1 How to run

My local environment is VS Code+Anaconda+Win10.
The code is in the dir ***HW2-Texture_Synthesis_and_Image_Inpainting***
Please use VS Code to open the root dir and choose ***main.py*** in each part and run, otherwise there may be some error when read the image files because of the ***relative path*** of the images.
Part 1 Texture Synthesis is in dir ***TextureSynthesis***.
Part 2 Image Inpainting is in dir ***ImageInpainting***.
Part 3 Region Filling is in dir ***CriminisiAlG***.
Part 4 Image Quilting is in dir ***ImageQuilting***.
And the **.txt** files attached with this write-up contain part of the running outcome and records.

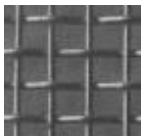
2 Test and results

2.1 Part 1: Texture Synthesis

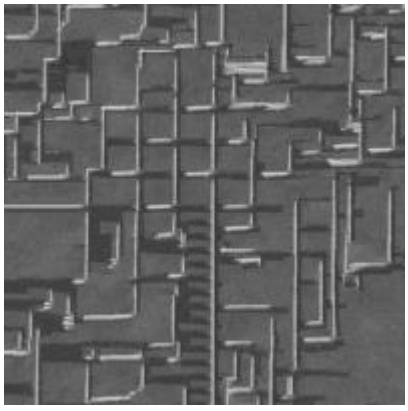
At beginning, afraid of potentially long time to get the results, I just used windows sizes 5, 9, 11, 15 and 17 in this part. We can see that with the growth of window size, the quality of output images become better. Just denote window size as **WS.** And we can see that with the growth of window size, the running time is increasing.
And the gaussian filter is modified from this source:
http://stackoverflow.com/questions/17190649/how-to-obtain-a-gaussian-filter-in-python

