Texture Synthesis using TSVQ and Target Re-synthesis

Anteneh Addis Anteneh[*]
CMSC 635 - Project

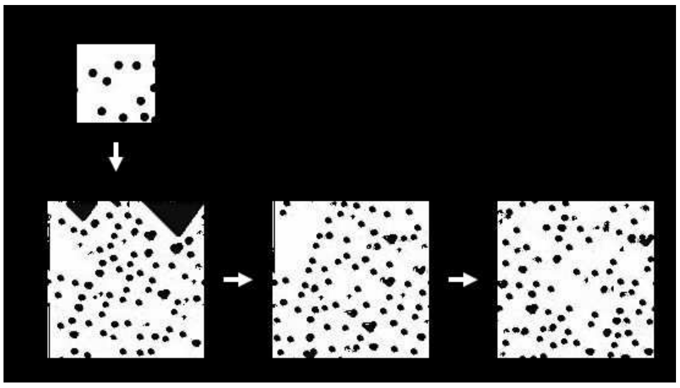


Figure 1: Texture synthesis through TSVQ and re-synthesis.

Abstract

This paper presents a simple texture synthesis method that is fast, efficient, and optimized. Given an input texture, the algorithm will synthesize a texture image of specified size by matching output image pixel neighborhoods with input pixel neighborhoods. The pixel-neighborhood matching will be aided by the use of a Tree-Structured Vector Quantization (TSVQ) method which will allow the algorithm to avoid exhaustively searching the input image pixels. TSVQ method allows the algorithm to construct a binary tree that will serve as a codeword for the pixel types in the input image. Re-synthesis will be performed on output textures a number of times to measure its effectiveness as an optimizing option. The presented method combines ideas of synthesis by example with a fast texturing algorithm.

Keywords: texture synthesis, re-synthesis

1 Introduction

This paper presents a method for efficient texture synthesis that combines a number of different approaches in texture synthesis with Wei and Levoy's fast texturing algorithm at its core. The method will implement a variation of the tree structured vector quantization method as described in [Wei and Levoy 2000]. The method will build the search tree based on the neighborhood value, $NV(p_{xy})$, for each pixel p_{xy} in the sample texture. The neighborhood value will be completed as a vector norm of the RGB values in each neighborhood picture. Based on these values, the search tree will be constructed along the lines of the median cut algorithm [Heckbert 1982]. An initial target image, I_t, will be synthesized on an input image of white noise. A second round of synthesis will be performed using image I_t as the input instead of white noise, producing the target image T. The image T will be re-synthesized a number of times in the experimental phase to see the effects of re-synthesis as an effective optimizing option.

The motivation for this work is to combine different aspects of synthesis and vector quantization to have an algorithm that is both fast and also produces quality synthesized images. The re-synthesis of the initial target image aids in iteratively improving the similarity of the target texture to the sample texture while preserving its difference/randomness that was dictated by the initial input noise image. The next section of this paper will go over some of the related work in this area with an emphasis on their importance to this work.

2 Related Work

This section will summarize previous work that this paper builds on and other related papers.

Color Vector Quantization: Vector quantization is a process of mapping vectors of the same size into a group of representative vectors called code words to build a code map for the whole vector space. The median-cut algorithm presented in [Heckbert 1982] is a quantizing method that subdivides color space into smaller and smaller bins from which a binary search tree is built from. The original bin is the color space containing all pixels, and this bin is split into two bins of more or less equal size based on the median value of a range of color values. Wie and Levoy's algorithm uses the Tree-Structured Vector Quantization as defined in [Gresho and Gray 1992] and uses the nearest point algorithm to match neighborhoods [Nene and Nayar 1997].

Mutiresolution/Pyramid-Based Sampling: Heeger and Bergen present a method for texture synthesis by matching the textural features of an input texture to an input noise image [1995]. The algorithm analyzes the input to get a number of texture parameter values and store them in an image pyramid. DeBonet presents a synthesis method through multiresolution pyramids to capture details of the image at different levels [DeBonet 1997].

