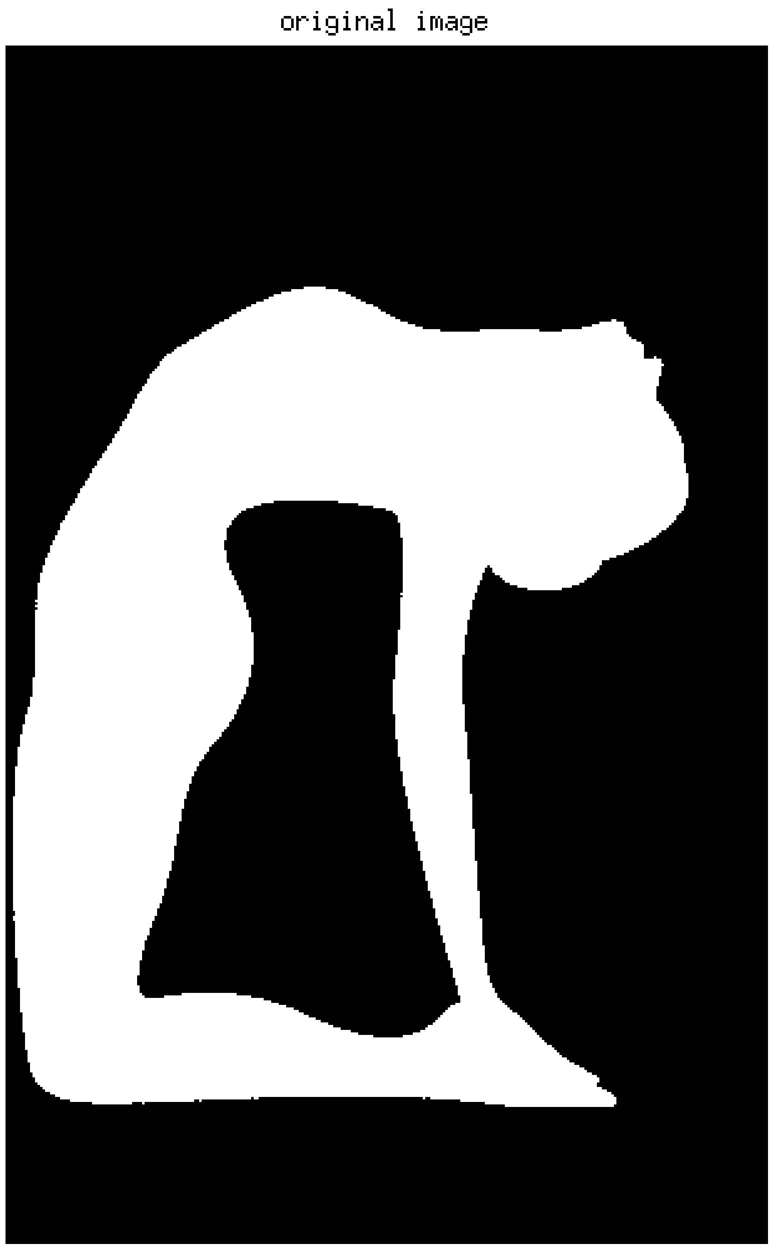
Table of Contents

```
function boundaryDescriptor()
```

