https://doi.org/10.2352/ISSN.2470-1173.2017.17.COIMG-436

# Non-Parametric Texture Synthesis Using Texture Classification

*Kyle Ziga[1], Judy Bagchi[2], Jan P. Allebach[1], Fengqing Zhu[1]*

*[1]School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA*

*[2]Dzine Steps LLC, Spring, TX*

## Abstract

*In this paper we present a method of texture synthesis which removes the need for users to set, or even understand, parameters which have an impact on the synthesized output. We accomplish this by first classifying each input texture sample into one of three texture types: regular, irregular and stochastic. We found that textures within a class were synthesized well with similar parameters. If we know the input texture class, we can provide a good starting set of parameters for the synthesis algorithm. Instead of requiring a user to manually select a set of parameters, we simply ask that the user tell us whether the synthesized texture is satisfactory or not. If the output is not satisfactory, we adjust parameters and try again until the user is happy with the output. In this implementation we use the image quilting method in [1], a texture synthesis algorithm, as well as texture classification. With small adjustments our method can be applied to other texture synthesis methods.*

## 1 Introduction

Texture synthesis has a variety of applications in areas such as computer vision, computer graphics and image processing. These applications include occlusion fill in by image inpainting, texture mapping, image compression and more. Also, with the growth of augmented and virtual reality applications, efficient texture synthesis algorithms have become more important. Texture synthesis is the procedure of "growing" a large texture image given a small sample texture. An example texture synthesis result can be seen in Figure 1 The resulting texture should not be a repetition of the sample pattern. Instead it should look as though it was created by the same underlying process. In this way only small texture samples are needed to generate large sized texture data. Current methods require a user to understand and carefully set parameters to produce satisfactory results. Removing the need for a user to tune parameters, while producing comparable results, will significantly improve existing texture synthesis methods.

Texture synthesis algorithms can be separated into two main categories: pixel-based methods and patch-based methods. These methods copy one pixel from the input texture sample to the output synthesized sample based on some defined condition. Efros and Leung [2] were the first to use this pixel-based technique. Their approach begins with a single seed pixel and grows the synthesized texture from that starting location. To synthesize a pixel their algorithm finds all neighborhoods from the sample texture which are similar, by some criteria, to the neighborhood in the synthesized texture with the pixel to be synthesized at the center. Once all candidate neighborhoods from the input texture are collected, one is chosen at random to prevent repetition in the output. The center pixel of the chosen neighborhood is then taken as the new synthesized pixel value in the output and the algorithm moves on to the next pixel location. This process can be extremely slow, limiting the practical application of such methods. Wei and Levoy [3] address this issue of speed by extending the previous method by using tree-structured vector quantization to speed the process of searching for candidate neighborhoods. Their method reports output quality equal or better to previous techniques while running two orders of magnitude faster.

The second category, patch-based methods, include the most recent techniques developed. These methods find and copy an entire patch from the input image. The patches are placed into the output image and then the transition from one patch to another must be taken into account so as to hide the seams between patches. The way the patches are made to transition smoothly differs between approaches. In [4] the boundary artifacts are removed by blending the transition areas. They use feathering, or blurring, across the patch boundaries in order to create smooth transitions from one patch to the next. Efros and Freeman in [1] allow neighboring patches to slightly overlap, then compute a similarity metric for the overlap regions of two patches. Using this metric they construct a list of all patches which meet the criteria, then randomly select one patch to avoid repetition in the synthesized texture. They then perform a boundary cut which minimizes the error in the overlap regions of two patches, finding the best seam. Selected parameters for this method highly affect the output, as illustrated in Figure 2. The two previous methods both use regular and constant sized patches, generally square. In [5] the boundary cut is extended further. They use irregular

