Visual Computing 2016
Image Processing Assignment

Siyu Zhou
77729957
szhou7

Part 0. Getting Started

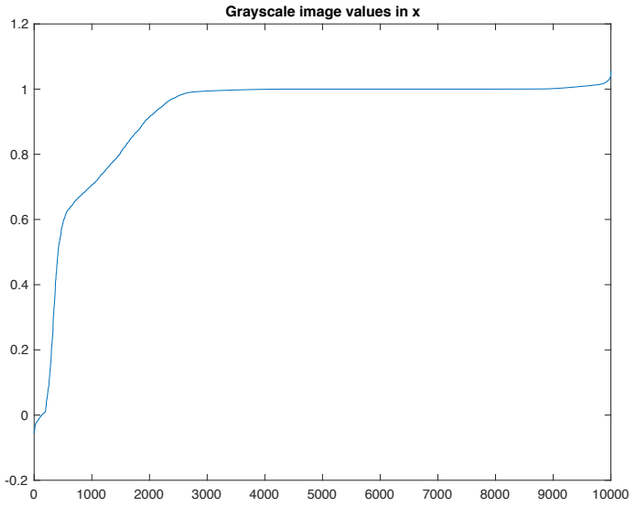
a. First read the image use: $[image, map] = imread('');$
and we have to check the input image use: if $map > 1;$
if it is a grayscale image. If not, transform the input image to grayscale type use $rgb2gray(image)$, else remain the same input image. Then transform the grayscale image to double type use $gray = im2double(image)$. Later we have to resize the image to 100×100 matrix form.

$$gray = imresize(gray, [100, 100]);$$

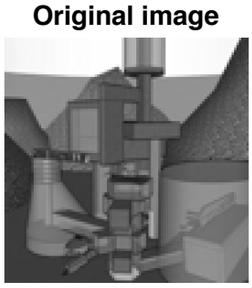
To put all the intensities in the grayscale image into a single 10000-dimensional vector, we use:

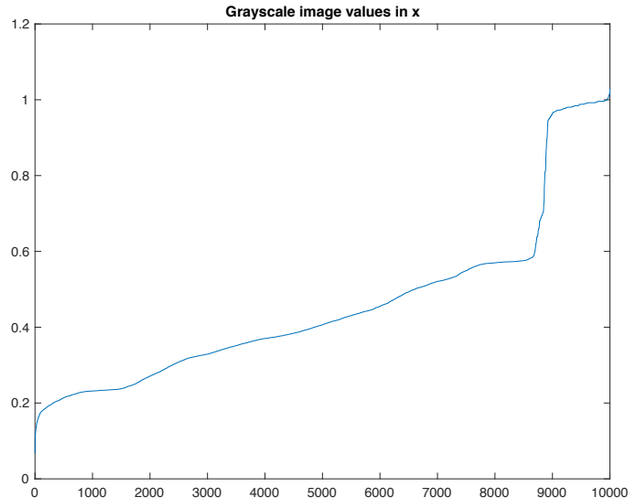
$$gray2 = gray';$$
$$y = gray(:)';$$

We use $sort(y)$ to sort the intensities in vector y. Then $plot(x, ysort)$. (Attention, x should have the same number of element as ysort, thus $x = 1:1:(row * column)$, row and column are the row number and column number of gray use $[row, column] = size(gray)$).
Here is the result output of each image in gallery.
Kitty:



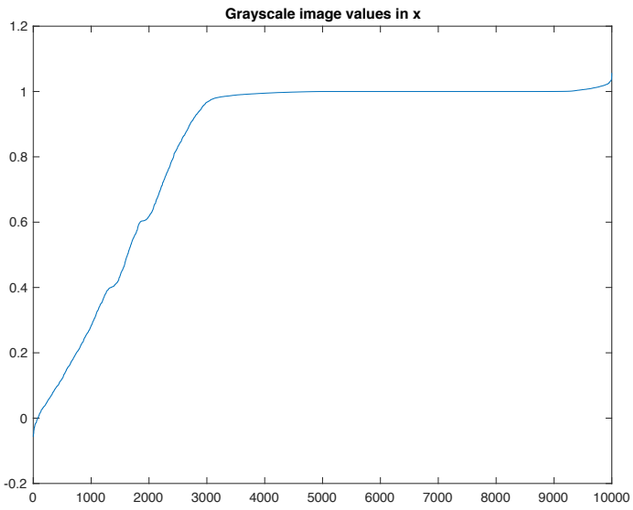
Polarcities:





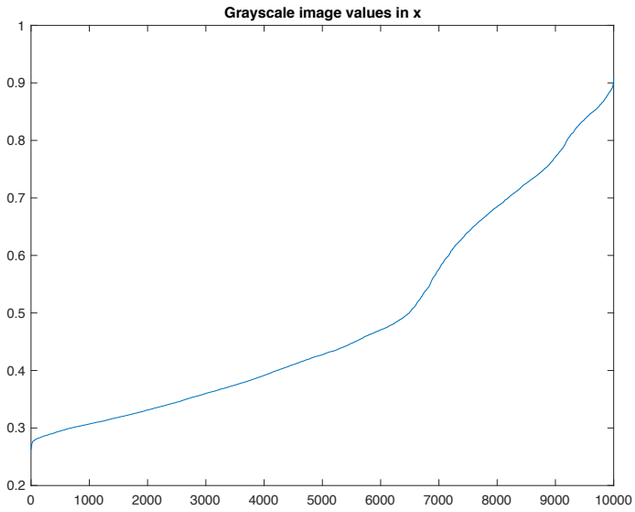
Cartoon:

Original image



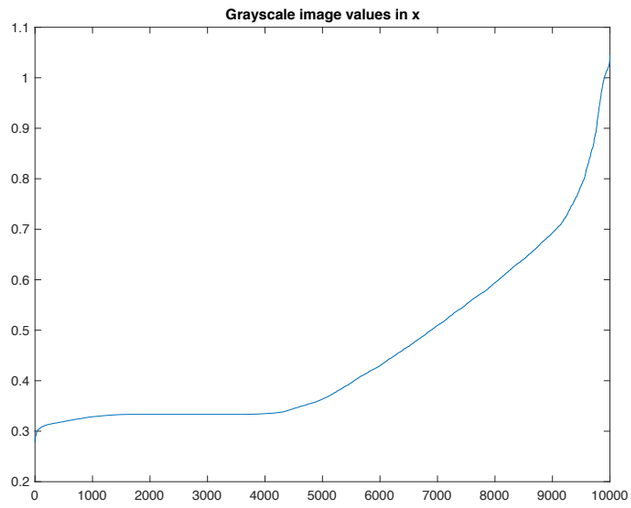
Flowergray:

Original image



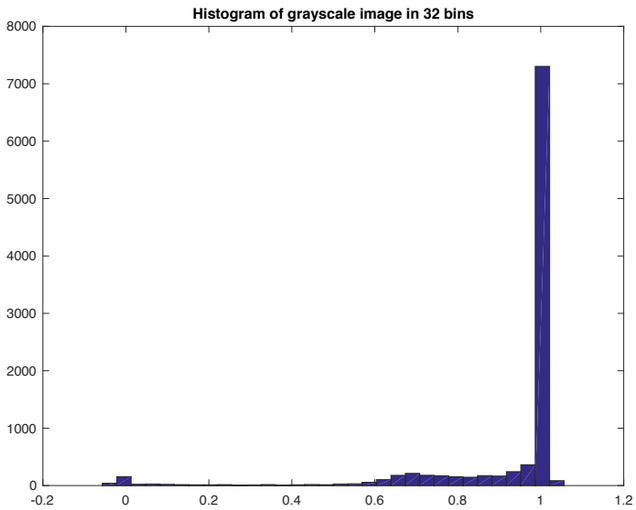
Text:

Original image



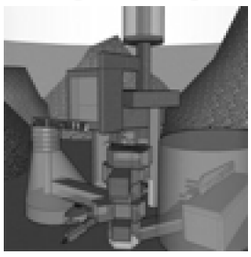
b. To show the histogram of a grayscale image, we can directly use the function: hist(y, binsnumber). Here are the results of each image in gallery.
Kitty:

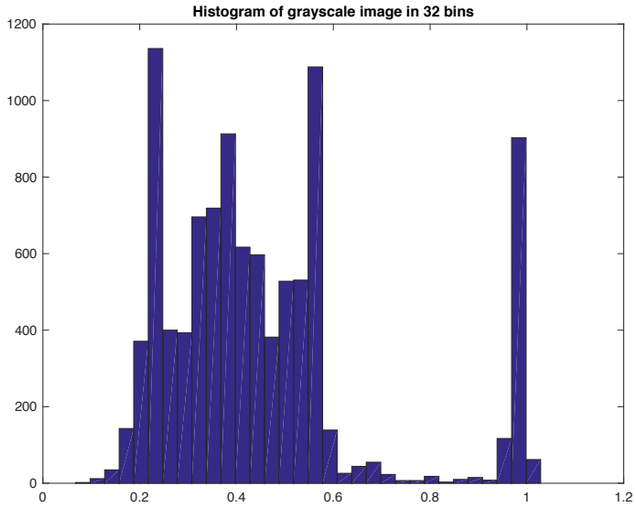
Original image



Polarcities:

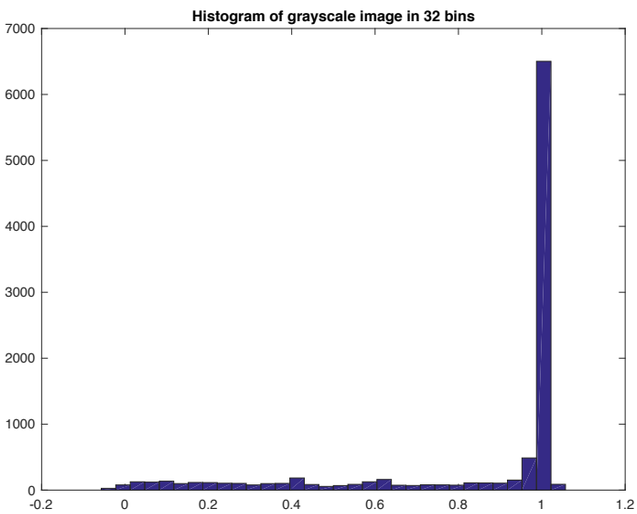
Original image





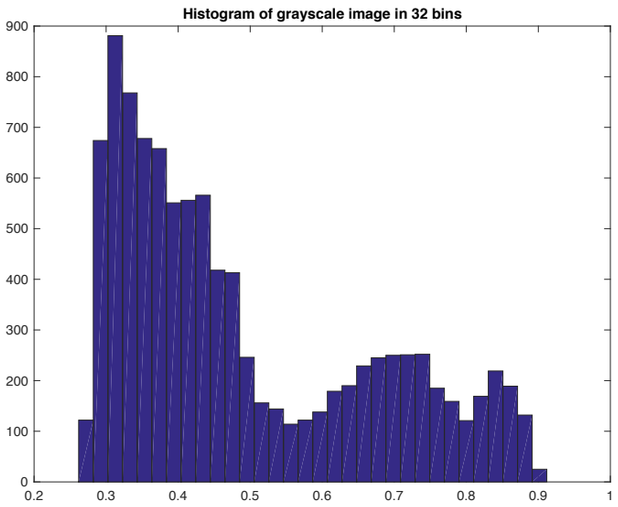
Cartoon:

Original image



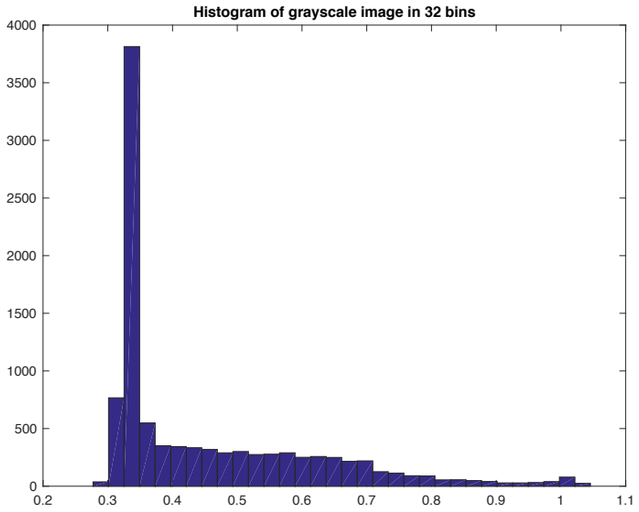
Flowergray:

Original image



Text:

Original image



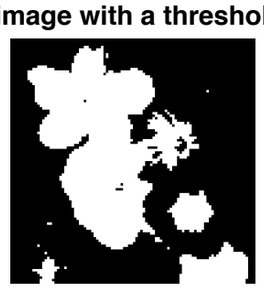
c. To show a binary image with specific threshold, I use $\text{binaryimage} = (\text{gray} > 0.5)$. So here are the results of each test image's binary image with a 0.5 threshold.

Binary image with a threshold of 0.5

Binary image with a threshold of 0.5

Binary image with a threshold of 0.5

Binary image with a threshold of 0.5



Binary image with a threshold of 0.5

d. To implement the requirement, I first calculate the mean value of each image by using the function

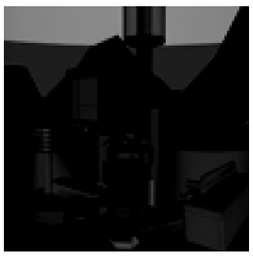
$$\text{average} = \text{mean2}(\text{gray});$$

and then I subtract the mean value from the original image and set the negative value to 0.

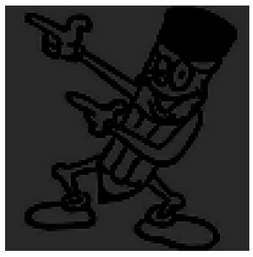
$$\text{subtract} = \text{gray} - \text{average};$$
$$\text{subtract}(\text{subtract} < 0) = 0;$$

Original image subtract mean value

Original image subtract mean value



Original image subtract mean value



Original image subtract mean value

Original image subtract mean value

e. I generate a vector $z = (1:8)$, and then $\text{reshapez} = \text{reshape}(z, [4, 2])$.
And then the final out put is

1	3	5	7
2	4	6	8

f. To create a vector that only has odd numbers:

$$\text{odd} = (1:2:99);$$

and the extracted image is calculated by:

$$\text{extract} = \text{gray}(1:2:\text{end}, 1:2:\text{end}, :);$$

Here are the extracted image of each test image.

