Non-parametric Texture Synthesis
EE4212 Part 2 Assignment 1

Luca Krueger

April 4, 2020

1 Algorithm

This project covers an implementation of an algorithm for texture growing as proposed by Alexei A. Efros et. al in the paper *"Texture Synthesis by Non-parametric Sampling"*.[1]

To find a new pixel, the algorithm compares the neighborhood in the new pixel with neighborhood of each pixel of the source image. The pixel with the closest neighborhood becomes the source pixel and is copied to the new position. These steps are repeated until every pixel of the image is filled. In the beginning the source image is placed in the upper left corner. The size of the synthesized image can be specified by the parameter `SCALE`. The neighborhood of a pixel is compared by determining the pixel-wise L2-distance.

Obviously the neighbourhood of a pixel must have at least one non-zero pixel. Therefore the implementation only selects new pixels bordering to already synthesized pixels of the image. In addition to that it uses a mask $M \in [0,1] \times n \times n$ which exludes pixels with either zero value in the source or the target pixel from the neighbourhood.

The final image is filled starting from the source image from top to bottom and then from left to right.

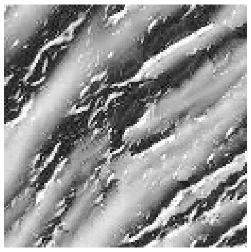
2 Performance and Accuracy

The accuracy of the algorithm highly depends on the `WINDOW` size used and the repetition of a particular pattern within a texture. I observed that high repititions of small patterns require a smaller `WINDOW` size for better performance whereas large patterns are synthesized more accurately using a larger parameter. (Figure 2, 3)

My first implementations of this algorithm weighted the pixel-wise distances with a $2D$ gaussian kernel. But this method only reduced the effective window size in a radial manner and did not improve accuracy. Therefore it is not used in the final implementation, which also gives a performance speedup.

The function `find_best_match` consists of a loop itaration over almost every pixel in the source image. The function `compute_distance` is called every iteration. To compute the distance, several element wise operations such as element-wise multiplication or computation of distance are required. Hence, this function scales to $\mathcal{O}(n^2)$, where n is the `WINDOW` size. Since the function `find_best_match` is called for each pixel, namely at a scale of $\mathcal{O}((s*(SCALE-1)^2)) = \mathcal{O}(s^2)$, where s is the source image size, the overall algorithm scales to $\mathcal{O}(s^4 \times n^2 \times SCALE^2) = \mathcal{O}(s^4 * n^2)$. This scaling can be slightly improved by making use of opencv's optimized math functions, but still requires a significant amount of time for larger images. The impact of the `WINDOW` size is not significant for the overall performance, because $s >> n$ for the general case.

(a) original


(b) `WINDOW` size 5

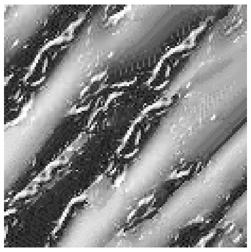
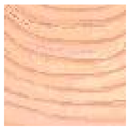
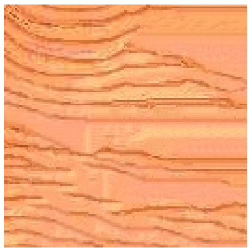

(c) `WINDOW` size 11

Figure 1: Texture 1


(a) original


(b) `WINDOW` size 5

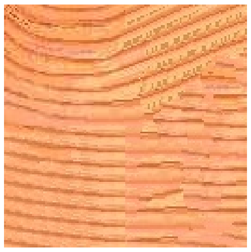
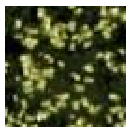
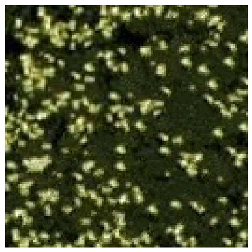

(c) `WINDOW` size 25

Figure 2: Texture 2


(a) original


(b) `WINDOW` size 5

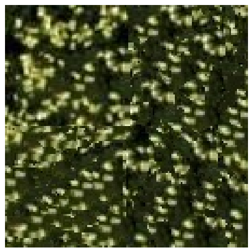
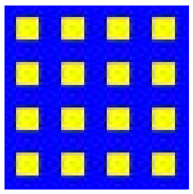

(c) `WINDOW` size 25

Figure 3: Texture 3


(a) original

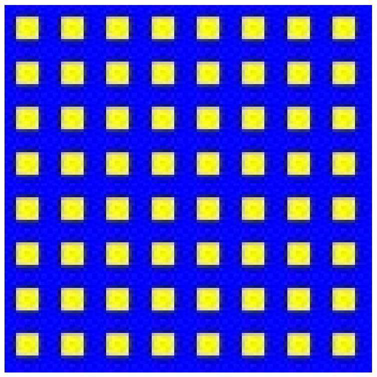
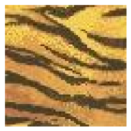
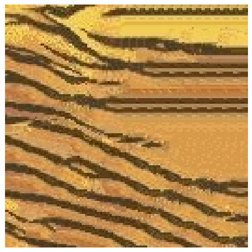

(b) `WINDOW` size 35

Figure 4: Texture 4


(a) original


(b) `WINDOW` size 11

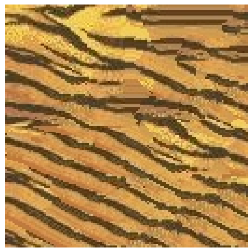
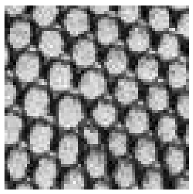
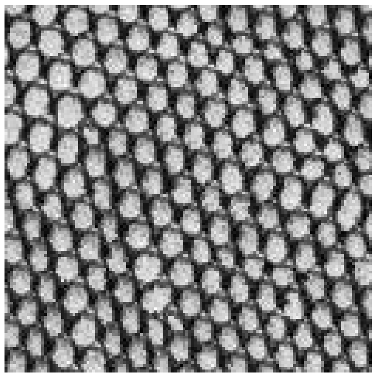

(c) `WINDOW` size 25

Figure 5: Texture 5



(a) original



(b) `WINDOW` size 11

Figure 6: Texture 6

References

[1] Alexei Efros and Thomas Leung. Texture synthesis by non-parametric sampling. In *In International Conference on Computer Vision*, pages 1033–1038, 1999.