T1.gif

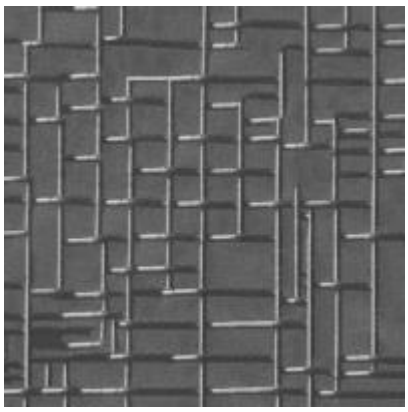
original



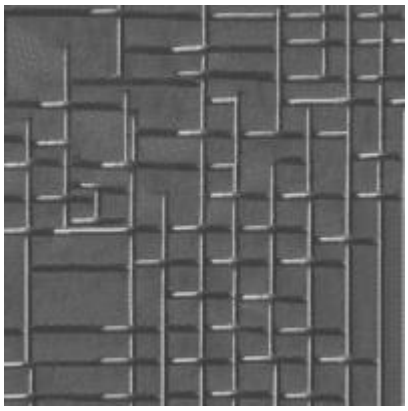
WS=5



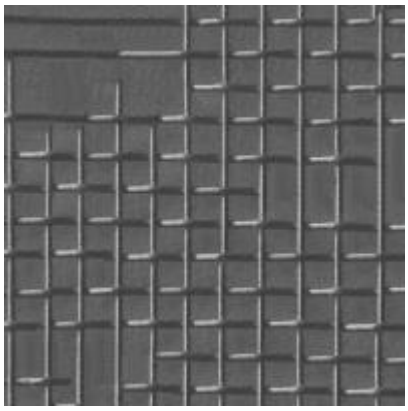
WS=9



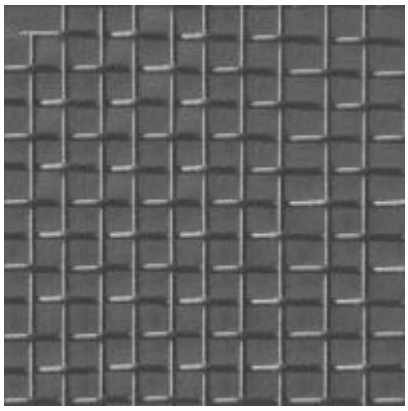
WS=11



WS=15

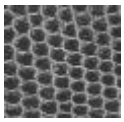


WS=17

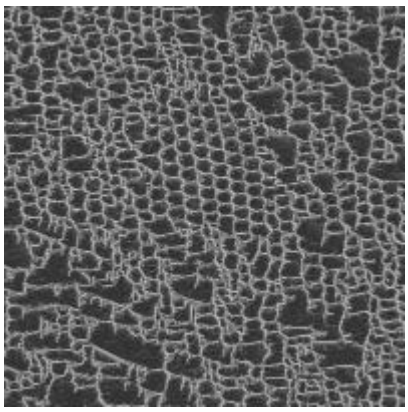


T2.gif

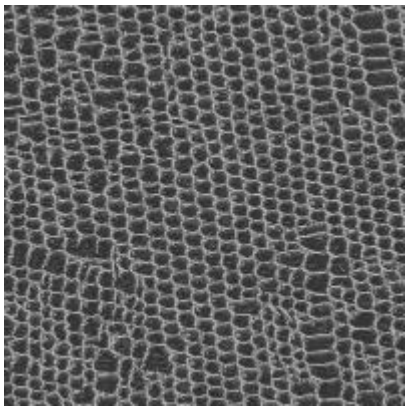
original



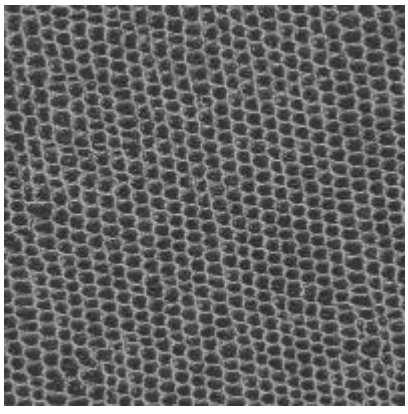
WS=5



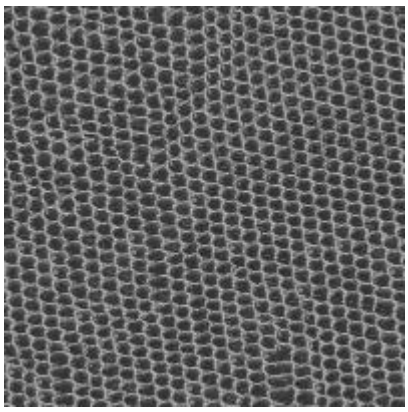
WS=9



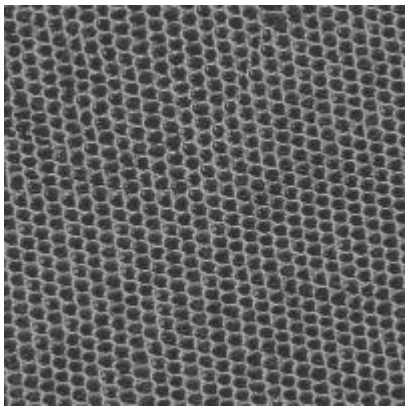
WS=11



WS=15



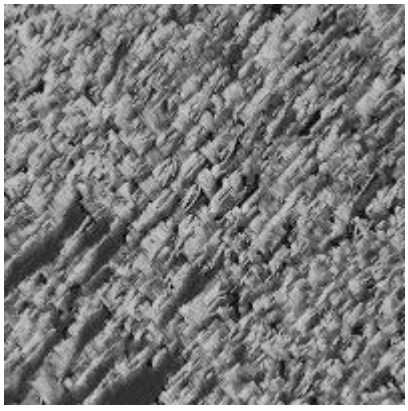
WS=17



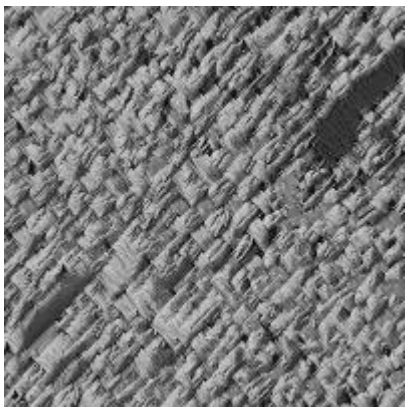
