CSC420 A1

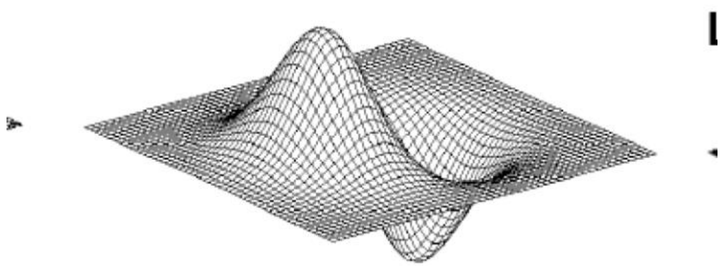
1.
   a. If h is not separable the computational cost will be h^2.
   b. If h is separable the computational cost will be 2h.
2. Steps of canny edge detection:
   a. Apply derivative gaussian filter to original image to smooth the image and reduce noise
   b. Find magnitude and orientation of gradient, a higher gradient meaning the pixel value is changing more rapidly, which is more likely an edge, and orientation presents the orientation of the edge.
   c. Apply non-maxima suppression, only take the pixel that's local maximum along the gradient direction, in this way we can reduce the thickness of an edge
   d. Hysteresis thresholding, when starting a curve we use a high threshold, therefore we can eliminate some really thin edges, but when continuing the edge we use a low threshold and thus we don't lose an edge because it's thin

   To implement:

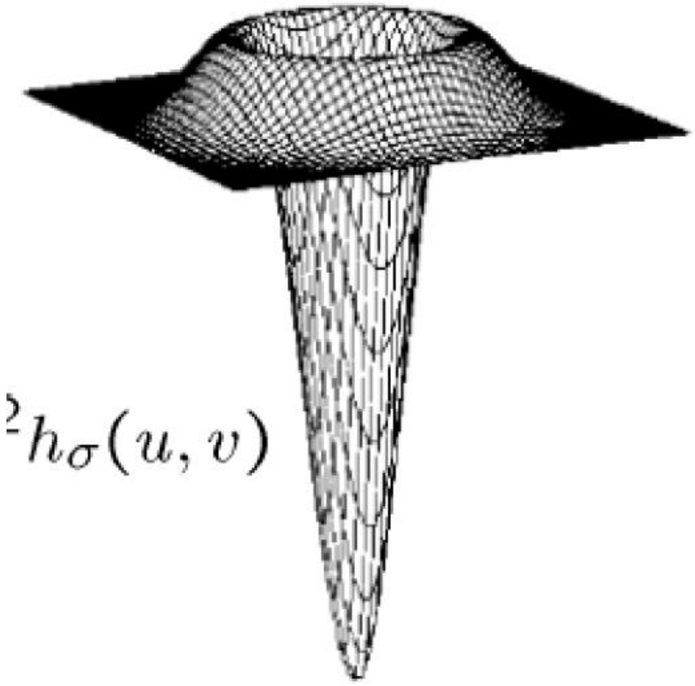
   i. For each pixel in image I, collects neighbors, apply a derivative gaussian filter, then form a new value with the dot product of its neighbors and the filter
   ii. Using a $\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$ and $\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$ we can find horizontal and vertical gradient.
   iii. For non-maxima suppression we can calculate the second order derivative of gradient, and if it's local maximum we keep it, else set it to 0 (black)
   iv. Start the edge only if it's gradient value is greater than the high threshold, if the edge only survives the low threshold, keep it only when an existing edge is connected with it

