**Your Name:** Yen-Ting Liu

**Your NetId:** ytliu2

# Part 1 Hybrid Images:

You will provide the following for ***3 different examples***(1 provided pair, 2 pairs of your own):

* two input images
* two *filtered* input images
* two generated hybrid image (at different resolutions, similar to images C and D above)
* two σ values (one for each filter)

You will provide the following as further discussion overall:

* Explanation of how you chose the σ values
* Discussion of how successful your examples are + any interesting observations

## Example 1:

|  |  |  |
| --- | --- | --- |
| **Input (Hi-Freq)** | | **Hybrid** |
|  |  |  |
| **Input (Lo-Freq)** | | **1/5 Hybrid** |
|  |  |  |

## Example 2:

|  |  |
| --- | --- |
| **Input (Hi-Freq)** | **Hybrid** |
|  |  |
| **Input (Lo-Freq)** | **1/3 Hybrid** |
|  |  |

## Example 3:

|  |  |  |
| --- | --- | --- |
| **Input (Hi-Freq)** | | **Hybrid** |
|  |  |  |
| **Input (Lo-Freq)** | | **1/3 Hybrid** |
|  |  |  |

## Discussion:

TODO

# Part 2 Scale-Space Blob Detection:

You will provide the following for ***8 different examples***(4 provided, 4 of your own):

* original image
* output of your circle detector on the image
* running time for the "efficient" implementation on this image
* running time for the "inefficient" implementation on this image

You will provide the following as further discussion overall:

* Explanation of any "interesting" implementation choices that you made.
* Discussion of optimal parameter values or ones you have tried

In following examples, blob detections all use

* Minimum sigma 2
* Maximum sigma 50
* 32 pyramid levels
* Intensity threshold 0.5
* Execute 10 times and take the runtime average of function blob\_detection

The inefficient implementation is named as *sigma*, the efficient one is named as *resize*. Input image resolution is listed below the header of each example.

Parameters for these examples are not tuned, in order to have a general idea how the detector perform across different input. Tuned results are provided in the discussion.

## Example 1:

|  |  |  |
| --- | --- | --- |
| **Original**  493 x 356 | **Circles (sigma)**  2.33 s | **Circles (resize)**  0.49 s |
|  |  |  |

## 

## Example 2:

|  |  |  |
| --- | --- | --- |
| **Original**  640 x 480 | **Circles (sigma)**  4.27 s | **Circles (resize)**  1.03 s |
|  |  |  |

## Example 3:

|  |  |  |
| --- | --- | --- |
| **Original**  500 x 335 | **Circles (sigma)**  2.24 s | **Circles (resize)**  0.50 s |
|  |  |  |

## Example 4:

|  |  |  |
| --- | --- | --- |
| **Original**  328 x 357 | **Circles (sigma)**  1.59 s | **Circles (resize)**  0.35 s |
|  |  |  |

## 

## Example 5:

|  |  |  |
| --- | --- | --- |
| **Original**  384 x 303 | **Circles (sigma)**  1.60 s | **Circles (resize)**  0.37 s |
|  |  |  |

*“British Museum, London, England”. Brooklyn Museum, Goodyear [from Pompeii].*

## Example 6:

|  |  |  |
| --- | --- | --- |
| **Original**  741 x 500 | **Circles (sigma)**  5.12 s | **Circles (resize)**  0.98 s |
|  |  |  |

*Stereo motorcycle. Middlebury 2014 stereo benchmark.*

## Example 7:

|  |  |  |
| --- | --- | --- |
| **Original**  512 x 512 | **Circles (sigma)**  3.72 s | **Circles (resize)**  0.80 s |
|  |  |  |

*Photograph of Eileen Collins. From the NASA Great Images database.*

## Example 8:

|  |  |  |
| --- | --- | --- |
| **Original**  600 x 400 | **Circles (sigma)**  3.24 s | **Circles (resize)**  0.66 s |
|  |  |  |

*Pikolo Espresso Bar. CC0 by photographer Rachel Michetti.*

## Discussion:

* Discussion of optimal parameter values or ones you have tried

**Implementation choices**

* To maximize the benefit from downscaling the image, I repeatedly resize instead of starting from the original size, see the following snippet (shape is the next-smaller shape in each layer)

shape0 = im.shape

for \_ in range(max\_layer):

im = resize(im, shape0)

yield im

…

im = **resize**(**im**, shape)

This may cause sampling error to accumulate with each layer.

* When downsizing the image, the image is not isotropically downsized,

**np.ceil**(np.array(im.shape) / downscale)

This causes the effective downscale to change slightly across each layer. Therefore, effective sigma is slightly different.

* Since the requirement for kernel is repeatedly mentioned in Piazza, to avoid penalty, I explicitly uses the truncate parameter

gaussian\_laplace(…, **truncate=6**)

to make sure the kernel is indeed cutoff at .

* I output each Laplacian pyramid layer with sigma multiplied,

yield im **\* sigma\*\*2**

so each scale is directly comparable during blob detection.

* This won’t directly impact time complexity, but consider that we only work with each layer independently, I utilize yield keyword

for \_ in range(max\_layer):

…

**yield** im \* sigma\*\*2, sigma

to have some flexibility at optimizing memory usage downstream.

* From Piazza discussion thread @135, it is my understanding that reference implementation prefers create scale threshold (scale\_max) by non-maximal suppressed value (pyramid). Therefore,

pyramid = rank\_filter(

pyramid0,

-1,

footprint=np.ones((3, ) \* (pyramid0.ndim-1) + (1, )

)

scale\_max = np.max(**pyramid**, axis=-1)

is later compare with the original pyramid value (pyramid0),

for I in range(n\_sigma):

pyramid[…, i] = (**pyramid0[…, i] == scale\_max**) \* im

* *(Cont.)* It is my understanding that reference design uses original pixel intensity (im) as the threshold to whether this circle is kept,

pyramid[…, i] = (pypramid0[…, i] == scale\_max) **\* im**

…

maxima = np.where(pyramid **> threshold**)

aka, we only output detected circles for (sufficiently) bright features.

**Optimal parameters**

Since we know intensity threshold simply pick out circles correspond to bright objects, we can fix this value and only fine tune parameters related to sigma. Without additional filtering strategy to remove spurious edge responses, we simply adjust the minimum sigma,

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

From here, we can see that choosing a sigma range close to feature we want is important in this naïve implementation.

# Bonus:

## Hybrid Images Extra Credit

* Discussion and results of any extensions or bonus features you have implemented for Hybrid Images

## Blob-Detection Extra Credit

* Discussion and results of any extensions or bonus features you have implemented for Blob-Detection