Matthew Vaysfeld
I pledge my honor that I have abided by the Stevens Honor System

Code

```
function main = main(image,sigma,threshold)
%Default Values
arguments
    image string = "plane.pgm";
    sigma string = "2";
    threshold string = "70";
end
sigma = str2double(sigma);
threshold = str2double(threshold);

%Read Image
image = imread(image);

%Gets the magnitude matrix using the gaussian image and applying
both
%sobels on it (gaussian done inside magnitude function)
sobel_matrix = magnitude(double(image),sigma,threshold);

%Finds Orientation Matrix
orientation_matrix = orientation(double(image),sigma);

%Uses Magnitude Matrix and Orientation Matrix to find the Non-
Maximum
%suppresion of the image
non_max = non_maxsuppression(sobel_matrix,orientation_matrix);


imshow(non_max);
end


%Pads Matrix
function padded_array = pad(image,sigma)
    %original rows, original colums
    [or,oc] = size(image);
    new_sigma = 5*sigma;
    m = zeros(or+(2*new_sigma), oc+(2*new_sigma), "double");
    m(new_sigma+1:or+new_sigma,new_sigma+1:oc+new_sigma)=image;

    %Top Left Corner
    for r = 1:new_sigma+1
        for c = 1:new_sigma+1
        m(r,c) = m(new_sigma+1,new_sigma+1);
   end
end
%Left side
for r = new_sigma+1:or+new_sigma-1
    for c = 1:new_sigma+1
        m(r,c) = m(r,new_sigma+1);
    end
end
 %Bottom Left Corner
 for r = or+new_sigma:or+(2*new_sigma)
    for c = 1:new_sigma+1
        m(r,c) = m(or+new_sigma,new_sigma+1);
   end
end
 %Top
for r = 1:new_sigma
     for c = new_sigma+1:oc+new_sigma
         m(r,c) = m(new_sigma+1,c);
     end
 end
 %Top Right Corner
 for r = 1:new_sigma+1
    for c = oc+new_sigma+1:oc+(2*new_sigma)
         m(r,c) = m(new_sigma+1,oc+new_sigma);
    end
 end
 %Right Side
 for r = new_sigma+1:or+new_sigma-1
     for c = oc+new_sigma+1:oc+(2*new_sigma)
         m(r,c) = m(r,oc+new_sigma);
     end
 end
 %Bottom Right Corner
 for r = or+new_sigma:or+(2*new_sigma)
     for c = oc+new_sigma+1:oc+(2*new_sigma)
         m(r,c) = m(or+new_sigma,oc+new_sigma);
    end
 end
 %Bottom
 for r = or+new_sigma:or+(2*new_sigma)
     for c = new_sigma+1:oc+new_sigma
         m(r,c) = m(or+new_sigma,c);
    end
 end
padded_array=m;
```

```
end

%Applies any filter to an image
%Takes in a filter, image, and the extended image
function apply_filter = app(filter,image,ext_image)
    [row,col] = size(image);
    [re,ce] = size(ext_image);
    [~,len] = size(filter);
    newx = re - row;
    newy = ce - col;
    image2 = zeros(row, col);
    for r = 1:row
        for c = 1:col
            newr = r +((newx)/2);
            newc = c +((newy)/2);
            piece = ext_image((newr)-((len-1)/2):(newr)+((len-
1)/2),(newc)-((len-1)/2):(newc)+((len-1)/2));
            %mult_matrix = mult_matrices(filter,piece);
            mult_matrix = filter .* piece;
            image2(r,c) = sum(mult_matrix(:));
        end
    end
    apply_filter = image2;
end



%Gets the Gaussian Filter
function get_gaussian = gauss(sigma)
   size = 10*sigma;
   filter = zeros(size-1,size-1);
   for r = 1:size-1
        for c = 1:size-1
             x = abs(r - size/2);
             y = abs(c - size/2);
             filter(r,c) = (1/(2*pi*sigma))*exp((-(x^2 +
y^2))/(2*sigma^2));
        end
   end
   sumof = sum(filter(:));
   get_gaussian = filter;
end

%Applies the Gaussian filter
function appgauss = appgauss(image,sigma)
    %Do the Gauss
     gaussian = gauss(sigma);
     padded_matrix = pad(image,sigma);
     appgauss = app(gaussian,image,padded_matrix);
end

%Applies the Vertical Sobel Filter
function sob1 = sobel1(image,gaussimage)
     single_pad = pad(gaussimage,1);
     sobelfilt1 = [-1 0 1; -2 0 2; -1 0 1];