Subsample in half size

Subsample in half size

Subsample in half size

Subsample in half size

Subsample in half size

g. As we know from the Wikipedia, Gaussian Filter is an application from Normal Distribution in digital image processing. For Normal Distribution, nearly all of the values lie within the area $(\pm 3\sigma)$ of mean value μ (3-sigma rule). Thus, for a pixel in an image, only the pixels that within the 3σ distance are considerable. Therefore, we only have to calculate the $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$ matrix to guarantee the result.
To compare the results of the Gaussian Filter with different parameters, we choose ([3, 3], 0.5), ([12, 12], 2), ([18, 18], 3) as the hsize and sigma.
Here is the main code of the process of Gaussian filter:

$$\text{width1} = 3;$$
$$\text{sigma1} = \text{width1}/6;$$
$$\text{gaussian1} = \text{fspecial}('\text{gaussian}', [\text{width1 width1}], \text{sigma1});$$
$$\text{blurred1} = \text{imfilter}(\text{gray}, \text{gaussian1}, '\text{replicate}');$$

And here is the result of Gaussian filter with different widths:

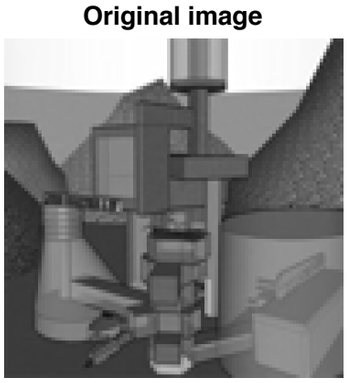
Original image

Width = 3

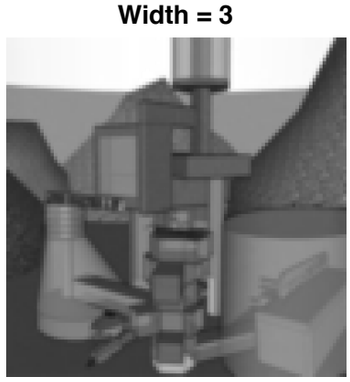
Width = 12

Width = 18

Original image



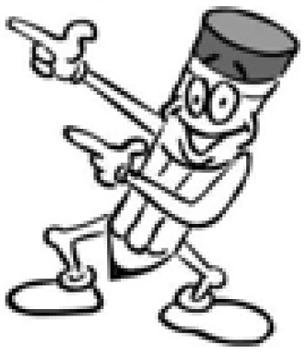
Width = 3



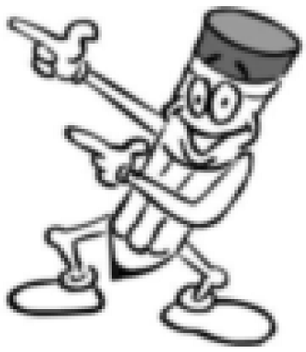
Width = 12

Width = 18

Original image

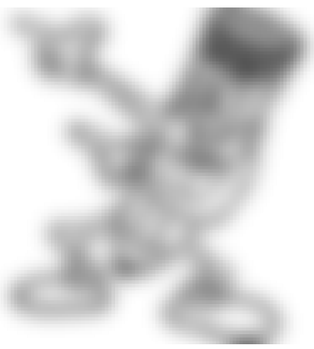


Width = 3

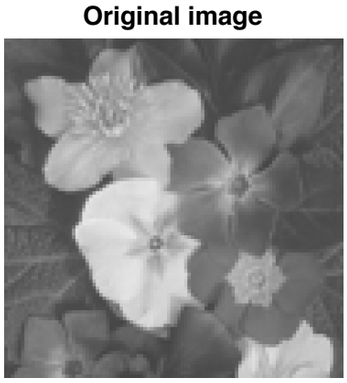


Width = 12

Width = 18

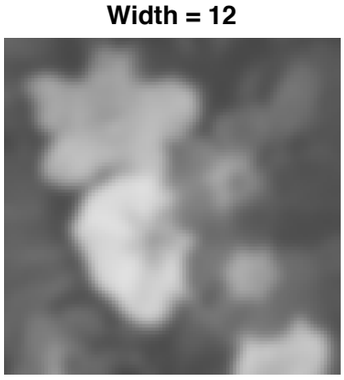


Original image

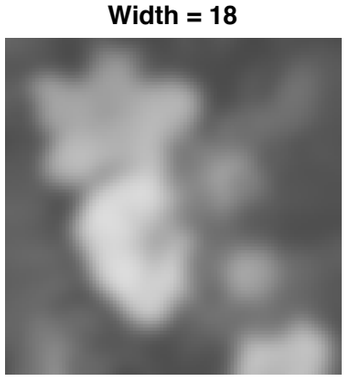


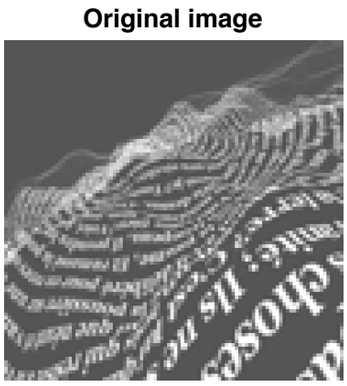
Width = 3

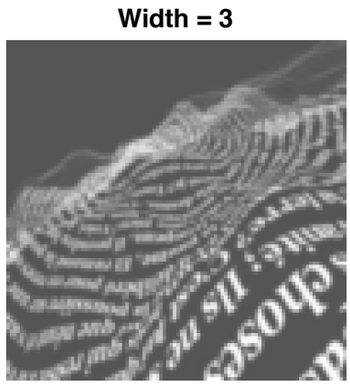
Width = 12

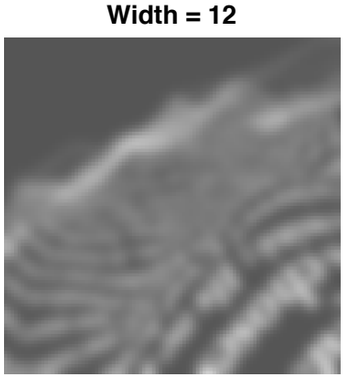


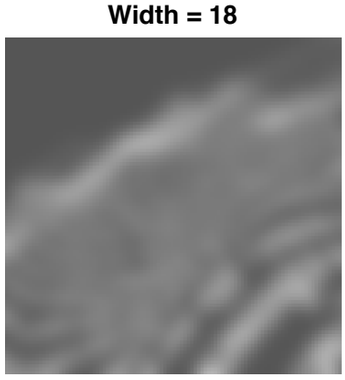
Width = 18









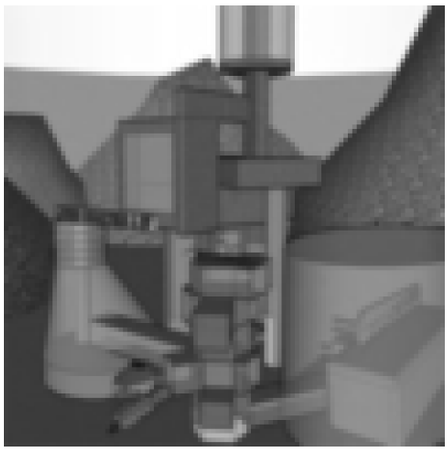


h. If I use conv2() function, and with the default method, I can see that there is black boundary in conv2() result. Because, the default method in conv2() output an image that is full filtered. Or I can change the method to ‘same’ then the output should be the same.
Here are the comparisons for each image:

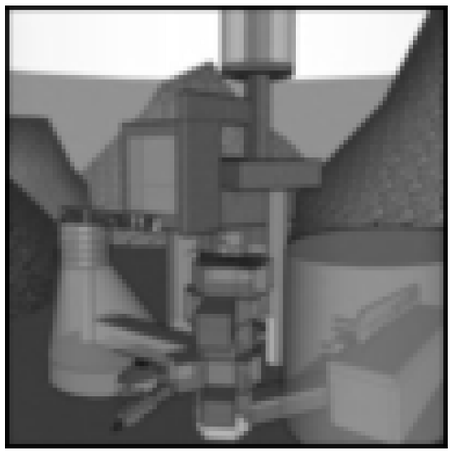
Guassian filter used fspecial and imfilter

Gaussian filter used fspecial and conv2

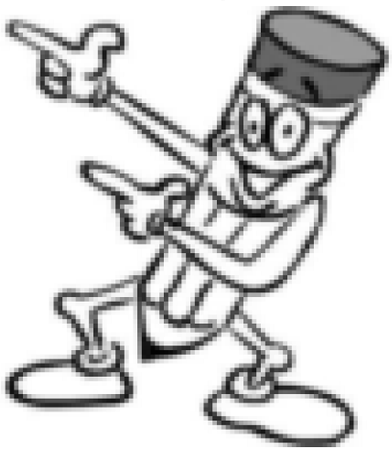
Guassian filter used fspecial and imfilter



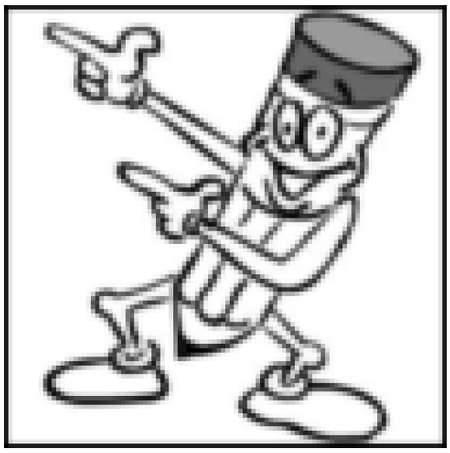
Gaussian filter used fspecial and conv2



Guassian filter used fspecial and imfilter



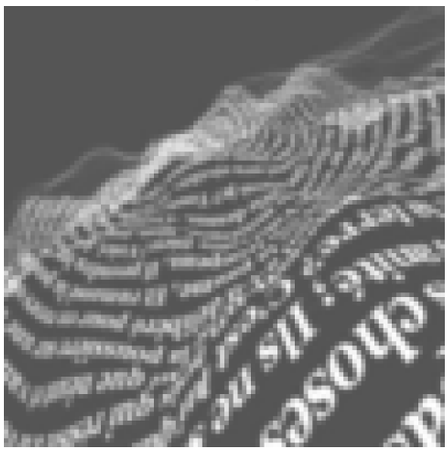
Gaussian filter used fspecial and conv2



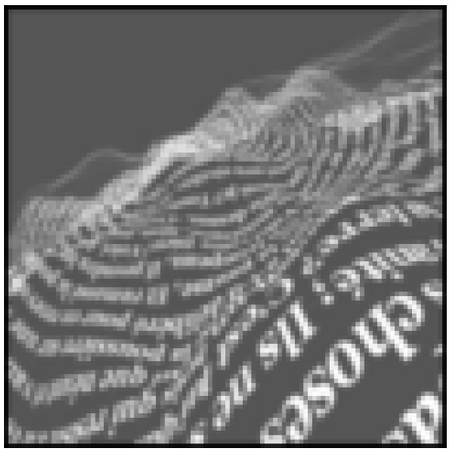
Guassian filter used fspecial and imfilter

Gaussian filter used fspecial and conv2

Guassian filter used fspecial and imfilter



Gaussian filter used fspecial and conv2



Part1: Gaussian Pyramid

To generate Gaussian Pyramid, first I have to transfer the input image to grayscale and double type.

```
image2 = rgb2gray(image);
gray = im2double(image2);
```

And then resize it to 2^N in each dimension.

```
[row, column] = size(gray);
n = 1;
while n < row && n < column
  n = n * 2;
end
n = n/2;
max = n;
```

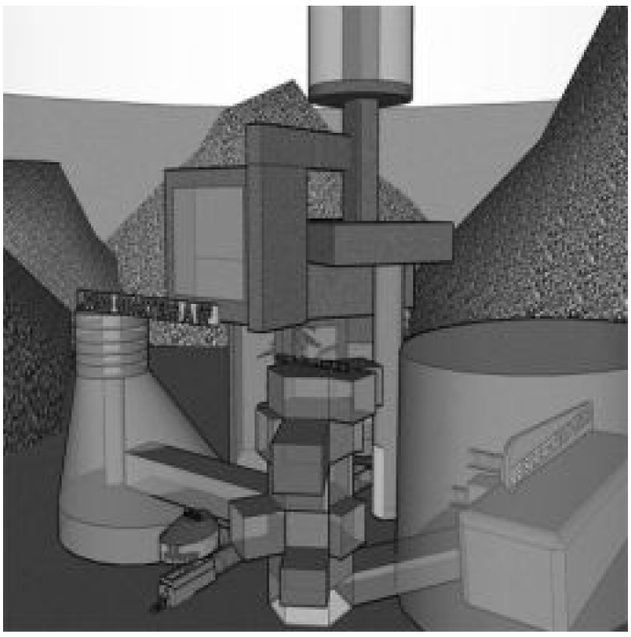
To realize the Gaussian Pyramid, I use a while loop to keep calculating until the size of the image (or the level of the pyramid is coming to the top) is less than 1. Because we are required to output the pyramid with same size, so I resize the each level of the pyramid with the 'bilinear' method.

```
nextlevel = gray;
while n > 1
  nextlevel = impyramid(nextlevel, 'reduce');
  [r, c] = size(nextlevel);
  display([r, c]);
  figure, imshow(nextlevel);
  title('Next level image');
  n = n/2;
  img = imresize(nextlevel, [max, max], 'bilinear');
  [r, c] = size(img);
  display([r, c]);
  figure, imshow(img);
  title('Next level image with same size');
end
```

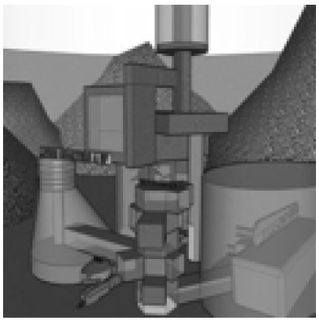
And here are the Gaussian Pyramid of each test image.

