Lab. Practice #5

Berk ARSLAN

Table of Contents

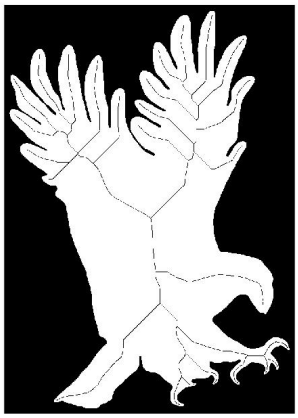
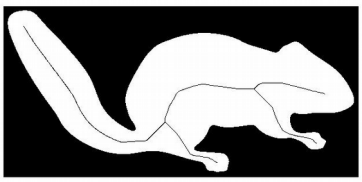
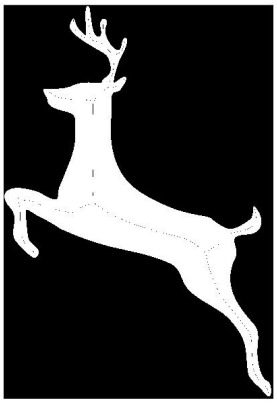
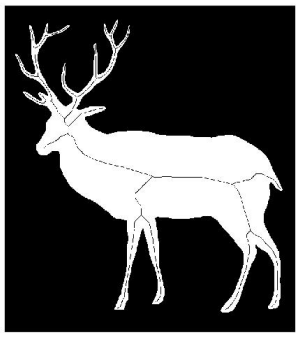
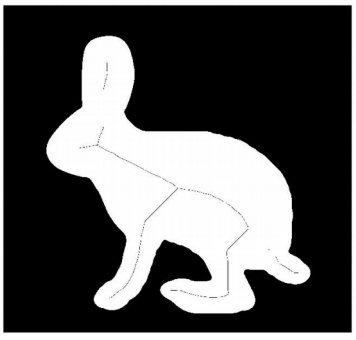
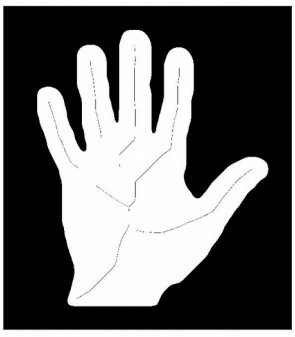
Thinning....2
Comparing with MATLAB Function....4
Code....7
    Implementation....7
    Thinning....7
    Pre-Table....9

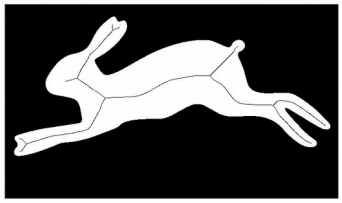
Thinning

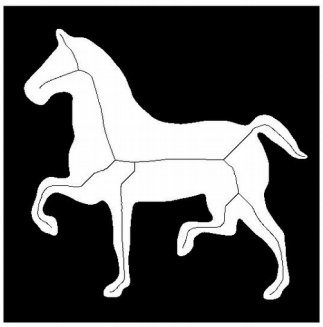
In the code, the Zhang-Suen algorithm is used which is published in 1984 in Communications of the ACM magazine. In pre-table function, a matrix is prepaired to boost the algorithm then in the algorithm the array compared with pre-tabled values and decided to clear the center pixel.

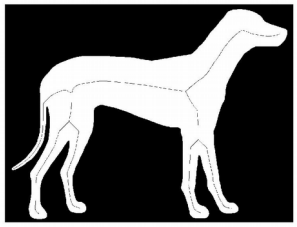
Image is thinned in two different direction which are from north-west to south-east and opposite direction. Firstly just one direction it is thinned then in the other direction thinned it again. -Some of the im2col and col2im fucntions could be eleminated however peripheral pixels must be calculated-. If previous thinning image is the same with current image, it stops the thinning loop. In addition, the tic-toc matlab functions give the elapsed time.

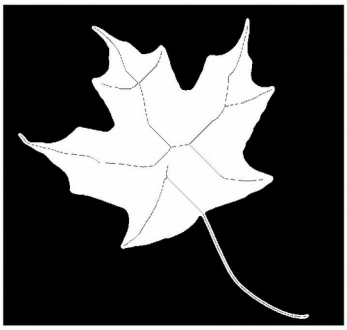
The pre-table function defines a multiplier which provides an specified array with 9 bits for a centered and peripheral pixels when multiplied by patch from image. Also it tests for each possible situation of patch the decision of clearing and exports two matrices include all possibilities to provide a comparing chance to thinning algorithm.