     sob1 = app(sobelfilt1,gaussimage,single_pad);
     sob1 = double(sob1);

end

%Applies the Horizontal Sobel Filter
function sob2 = sobel2(image,gaussimage)
     single_pad = pad(gaussimage,1);
     sobelfilt2 = [1 2 1; 0 0 0; -1 -2 -1];
     sob2 = app(sobelfilt2,gaussimage,single_pad);
     sob2 = double(sob2);
end

 function magnitude = magnitude(image,sigma,threshold)
     %Apply gauss on image, apply both sobel on gauss image,
calc gradient
     %magnitude for each using formula, under certain threshhold
get rid of
     %pixels
     [row,col] = size(image);

     %Do the Gauss
     gaussimage = appgauss(image,sigma);

     %App both Sobs
     sob1 = sobel1(image,gaussimage);
     sob2 = sobel2(image,gaussimage);
     image2 = gaussimage;

     %Gradient Matrix
     for r = 1:row
         for c= 1:col
             image2(r,c) = sqrt(((sob1(r,c))^2) +
((sob2(r,c))^2));
             %Under threshold remove
             if(image2(r,c)<threshold)
                image2(r,c) = 0;
             end
         end
     end

     magnitude = image2;

 end

 %Gets the orientation matrix for non-max suppression
 function orientation = orientation(image,sigma)
    [row,col] = size(image);
    gaussimage = appgauss(image,sigma);
    sob1 = sobel1(image,gaussimage);
    sob2 = sobel2(image,gaussimage);
    image2 = gaussimage;
    for r = 1:row
         for c= 1:col
             image2(r,c) =
atan(double(sob2(r,c))/double(sob1(r,c)));
         end
    end
    orientation = image2;
 end


%Gets the image with non-maximum suppresion
function non_max = non_maxsuppression(magnitude,orientation)
    [row,col] = size(orientation);
    m = zeros(row,col, "double");
    magnitude2 = pad(magnitude,1);

    for r = 1:row
         for c= 1:col
             newr=r+3;
             newc=c+3;
             %Horizontal
             if((orientation(r,c) >= (-pi/2)) &&
(orientation(r,c) <= (-3*pi/8)))
                 if(magnitude2(newr,newc) > magnitude2(newr-
1,newc) && magnitude2(newr,newc) > magnitude2(newr+1,newc))
                    m(r,c) = 255;
                 else
                    m(r,c) = 0;
                 end
             end
             %Vertical
             if( (orientation(r,c) > (-3*pi/8)) &&
(orientation(r,c) <= (-pi/8)))
```

```
                 if(magnitude2(newr,newc) > 
magnitude2(newr,newc-1) && magnitude2(newr,newc) > 
magnitude2(newr,newc+1))
                    m(r,c) = 255;
                 else
                    m(r,c) = 0;
                 end
             end
             %Left Diagonal
             if( (orientation(r,c) > (-pi/8)) && 
(orientation(r,c) <= (pi/8)) )
                 if(magnitude2(newr,newc) > magnitude2(newr-
1,newc-1) && magnitude2(newr,newc) > magnitude2(newr+1,newc+1))
                    m(r,c) = 255;
                 else
                    m(r,c) = 0;
                 end
             end
             %Right Diagonal
             if( (orientation(r,c) > (pi/8)) && 
(orientation(r,c) <= (3*pi/8)))
                 if(magnitude2(newr,newc) > 
magnitude2(newr+1,newc-1) && magnitude2(newr,newc) > 
magnitude2(newr-1,newc+1))
                     m(r,c) = 255;
                 else
                    m(r,c) = 0;
                 end
             end
             %Also Horizontal
             if( (orientation(r,c) > (3*pi/8)) && 
(orientation(r,c) <= (pi/2)))
                 if(magnitude2(newr,newc) > magnitude2(newr-
1,newc) && magnitude2(newr,newc) > magnitude2(newr+1,newc))
                    m(r,c) = 255;
                 else
                    m(r,c) = 0;
                 end
             end

         end
    end
         non_max = m;

    end
```