Pixel Based Synthesis: Wei and Levoy extend earlier exhaustive search algorithms [Efros and Leung 1999] by implementing a multi-resolution synthesis pyramid to allow the use of smaller neighborhood sizes [Wei and Levoy 2000]. They also use tree-structured vector quantization (TSVQ) to accelerate the runtime of the algorithm. They have shown that synthesis results using TSVQ are comparable in quality to the exhaustive search method but with a synthesis time magnitudes faster than the latter. Ashikhmin [2001] reduces the search state by using information from earlier neighborhood comparisons. Hertzmann et. al. present a synthesis method by which an input image is synthesized through image analogies of input texture-synthesized texture example sets [2001].

Patch Based Synthesis: These methods perform synthesis by using patches from the sample texture. Kwatra et. al. present a method of synthesis where irregular patches are copied to the target image and

[*]e-mail: anteneh1@umbc.edu

seams are corrected through a graphcut algorithm [2003]. Praun et. al. present a similar method where an arbitrary surface mesh is covered by repeatedly pasting texture patches from the input[2000]. Seams are corrected using alpha blending.

Pixel-Patch/Hybrid Synthesis: Patch-based synthesis methods have proved to be fast and efficient algorithms for texture synthesis. Pixel-based searches are in most cases slower put provide the advantage of optimal target images. Pixel-based images have the disadvantage of blurring fine detail, while patch-based methods usually produce unwanted textures along overlapping seams. Nealen and Alexa's method presents a patch-based synthesis method that uses pixel-based re-synthesis to eliminate errors in overlapped patch regions [2003]. Ashikhmin's pixel-based method also includes the use of irregular shaped patches to eliminate synthesis errors [2001].

3 Implementation

As mentioned above, this work will be a variation on the TSVQ method of the WL algorithm. The method has three different phases: 1) the binary search tree is constructed from the input image, 2) a white noise image is converted into the initial synthesized image I_t by using the search tree, 3) re-synthesis is implemented once, and repeated if necessary.

3.1 Tree Construction and Synthesis

The completed search tree is a binary tree that is used as the codebook to classify all of the pixels in the sample image with respect to its neighborhood. This section will first look at initial attempts at tree construction. Then a working implementation for the search tree will be discussed.

3.1.1 Initial Attempts at Tree Construction

Initial attempts of tree construction focused on quantizing neighborhoods into a linear value. The trees were constructed such that leaf nodes had a number of pixels available to choose from. When a search reaches a leaf it may choose from one of those pixels in a number of ways to complete the match. Figure 2 shows a completed tree with median values on top of each node. One of the goals of this project is to develop a neighborhood classification method that is efficient and computationally inexpensive. An important part of this classification step that was explored in the initial stages was defining an equation or an algorithm that will adequately represent the characteristics of a single neighborhood. A number of classification methods have been explored, all of them based on the RGB values inside a neighborhood vector.

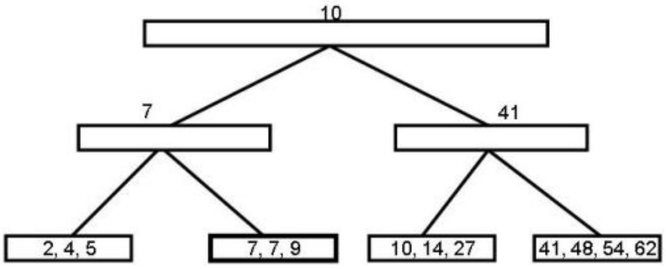


Figure 2: A tree with minimum bin size = 3.

The first method explored was to use the ranges of color values at each position in the neighborhood vector. The color (red, green, or blue) with the highest range at position i in the neighborhood vector becomes the splitting color and the median of that range becomes the splitting median. This method proved to be a poor representation of neighborhood values as searches went to one or a few pixel values and resulting in a single color image after the first few rows.

A second group of methods were considered by applying the vector norm to different combinations of values in the neighborhood vectors. The vector norm value of each L-shaped neighborhood around each pixel was calculated. The neighborhood value for each pixel was calculated by applying the L2 norm formula first to each RGB value, and then to these values over the vector space. The neighborhood value calculated using this method was not a good representation of a neighborhoods as can be seen in example synthesis run shown in Figure 3.