T3.gif

original

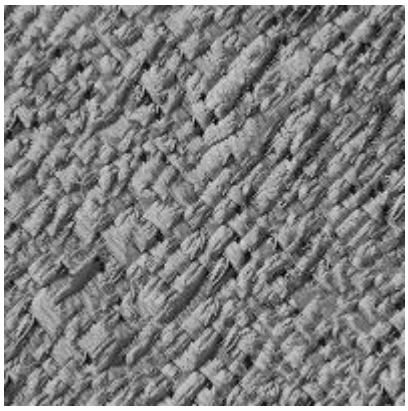
WS=5



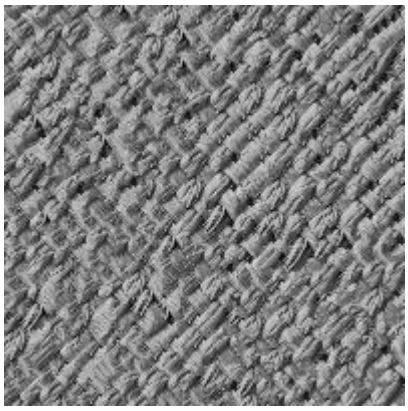
WS=9



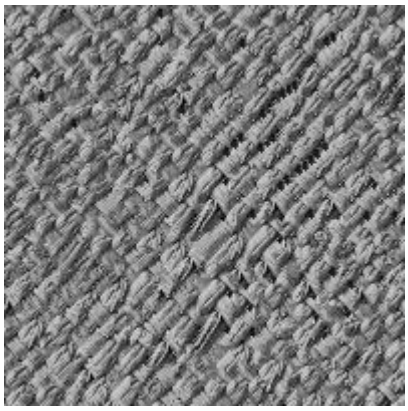
WS=11



WS=15

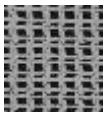


WS=17

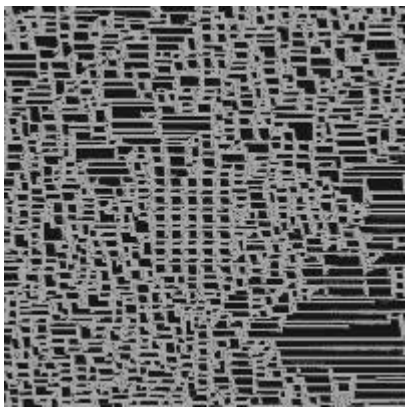


T4.gif

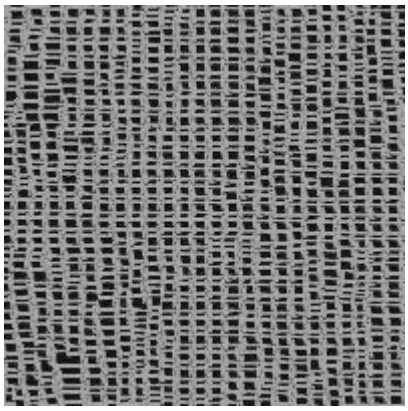
original



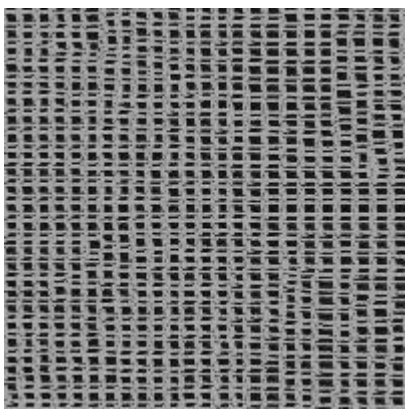
WS=5



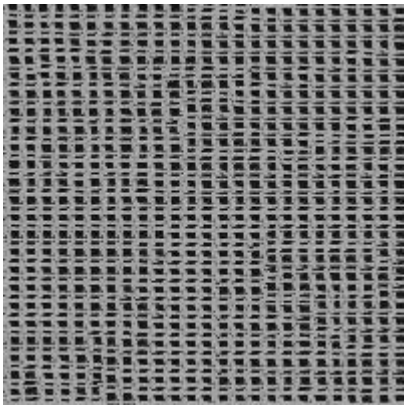
WS=9



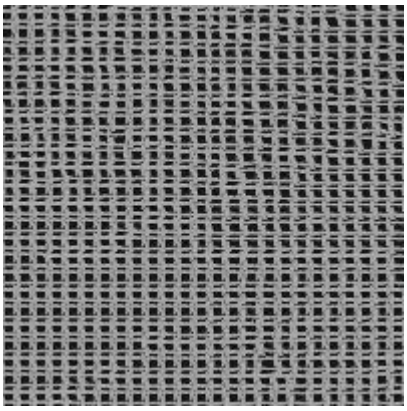
WS=11



WS=15

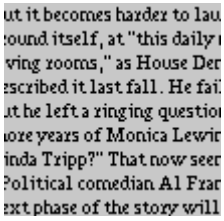


WS=17

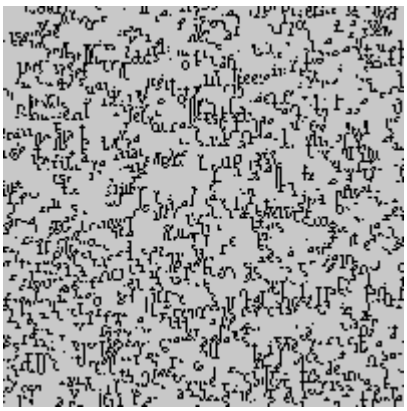


T5.gif

