1 Introduction

The assignment consists of two sub-tasks, where the first one requires to implement skeletonisation procedure given steps in the task description while the second task requires to implement the Fourier descriptor and inverse Fourier descriptor procedures.

Solution to every task is given in the appendix, while visual proof together with the solution description are presented in the body of the report. Observations will be discussed separately in the discussion section.

2 Description

Sub-Task 1: Step 1 Apply Gaussian blur using variable kernel size based on the original image

Step 2 Flag border pixels for deletion if they satisfy following conditions: (i) border pixel has between 2 and 6 non-zero pixels in its neighbourhood; (ii) border pixel has exactly 1 combination of pixel in its neighbourhood, where one of the pixels is zero and the next one is non-zero; (iii) pixel (2,4,6 and 4,6,8 CW) product equals 0

Step 3 Remove flagged pixels

Step 4 Flag border pixels for deletion that satisfy conditions in Step 1 with only difference in that pixels 2,4,8 and 2,6,8 CW product must be equal 0.

Step 5 Remove flagged pixels

Sub-Task 2: Step 1 Apply Gaussian blur using variable kernel size based on the original image

Step 2 Represent boundary by a sequence of points

Step 3 Transform every boundary point to a complex number

Step 4 Take 1D Fourier transform of resulting signal

Step 5 Truncate signal in Fourier domain

Step 6 Apply inverse Fourier transform to reconstruct the image
3 Experiments

Experiments show visual proof of developed algorithm for every sub-task. Images used for experimentation were provided by the lecturer.

3.1 Sub-Task 1 — Skeletonisation

1. Experiment 1

- a) Chromosome
- b) Noisy circle
- c) Bone
- d) Bust
- e) Maple leaf
- f) Noisy triangle
- g) Noisy square
3.2 Sub-Task 2 — Fourier Descriptor

- Original
- Gaussian Blurred Binary
- Post Fourier transform

a) No transformation  
b) 180° Rotation  
c) Scaling  
d) Affine Transformation 1
Image Processing - Coursework 3 - Boris Mocialov (bm4)

MARCH 21
4 Discussion

During skeletonisation process, 50 algorithm iterations had been used, which was expected to ensure complete thinning for all the images considered. In the case of a chromosome silhouette image, silhouette boundary is thinned, but not destroyed. In the case of the filled circle with noisy background, the circle is thinned from both sides, while the noise is not changed. The bone image is thinned, exposing the details (thick sides of the bone). In the case of images with excessive filled area (e.g. bust, maple leaf, noisy triangle, noisy square), the thinning process exposes carcass of the image.

Initially, every image is blurred in order to smooth boundaries as to avoid unwanted branches after the thinning is performed (gives clearer final carcass of the image). From the results, it can be seen that most of the resulting skeletons produce sufficient carcasses of images. In case of the circle, few unwanted branches are left out. This may be due to small edges present on the boundary of the circle that are preserved during skeletonisation until the final skeleton carcass.

For Fourier descriptor case, the image is first blurred for the same purpose as in skeletonisation process. Later, the shape is extracted (black boundary) while the boundary descriptor is truncated in Fourier domain and the boundary is reconstructed using inverse Fourier transform. The task is to show whether the descriptor is invariant to the cases, such as rotation, scaling, affine transformation, and translation. A single image (chromosome) is used to verify the invariance and results are presented in the Experiments section. From the resulting images it can be seen that the descriptor is relatively invariant to considered transformations. In the case of the scaling and affine transformation, some of details are lost, but the general shape is preserved.
Appendices

Skeletonisation script:

```matlab
N = 50; % thinning iterations

image=(imread('image.tif'));

imageorig=image;

[rows, cols] = size(image); % size of the image
n=0; % iteration counter
while n<N
    for step = 1 : 2 % 2 algorithm steps
        marked_for_deletion = zeros(rows,cols); % matrix to hold pixels that were marked for deletion

        for x = 2 : rows - 1;
            for y = 2 : cols - 1;
                delete = 0; % delete pixel flag initially set to 0

                % CONDITIONS FOR DELETION BELOW
                % 1
                N_p = 0;
                for i = -1 : 1
                    for j = -1 : 1
                        if i ~= 0 && j ~= 0 && image(x+i,y+j) > 0
                            N_p = N_p + image(x+i,y+j);
                        end
                    end
                end
                % 2
                if (N_p >= 2 && N_p <= 6)
                    delete = delete + 1;
                end

                % 3
                zero_to_one_counter = 0;
                if image(x-1,y-1) > 0 && image(x,y-1) == 0
                    zero_to_one_counter = zero_to_one_counter + 1;
                end

            end
        end

    end

end
```