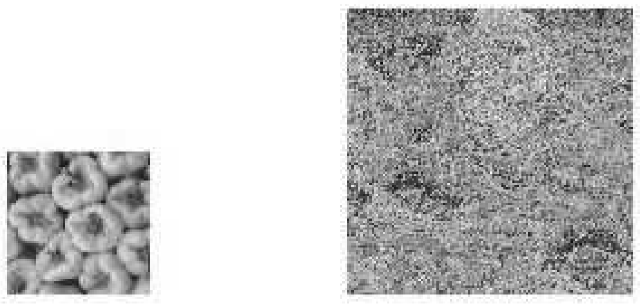


Figure 3: Synthesis Results from TSVQ using L2 norm.

A third method that was explored was to use the RGB values in each neighborhood pixel, and maintaining each color as a separate value to identify each neighborhood. To achieve this, the red, green, and blue values for each pixel in the neighborhood were put into a vector and the vector norm for each was calculated. This resulted in an [R, G, B] like representation for each neighborhood. These values were used to construct the search tree using a median-cut like algorithm for the color value with the highest range at each level of recursion. The results for this method were also poor but showed improvement from previous methods. Results from this method are shown in Figure 4 and Figure 5.

3.1.2 Tree Construction using limited neighborhood comparison

The final version of the search tree was constructed by comparing the pixel neighborhoods in the sample image to each other. After the initial methods were tried as mentioned above, it became clear that some level of comparison was necessary to construct a meaningful search tree. The algorithm described is implemented along the lines of [Schmidt 2005].The tree was constructed by clustering similar nodes together and then iteratively dividing those nodes until a leaf node is reached. In this implementation of the tree each leaf node contains either one or two pixel matches (in which case the target neighborhood would choose the best match).

Bins:A bin is a node in the search tree. Each bin has a pointer to its left and right child. In addition this node has a divider pixel - the pixel coordinate which specifies the neighborhood associated with each bin around the specified pixel. A bin also contains a list of pixels before it is split.

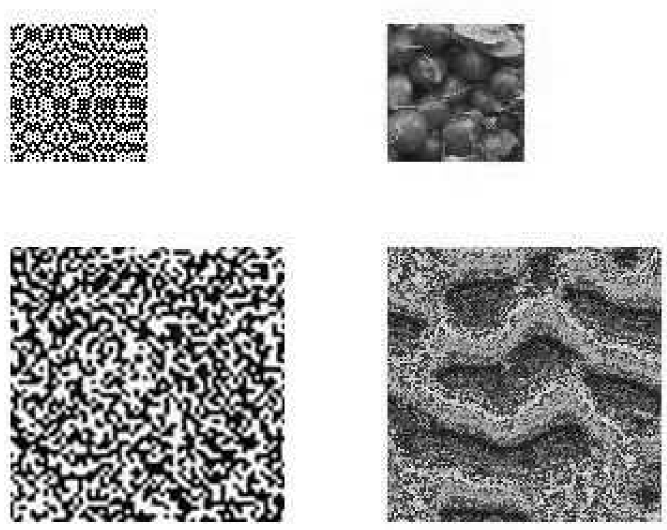


Figure 4: Synthesis results from TSVQ using RGB-like representation.

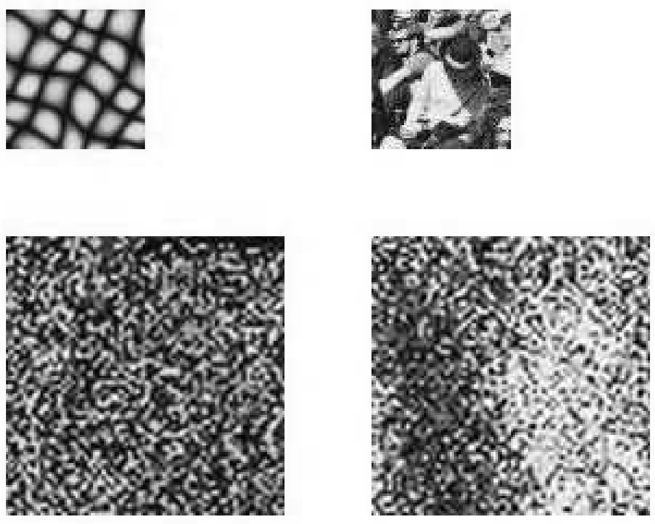


Figure 5: Synthesis results from TSVQ using RGB-like representation.