3. Since we know LoG: $\nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

   And the first order derivative of gaussian looks like this



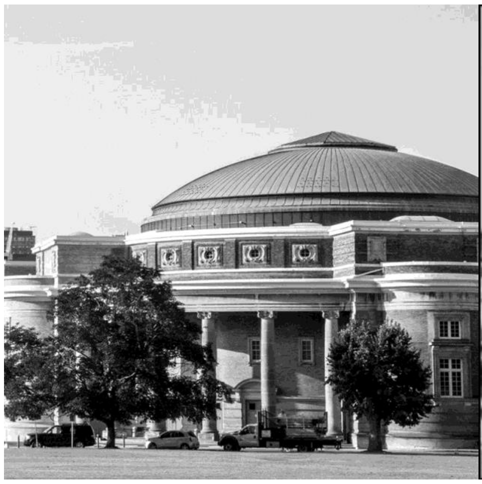
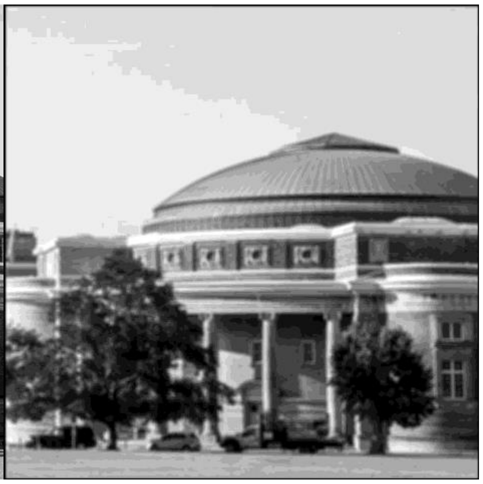
Thus Laplacian of Gaussian is just the second order derivative looks like following, it values the pixel at center much higher than surrounding pixels, thus if there is a rapid change on a pixel this will return a very negative number, in another word, LoG is very sensitive of pixel rapid change, and thus can be used for edge detection.



4.

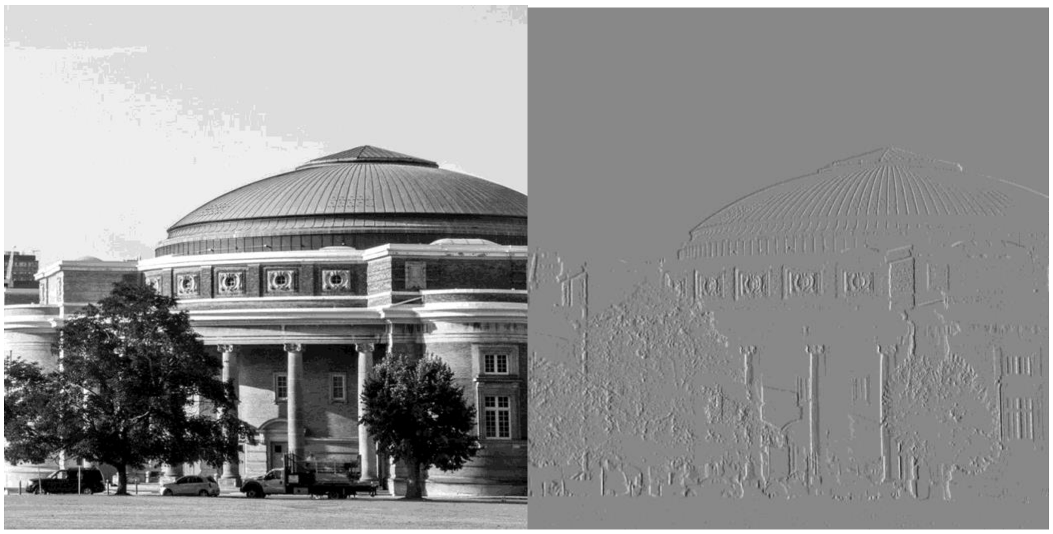
a. *MyCorrelation*, Using the gray.jpg image, with filter 7 x 7 $\begin{bmatrix}1 & \cdots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 1\end{bmatrix}$