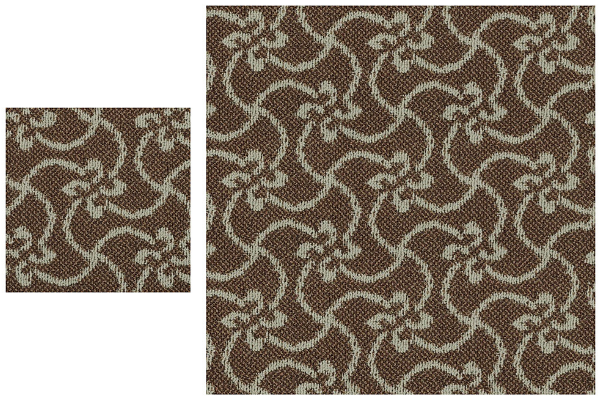
Figure 1: An example texture synthesis result. The left is the original texture, the right is the synthesized texture at twitce the size of the original.

patches without a constant patch size in order to find the optimal seam between patches. A graph cut approach is used to determine the optimal patch seam for any given region of the output texture.

Some methods work better for certain types of textures while some textures are extremely difficult to synthesize regardless of the method used. However, these results come after the process of working with the algorithms and understanding how parameters affect the output. Such a process is tedious, burdensome to untrained users, and difficult to adapt to real-life appliactions. Our work is based on the image quilting method described in [1]. In this case the parameters include: size of the patch, overlap area and error tolerance for the overlap regions. For someone very familiar with the details of the texture synthesis algorithm being used paramter tuning is not a big issue, but for untrained users this can be a frustrating trial and error process. We propose a solution to this problem by first classifying the input texture into a predetermined texture type. Once the texture type is known, an initial estimate of parameters for the image quilting method can be set without any user input. This allows for texture synthesis to be carried out automatically in applications where a user does not have a technical understanding of the algorithm. The only user input required is feedback of whether the synthesized result is satisfactory.

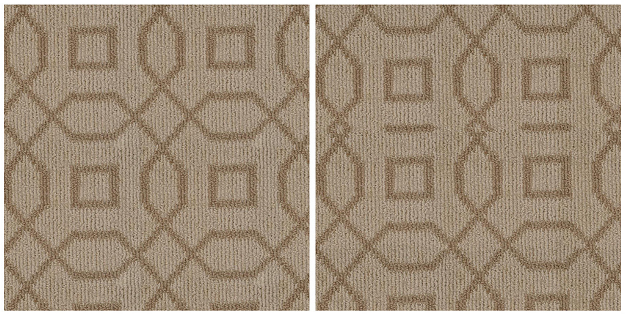
Figure 2: Example of patch size effects. Left: Patch size of 500x500 pixels (Structure retained), Right: Patch size of 450x450 pixels (Structure not retained)

## 2 Texture Classification

We found that the texture synthesis parameters which produce the best results strongly depended on certain characteristics of the input texture. For example, if the input texture has a strong repeating pattern, the patch size must be on the order of the fundamental repeated element, visualized in the right image of Figure 6. If this is not done correctly, the structure of the pattern would not be maintained as seen in Figure 2. It was also noticed that in highly structured textures the error tolerance in the overlapping regions of patches needed to be very strict. Failure to do this would also result in the structure of the pattern not being kept. On the other end of the spectrum we found that if the input texture was very random and noise-like, the parameters needed to be more relaxed. The output for these types of textures depended on the error tolerance for the overlap regions being much lower. If this was not done the output would look repetitive and a clear difference from the input image would be noticeable. These obvservations led us to create a preprocessing step in the image quilting algorithm [1] in which we first identify the type of texture the input is. Once we know which type of texture the input is we can predict a good starting point for the image quilting parameters. Then, with simple user feedback, we can adjust the parameters accordingly until the synthesized output is satisfactory.