Polarcities:

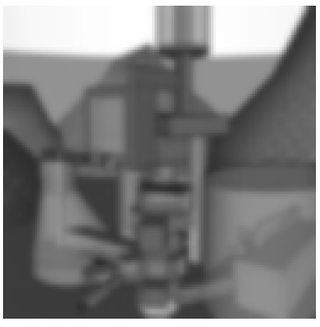
Original image



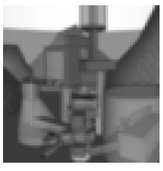
Resized image



Next level image with same size



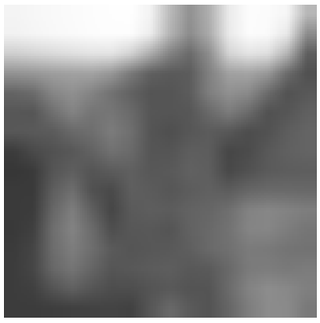
Next level image



Next level image with same size

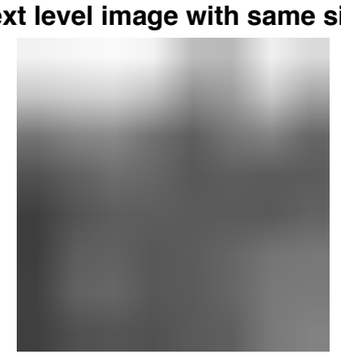
Next level image

Next level image with same size



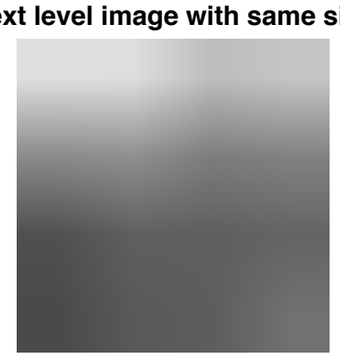
Next level image

Next level image with same size



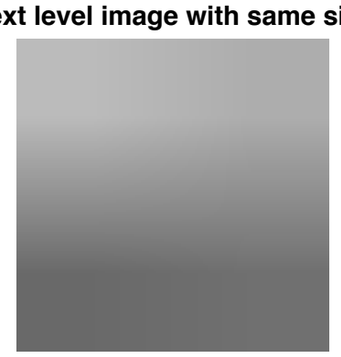
Next level image

Next level image with same size



Next level image

Next level image with same size



Next level image

Next level image with same size

Next level image

kitty:

Original image

Resized image

Next level image with same size

Next level image

Next level image with same size

Next level image

Next level image with same size

Next level image

Next level image with same size

Next level image

Next level image with same size

Next level image

Next level image with same size

Next level image

Next level image with same size

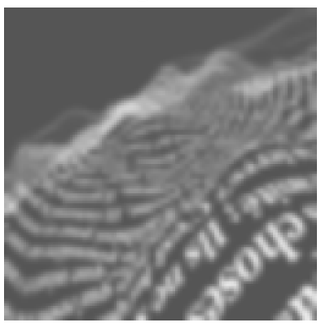
Next level image

Text:

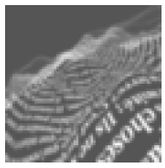
Original image

Resized image

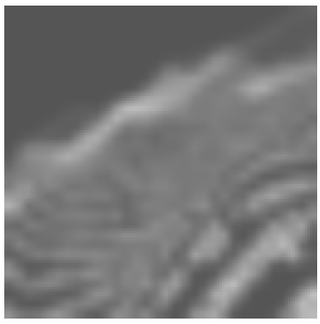
Next level image with same size



Next level image

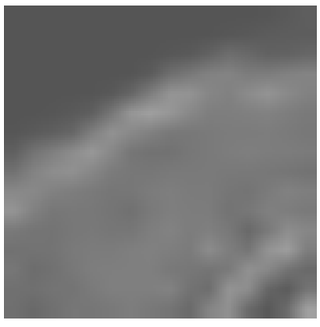


Next level image with same size



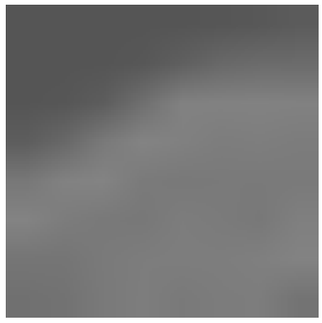
Next level image

Next level image with same size



Next level image

Next level image with same size

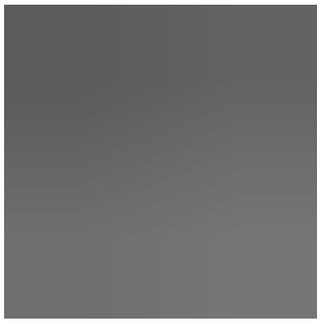


Next level image

Next level image with same size

Next level image

Next level image with same size



Next level image

Next level image with same size

Next level image

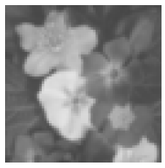
Flowergray:

Original image

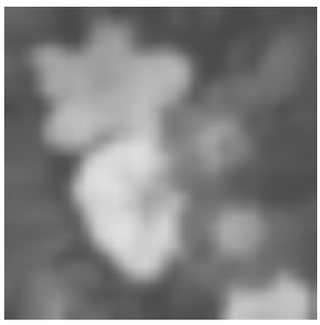
Resized image

Next level image with same size

Next level image

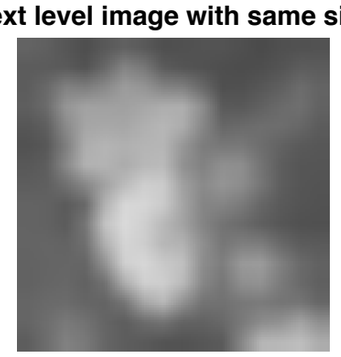


Next level image with same size



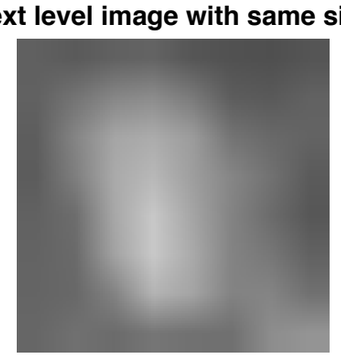
Next level image

Next level image with same size



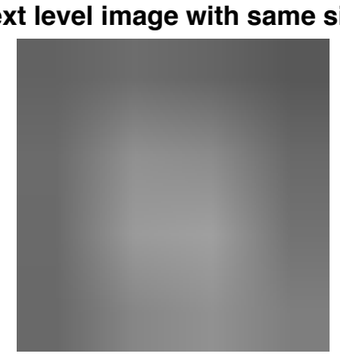
Next level image

Next level image with same size



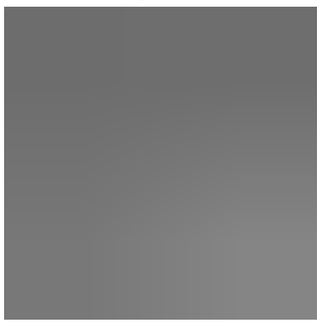
Next level image

Next level image with same size



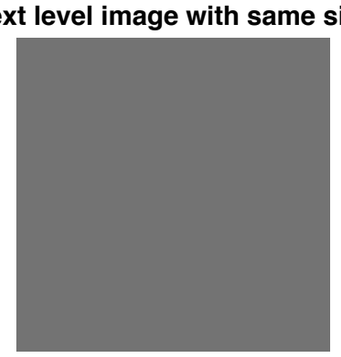
Next level image

Next level image with same size



Next level image

Next level image with same size



Next level image

Cartoon:

Original image

Resized image

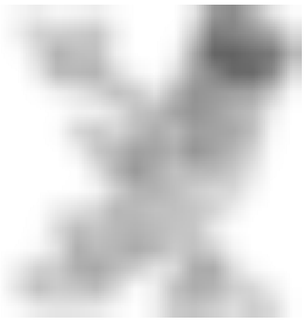
Next level image with same size

Next level image

Next level image with same size

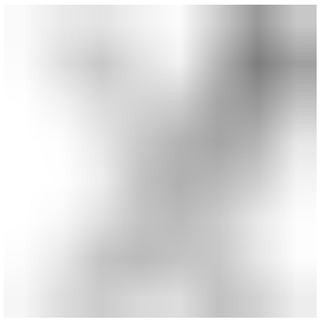
Next level image

Next level image with same size



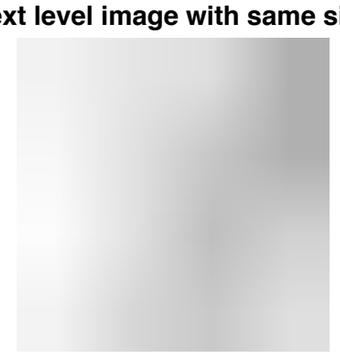
Next level image

Next level image with same size



Next level image

Next level image with same size

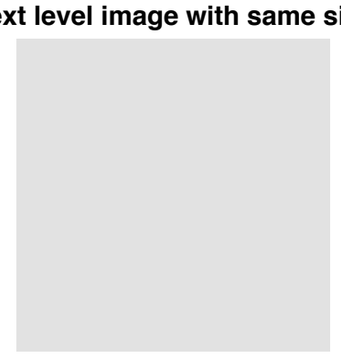


Next level image

Next level image with same size

Next level image

Next level image with same size



Next level image

Part2: Laplacian Pyramid

To calculate Laplacian Pyramid, we just need to do the subtraction between each two consecutive levels from Gaussian Pyramid. Thus I store the Gaussian Pyramid each level's image into a cell:

```
gaussian = cell(j, 1);
```

j is the total level number of the Gaussian pyramid.

Then I just need to do a loop to calculate the subtractions and store the results into another cell with the same level size (the top level of Laplacian pyramid is the same as the top level of Gaussian pyramid). Here is the main process of the generation of Laplacian pyramid:

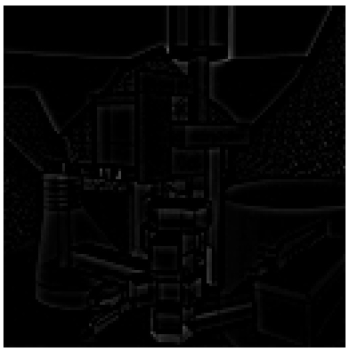
```
lap = cell(j, 1);
```

```
lap{j} = gaussian{j};
for n = 1 : j − 1
  lap{n} = gaussian{n} − gaussian{n + 1};
end
for n = 1 : j
   figure, imshow(lap{n});
   title('Laplacian pyramid')
end
```

Here is the Laplacian pyramid of each test image:

Polarcities:

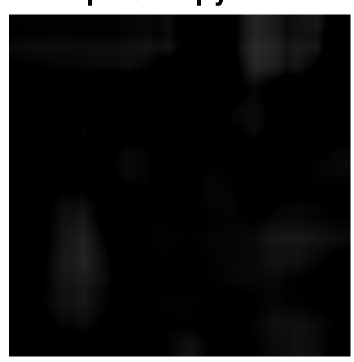
Laplacian pyramid



Laplacian pyramid

Laplacian pyramid

Laplacian pyramid

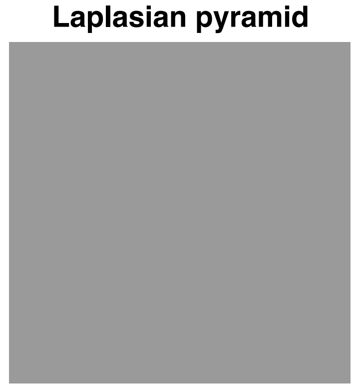


Laplacian pyramid

Laplacian pyramid

Laplacian pyramid

Laplacian pyramid



Text:

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

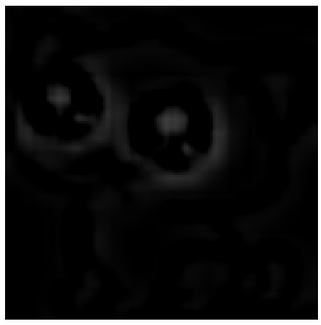
Laplasian pyramid

kitty:

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid



Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

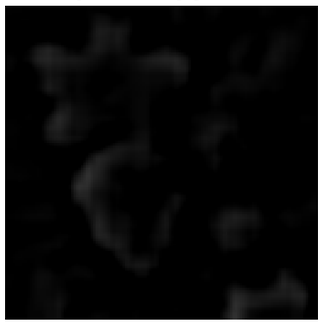
Laplasian pyramid

Flowergray:

Laplasian pyramid

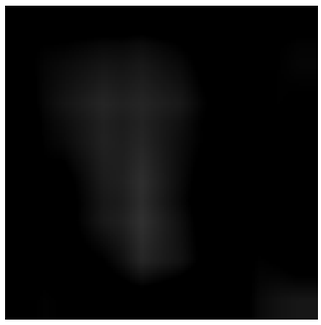
Laplasian pyramid

Laplasian pyramid

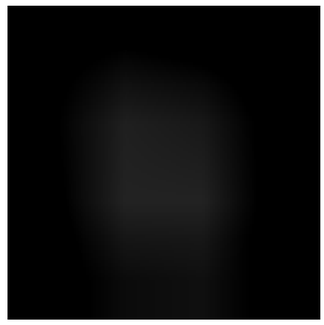


Laplasian pyramid

Laplasian pyramid



Laplasian pyramid



Laplasian pyramid

Laplasian pyramid

Cartoon:

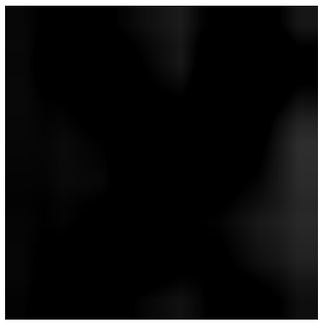
Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

Laplasian pyramid



Laplasian pyramid

Laplasian pyramid

Laplasian pyramid

Part3 Multi-Scale Edge Detection

In this part, I follow the steps in the description to finish:

Step1: Apply the Laplacian operator to every level of the Gaussian pyramid that generated in part1. So I use the conv2() function and the operator to calculate in a loop for each level of Gaussian pyramid.

Step2: change the output image from step1 to binary image use imbinarize(), the threshold is 0.

Step3: use edge() function and ‘zerocross’ method to detect zero crossing from the image of step2.