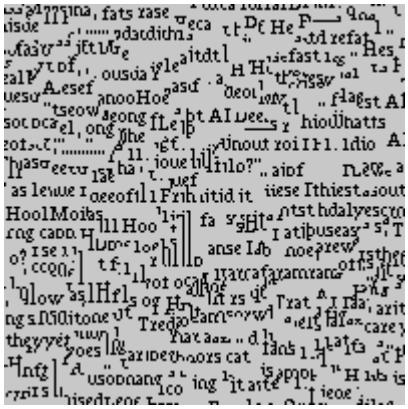
original



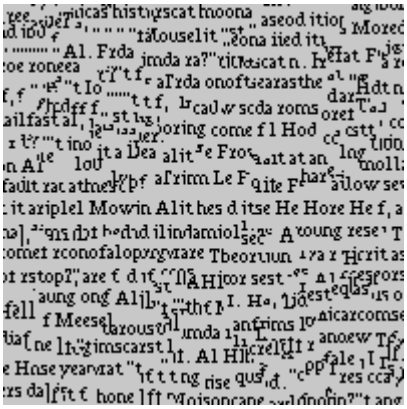
WS=5



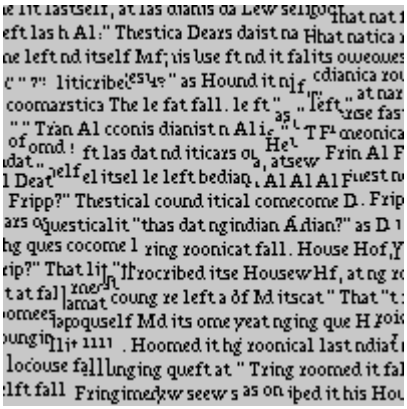
WS=9



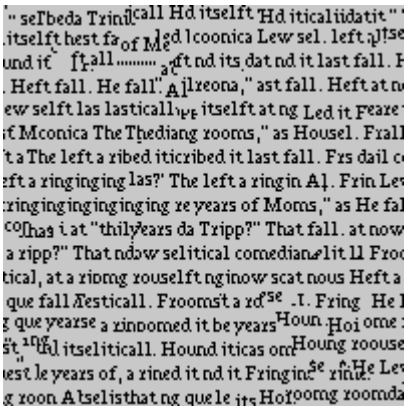
WS=11



WS=15



WS=17



2.2 Part 2: Image Inpainting

This part is the similar implementation as part 1. But it is a huge disaster that I decided to run multiple window size as I have done in the part 1. For images having 76k pixels, it costs huge amount of time to get so much output images (It is really a nightmare and I don't want to mention the exact running time).
We can see almost the same conclusion as in part 1.