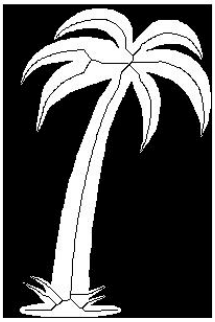







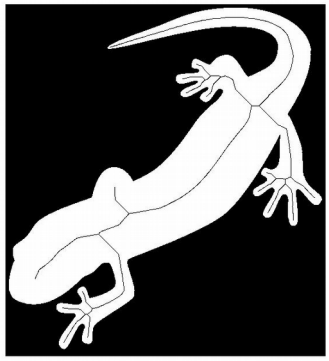


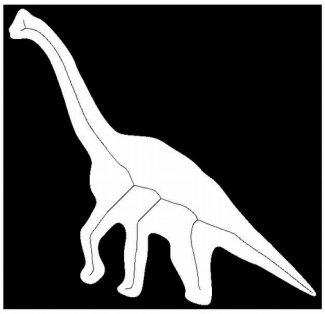


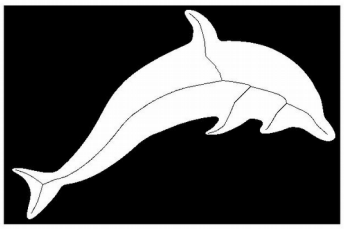


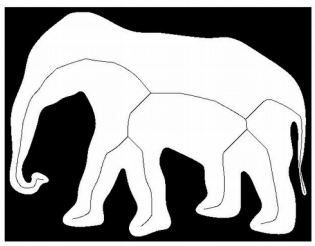






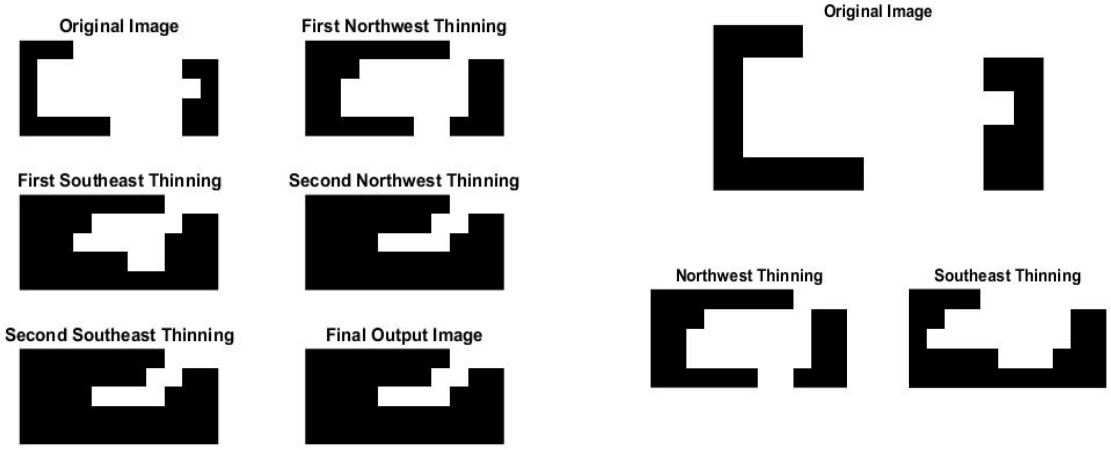






In the random example, we can see that the pixel which is one and has neighbours which are in sum is greater than two and less than six and due to the direction ('P_2*P_4*P_6' and 'P_4*P_6*P_8' for east and south, 'P_2*P_6*P_8' and 'P_2*P_4*P_8' for west and north) and has '0 -> 1' pattern in the neighbours is deleted step by step. That means in the northwest direction left and up side are deleted as a line also the left-down pixel and in the southeast direction is vice versa.

This progress continues until when the input and output of progress are same which means there would not be deleted any pixel.