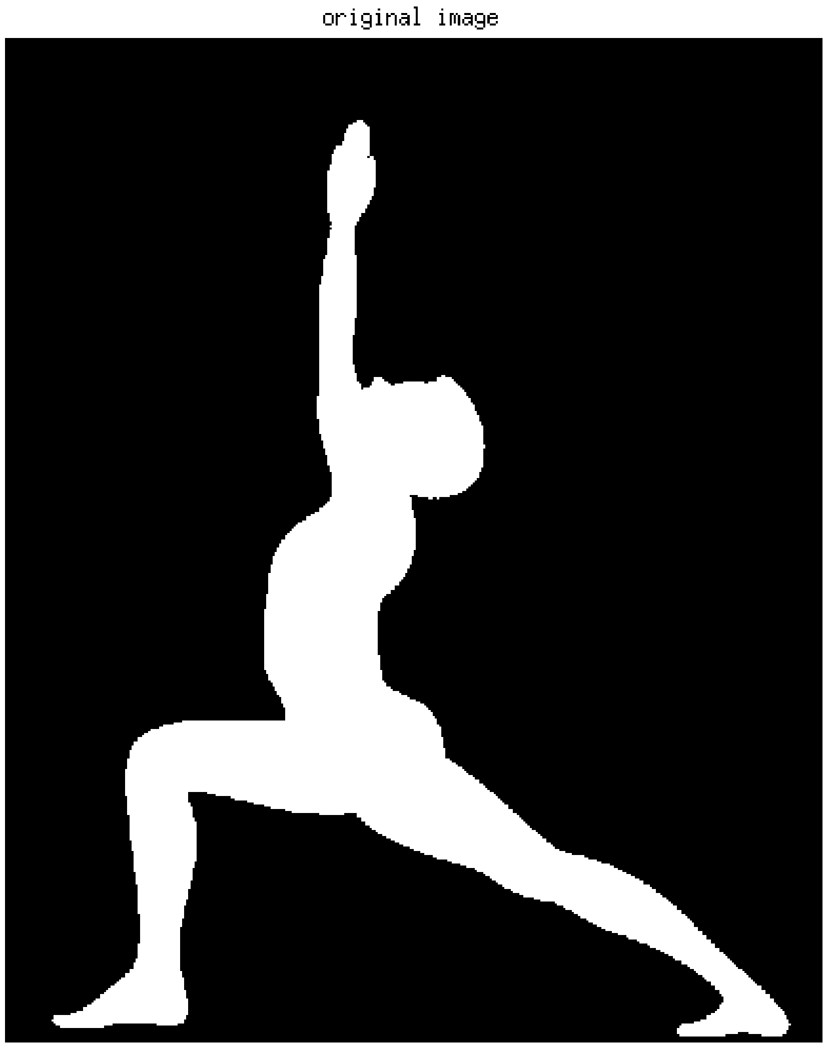
The method used to solve the problem is Moore's Boundary Tracking Algorihm for boundary detection, resampling the boundary detected, ad finally using chain code for the resampled boundary. The problem with this approach is that downsampling approximates the shape of the boundary and thus optimal sampling size is to be determined. Also similar asanas will result in similar shape od approximated boundary, thus making classification diffuclt.

```
clc;
close all;

asana = {'Ustrasana';'veerbhadrasan'; 'vrikhsasana';'trikonasana'};

r = 1;
for m = 1:4

    im1 = imread(['yogasan/y' num2str(m) '.jpg']);
    bw1 = im2bw(im1);
    bw1 = 1 - bw1;

    figure;
    imshow(bw1);
    title('original image');
    [m,n] = size(bw1);

    %bw1 = [0,0,0,0,0,0,0; 0,0,1,1,1,1,0; 0,1,0,0,1,0,0; 0,0,1,0,1,0,0; 0,1,0,0,1,
    %[m,n] = size(bw1);
    boundary = [];
    %get first 1 position of image
    flag = 0;
    for i = 1:m
        for j = 1:n
            if bw1(i,j) == 1
                loc = [i j];
                flag = 1;
            end
            if flag == 1
                break;
            end
        end
```

```
end

b0 = loc;
c0 = [loc(1)-1 loc(2)];
```

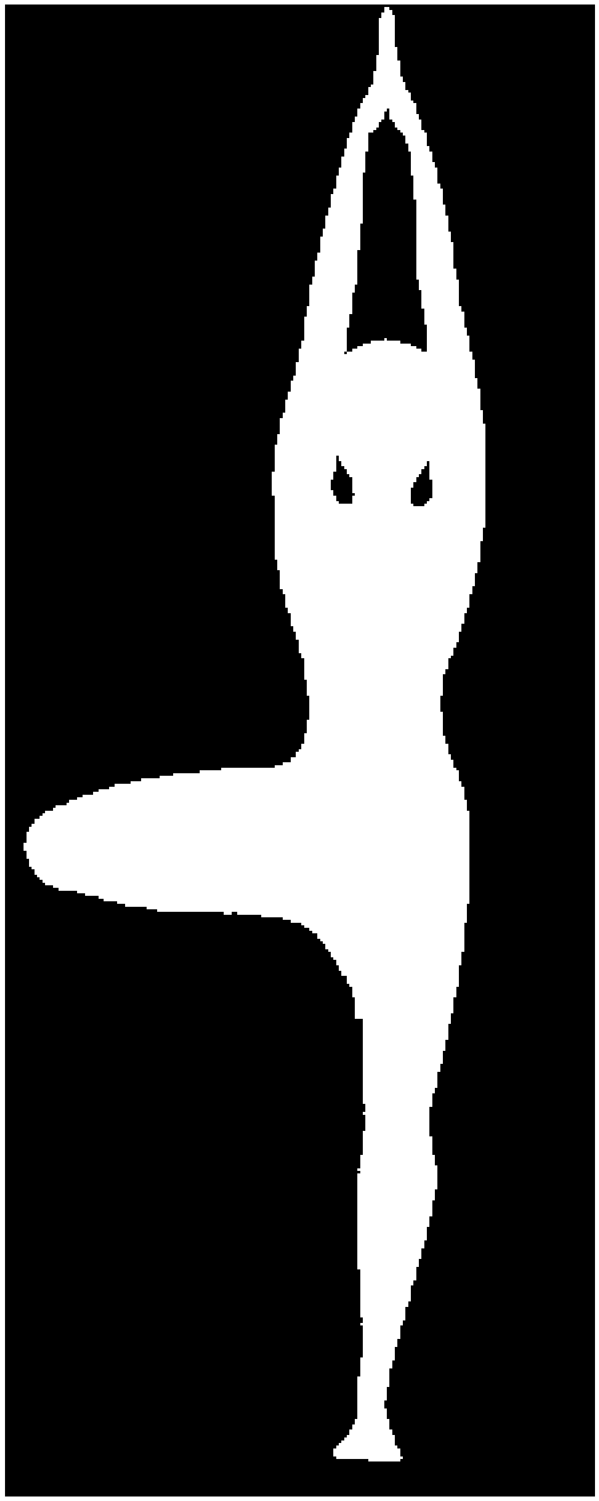


```
Warning: Image is too big to fit on screen; displaying at 67%
```



```
Warning: Image is too big to fit on screen; displaying at 67%
```