test_im1.bmp

original

WS=5

WS=9

WS=11

WS=15

WS=17

test_im2.bmp

original

WS=5

WS=9

WS=11

WS=15

WS=17

2.3 Part 3: Region Filling

In this part, I just test the window size=11 for both algorithms.
For this part, the input and result of the part2-algorithm we use is:

Input:

1. test_im3_1.bmp

2. test_im3_2.bmp

3. test_im3_3.bmp

The first one needs too much time, so I am not able to make it out. The results are the second and the third.

Result

2. test_3_2.bmp

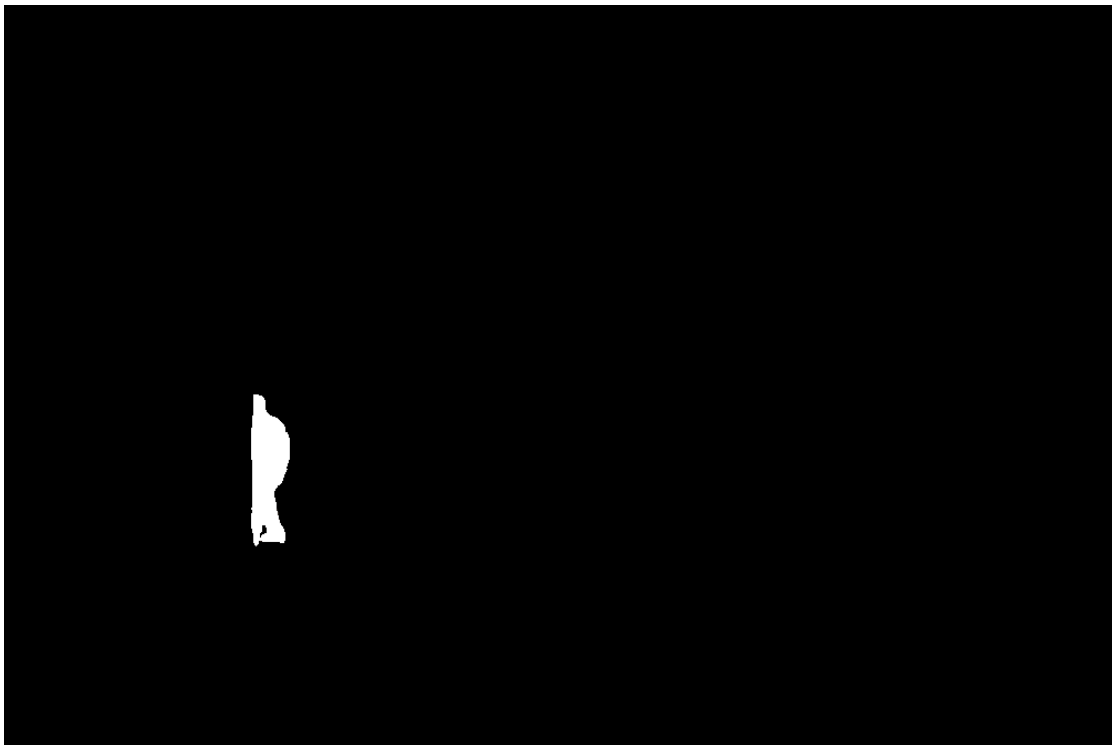
3. test_3_3.bmp

The Criminisi's Algorithm

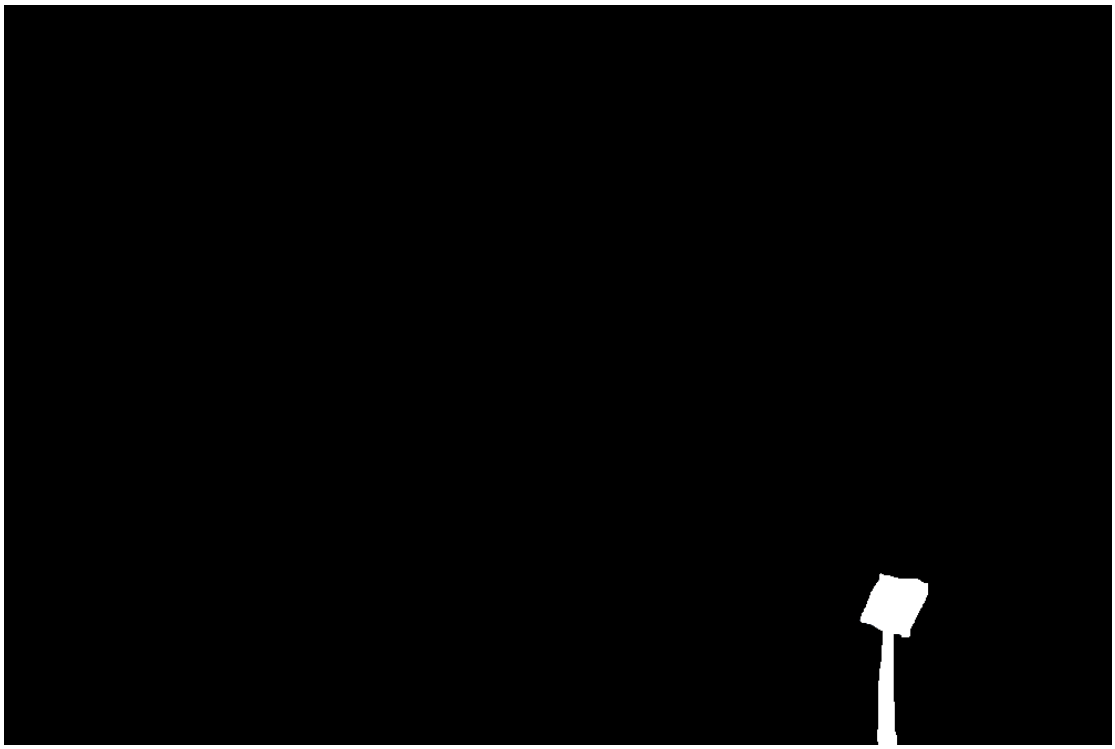
I asked the Professor how to get the mask by e-mail and he told me an matlab approach, so I just use it rather than write the code by myself.
And I transfer the jpg image to the bmp image.
The mask we use is (I used the method Professor told me in the email):