Resulting Images

Part1 Gaussian Filters

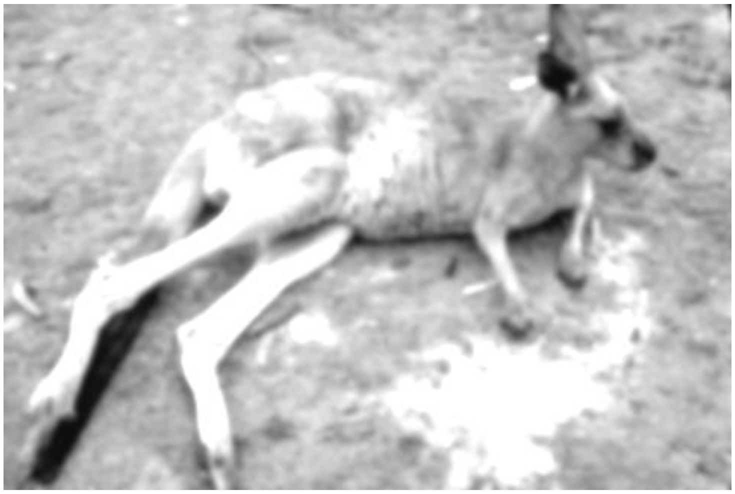


Image: kangaroo

Sigma : 2

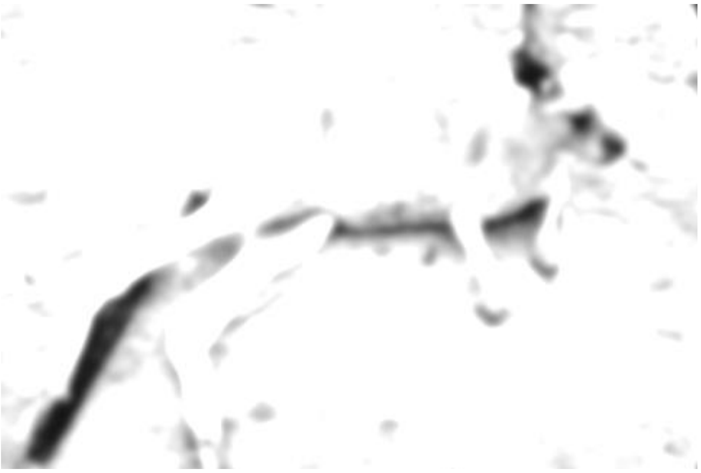


Image: kangaroo

Sigma : 3

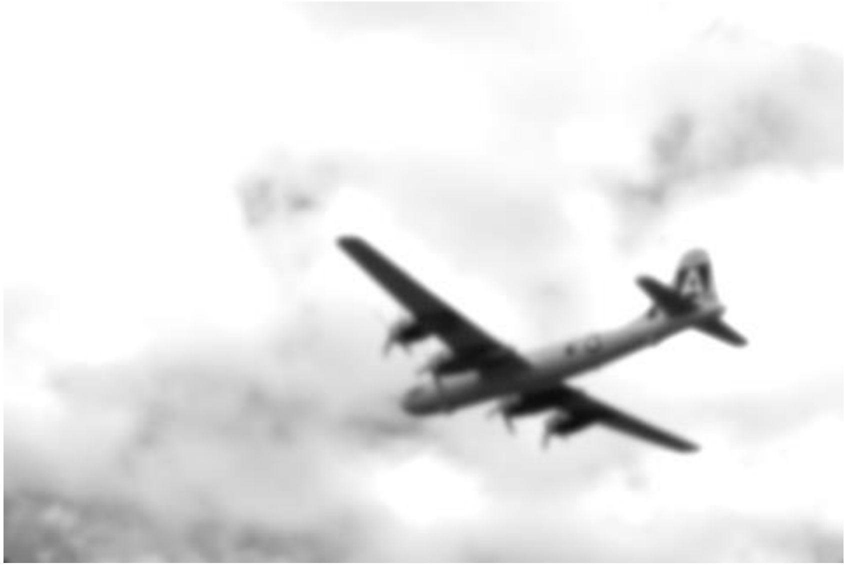


Image: plane

Sigma : 2

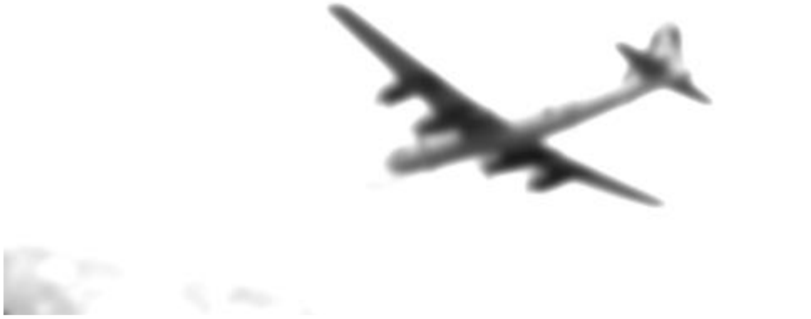


Image: plane

Sigma : 3

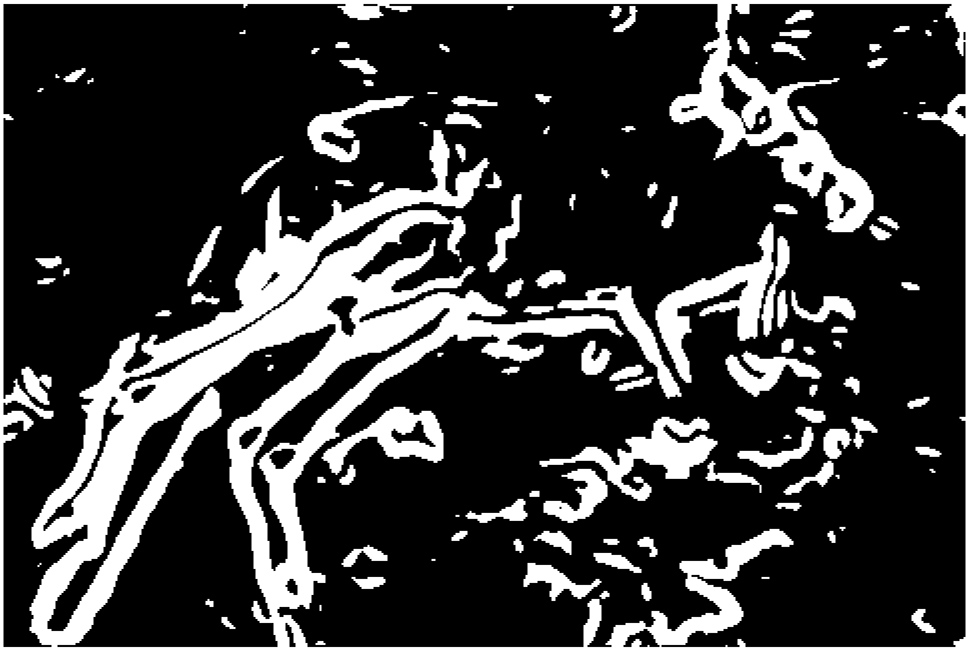
Image: red

Sigma : 2

Image: red

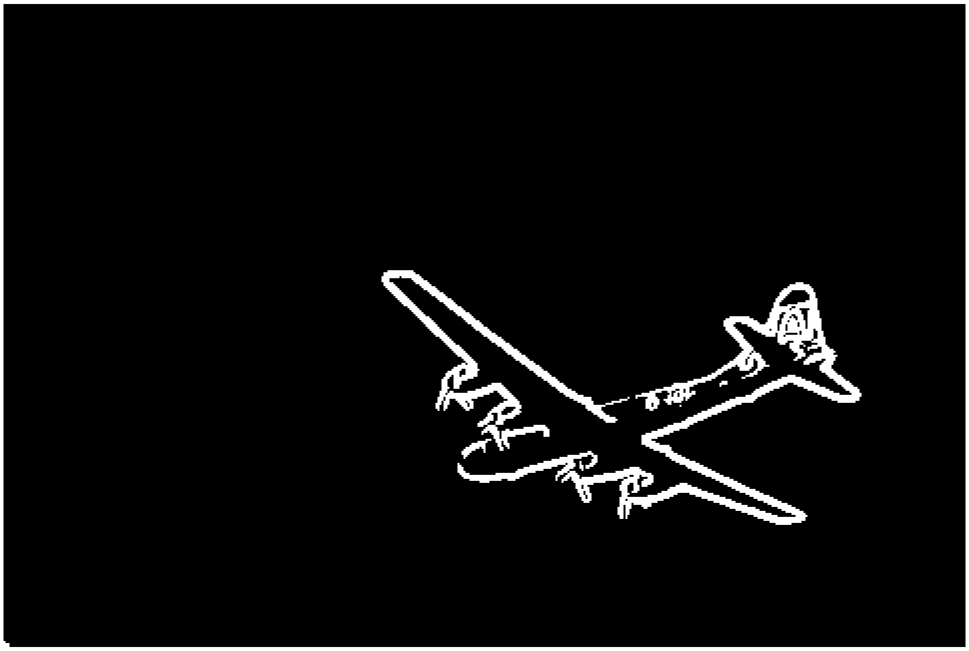
Sigma : 3

Part 2 Sobel Magnitude



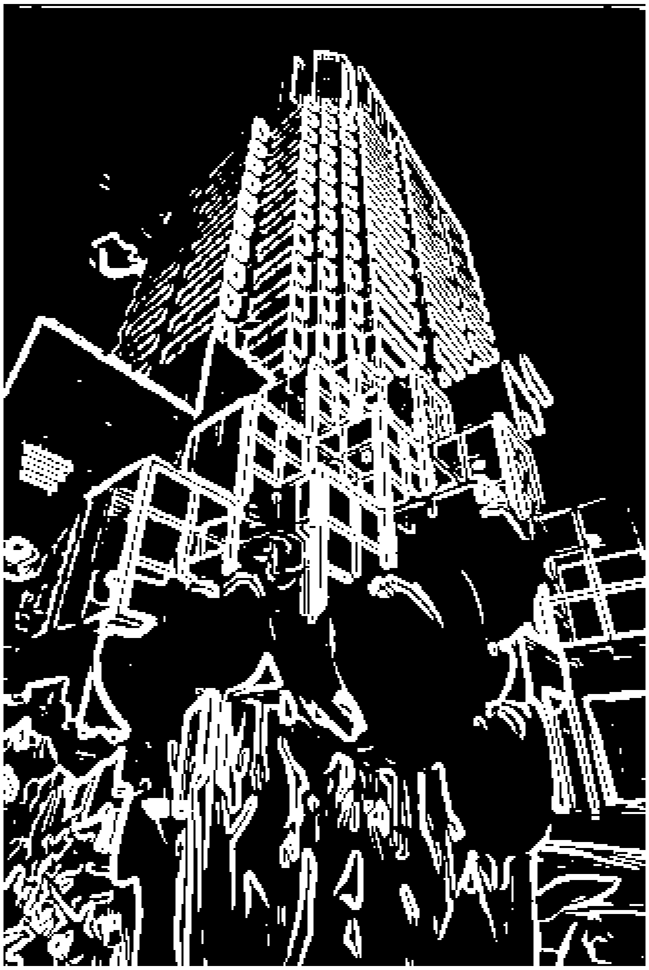