original image



```
Warning: Image is too big to fit on screen; displaying at 67%
```

search for b1 and c1

```
flag = 0;
b1 = c0;
%eight nearest negihbour search

i = -1;
for j = -1:1
    c1 = b1;
    b1 = [loc(1)+i, loc(2)+j];
    if bw1(loc(1)+i, loc(2)+j) == 1
        flag = 1;
        c = c1;
        b = b1;
    end
```

```
        if flag == 1
            break;
        end
    end

    if flag ~= 1
        c1 = b1;
        b1 = [loc(1), loc(2)+1];
        if bw1(loc(1), loc(2)+1) == 1
            flag = 1;
            c = c1;
            b = b1;
        end

    end

    if flag ~=1
       i = 1
       for j = -1:1
           c1 = b1;
           b1 = [loc(1)+i, loc(2)+j];
           if bw1(loc(1)+i, loc(2)+j) == 1
               flag = 1;
               c = c1;
               b = b1;
           end
           if flag == 1
               break;
           end
       end
    end
```

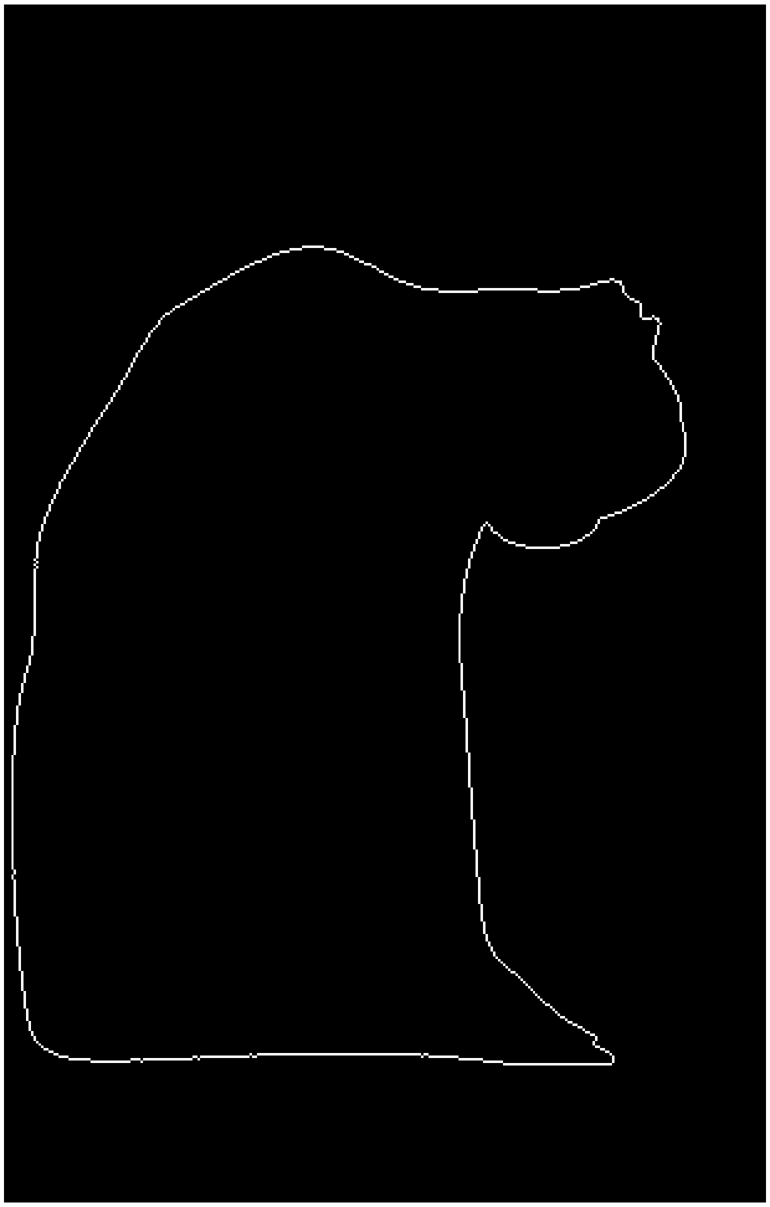
search for other boundary points

```
    boundary = [boundary; b];
    cIndx = findIndex(b-c);
    indx = [-1 -1; -1 0; -1 1; 0 1; 1 1; 1 0; 1 -1; 0 -1];


    while ~isequal(b,b0)
        b1 = b;
        c1 = c;
        flag = 0;
        cIndx = findIndex(c-b);
        %find next broder point from eight neighbours of current border point
        for i = 1:8
             %neighbour point is a border point
            if(bw1(c1(1), c1(2)) == 1)
                 flag = 1;
            end
            %change starting neighbour point for new border point and break
            if flag == 1
                 preIndx = mod(cIndx-1,9);
                 if cIndx == 1
                     preIndx = 8;
                 end

                c = b+indx(preIndx,:);
                b = c1;
                boundary = [boundary; b];
                break;
            end

            %search neighbours in clockwise direction
             cIndx = mod(cIndx+1,9);
            if cIndx == 0
              cIndx = 1;
            end
             c1 = b+indx(cIndx,:);

        end

    end
```