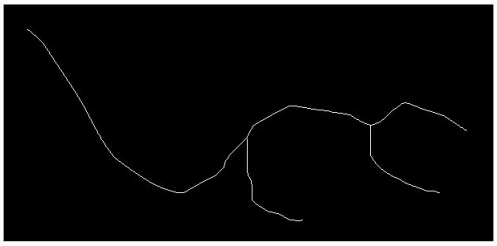


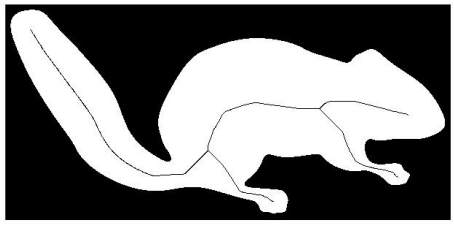
Comparing with MATLAB Function

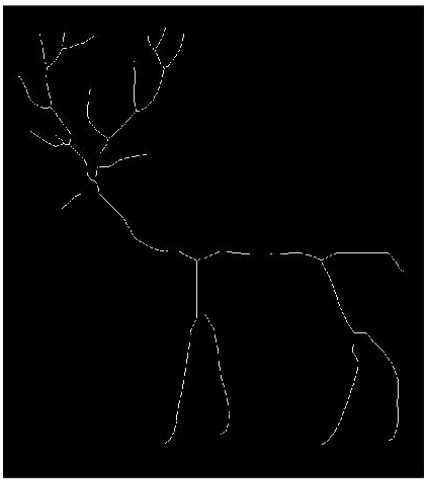
MATLAB uses an algoritm which is published in Thinning Methodologies-A Comprehensive Survey,IEEE in 1992 by Louisa Lam, Seong-Whan Lee, and Ching Y. Wuen.

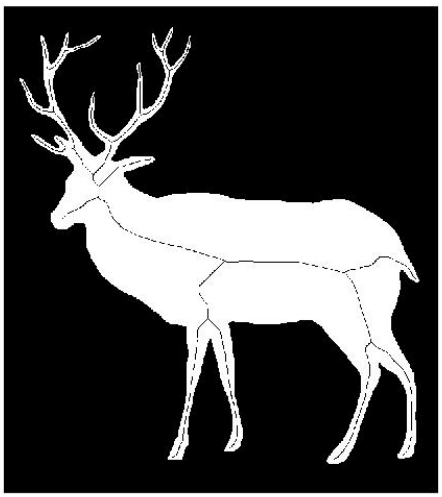
Branching and computing time is pretty different in the algorithm as seen in the example outputs. The MATLAB algorithm is really faster and attends more branching.

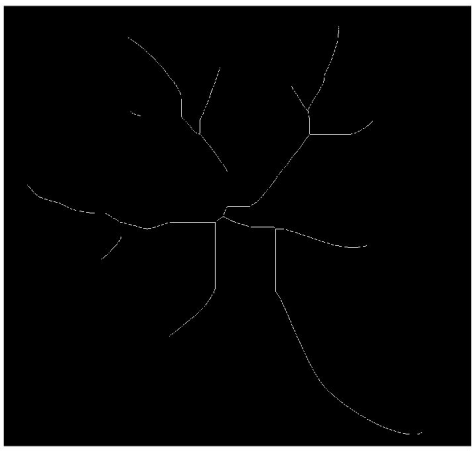
	MATLAB	**Zhang-Suen**
Chipmunk	0.22 seconds	15 seconds
Deer	0.35 seconds	31 seconds
Leaf2	0.66 seconds	265 seconds
Deer_Matt_Todd_01	6.8 seocnds	2951 seconds
Elephant	0.49 seconds	25 seconds