### 2.1 Texture Types

Textures can generally be classified into two types: stochastic and regular. However, most real-world textures fall somewhere between these two classes forming a spectrum of texture types [6]. In our preprocessing step we classify three texture types: stochastic, irregular and regular. An example of each texture type is shown in Figure 3. These three classes were selected based on similar parameters needed to produce the best texture synthesis results. The application of our work is in the field of interior design, as such we can provide examples of each type of texture relating to interior design. Stochastic textures are random, noise-like textures, commonly found in carpets. Irregular textures fall between stochastic and regular texture types. Textures that fall under this category do not have a clear repeating structure, but also are not completely noise-like. Examples of this type of texture are marble, granite, or natural stones. Regular type textures have very strong repeating structures, like the square patterns shown in the right image of Figure 3. A single element of this repeating pattern, or the fundamental repeating pattern, can be identified and extracted. An example of the fundamental repeating pattern can be seen highlighted in green in the right image of Figure 6.

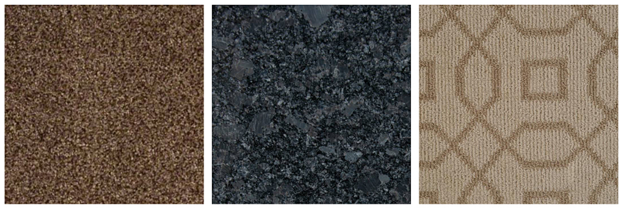
Figure 3: Example texture classes. Left: Stochastic, Center: Irregular, Right: Regular

### 2.2 Local Binary Patterns

To perform the texture classification we made use of local binary patterns [7]. A texture feature vector could be extracted from the input image by using the local binary pattern operator on neighborhoods within the input image, then constructing a histogram for the transformed image. This method was selected for computational simplicity. Generating a local binary pattern image is done by encoding each pixel based on the pixel's relation to neighboring pixels. For a simple explanation of local binary pattern encoding we will examine a 3x3 neighborhood from a texture image as shown in Figure 4. Each of the eight neighbors is compared to the center pixel. Every neighboring pixel is assigned to one bit in an 8-bit binary number. If the neighboring pixel is greater than the center pixel, that pixel is assigned a value of one. If the neighboring pixel is less than the center pixel, a value of zero is assigned to that pixel. Once all comparisons have been made the center pixel is assigned the integer value of the 8-bit codeword by concatenating the newly assigned values of the neighboring pixels. The order of the neighboring pixel values used when generating the codeword is arbitrary but must be kept consistent for all pixels. The neighborhood for performing the encoding can be modified in two ways. First, a circular neighborhood can be used where only a radius is defined. Since the points of this neighborhood will not lie on the rectangular grid of the image, pixel values for neighboring locations will be interpolated. Second, the number of neighbors does not need to be set to eight. A different amount of neighbors will change the number of possible encoded values in the output image.

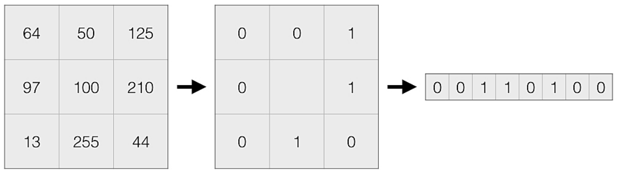

Figure 4: Left: Example 3x3 pixel neighborhood, Center: Comparison of neighbors to center pixel, Right: Local binary pattern codeword

Once this process is completed we have an image in which each pixel value is the local binary pattern encording. We now need to construct the texture feature vector from this image. The feature vector is constructed by concatenating histograms for non-overlapping windows within the local binary pattern image. For our 3x3 neighborhood example we selected a window size of 15x15 pixels following [7]. For each window we could construct a histogram of length 256. The histogram for each window would be concatenated to form the global feature vector for the texture image. This feature vector could then be used by any machine-learning method to classify the texture.

### 2.3 Classifying

We use a simple approach to classify textures as a proof of concept. In order to classify input texture images of unknown texture type, sample textures belonging to each class must first be collected. We used four samples from the stochastic class, ten samples from the irregular class and eight samples from the regular class. The local binary pattern feature vector was then computed for each training image. When the class of an unknown texture is desired, the same steps are followed to compute the feature vector of the unknown texture. Then the feature vector is compared with the training data to determine which class the new texture belongs to. In this paper we compute the chi-square distance between the feature vectors given chi-square distance is used to measure distance between histograms. We then use a nearest neighbor approach and determine the class that the new texture belongs to as the one with the minimum average chi-square distance.

