**Square Tracing Algorithm:**

**% Main part**

% I = imread('binaryImage.png'); % you can change this input

I = [0 0 0 0 0 0 0 0;

0 0 0 1 0 1 0 0;

0 0 0 1 1 1 0 0;

0 0 0 0 1 0 1 0;

0 0 1 0 1 1 0 0;

0 0 1 1 1 0 1 0;

0 0 0 0 0 0 0 0];

I = im2bw(I,0.5); %#ok<IM2BW>

tic

[R, C] = size(I);

T = zeros(R+2, C+2);

for i = 1 : R

for j = 1 : C

T(i+1, j+1) = I(i, j);

end

end

% From bottom to top and left to right

% scan the cells of T until a black pixel, s,

% of P is found.

[L, P] = size(T);

for i = L : -1 : 1

for j = 1 : P

[Y, X] = scan(T, i, j);

if Y ~= -1

break;

end

end

if Y ~= -1

break;

end

end

B = []; % Set B to be empty

S = [Y, X]; % to be the starting pixel, s

B = [B, S];

P = [Y, X-1]; % Set the current pixel, p

path = 'W';

while ~isequal(P, S) % While p not equal to s do

Y = P(1); X = P(2);

if T(Y, X) == 1 % If the current pixel p is black

B = insert(B, P); % Insert s in B

[X, Y, path] = left(X, Y, path); % Turn left visit the left adjacent pixel of p

else

[X, Y, path] = right(X, Y, path); % Turn right

end

P = [Y, X]; % Update p i.e. set it to be the current pixel

end

B = B - 1;

toc

B\_O = visualization(B, T); % visualization boundary

fprintf("Square Tracing Algorithm\n");

fprintf(" the coordinate:\n")

disp(B);

fprintf(" the coordinate axis:\n")

disp(B\_O);

subplot(1,2,1), imshow(T,'InitialMagnification','fit'); title('original image');

subplot(1,2,2), imshow(B\_O,'InitialMagnification','fit'); title('boundary image');

**Function:**

function B = insert(B, P)

[R, ~] = size(B);

if B(R, 1) ~= P(1) || B(R, 2) ~= P(2)

B = [B; P];

end

end

function [X, Y, path] = left(X, Y, path)

if path == 'N'

X = X-1;

path = 'W';

elseif path == 'E'

Y = Y-1;

path = 'N';

elseif path == 'S'

X = X+1;

path = 'E';

elseif path == 'W'

Y = Y+1;

path = 'S';

end

end

function [X, Y, path] = right(X, Y, path)

if path == 'W'

Y = Y-1;

path = 'N';

elseif path == 'N'

X = X+1;

path = 'E';

elseif path == 'E'

Y = Y+1;

path = 'S';

elseif path == 'S'

X = X-1;

path = 'W';

end

end

**Function (for both Square and Moore Tracing):**

function [Y, X] = scan(T, Y, X)

[R, C] = size(T);

if Y > 1 && T(Y-1, X) == 1

Y = Y-1;

elseif X > 1 && T(Y, X-1) == 1

X = X-1;

elseif Y < R && T(Y+1, X) == 1

Y = Y+1;

elseif X < C && T(Y, X+1) == 1

X = X-1;

else

Y = -1;

X = -1;

end

end

function B\_O = visualization(B, T)

[R, C] = size(T);

B\_O = zeros(R, C);

[B\_r, ~] = size(B);

for B\_i = 1 : B\_r

B\_x = B(B\_i,1);

B\_y = B(B\_i,2);

B\_O(B\_x, B\_y) = 1;

end

end

**Moore Neighborhood Tracing Algorithm:**

% T = imread('binaryImage.png');

I = [0 0 0 0 0 0

0 0 0 1 0 0

0 0 1 1 1 0

0 0 0 1 1 0

0 0 1 1 0 0

0 0 1 0 0 0

0 0 0 0 0 0];

I = im2bw(I,0.5); %#ok<IM2BW>

tic

[R, C] = size(I);

T = zeros(R+2, C+2);

for i = 1 : R

for j = 1 : C

T(i+1, j+1) = I(i, j);

end

end

% From bottom to top and left to right scan the cells of

% T until a black pixel, s, of P is found

[L, P] = size(T);

for i = L : -1 : 1

for j = 1 : P

[Y, X] = scan(T, i, j);

if Y ~= -1

break;

end

end

if Y ~= -1

break;

end

end

B = []; % Set B to be empty.