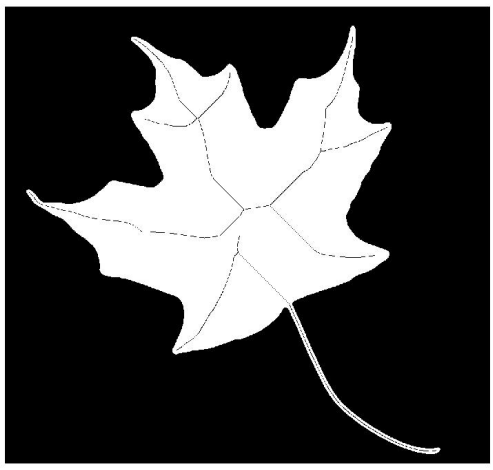


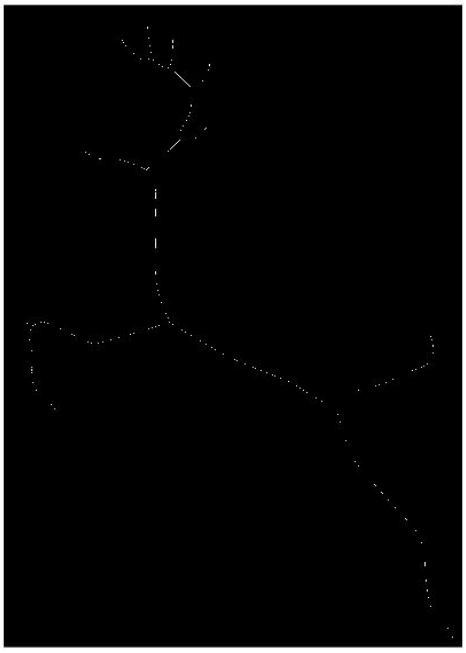


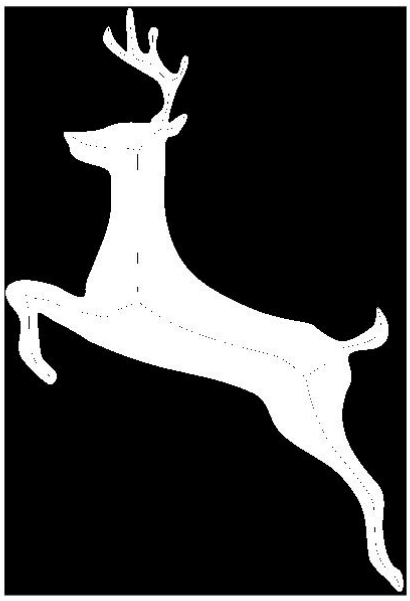


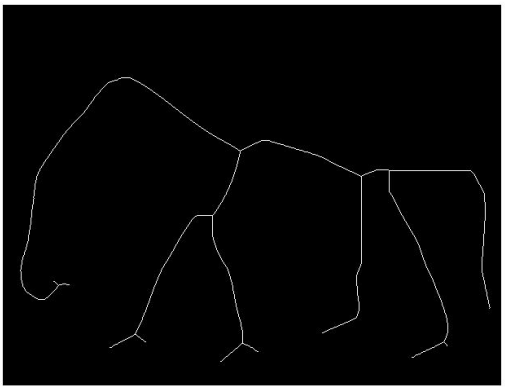


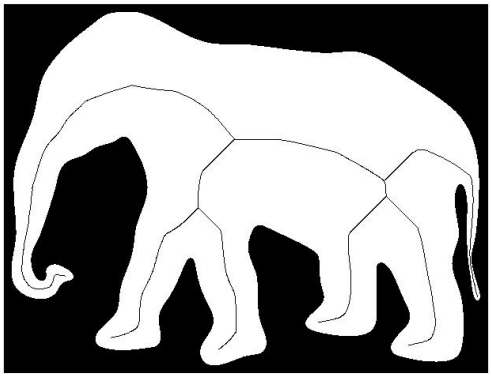












Code

Implementation

```
image_original = imread('<picture_path>');

%pre-table function is called
[multiplier, image_north_west, image_south_east] =
pre_table(numel(image_original));
%thinning operation is implemented
[thinned_image, time] = thinning(image_original, multiplier, image_north_west,
image_south_east);

%image with skeleton
imshow(image_original-thinned_image);
```

