CSE 559A Project : Image Quilting for Texture Synthesis and Transfer

Seunghwan "Nigel" Kim
seunghwan.kim@wustl.edu

Abstract

In this report, I present a concept of synthesizing new image from using patches of an existing textural image as presented in a paper by Efros and Freeman[1]. I focus particularly on the output texture being synthesized smoothly, without any means of smoothing edge artifacts. First, naive approaches of synthesizing textures from previous works are investigated. Second, I extend the algorithm to implement image quilting, which is a fast and simple way of constructing a new texture image that works well on smoothing the edge artifacts. Finally, I extend the image quilting algorithm to integrate with a target image, and while keeping the target images properties, transfer a source texture on to the target image. Then I demonstrate how texture transfer can be applied in different angles to reproduce an image.

1 Introduction

In a visual world, color and shape properties of a scene play a direct role in humans visual perception of gaining information. In perceiving shape, we can extend our view microscopically to gain textural information that often gives us direct insight on the material properties of a scene or an object. The existence of textural information contributes the most to a human viewer on whether we will perceive the scene as real or artificially generated. Therefore, in a computer-generated scene, textural information can be considered to play the most important role in persuading the viewer. In this report, I will investigate a paper on Image Quilting for Texture Synthesis and Transfer by Efros and Freeman[1] to implement and experiment with a novel texture synthesis algorithm to generate as many new texture samples as we desire by patching arbitrary image blocks from an input texture source. I will take an incremental approach to the texture synthesis problem by starting from implementing naive algorithms that were presented in previous works, then extend it to *image quilting* algorithm by Efros and Freeman[1] that produces a very good result. Then, I will integrate the concept of *correspondence map* with image quilting algorithm to generalize the algorithm to not only synthesize but transfer texture onto an existing image while minimizing the loss of its visual appearances.

2 Background & Related Work

Texture analysis started from a simple concept of matching appropriate statistics of two texture, and Bela Julesz[2] proposed that by a careful statistically matching, we can make two textures to be perceived to be very similar to human eye. This was not a direct approach to the problem but rather a forced approach because it was essentially taking a random noise image and modifying them to be statistically similar to the target texture in terms of histograms of filter responses at multiple scales and orientations. Heeger and Bergen[3] took this approach and produced good synthesis results for stochastic textures, but since this does not consider the local relationship along scales and orientations, it does not perform well on structured textures. Beginning with Xu el.al[4], a Chaos mosaic was proposed, which takes a very simple approach of taking random patches from the source texture and placing it randomly onto a new frame to synthesize a texture. They smoothed out the blocks using alpha blending to avoid edge artifacts. As before, it produced good results on stochastic texture(where there is almost no edge artifact due to its textural property), but it failed on structured textures due to the edge artifacts being produced from random extraction of source texture patches. Efros and Leung[5] then took an approach to go through one-pixel-at-a-time, looking at its conditional distribution by a search through the texture source and finding a best-matching neighboring region. This local approach could be applied to wide range of textures, but it was very slow due to its greedy approach.

3 Previous Approach

3.1 Random Placement with Alpha Blending

As proposed by Xu el.al[4], I randomly sampled source texture patches(blocks) of size B and place it randomly

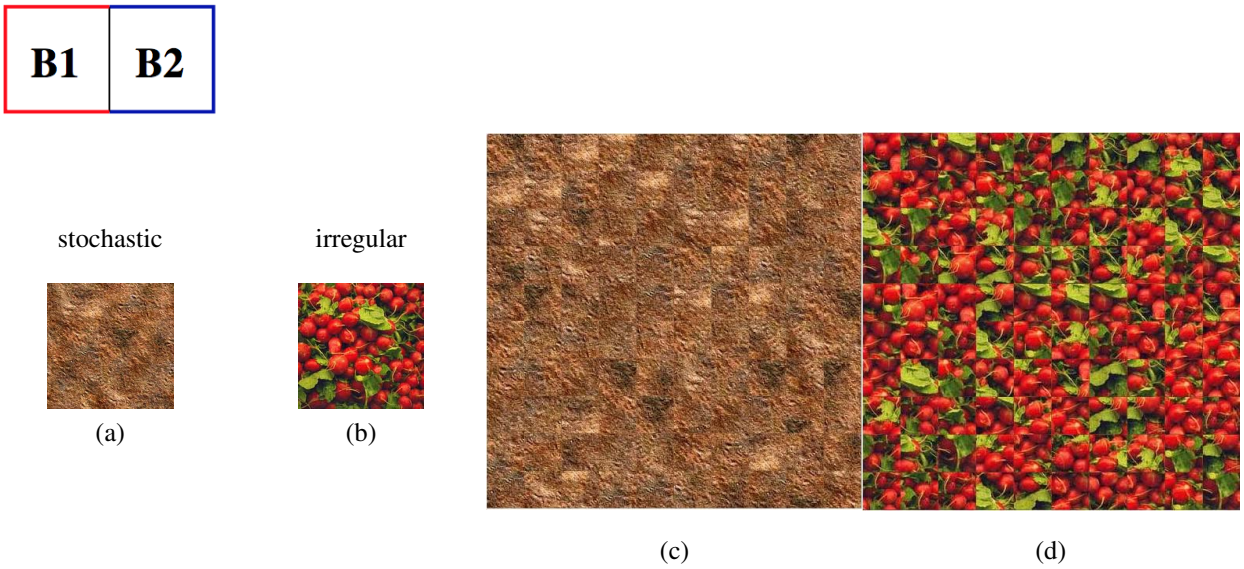


Figure 1: Random Placement with Alpha Blending. Square blocks from the input texture are patched together randomly to synthesize new texture: (a) stochastic texture input, (b) irregular texture input, (c) stochastic texture synthesized; edge artifacts are gone, (d) irregular texture synthesized; edge artifacts can be clearly seen.