S = [Y, X];

B = [B, S]; % Insert s in B.

C = [Y-1, X]; % Set c to be the next clockwise pixel.

startPoint = 6;

while ~isequal(S, C) % While c not equal to s do

n = [Y-1, X-1, Y-1,X, Y-1,X+1, Y,X+1, Y+1,X+1, Y+1,X, Y+1,X-1, Y,X-1]; % neigh

startPoint = startPoint - 1;

for i = 1 : 8

Y = n(startPoint \* 2 - 1);

X = n(startPoint \* 2);

C = [Y, X];

if isequal(S, C)

break;

end

if T(Y, X) == 1

B = [B; C]; %#ok<AGROW> %insert c in B

startPoint = backtrack(C, startPoint, n); % backtrack (move the current pixel c to the pixel from which p was entered)

break;

else

if startPoint == 8

startPoint = 1;

else

startPoint = startPoint + 1; % advance the current pixel c to the next clockwise pixel

end

end

end

end

toc

fprintf("Moore Neighborhood Tracing Algorithm\n");

B\_O = visualization(B, T);

fprintf(" the coordinate:\n")

disp(B);

fprintf(" the coordinate axis:\n")

disp(B\_O);

subplot(1,2,1), imshow(T,'InitialMagnification','fit'); title('original image');

subplot(1,2,2), imshow(B\_O,'InitialMagnification','fit'); title('boundary image');

**Function:**

function start = backtrack(C, start, neigh)

posi = [8, 8, 2, 2, 4, 4, 6, 6];

if start \* 2 - 3 < 1

C = [neigh(15), neigh(16)];

else

C = [neigh(start \* 2 - 3), neigh(start \* 2 - 2)];

end

start = posi(start);

end

**Function (for both Square and Moore Tracing):**

function [Y, X] = scan(T, Y, X)

[R, C] = size(T);

if Y > 1 && T(Y-1, X) == 1

Y = Y-1;

elseif X > 1 && T(Y, X-1) == 1

X = X-1;

elseif Y < R && T(Y+1, X) == 1

Y = Y+1;

elseif X < C && T(Y, X+1) == 1

X = X-1;

else

Y = -1;

X = -1;

end

end

function B\_O = visualization(B, T)

[R, C] = size(T);

B\_O = zeros(R, C);

[B\_r, ~] = size(B);

for B\_i = 1 : B\_r

B\_x = B(B\_i,1);

B\_y = B(B\_i,2);

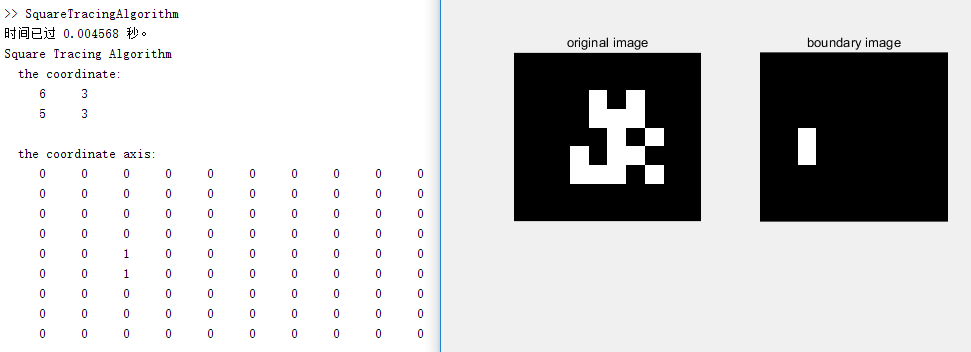
B\_O(B\_x, B\_y) = 1;