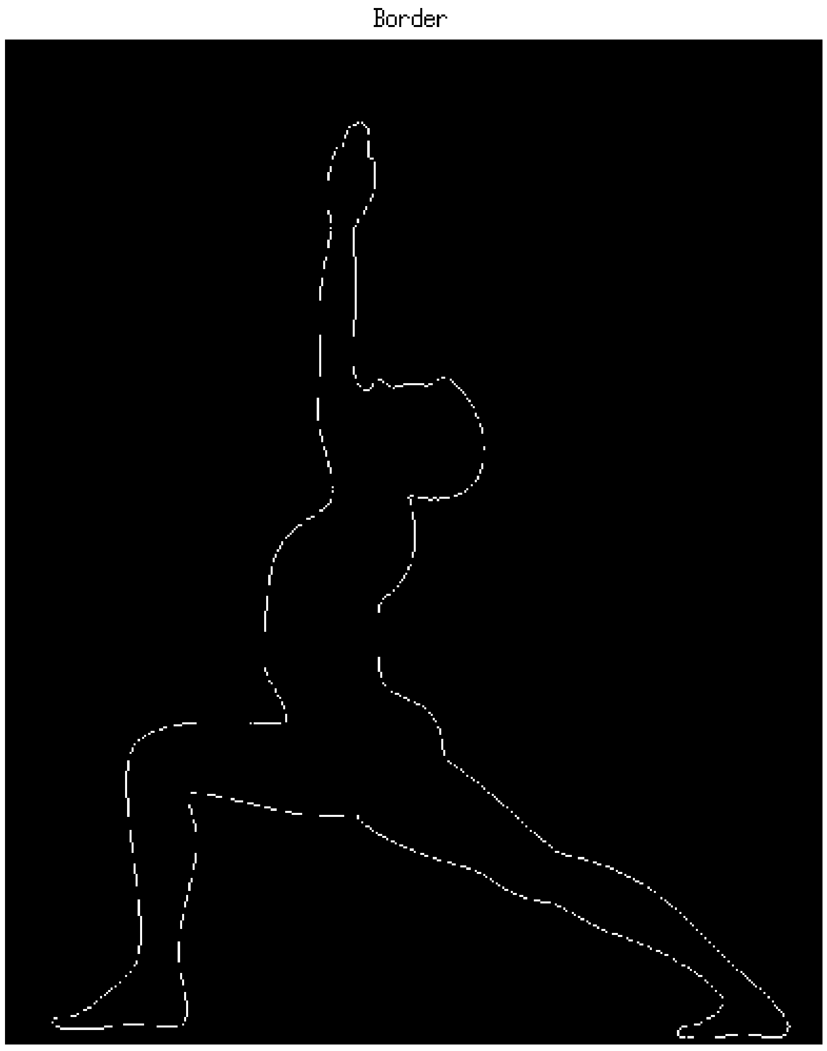
make border image

```
border = zeros(size(bw1));
for i = 1:length(boundary)
    border(boundary(i,1), boundary(i,2)) = 255;
end

figure;
imshow(border);
title('Border');
```