Sigma = 2

Threshold = 70



Sigma = 1

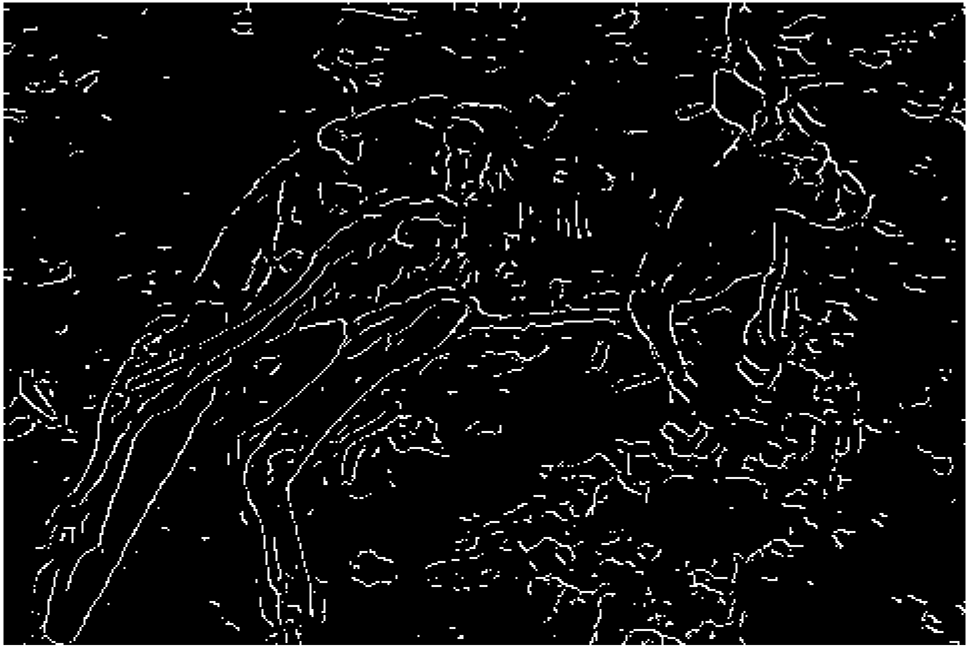
Threshold = 70



Sigma = 1

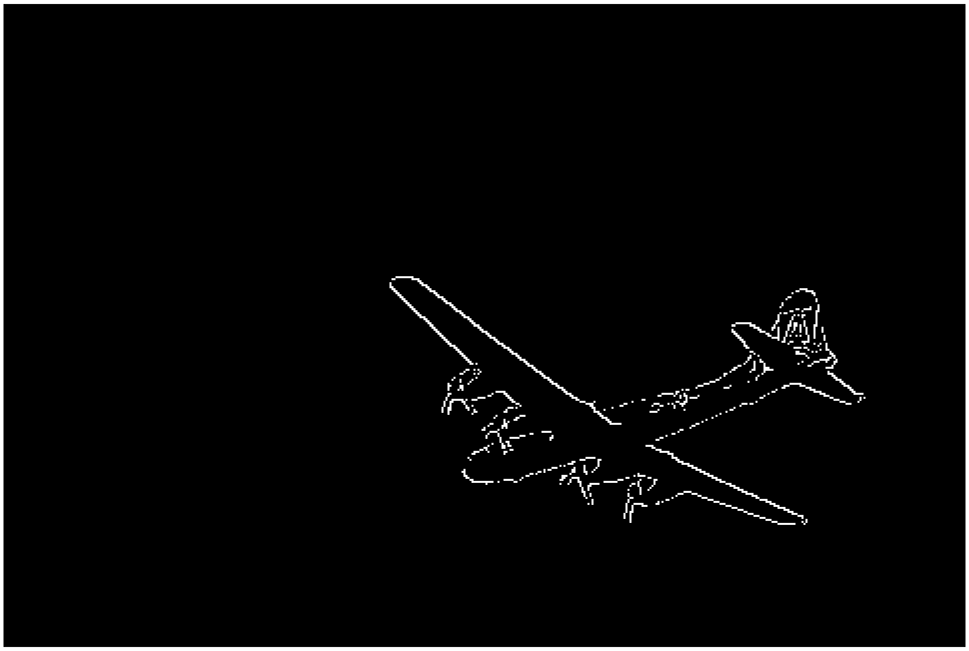
Threshold = 70

Part 3 : Non-maximum Suppression



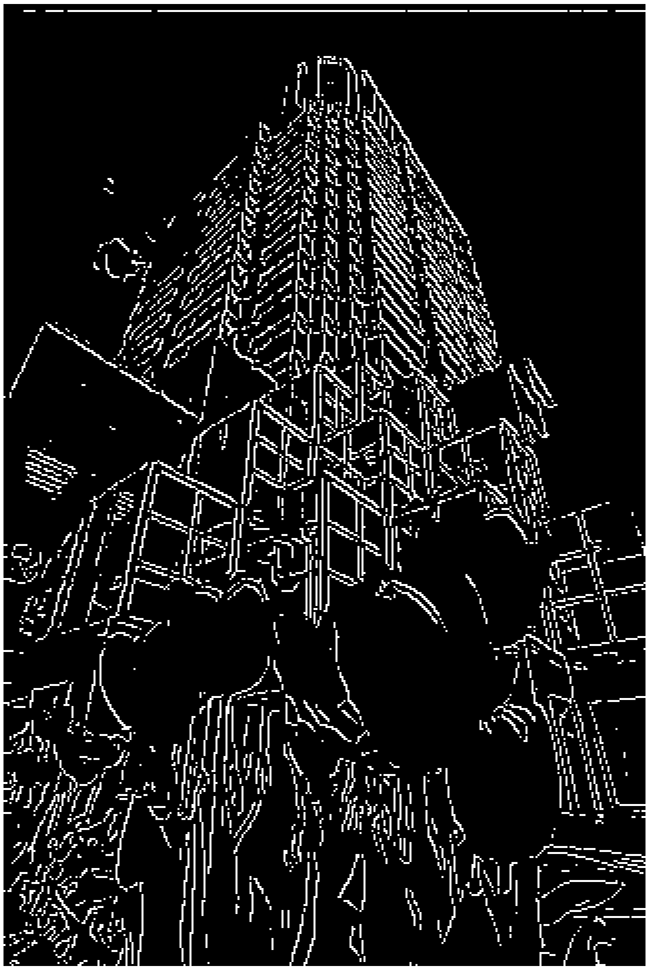
Sigma = 2

Threshold = 70



Sigma = 1

Threshold = 70



Sigma = 1

Threshold = 70

I split up the code into 3 major steps as outlined by the homework. First I applied the gaussian filter to the image, then found the magnitude matrix of the image, and