Image Processing - Coursework 3 - Boris Mocialov (bm4)

MARCH 21
if image(x-1,y) > 0 && image(x-1,y-1) == 0
    zero_to_one_counter = zero_to_one_counter + 1;
end
if image(x-1,y+1) > 0 && image(x-1,y) == 0
    zero_to_one_counter = zero_to_one_counter + 1;
end
if image(x,y-1) > 0 && image(x+1,y-1) == 0
    zero_to_one_counter = zero_to_one_counter + 1;
end
if image(x,y+1) > 0 && image(x-1,y+1) == 0
    zero_to_one_counter = zero_to_one_counter + 1;
end
if image(x+1,y-1) > 0 && image(x+1,y) == 0
    zero_to_one_counter = zero_to_one_counter + 1;
end
if image(x+1,y) > 0 && image(x+1,y+1) == 0
    zero_to_one_counter = zero_to_one_counter + 1;
end
if image(x+1,y+1) > 0 && image(x,y+1) == 0
    zero_to_one_counter = zero_to_one_counter + 1;
end
%4
if zero_to_one_counter == 1
    delete = delete + 1;
end
if step == 1
%5
    if image(x-1,y) * image(x,y+1) * image(x+1,y) == 0
        delete = delete + 1;
    end
%6
    if image(x,y+1) * image(x+1,y) * image(x,y-1) == 0
        delete = delete + 1;
    end
else
%5
    if image(x-1,y) * image(x,y+1) * image(x,y-1) == 0
        delete = delete + 1;
    end
%6
    if image(x-1,y) * image(x+1,y) * image(x,y-1) == 0
        delete = delete + 1;
end

end
end

if delete == 4
    marked_for_deletion(x,y) = 1;
end
end
end

%the actual deletion of the pixel that satisfied all the conditions above
for x = 2 : rows - 1;
    for y = 2 : cols - 1;
        if marked_for_deletion(x,y) > 0
            image(x,y) = 0;
        end
    end
end
end

%increment iteration
n=n+1;
end

%show results
subplot(1,2,1);
imshow(~imageorig);
subplot(1,2,2);
imshow(~image);
Fourier descriptor:

```matlab
I = imread('image.jpg');
%
pre-processing
I = 1-im2double(I);
h = fspecial('gaussian',32,15);
I = imfilter(I,h,'replicate');
I = im2bw(I,1.7*graythresh(I));

% Find a starting point on the boundary
[rows, cols] = find(I~=0);
[rows2, cols2] = size(I);
contour = bwtraceboundary(I, [rows(1), cols(1)], 'N');

% Subsample the boundary points so we have exactly 128, and put them into a
% complex number format (x + jy)
sampleFactor = length(contour)/128;
dist = 1;
for i=1:128
c(i) = contour(round(dist),2) + 1j*contour(round(dist),1);
dist = dist + sampleFactor;
end

% Fourier transform
C = fft(c);
%alternative
%C = zeros(1,128);
%for i=1:128
%  C(i) = C(i) + c(i) * exp((-2*pi*(1j))/128);
%end

% Chop out some of the smaller coefficients (less than umax)
% umax = 32;
umax = 10;
Capprox = C;
for u=1:128
  if u > umax && u < 128-umax
    Capprox(u) = 0;
  end
end

% Take inverse fft
```

Image Processing - Coursework 3 - Boris Mocialov (bm4)
cApprox = ifft(Capprox);

% alternative
% cApprox = zeros(1,128);

% for i=1:128
% cApprox(i) = cApprox(i) + Capprox(i) * exp((2*pi*(1j))/128);
% end

% Show original boundary and approximated boundary
subplot(1,2,1);
imshow(imcomplement(bwperim(I)));
hold on, plot(cApprox,'r');
subplot(1,2,2);
imshow(I);

imshow(I);