Building the Tree: The root bin is defined to contain no dividing neighborhood vector and is initialized to a neighborhood vector with zero values for each pixel color. The root bin also contains all the pixels of the original sample image. To generate the first two children, all pixels contained within the bin are used to obtain the neighborhoods associated with them (in this implementation all sample neighborhoods are constructed ahead of time). Each neighborhood is compared to the dividing neighborhood using sum of squared differences equation (L2 difference). The neighborhood with lowest difference value will become the left/nearest bin while the neighborhood with the highest difference value will become the right/farthest bin. After the left and right nodes are initialized the pixel values contained within the parent node are split and transferred into these bins. Pixels are split by calculating their difference with the left and right bins and transferring them to the node they are closest to. The above procedure is performed recursively on child nodes until a node either has no more pixels in its pixels list or only one.

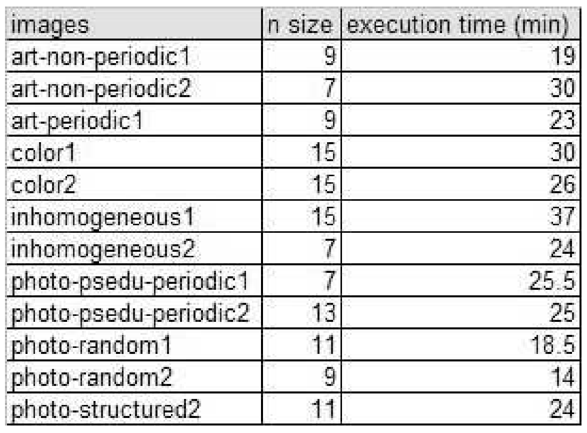
Searching the Tree and Matching a Pixel: After the tree has been constructed, the algorithm can begin its synthesis phase. For each pixel in the output image, a target neighborhood *TN* of size n is build around it. *TN* is used to search the tree through comparisons. At each bin the difference between TN and the dividing neighborhood of a child node. The search algorithm will iteratively search the tree by following the path of a neighborhood it is closest to between each child node. When a search reaches a leaf node it will return a matching pixel. If a leaf node contains only one pixel in addition to its divider pixel, two more comparisons will be made with *TN* to find the best match.

3.2 Texture Re-synthesis

A second round of synthesis is conducted on the first synthesized image in the same way as described above. The motivation for doing this is to optimize the synthesized image so that any irregularities caused during the first run of optimization will be minimized during multiple runs as the synthesized texture becomes more refined since its current pixel neighborhoods are more similar to the input image. The final implementation of this algorithm contains three synthesis iterations.

4 Results and Analysis

Testing was conducted on different sets of images chosen from the seven class of images on (http://www.cns.nyu.edu/ eero/texture). The runtimes for tree construction and synthesis vary depending on the neighborhood size and the size of the output image. The average runtime for synthesizing (with no re-synthesis) an image of size 128x128 with a neighborhood size of 15 is about 5 minutes. The average time to construct the tree is about 80 seconds. With re-synthesis the runtimes seem to increase according to the number of iterations, ranging on average from 14 - 25 minutes for three synthesis iterations. The table in Figure 6 shows the rounded execution times (in minutes) along with the neighborhood sizes used for the example result images used in this paper.



images	n size	execution time (min)
art-non-periodic1	9	19
art-non-periodic2	7	30
art-periodic1	9	23
color1	15	30
color2	15	26
inhomogeneous1	15	37
inhomogeneous2	7	24
photo-psedu-periodic1	7	25.5
photo-psedu-periodic2	13	25
photo-random1	11	18.5
photo-random2	9	14
photo-structured2	11	24

Figure 6: Execution times for different textures.

Example synthesis images using the algorithm described above are shown staring from Figures 7. For all figures the top images are the sample textures while the others are results of re-synthesis shown in order by the arrows. The TSVQ search algorithm produces good results for most textures although it is not optimal as compared to the exhaustive search comparison method. The result of re-synthesis is mixed for different types of textures. Figure 7 is a poor image for the initial synthesis and re-synthesis does not improve it anymore. Figure 8 is much better with the original synthesized image being greatly optimized over iterations. The same goes for Figure **??**, with the heart shaped splotches decreasing in size. The color textures produce good results with Figure 10, the least helped by re-synthesis. In Figure 11 re-synthesis optimizes the texture by correcting errors at the beginning of synthesis. The third iteration seems to be giving the texture more of its original randomness. Synthesis results for inhomogeneous textures are very poor, with the re-synthesis worsening the target texture. Figure 15 is also a poor result with re-synthesis not being able to have much effect. In Figure 16 re-synthesis has eliminated some of the blurring along the edges of the target image, with the optimal one being the second iteration. Re-synthesis also helps in Figure 18 with the target image gaining more of the original pattern.