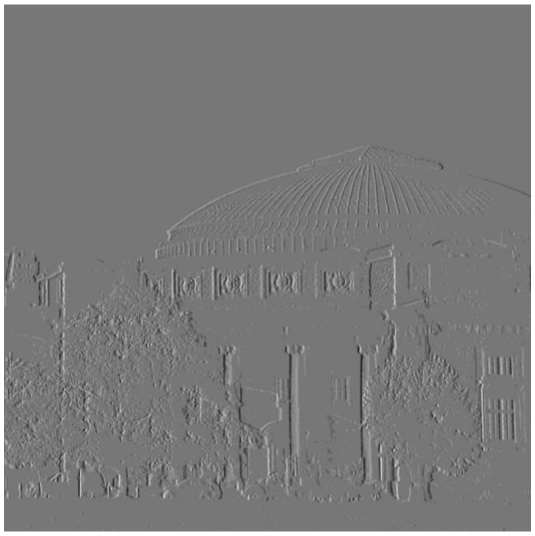
Original image

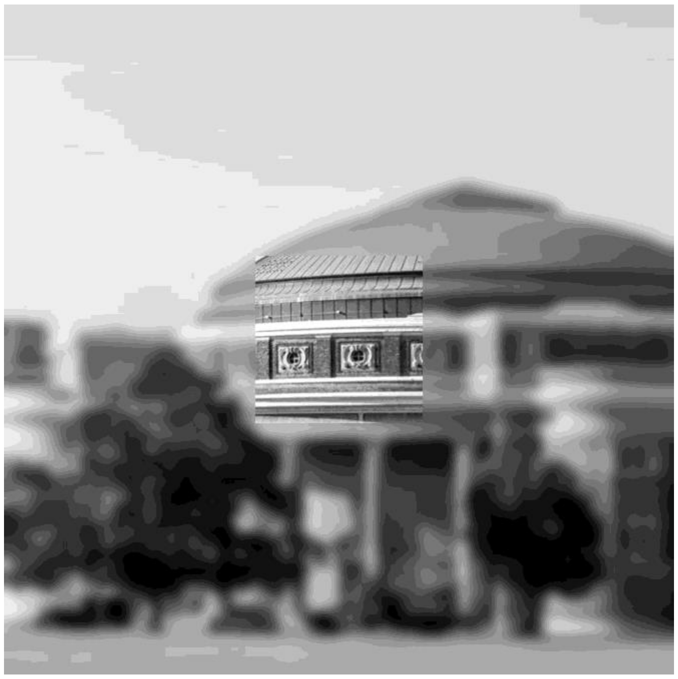
Upper left: original
Upper right: same
Bottom left: full
Bottom right: valid

b. *MyConvolution*, using the same gary.jpg, using filter h: $\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$ to indicate its difference with correlation




Upper left: original
Upper right: convolution using filter
Bottom left: correlation using filter

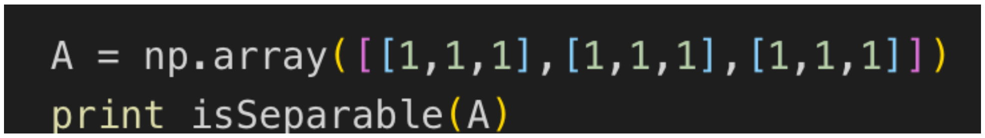
c. Portrait mode, using a gaussian filter, with sigma = 7. The mask is manually segmented 200 * 200 matrix. The reason choosing gaussian filter is because box filter will reduce image detail where gaussian would not.



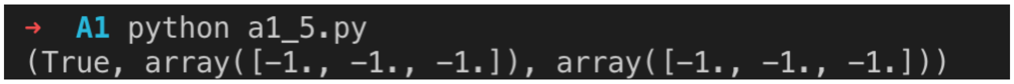
5.

a. A separable m x m matrix is a matrix which can be represented by a dot product of two one dimensional 1 x m matrices.

b. First example, matrix is $\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$ and is separable, the function returned true and horizontal and vertical filters both -1 -1 -1.

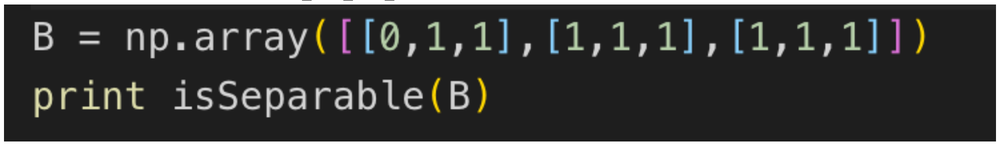


```
A = np.array([[1,1,1],[1,1,1],[1,1,1]])
print isSeparable(A)
```

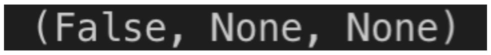


```
➜  A1 python a1_5.py
(True, array([-1., -1., -1.]), array([-1., -1., -1.]))
```