then found the non-maximum suppressed image. I further split up my work into smaller steps.

Step 1 (Finding the Gaussian):

In order to do this I first padded the matrix as described in the assignment ("replicate boundary pixels when the filter window falls out of bounds"). Then I created the gaussian matrix based on the sigma inputed. **Note: my gaussian matrix is of size 10*sigma, with a half length of 5*sigma, since this way the gaussian matrix sums to 1.**

Then I created a function that applied all filters given the image, the filter, and the padded image. This was later used to apply the sobel filters, but in this case it was used to create a function, appgauss, that would apply the gaussian filter generated by the sigma inputted onto an image.

Step 2 (Finding the Magnitude Matrix):

In order to find the Magnitude matrix I used my apply filter function (app) to apply both the vertical and horizontal sobel filters onto the image which has had the gaussian applied to it. Then I calculate the magnitude using the formula `sqrt(((sob1(r,c))^2) + ((sob2(r,c))^2)).`

Step 3 (Finding the non-maximum suppression image):

This step used all of the previous functions in order to be accomplished. I also generate an orientation matrix. With both the magnitude and orientation matrices I created an algorithm that identifies which direction the pixel is and then check the neighbors based on that. If it is greater than both of its neighbors I set the pixel to 255 (white) and otherwise 0 (black).