In general the re-synthesis step has optimized the target image in varying degrees for various texture types. But the amount of time it takes for re-synthesis means that we can not continue to perform numerous iterations without eventually losing the advantage of fast texturing through TSVQ. Other optimization methods in combination with re-synthesis may be used to perform a minimal number of optimizations and to gain the optimal target image.

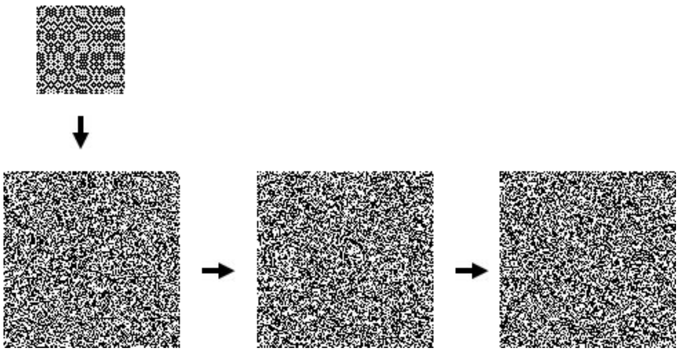


Figure 7: Results for art-non-periodic1.

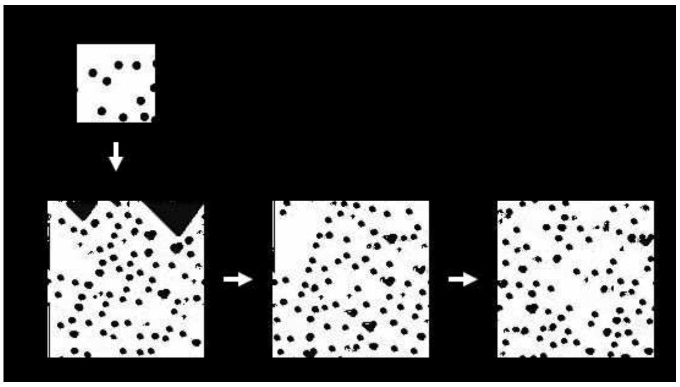


Figure 8: Results for art-non-periodic2.

5 Summary and Future Work

Summary: This paper presents a method to combine Wei and Levoy's fast texturing algorithm using search trees with a simple optimization heuristic. The re-synthesis phase allows for the optimization of a target texture incorporating the ideas in [Ashikhmin 2001] [Nealen and Alexa 2003] of using analogies of and pre-drawn target resembling textures. The search tree is constructed by utilizing simple L2 difference comparison between pixel neighborhoods in the sample image. The search is also performed by calculating very few number of L2 comparisons (as compared to the exhaustive method) to reach a matching neighborhood and pixel. This work has also shown that in most cases re-synthesis of target textures can be used as an option for texture optimization. But the amount of time re-synthesis takes also makes it only a partial optimizing option in combination with other algorithms.

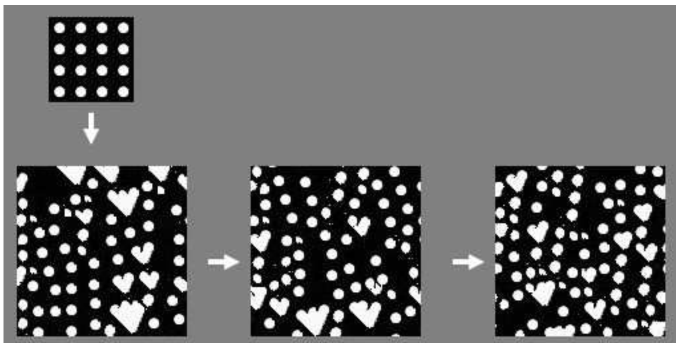


Figure 9: Results for art-periodic.