## 3 Our Approach

The intent of this paper is provide a method which eliminates the need for the user to perform tedious parameter tuning while still producing high quality texture synthesis results. However, even the latest texture synthesis algorithms do not succeed in synthesizing every input texture. For this reason we still need to use some user information to steer our texture synthesis method towards a more appropriate set of parameters. In our method the user simply needs to respond to the question of a satisfactory output with a ”yes” or ”no” after the synthesis has taken place. If answered ”yes”, the program will terminate, if ”no”, the parameters from the previous run will be adjusted and synthesis will start over until the results satisfy the user.

Our method follows these steps to synthesize a texture:

1. Compute the local binary pattern image of the input texture.
2. Use the local binary pattern image to construct the texture feature vector for the input texture.
3. Classify the input texture as one of the three texture classes.
4. Set initial parameters for texture synthesis based on the texture class.
5. Run texture synthesis algorithm with defined parameters.
6. When synthesis is complete ask user if the result is satisfactory.
   (a) If yes, quit.
   (b) If no, update parameters and go to step 5.

### 3.1 Parameters

Based on the class of texture the input image has been identified, along with the user feedback, we were able to develop a strategy to set the parameters for the image quilting algorithm which produce the best results. Since there is variability in the best parameters even within a texture class, the user feedback will allow the parameters to be updated. The table in Figure 5 shows the initial set of parameters used once the texture class is identified. The patch size parameter is defined as a percentage of the input image size, the overlap is defined as a percentage of the patch size and the error tolerance is a raw difference in a selected error metric between patch overlap regions.

| Class | Patch Size | Overlap | Error Tol. |
|---|---|---|---|
| Stochastic | 0.4 | 0.05 | 0.1 |
| Irregular | 0.5 | 0.1 | 0.01 |
| Regular | 0.9 | 0.1 | 0.001 |

Figure 5: Initial parameters for each texture class.

The user response to the output of the texture synthesis tells our method what to do next. If the user says that the output is not satisfactory the parameters need to be adjusted to try and correct the error. The overlap area still remains to be calculated as a percentage of the patch size. Error tolerance also does not need to be adjusted once the texture class is found. The only parameter that needs to be adjusted is the patch size. The adjustment is made as a percentage of the previous iteration’s patch size. The new patch size is ninety percent of the previous patch size for each class. It was found that this parameter plays the most important role in the synthesis results. For regular textures the patch size needs to be on the order of the fundamental repeating structure or the pattern will not be kept. An error in a repeated regular pattern is easily perceived by humans. Generally, the patch size must be large enough to capture the structures within the input texture, but small enough that the output image does not look too similar to the input texture. The output must look like a different image generated by the same process. To achieve this our method starts with an over estimation of the patch size and slowly decreases the size until the output is satisfactory. Figure 6 demonstrates this procedure. Each box represents the patch size for one round of texture synthesis. When the patch size is set to the size of the green box it is on the order of the fundamental repeating structure, as demonstrated by the right image, and the synthesized texture is accepted. After each round the patch sized is reduced to ninety percent of the previous size regardless of which class the input texture belongs to.

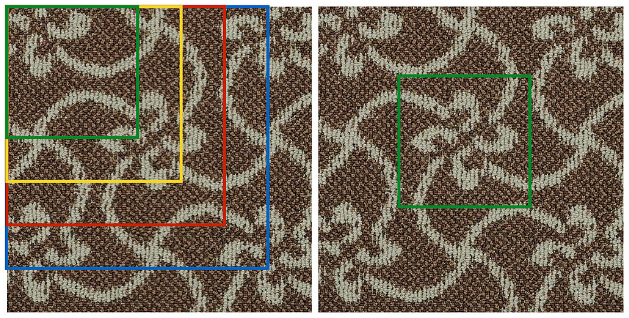
Figure 6: Illustration of patch size progression through user feedback. Left: Each box represents the patch size for each iteration of texture synthesis, Right: Green box shows appropriate patch size on the order of the fundamental repeating structure.