end

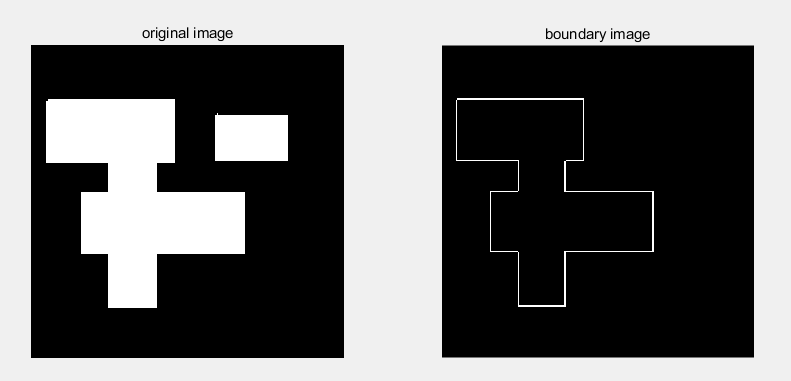
end

**Output**

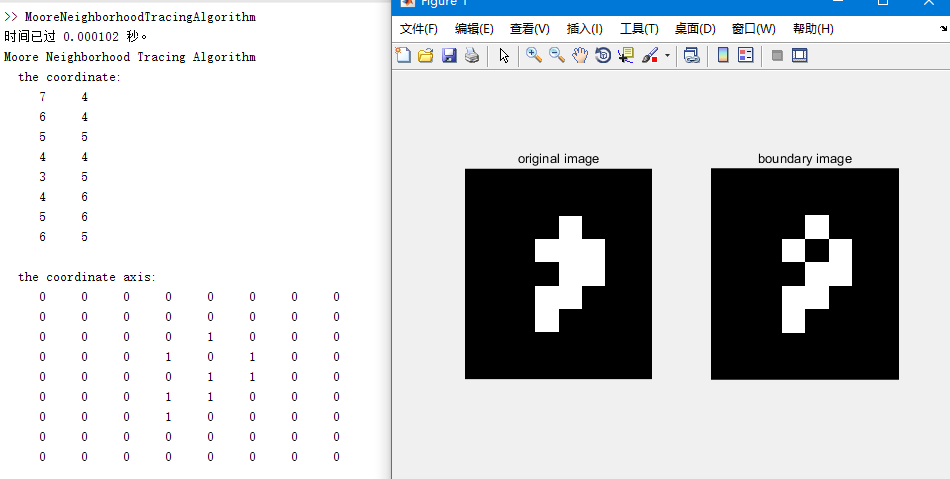
Square Tracing Algorithm

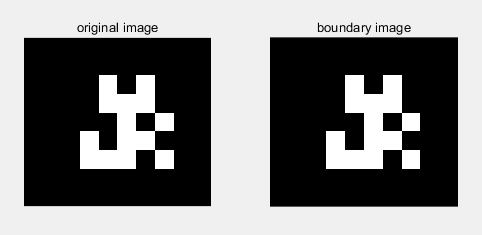


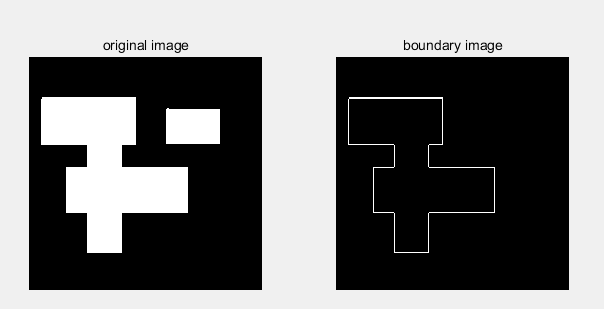
Change input binary image:



Moore Neighborhood Tracing Algorithm







**Analysis:**

2.

Moore tracing is faster than square tracing.

Moore have effect computing time.

Moore Algorithm is superior to square algorithm.

**2.**

Square and Moore can get the boundary of the image

But, square can’t search diagonal of pixels.

And, both of them only search one object. It’s cannot get more than one boundary of object.