Thinning

```
function [ output_image, elapsed_time ] = thinning( image_original, multiplier,
      image_north_west, image_south_east)

%initialized main variables
    continue_iteration = 1;         %iteration decider
    input_image = image_original;    %initialize input image as original image
    output_image = input_image;      %initialize output image as original image
    %start timer
    tic

    while continue_iteration == 1
        %set input image the previous output image
        input_image = output_image;
        %first imply the north and west direction then imply south and east
        %direction
        first_thinning = thin_north_west(input_image, image_north_west,
      multiplier);
        output_image = thin_south_east(first_thinning, image_south_east,
      multiplier);
        %test input and output image are equal or not
        continue_iteration = ~isequal(input_image,output_image);
    end
%stop timer
elapsed_time = toc;

end


function [ output_image ] = thin_north_west( input_image, image_north_west,
      multiplier )

%padding image
    padded_input_image = padarray(input_image,[1 1]);
    padded_image_size = size(padded_input_image);
    %convert image to columns
    image_columns = im2col(padded_input_image,[3 3]);

    %create P matrix
    P = [image_columns(5,:); image_columns(2,:); image_columns(3,:);
      image_columns(6,:); image_columns(9,:); image_columns(8,:);
```

```
      image_columns(7,:); image_columns(4,:); image_columns(1,:) ];

    %multiply P with pre-defined multiplier matrix and add 1 to specify
    %each P value and convert sum of them to decimals
    columns_in_decimal = sum(multiplier .* P , 1) + 1;
    %each column of pre-defined image creates new pixel according to
    %columns_in_decimal variable
    image_columns = image_north_west(columns_in_decimal);
    %new array of image created by pre-defined image converting to filtered
    %image
    output_image = col2im(image_columns,[3 3],  padded_image_size);
end

function [ output_image ] = thin_south_east( input_image, image_south_east,
      multiplier )

%padding image
    padded_input_image = padarray(input_image,[1 1]);
    padded_image_size = size(padded_input_image);
    %convert image to columns
    image_columns = im2col(padded_input_image,[3 3]);

    %create P matrix
    P = [image_columns(5,:); image_columns(2,:); image_columns(3,:);
      image_columns(6,:); image_columns(9,:); image_columns(8,:);
      image_columns(7,:); image_columns(4,:); image_columns(1,:) ];

    %multiply P with pre-defined multiplier matrix and add 1 to specify
    %each P value and convert sum of them to decimals
    columns_in_decimal = sum(multiplier .* P , 1) + 1;

    %each column of pre-defined image creates new pixel according to
    %columns_in_decimal variable
    thinned_image_columns = image_south_east(columns_in_decimal);

    %new array of image created by pre-defined image converting to filtered
    %image
    output_image = col2im(thinned_image_columns,[3 3],  padded_image_size);

end
```

Pre-Table

```
function [multiplier, image_north_west, image_south_east] = pre_table
      ( image_size )
    %pre allocate the P matrix
    P = zeros(9,511);
    %for each column
    for i = 0:511
        %each value converted to binary as a column
        P(:,i+1) = transpose(dec2binvec(i,9));
    end

    %prepare a multiplier from 2^0 to 2^8
    multiplier = zeros(9,1);
    for i = 0:8
    multiplier(i+1) = 2^i;
    end

    %copy this multiplier to each column
    multiplier = repmat(multiplier,1,image_size);

    %the last P is same with P_2
    P(10,:) = P(2,:);
    % B is the summation from P_1 to P_9
    B = sum(P(2:end-1,:));
    %Describe each direction's P array
    P_east = [P(2,:);P(4,:);P(6,:)];
    P_south = [P(4,:);P(6,:);P(8,:)];
    P_west = [P(2,:);P(6,:);P(8,:)];
    P_north = [P(2,:);P(4,:);P(8,:)];

    %Then multiply column by colmun
    P_east = prod(P_east,1);
    P_south = prod(P_south,1);
    P_west = prod(P_west,1);
    P_north = prod(P_north,1);

    %copy original images to outputs of will be compared with original
    %image
    image_south_east = P(1,:);
    image_north_west = P(1,:);

    %define the counter of loop
    column_size = size(P);

    for i = 1:column_size(2)
        %if middle pixel is one
        if(P(1,i))
            %and if 2<=B<=6 and P_east and P_south are zero
            if(2<=B(i) && B(i)<=6 && P_east(i) == 0 && P_south(i) == 0)
                %the counter A is initilazed
                A = 0;
                %test 0->1 pattern in P array
                for k = 2:size(P(:,1),1)-1
                    if P(k,i) == 0 && P(k+1,i)==1
                        %if there is this pattern, then add 1 to A coutner
                        A = A+1;
                    end
                end

                if (A==1)
                    %if A is 1 then clear the center pixel of comparing
                    %matrix
                    image_south_east(i) = 0;
```

```
                end
            end

            %and test it for west and north direction
            %and if 2<=B<=6 and P_west and P_north are zero
            if(2<=B(i) && B(i)<=6 && P_west(i) == 0 && P_north(i) == 0)
                %the counter A is initilazed
                A = 0;
                for k = 2:size(P(:,1),1)-1
                    %test 0->1 pattern in P array
                    if P(k,i) == 0 && P(k+1,i)==1
                        %if there is this pattern, then add 1 to A coutner
                        A = A+1;
                    end
                end
                if (A==1)
                    %if A is 1 then clear the center pixel of comparing
                    %matrix
                    image_north_west(i) = 0;
                end
            end
        end
    end
end
```