Second example, matrix is $\begin{matrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$, and it's not separable, the function returned false



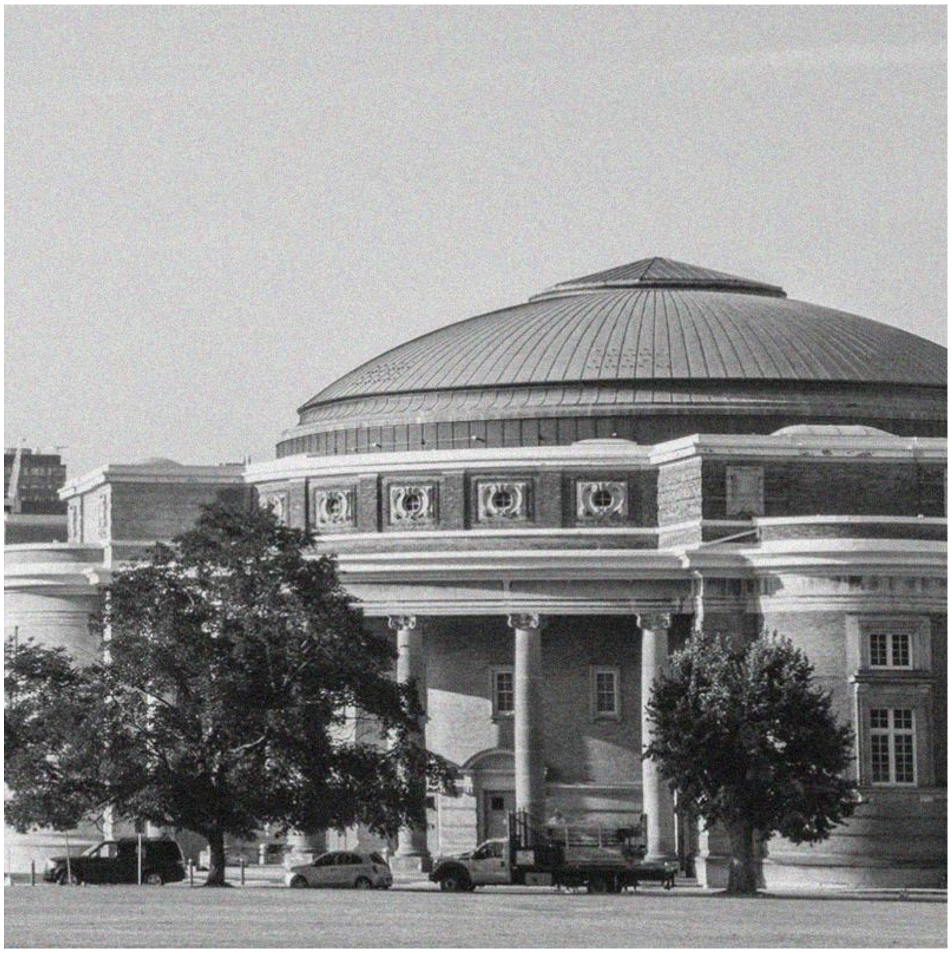
```
B = np.array([[0,1,1],[1,1,1],[1,1,1]])
print isSeparable(B)
```