## 4 Experimental Results

To test our method we used real world textures of varying texture types. The textures were selected to represent a range of those that were commonly found in the interior design industry such as flooring and countertops. No parameters for the image quilting method were needed as input from the user. The only user interaction was the feedback described in previoius sections.

Our classification results are satisfactory, though only tested on a small sample size. We used twenty-two training images received eighty percent accuracy when testing on images containing similar characteristics to the training set. As the sample size increases, intra-class variation may cause an issue using a nearest neighbor approach as discussed in this paper. In that case another classification method may be investigated.

The results of our method are shown in Figure 8. For stochastic type textures, (Figure 8 (a), (b), (c), (d), (e) and (f)), our method works very well. This is because a wide range of parameters would yield pleasing results. Any perceptual differences in the synthesized output for different parameters are difficult to detect. Structural textures with a strong repeating pattern are also very succesfully synthesized using our method. Unlike stochastic textures, structural textures are very sensitive to the parameters. This causes our methods to require approximately 3-5 iterations of user feedback in order to correctly synthesize most structural textures. Results of structural textures are shown in the third image in Figure 8 (j), (k), and (l).

The texture class that is challenging for the proposed method is the irregular texture. There are two reasons why these texture types are difficult to properly synthesize. The first is that they do not have a fundamental repeating pattern but they do contain some structure which can be easily seen when synthesized incorrectly. Working with a sample image limits the available data used to synthesize, creating the possibility of leaving out some structural characteristics which may be obvious to a human viewer. This can be seen in Figure 7 (a). The stones clearly have some structure, but not a well defined structure. In the synthesized image it is obvious that mistakes have been made when pebbles of different color and texture are attempted to be matches in an overlapping region. The second reason is that most real world irregular textures have multiple layers of textures within them. For example Figure 7 (b) is an irregular texture of stones, but within each stone there exists a stochastic type texture. Our method will try to synthesize the irregular structure as best it can, but in certain cases such as the one in the figure the mistakes are very obvious. This is not true in all cases. Figure 8 (g), (h), and (i) show cases where irregular textures are able to be correctly synthesized by our method. We believe that in order to overcome the challenge of irregular texture sythesis, methods other than patch based algorithms must be developed. The use of patches limits the total information available to that which exists in the input image. A model based method has the possibility to produce better results for irregular textures. One would need to extract an accurate model for the generation of the texture given in the input image, then synthesize a new image according to the model for the output.

## 5 Conclusion and Future Work

In this paper we have presented a method that allows a user to synthesize real world textures without the need to understand or tune parameters. We have done this by first classifying an input texture into one of three classes: stochastic, irregular and regular. Once the texture type is known, a starting estimate of the best parameters can be made automatically. Then with a simple user feedback we can adjust the parameters until a satisfactory result is produced. This allows for the integration of texture synthesis into tools where users may not have technical knowledge of the texture synthesis algorithm being used.

## References

[1] A. A. Efros and W. T. Freeman, “Image quilting for texture synthesis and transfer,” *ACM SIGGRAPH*, vol. 28, pp. 341–346, August 2001.

[2] A. A. Efros and T. K. Leung, “Texture synthesis by non-parametric sampling,” *The Proceeding of the Seventh IEEE International Conference on Computer Vision*, pp. 1033–1038, September 1999. Corfu, Greece.

[3] L.-Y. Wei and M. Levoy, “Fast texture synthesis using tree-structured vector quantization,” *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, vol. 28, pp. 479–488, July 2000.