Step4: calculate the local variance by using stdfilt() function from step1 and set a threshold. Then do a AND between the output of step3 and the previous part of this step. The final output should be the satisfied result. Here are the main code realizing this process.

```
h = [[−1/8 −1/8 −1/8]; [−1/8 1 −1/8]; [−1/8 −1/8 −1/8]];
while i < j + 1
  lapop = conv2(gaussian{i}, h, 'same');
  bichange = imbinarize(lapop, 0);
  findedge = edge(bichange, 'zerocross');
  lap{i} = lapop;
result{i} = findedge;
  i = i + 1;
end
final = cell(j, 1);
for m = 1: 1: j
  variance = stdfilt(lap{m});
  varianceth = (variance > 0.001);
  final{m} = result{m}&varianceth;
end
```

Here are the results:

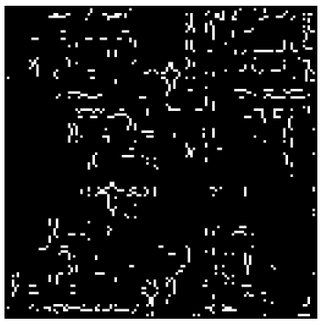
Cartoon: threshold = 0.001

Final result for the edge detection

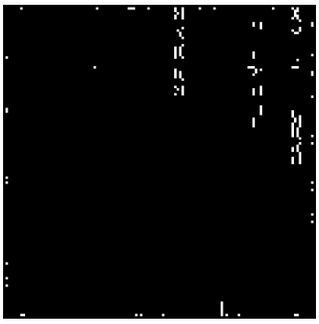
Final result for the edge detection

Final result for the edge detection

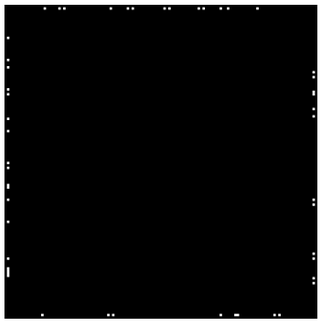
Final result for the edge detection



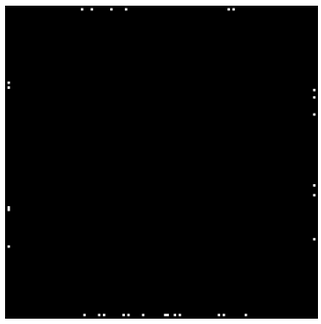
Final result for the edge detection



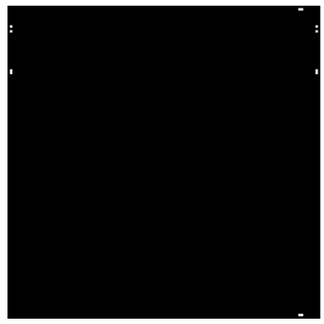
Final result for the edge detection



Final result for the edge detection



Final result for the edge detection

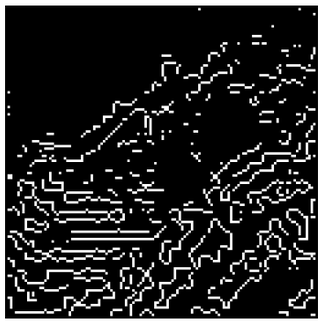


Text：threshold = 0.001

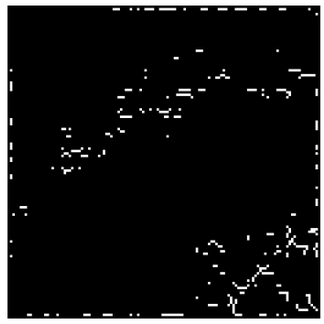
Final result for the edge detection

Final result for the edge detection

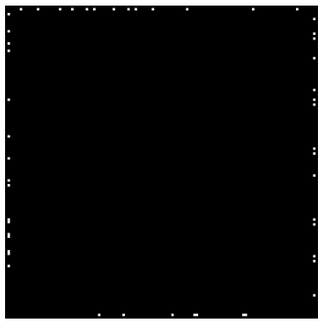
Final result for the edge detection



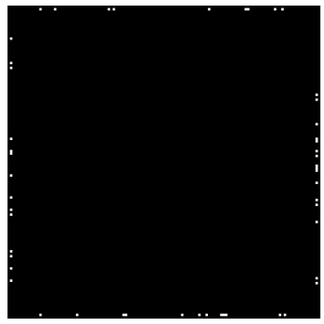
Final result for the edge detection



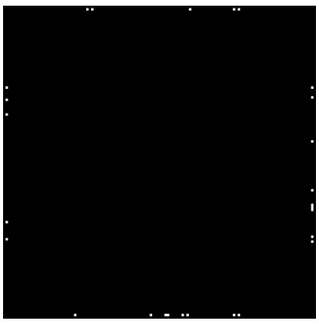
Final result for the edge detection



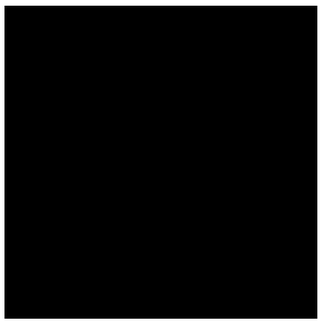
Final result for the edge detection



Final result for the edge detection



Final result for the edge detection

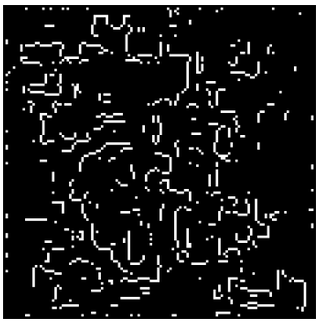


Flowergray: threshold = 0.001

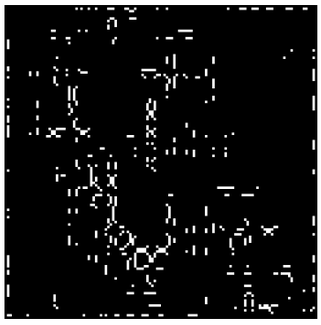
Final result for the edge detection

Final result for the edge detection

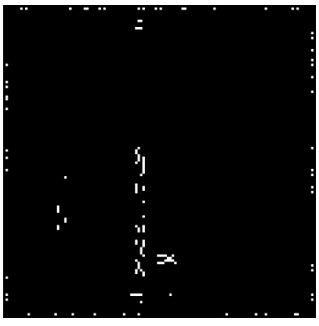
Final result for the edge detection



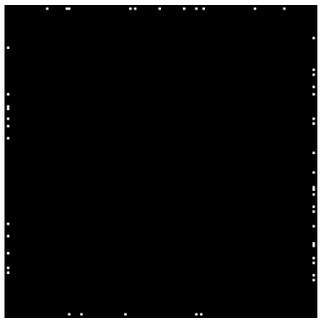
Final result for the edge detection



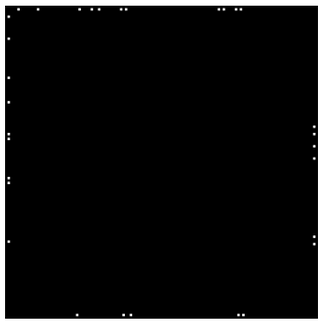
Final result for the edge detection



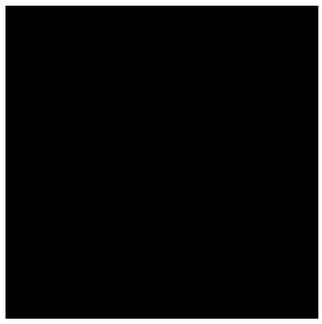
Final result for the edge detection



Final result for the edge detection



Final result for the edge detection



kitty: threshold = 0.001

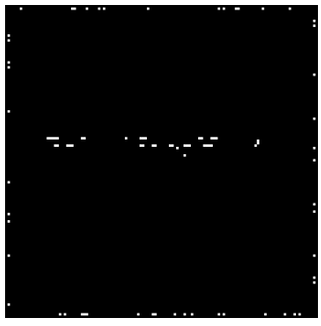
Final result for the edge detection

Final result for the edge detection

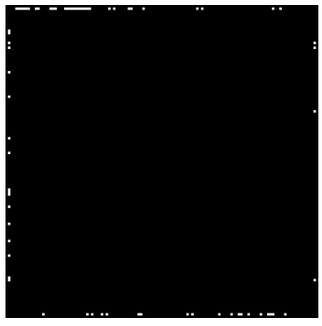
Final result for the edge detection