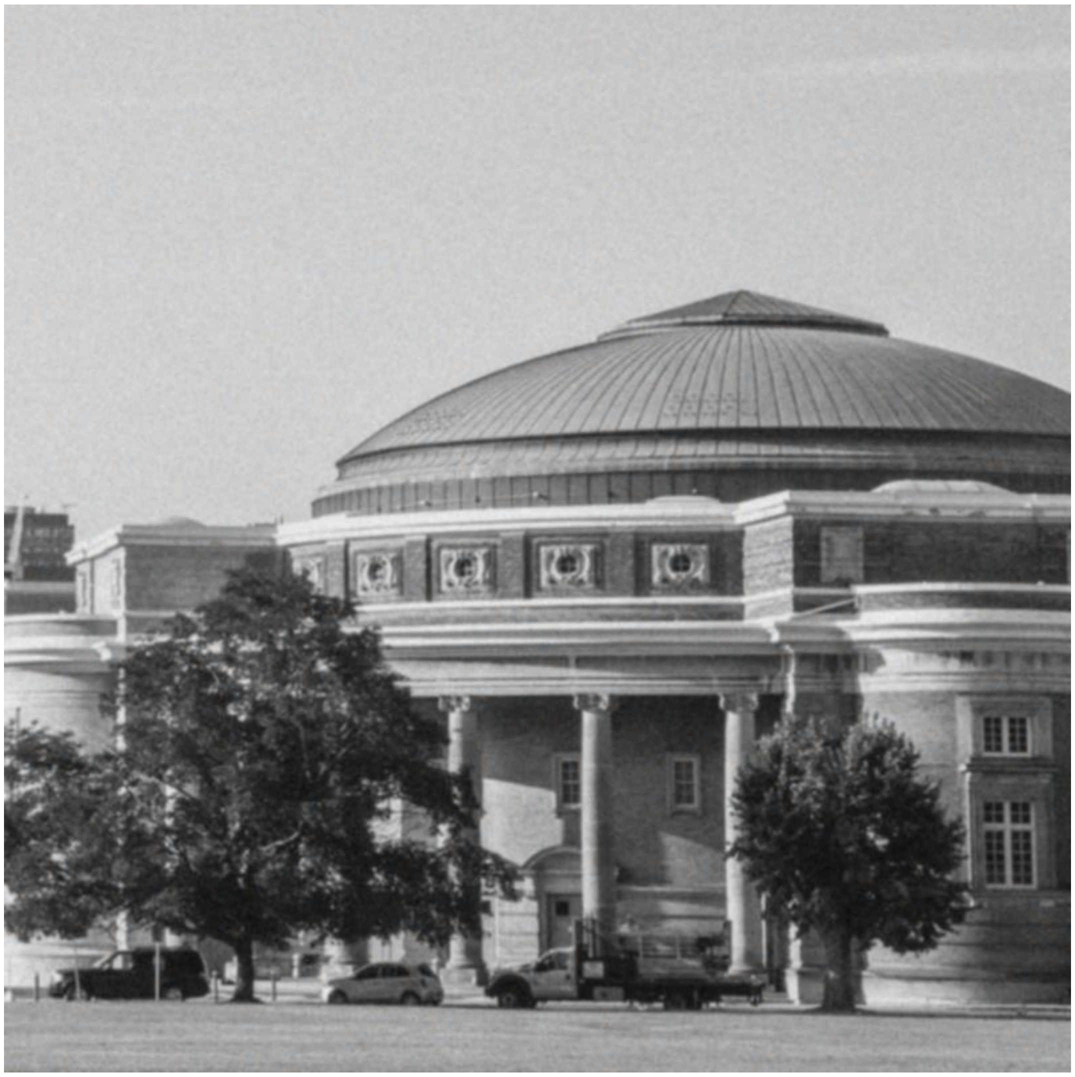
```
(False, None, None)
```

6.

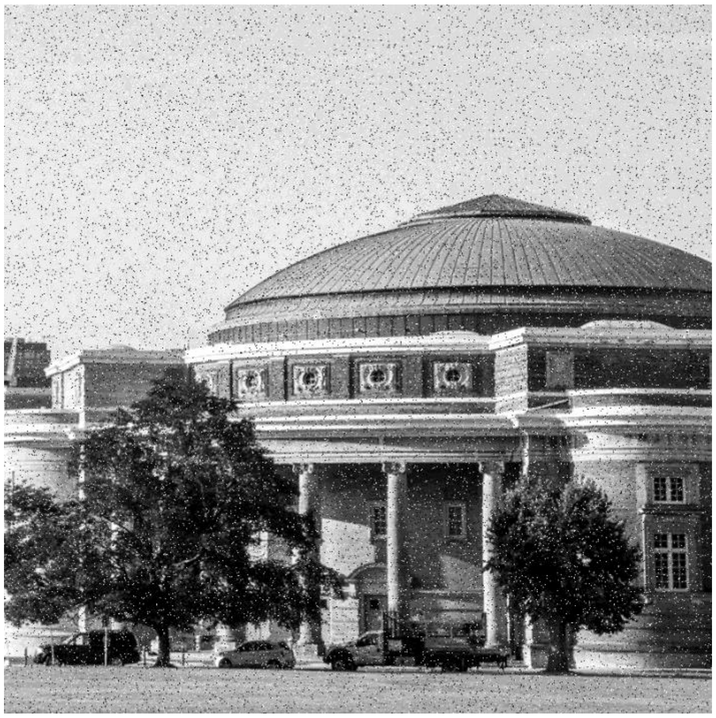
a. Random noised added to original image