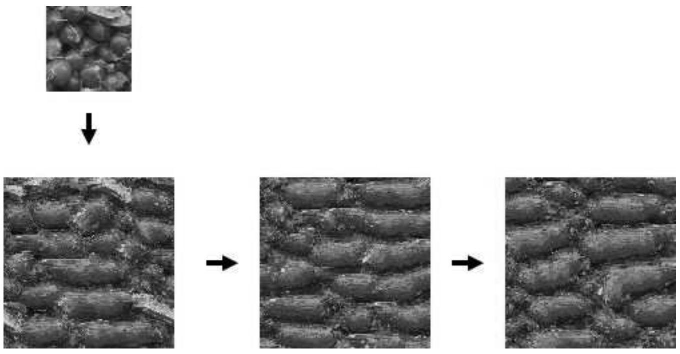


Figure 10: Results for color1.

Future work: Patch-based optimizations are strong options to use with a pixel based synthesis as they have proven to be much faster and equally optimal in general. One optimization method may involve only a second round of re-synthesis after a patch-based algorithm has fixed any trouble areas in the initial target texture.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, 217–226.

DEBONET, J. S. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques*, 361–368.

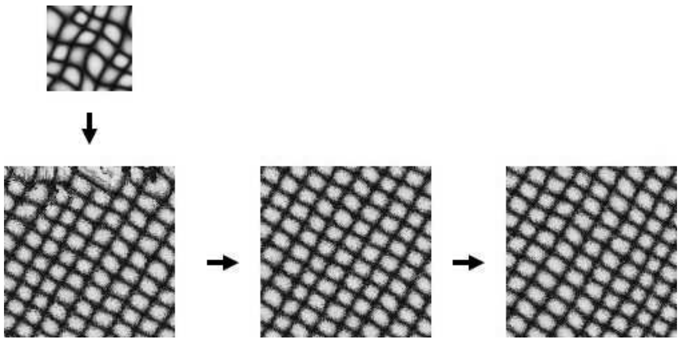


Figure 11: Results for color2.

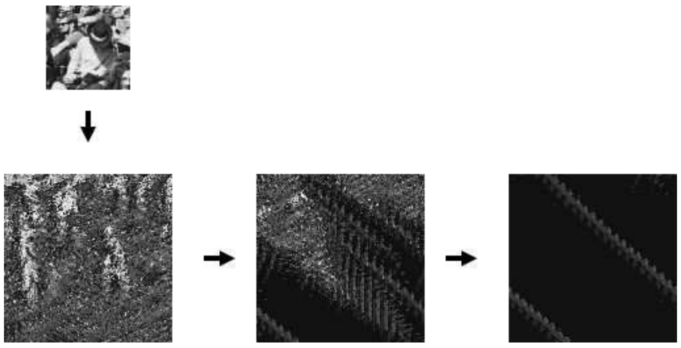


Figure 12: Results for inhomogeneous1.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *Proceedings of the IEEE International Conference on Computer Vision*, 1033–1038.

GRESHO, A., AND GRAY, R. M. 1992. *Vector Quantization and Signal Compression*. KluwerAcademic Publishers.

HECKBERT, P. S. 1982. Color image quantization for frame buffer display. 297–307.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques*, 229–238.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 327–340.

KWATRA, V., SCHDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. In *Proceedings of SIGGRAPH*, 277–286.

NEALEN, A., AND ALEXA, M. 2003. Hybrid texture synthesis. In *Proceedings of the 14th Eurographics workshop on Rendering*, 97–105.

NENE, S., AND NAYAR, S. 1997. A simple algorithm for nearest neighbor search in high dimensions. In *Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligenc*, 989–1003.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 465–470.

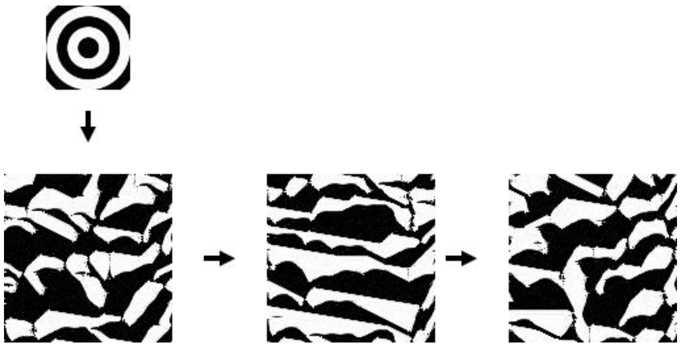


Figure 13: Results for inhomogeneous2.