Border

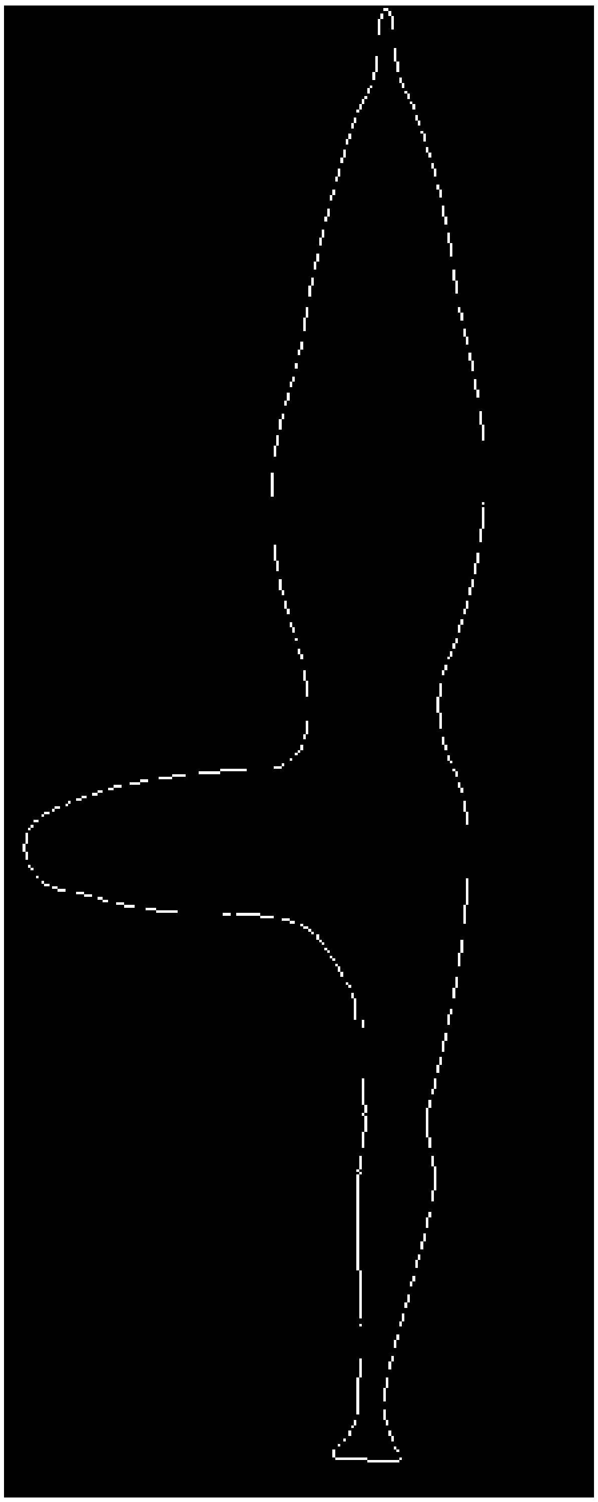


Warning: Image is too big to fit on screen; displaying at 67%

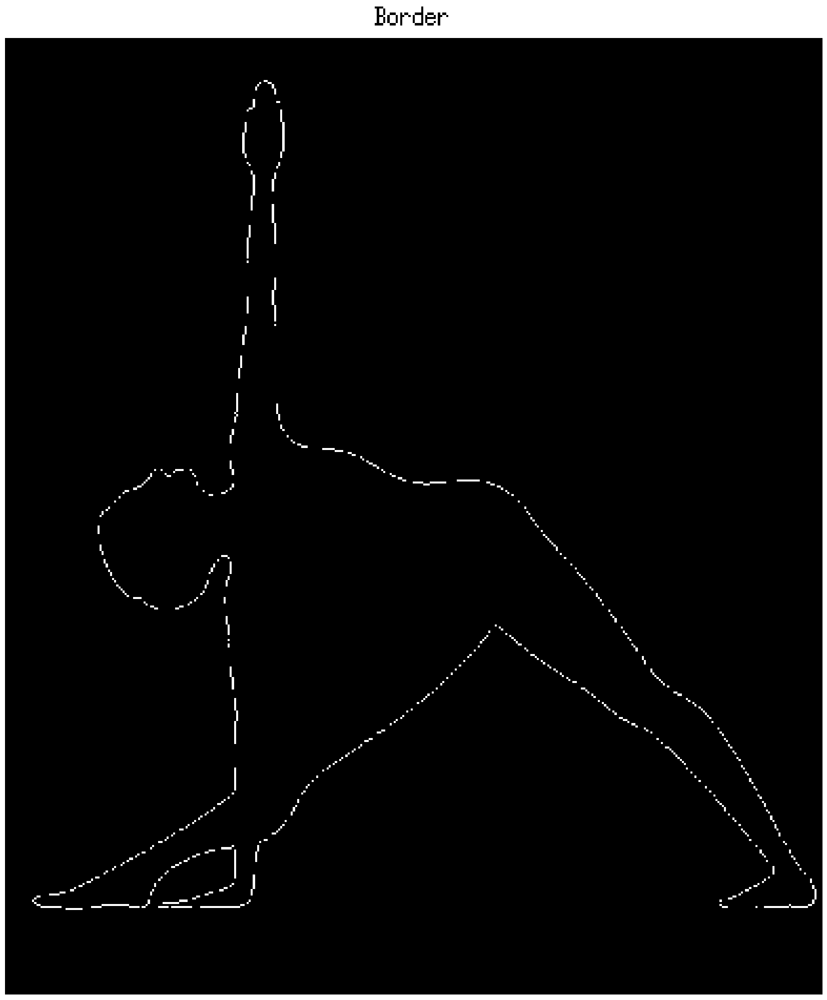


```
Warning: Image is too big to fit on screen; displaying at 67%
```