[4] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, “Real-time texture synthesis by patch-based sampling,” *ACM Transactions on Graphics*, vol. 20, pp. 127–150, July 2001.

[5] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick, “Graphcut textures: Image and video synthesis using graph cuts,” *ACM Transactions on Graphics*, vol. 23, pp. 277–286, July 2003.

[6] W.-C. Lin, J. Hays, C. Wu, V. Kwatra, and Y. Liu, “A comparison study of four texture synthesis algorithms on near-regular textures,” *ACM SIGGRAPH Posters*, p. 16, August 2004.

[7] T. Ojala, M. Pietikainen, and T. Mainpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 971–987, July 2002.

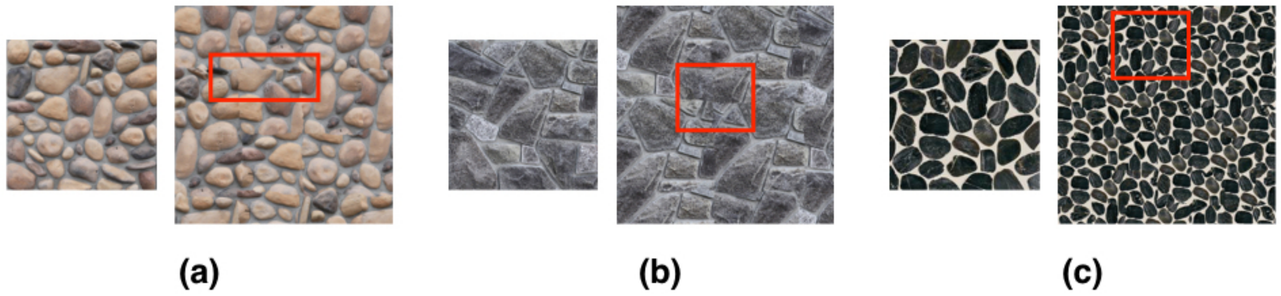

Figure 7: Experimental results: Examples of failed synthesis attempts. The smaller image is the input sample and the larger image is the synthesized output. Output synthesized at twice the input size. The red boxes highlight errors in the synthesized output.

## Author Biography

*Kyle Ziga received his BS in Computer Engineering from the Calumet campus of Purdue University in 2015. He is now working on his PhD at Purdue University West Lafayette in the field of Image Processing.*

*Jan P. Allebach is Hewlett-Packard Distinguished Professor of Electrical and Computer Engineering at Purdue University. Allebach is a Fellow of the IEEE, the National Academy of Inventors, the Society for Imaging Science and Technology (IS&T), and SPIE. He was named Electronic Imaging Scientist of the Year by IS&T and SPIE, and was named Honorary Member of IS&T, the highest award that IS&T bestows. He has received the IEEE Daniel E. Noble Award, the IS&T/OSA Edwin Land Medal, and is a member of the National Academy of Engineering. He currently serves as an IEEE Signal Processing Society Distinguished Lecturer (2016-2017).*

*Fengqing Zhu is an Assistant Professor of Electrical and Computer Engineering at Purdue University, West Lafayette, IN. Dr. Zhu received her Ph.D. in Electrical and Computer Engineering from Purdue University in 2011. Prior to joining Purdue in 2015, she was a Staff Researcher at Huawei Technologies (USA), where she received a Huawei Certification of Recognition for Core Technology Contribution in 2012. Her research interests include Image processing and analysis, video compression, computer vision and computational photography.*

*Judy Bagchi is founder and CEO of Dzine Steps, a cloud software provider to home builders and independent design centers nationwide. Prior to this Judy held Research & Development Management roles at Hewlett-Packard and Nortel. She is an industry veteran with more than 20 years of experience in the entire business value chain. She has a keen interest in and has led and participated in various activities supporting Women in Technology.*