Final result for the edge detection

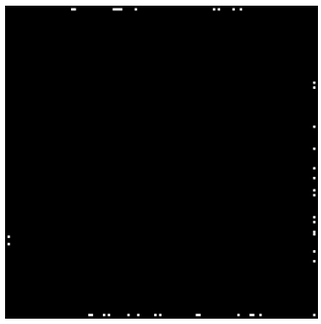
Final result for the edge detection



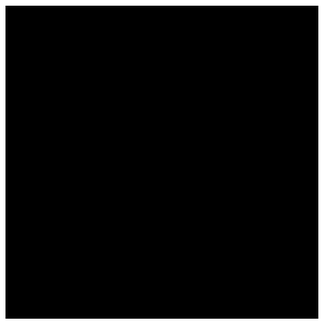
Final result for the edge detection



Final result for the edge detection

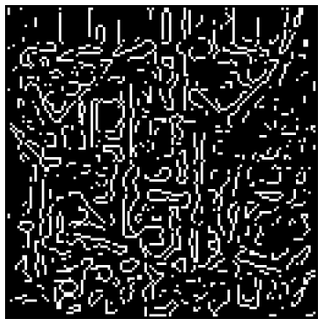


Final result for the edge detection

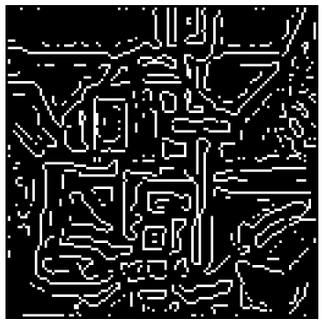


Polarcities: threshold = 0.001

Final result for the edge detection



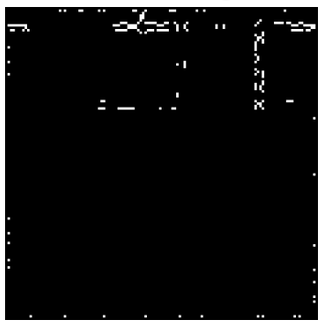
Final result for the edge detection



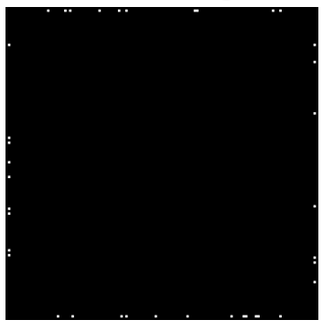
Final result for the edge detection

Final result for the edge detection

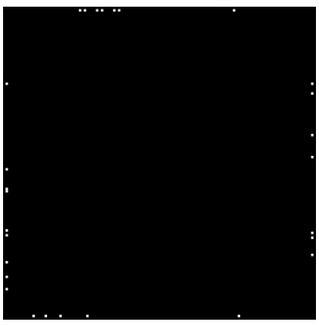
Final result for the edge detection



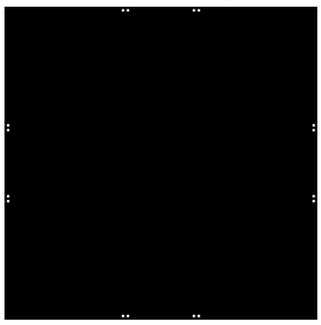
Final result for the edge detection



Final result for the edge detection



Final result for the edge detection



Part4: Multi-resolution Spline

Step1: the process to create the Laplacian pyramid is mentioned in part2.

Step2: creating a binary mask

```
bimask = [[ones(max1/2) zeros(max1/2)]; [zeros(max1/2) ones(max1/2)]];
```

Step3: Gaussian pyraimd is also mentioned before.

Step4: to calculate the new Laplacian pyramid, I have to do in loop. And in each cycle, calculate pixel by pixel from the laplacian pyramid of two images and gaussian pyramid of the binary mask. Then I have to add all these levels in the new pyramid up to generate the final result.

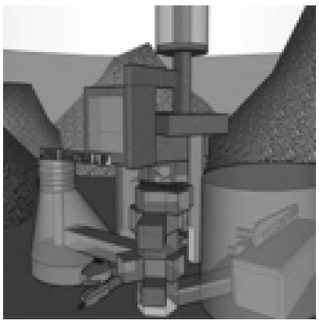
Here is the main code:

```
spline = cell(j1,1);
for i = 1:1:j1
  pre = zeros(max1);
  lap1pre = lap1{i};
  lap2pre = lap2{i};
  gaussian3pre = gaussian3{i};
  for row = 1:1:max1
    for column = 1:1:max1
      pre(row,column)
                = lap1pre(row,column)*gaussian3pre(row,column) + lap2pre(row,column)*(1
                - gaussian3pre(row,column));
    end
  end
  spline{i} = pre;
end
result = 0;
for t = 1:1:j1
  result = result + spline{t};
end
```

Here are the results of three pairs of images with different binary mask.

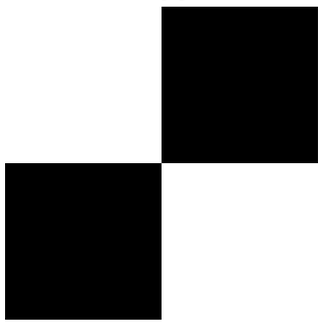
Polarcities and Kitty:

Fisrt resized image

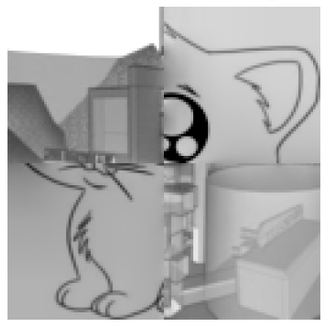


Second resized image

Bimask pyramid



Result of blending

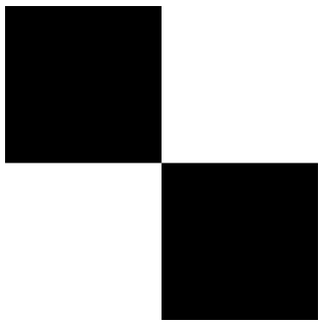


Cartoon and flowergray:

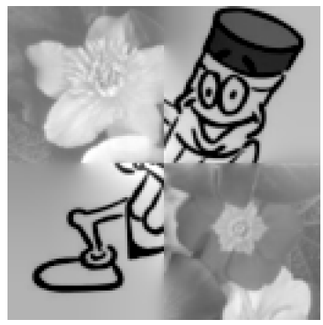
Fisrt resized image

Second resized image

Bimask pyramid



Result of blending



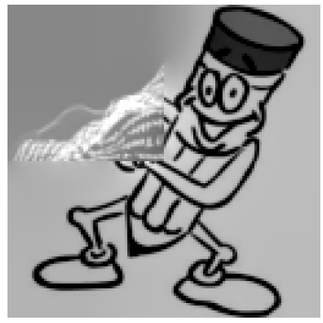
Cartoon and Text:

Fisrt resized image

Second resized image

Bimask pyramid

Result of blending



Referenc:

"Mathworks - Support". *Mathworks.com*. N.p., 2016. Web. 27 Sept. 2016.