Border



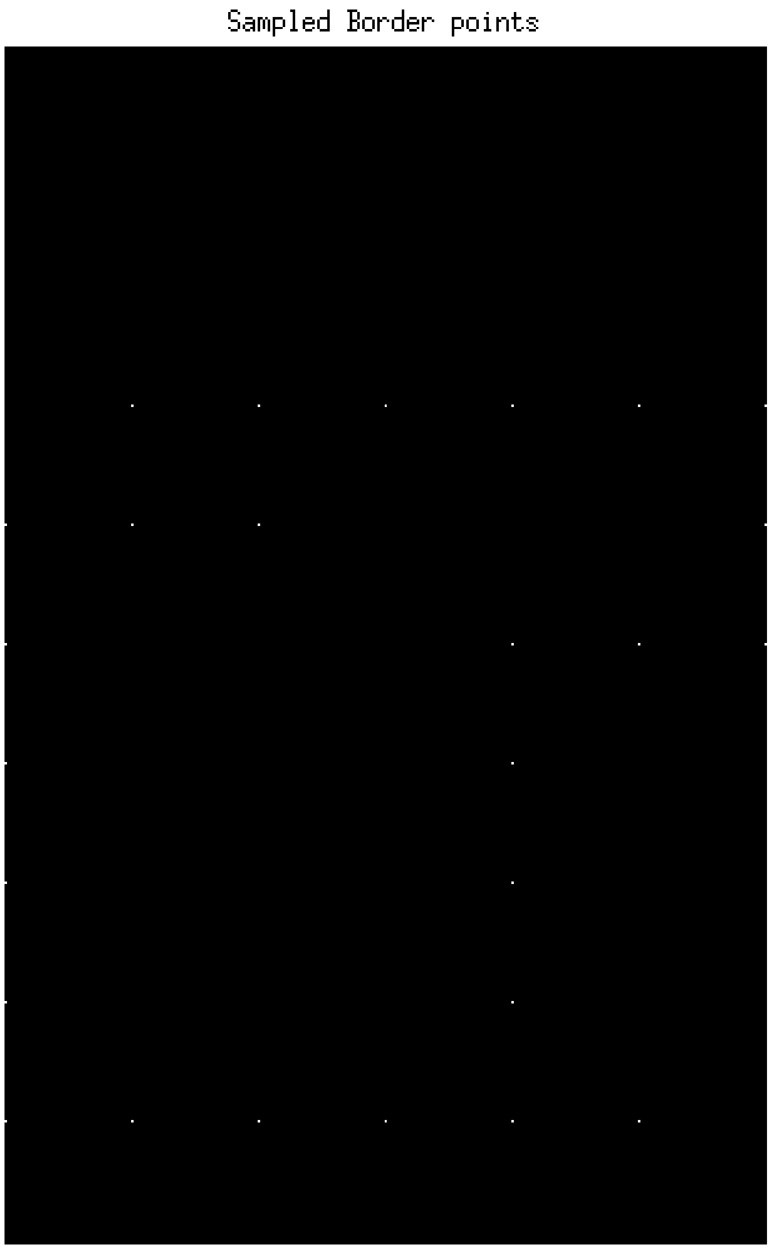
```
Warning: Image is too big to fit on screen; displaying at 67%
```



downsample border

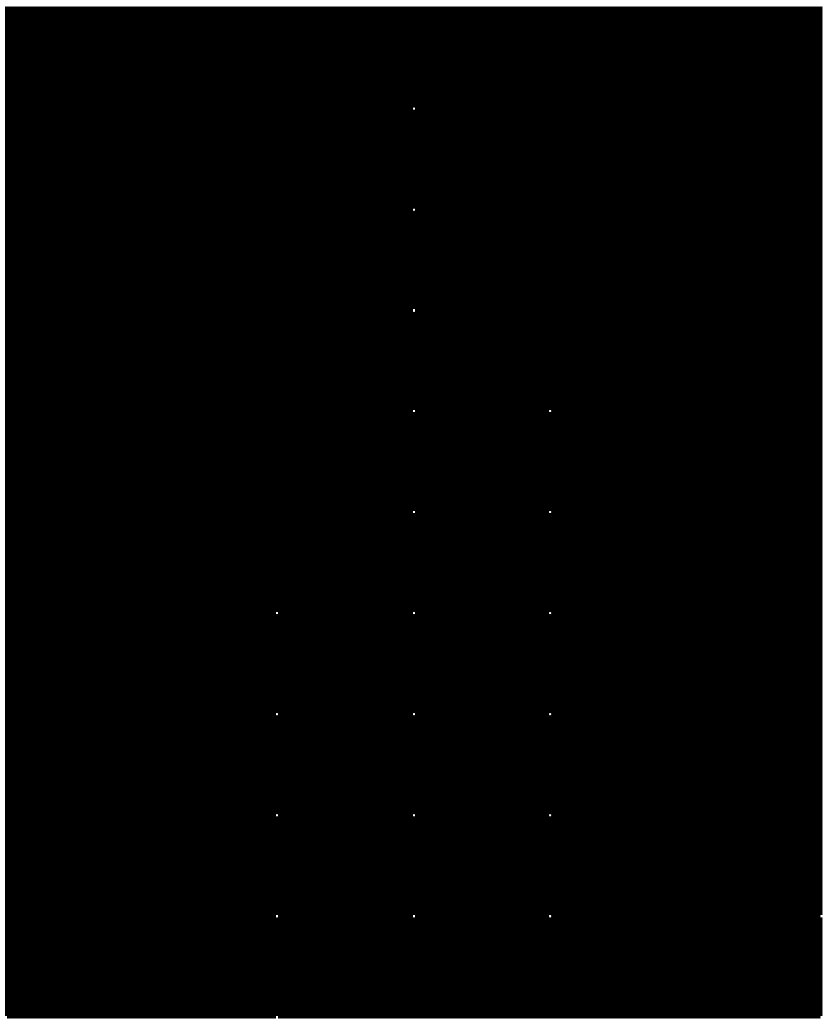
```
% sampled grid dim
rowGridLen = round(m/10);
colGridLen = round(n/6);
sampledBorder = [];

for i = 1:rowGridLen:m
    for j = 1:colGridLen:n
        border(round(i),round(j)) = 127;

         % find which corner of the grid border points in that grid lie
        for p = i:i+rowGridLen
            for q = j:j+colGridLen
```

```
                %check boundary of grid
                if p > m
                     p = m;
                end
                if q >n
                    q = n;
                end

                %find the corners where border points lie
                if(border(p,q) == 255)
                  if p < (i+rowGridLen)/2 && q < (j+colGridLen)/2
                      t = [i j];
                      sampledBorder = [sampledBorder; t];
                  elseif p >= (i+rowGridLen)/2 && q < (j+colGridLen)/2
                      t = [i+rowGridLen j];
                      sampledBorder = [sampledBorder; t];
                  elseif q < (j+colGridLen)/2 && q >= (j+colGridLen)/2
                      t = [i j+colGridLen];
                      sampledBorder = [sampledBorder; t];
                  else
                      t = [i+rowGridLen j+colGridLen];
                      sampledBorder = [sampledBorder; t];
                  end
                end
            end
        end
    end
end

% sampled border image
sampledOut = zeros(m,n);
sampledBorder = unique(sampledBorder,'rows');

for i = 1:size(sampledBorder,1)
   sampledOut(sampledBorder(i,1), sampledBorder(i,2)) = 255;

end
figure;
imshow(sampledOut);
title('Sampled Border points');
```