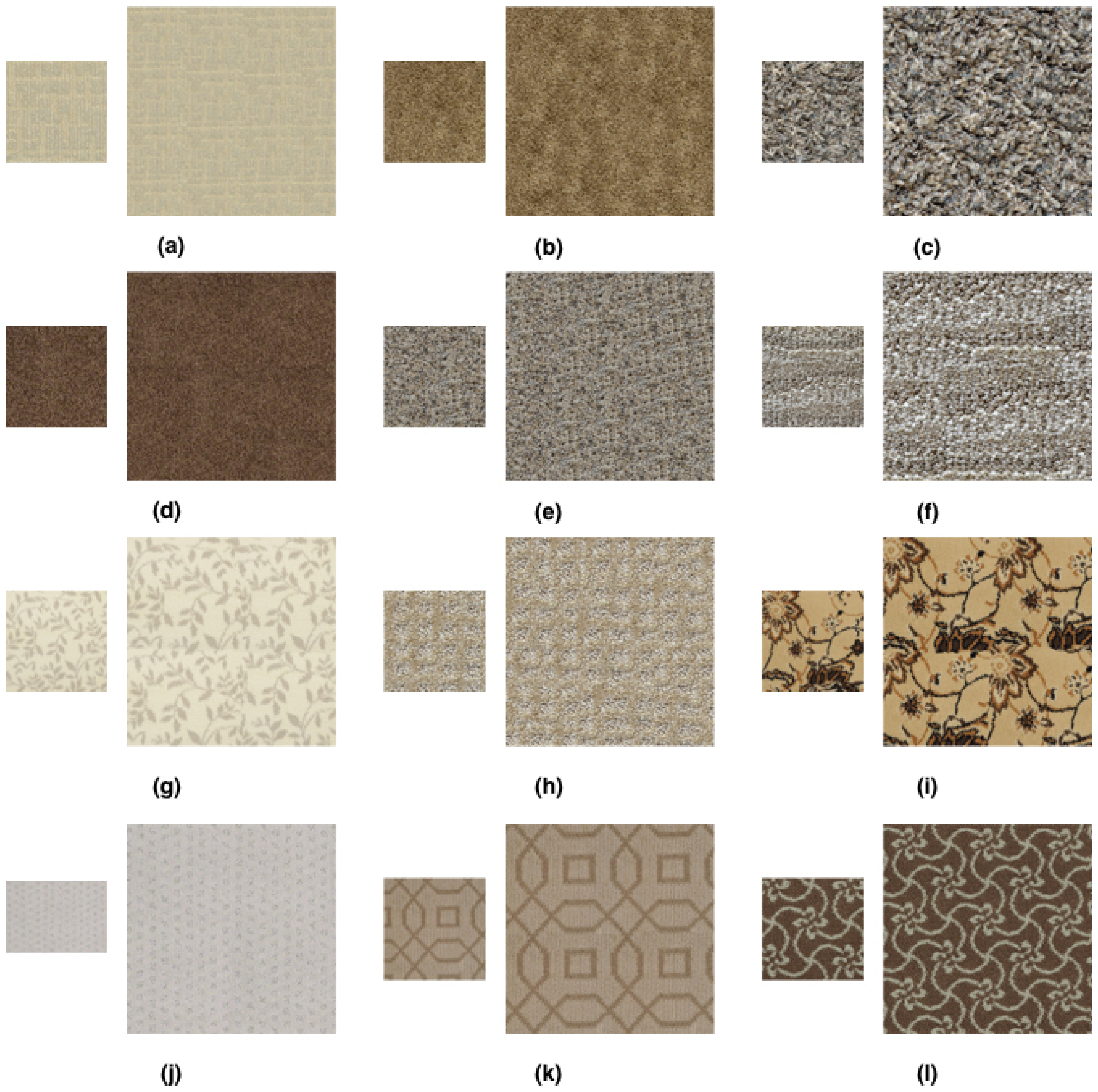

Figure 8: Experimental results: Examples of successful synthesis results. The smaller image is the input sample and the larger image is the synthesized output. Output synthesized at twice the input size.