be computed to reconstruct the image using different number of descriptors.

Results

First, the boundary will be extracted:

```
B = bwboundaries(g);

B1 = B{1}; % Main boundary is chosen
M = size(f,1);
N = size(f,2);
image = bound2im(B1,M,N);
figure()
```

```
imshow(image)
```

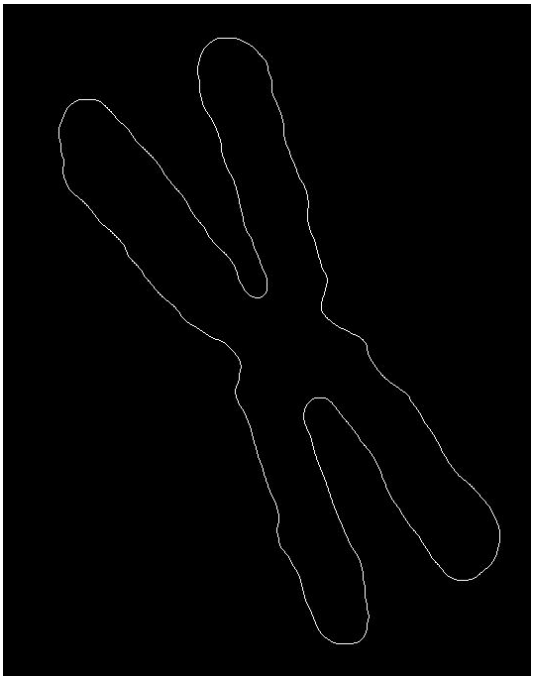

Figure 14 Boundary of the chromosome

Then, we will implement the Fourier transform using the function FourierDescriptor described in the Source code section. Doing it we obtain a sequence of 2208 complex numbers starting with 6+33i until -3.919-5.293i.

After that, we will use the function ifourierdescp(z,nd) where z is the sequence of descriptors and nd is the number of descriptors used for the reconstruction.

With nd = 1104, that is half of the total descriptors, the reconstructed image is almost equal to the real one:

```
d = size(z,1);
nd1 = round(d / 2);
s1 = abs(ifourierdescp(z,nd1));
image1 = bound2im(s1,M,N);
figure()
imshow(image1)
```

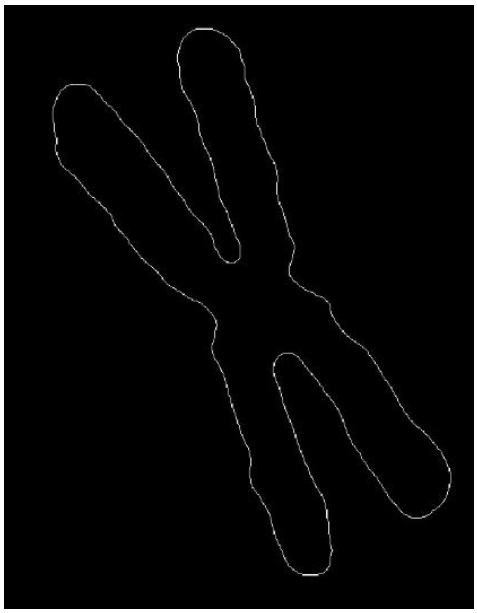

Figure 15 First reconstruction with half of the descriptors

Using nd = 22, or the 1% of the total descriptors, the reconstructed image shows the approximation of the shape of the real chromosome:

```
nd2 = round(d / 100);
s1 = abs(ifourierdescp(z,nd2));
image2 = bound2im(s1,M,N);
figure()
imshow(image2)
```

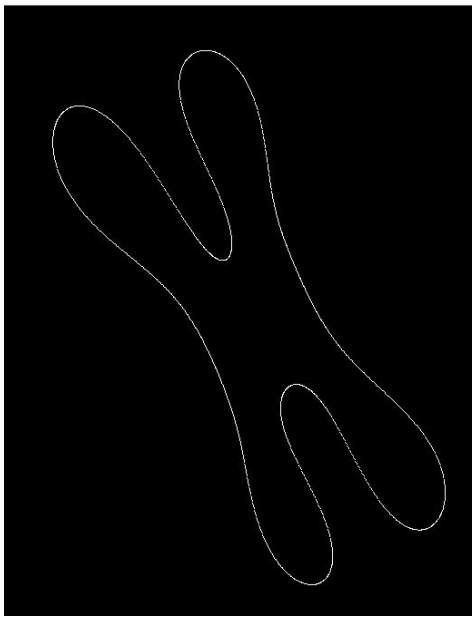

Figure 16 Second reconstruction with 22 descriptors

Source code

```
function z= FourierDescriptor(s)
[np,nc] = size(s);
if np/2 ~= round(np/2)
    s(end+1,:) = s(end,:);
    np=np+1;
end
x = 0:(np-1);
m = ((-1).^x)';
s(:,1) = m.*s(:,1);
s(:,2) = m.*s(:,2);
s = s(:,1) + i*s(:,2);
z = fft(s);
end
```

```
function s = ifourierdescp(z,nd)
np = length(z);
x = 0:(np-1);
m = ((-1).^x)';
d = round((np - nd)/2); % if nd is odd
z(1:d) = 0;
z(np - d + 1:np) = 0;
% Compute the Fourier inverse
zz = ifft(z);
s(:, 1) = real(zz);
s(:, 2) = imag(zz);
s(:, 1) = m.*s(:, 1);
s(:, 2) = m.*s(:, 2);
end
```

References

1) https://www.mathworks.com/