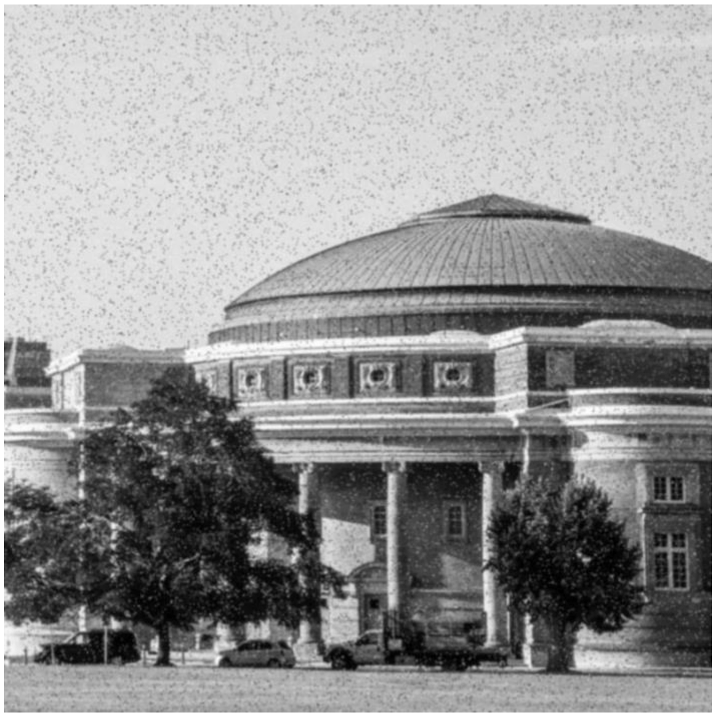
b. Denoise with a gaussian filter, since applying the gaussian filter can reduce the image's high frequency components, which is good for denoising.



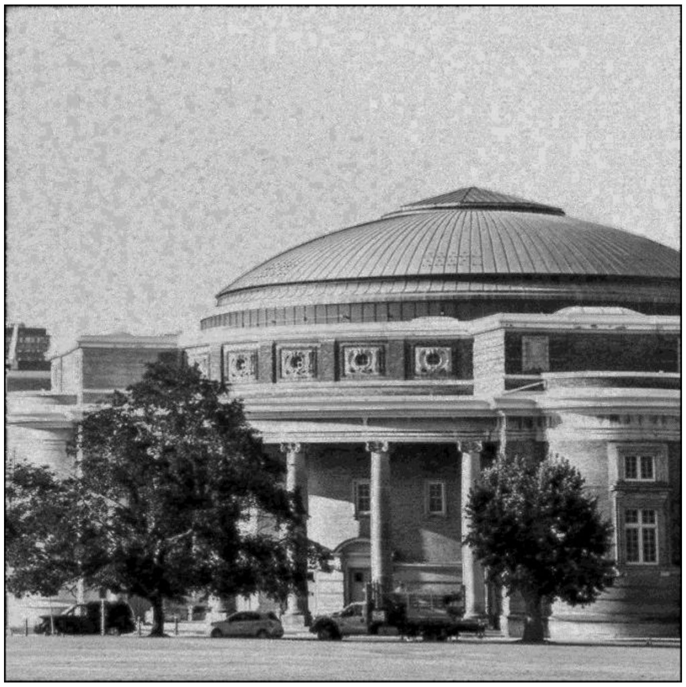
c. Salt-and-pepper noise



d. Try to remove salt-and-pepper noise with Gaussian filter picked in (b) doesn’t work



Instead I used a way to detect all black/white noise points in image and filter them with a 5 * 5 matrix with center to be 0.



And the result if better than simply applying Gaussian filter.

e.