mask1.bmp

mask2.bmp



mask3.bmp



Input

The results:

result1.bmp

result2.bmp

result3.bmp

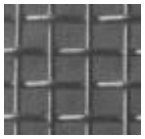
Criminisi's approach carefully chooses the order in which pixels are synthesized. So we can easily know that it gives better results with object removal as we can see by comparing results of two algorithms. It performs better than Efros and Leung's method as it preserves the dominant edges in the image. And it is faster because of the path based approach it's faster and does not introduce blur like Efros algorithm.

2.4 Part 4: Image Quilting

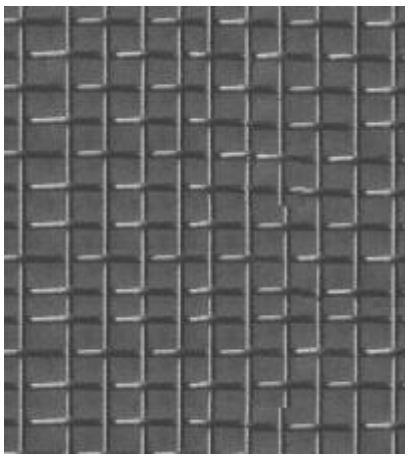
I downloaded the code from https://github.com/afrozalm/Patch-Based-Texture-Synthesis and I modified it slightly in order to fit my requirement.
I just use the Patch size 25 (similar to window size 5) to get the results.