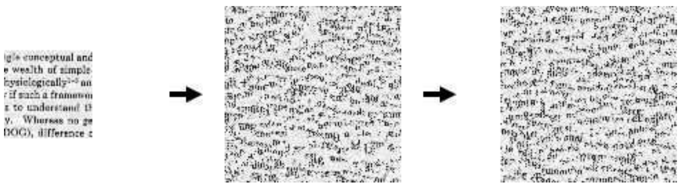


Figure 14: Results for photo-pseudo-periodic1.

SCHMIDT, J. 2005. Practical implementation of a texture synthesis algorithm. In *Proceedings of 9th Central European Seminar on Computer Graphics*.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the SIGGRAPH 2000*, 479–488.

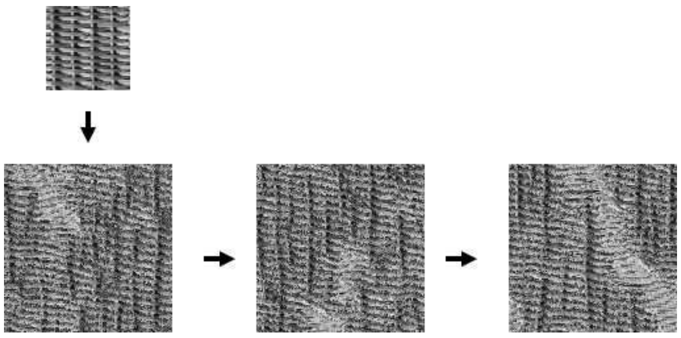


Figure 15: Results for photo-pseudo-periodic2.

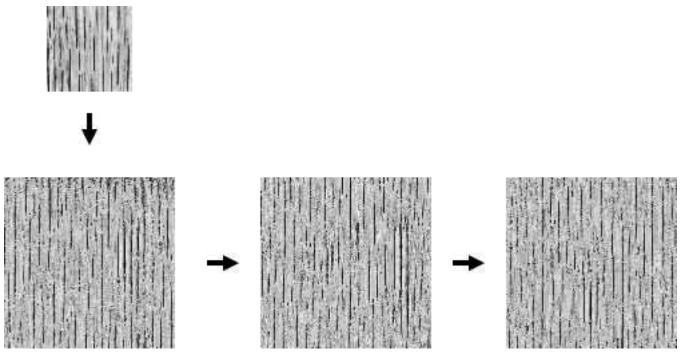


Figure 16: Results for photo-random1.

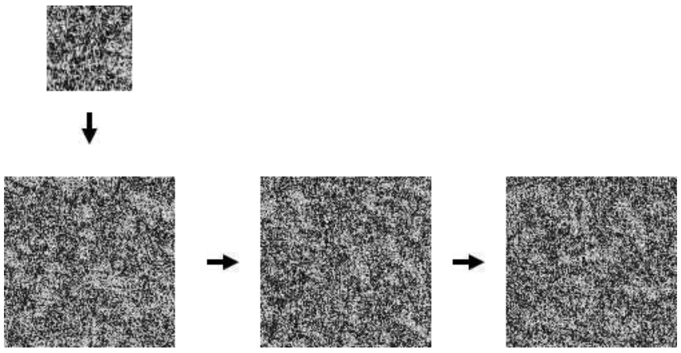


Figure 17: Results for photo-random2.

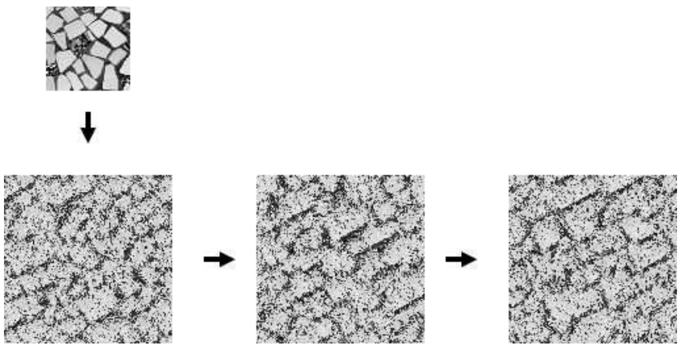


Figure 18: Results for photo-structured.