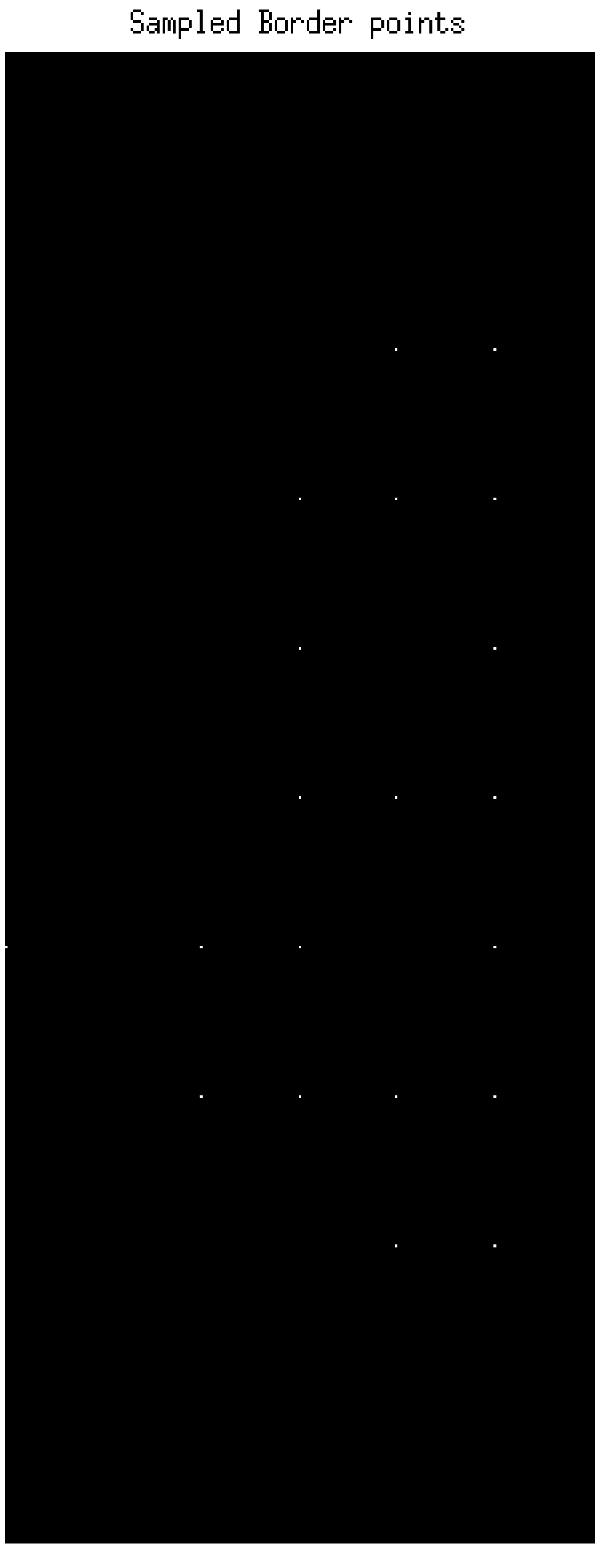


Warning: Image is too big to fit on screen; displaying at 67%

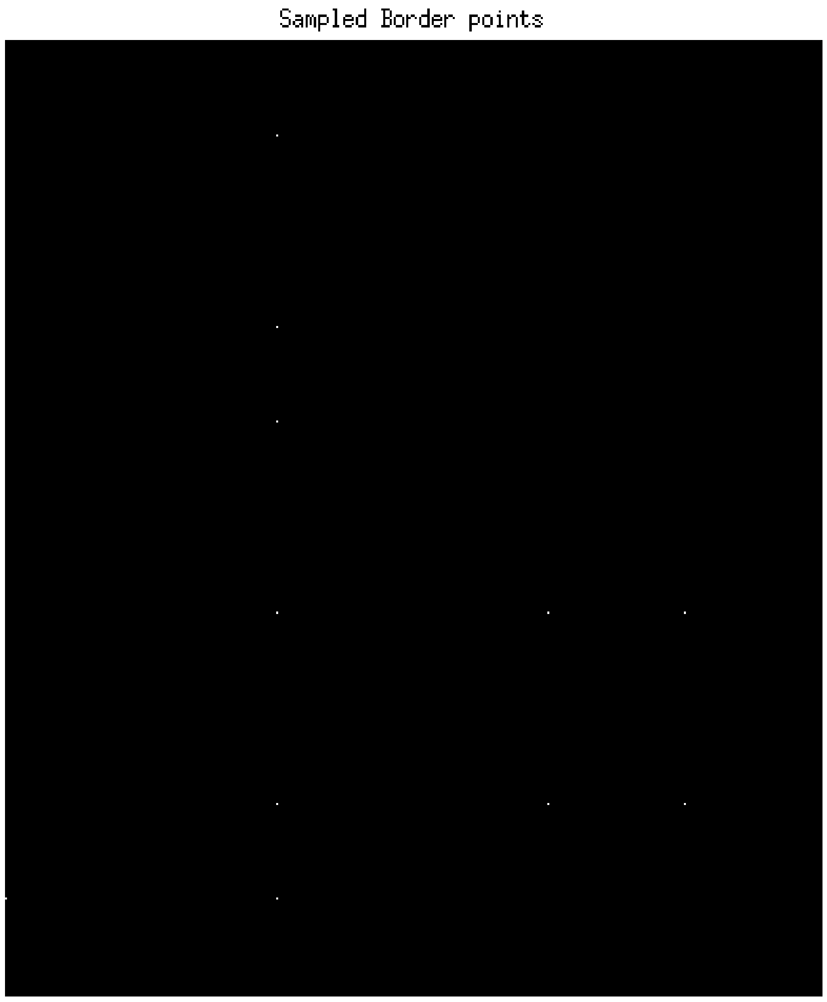
Sampled Border points



Warning: Image is too big to fit on screen; displaying at 67%


Sampled Border points

```
Warning: Image is too big to fit on screen; displaying at 67%
```



get chain code

```
disp(['chain code for image' asana(r)]);
chain = getChain(sampledBorder);
disp(chain);
```

```
    'chain code for image'    'Ustrasana'

     3
     2
     1
     1
     3
     1
     1
```

```
 1
 3
 1
 3
 1
 3
 1
 3
 2
 1
 1
 1
 1

'chain code for image'    'veerbhadrasan'

 2
 2
 2
 1
 2
 1
 3
 2
 1
 3
 3
 2
 1
 3
 3
 2
 1
 1
 3
 3
 2
 1
 1
 1
 3
 3
 3
 1
 1

'chain code for image'    'vrikhsasana'

 2
 1
 3
 2
 1
 3
 1
 3
 2
 1
 3
 3
 3
 3
 1
 3
 3
 2
 1
 2
 1
 2
 1
 2
 1

'chain code for image'    'trikonasana'

 2
 1
 2
 2
 1
 3
 2
 1
 1
 3
 2
 1
 1
 2
 1
 1
 1
 2
 1
 1
 1
 1
 3
 3
 2
 1
 1
 3
 3
```

```
     2
     1
```

get normalized chain

```
disp(['normalized chain code for image' asana(r)]);
nChain = normalizeChain(chain);
disp(nChain)
r = r+1;

save(num2str(r-1), 'nChain');
```

```
    'normalized chain code for image'    'Ustrasana'

     1
     1
     1
     1
     3
     2
     1
     1
     3
     1
     1
     1
     3
     1
     3
     1
     3
     1
     3
     2


    'normalized chain code for image'    'veerbhadrasan'

     1
     1
     1
     3
     3
     3
     1
     1
     2
     2
     2
     1
     2
     1
     3
```

```
 2
 1
 3
 3
 2
 1
 3
 3
 2
 1
 1
 3
 3
 2

'normalized chain code for image'    'vrikhsasana'

 1
 2
 1
 2
 1
 2
 1
 2
 1
 3
 2
 1
 3
 1
 3
 2
 1
 3
 3
 3
 3
 1
 3
 3
 2

'normalized chain code for image'    'trikonasana'

 1
 1
 1
 1
 3
 3
 2
```

```
                  1
                  1
                  3
                  3
                  2
                  1
                  2
                  1
                  2
                  2
                  1
                  3
                  2
                  1
                  1
                  3
                  2
                  1
                  1
                  2
                  1
                  1
                  1
                  2


    end

end
```

Published with MATLAB® 8.0