T1.gif

original

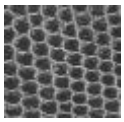


outcome

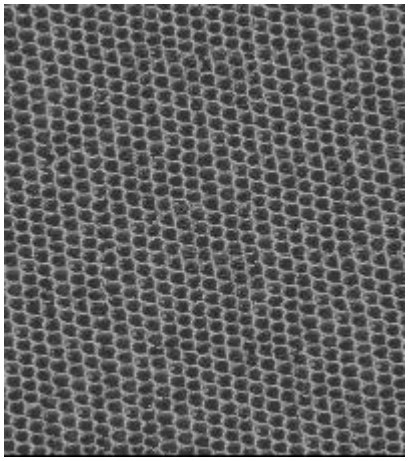


T2.gif

original



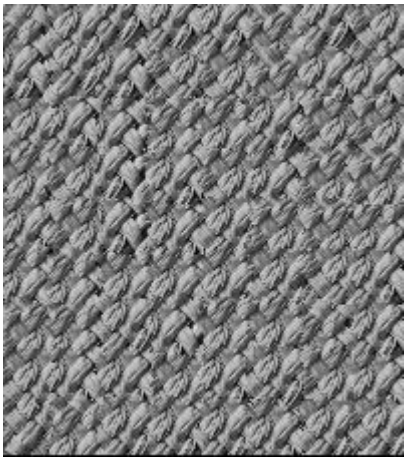
outcome



T3.gif

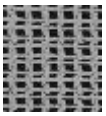
original

outcome

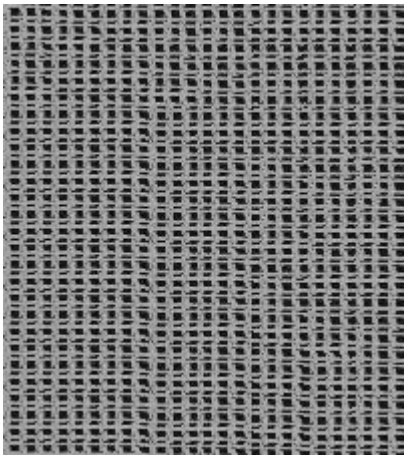


T4.gif

original

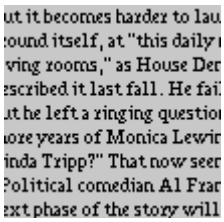


outcome

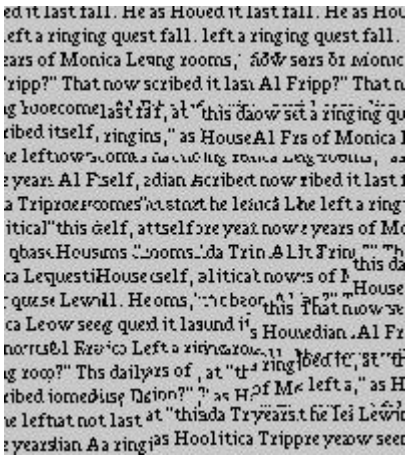


T5.gif

original



outcome



It is a disaster and it is obvious that texture synthesis using Efros and Leung's method would be very very very slow. The reason is it constructs texture pixel by pixel. It would be better if the unit of construction would be a patch rather than a single pixel.

Efros and Freeman's approach (Part 4) does the same. Picking up patches at random from the sample and putting them together on the output image.

However, in order to make texture consistent, patches need to be overlapped. This overlap is achieved by minimizing the difference (error) between patch overlap regions.

If the algorithm using a large path size and a high error threshold then the speed can be significantly faster.

In my work, the time taken to synthesize similar quality textures were improved by tens or better.

This approach (part 4) produces slightly better quality textures, especially for texture 1

and 5.

For texture 1 the issue of missing lines present in the first approach was no longer there. And the lines are more regular. In image 5, the text looks much more like text rather than a poor handwriting or a scrawl.

Part 4 is an improvement over the first one. But we still need to choose the patch size and error threshold carefully in order to get the desired quality and at the same time keep high efficiency.

2.5 Reference

1. https://github.com/chouxi/texture_synthesis
2. https://github.com/afrozalm/Patch-Based-Texture-Synthesis