along the side of an already synthesized block on the synthesized texture image. In each placement of the source texture patch, the concatenation edge will be alpha blended by a linear combination of the previous block's edge values and the placement block's edge values along the concatenation edge. As shown in Figure 1(a), this method works well with stochastic texture sources because it produces almost no edge artifacts when a source texture patch is randomly extracted. However, as shown in Figure 1(b), this method does not completely smooth out the edge artifacts produced when we use a non-stochastic texture source such as the ones with irregular patterns and structured patterns.

4 Proposed Approach: Texture Synthesis

4.1 Match Texture Block with Overlap Region

We can bring the previous approach of random placement along previously synthesized blocks edge and alpha blending to take into account overlapping placement of blocks. Instead of concatenating along the edge, we now have an overlap region of both previously synthesized block and a currently extracted source texture block. Therefore, in place of random placement of each new randomly extracted block, we can now search the source texture input space for possible neighbor blocks that by some measure matches with the already synthesized block along the overlap region. In my implementation, the matching constraint was set to be a block with error value within an error tolerance, which is 0.1 times the error of the best matching block(block that produces the least error). The L2 norm was used to compute the error between the already placed block and each block in all possible set of possible overlapping blocks. When there are multiple matches, one block is randomly selected among the matches as a neighbor, and pasted along the overlap region onto an already synthesized texture block.

Using this method, we can get an improved result than the previous method because we have taken into consideration a similarity measure to further smooth out the edge artifacts. Results are shown in Figure 2, where Figure 2(a) shows the synthesized texture using a stochastic texture source and Figure 2(b) shows the synthesized texture using a non-stochastic texture. However, we can still see that some edge artifacts are visible, and this calls for a further optimization.

4.2 Minimum Error Boundary Cut

We can optimize the previous algorithm with a more careful approach in the placement process. In the previous algorithm, by simply overlapping a matched candidate

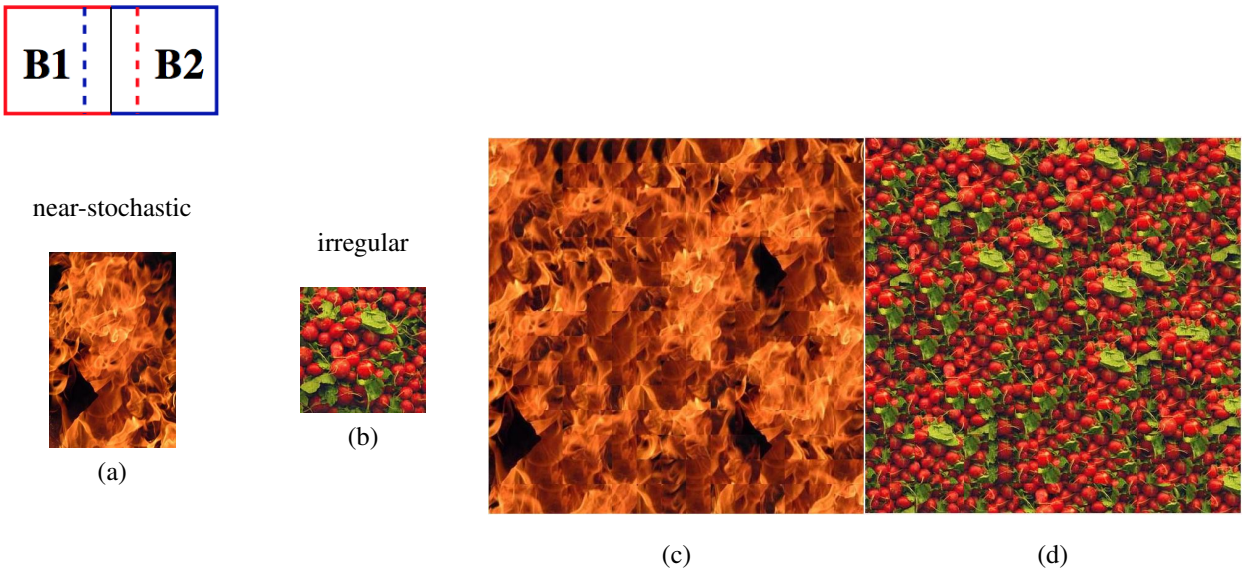


Figure 2: Neighboring blocks constrained by overlap. Square blocks from the input texture are chosen to match and overlap with its neighbor to synthesize new texture: (a) near-stochastic texture input, (b) irregular texture input, (c) near-stochastic texture synthesized; edge artifacts visible, (d) irregular texture synthesized; less edge artifacts than Figure1(d), but still visible.

block to the previously synthesized neighboring blocks resulted in a loss of the previously synthesized blocks' visual properties in the overlapping region, because a matched block will replace the previously synthesized blocks' overlapping region. To maintain both the previous block and current matched block's visual properties, Efros and Freeman[1] proposes to find a minimum error boundary along the overlapping region and make a cut along the path. The algorithm to find the minimum error boundary cut can be constructed using dynamic programming(below is for vertical cut):

Minimum Error Boundary Cut Algorithm(vertical):

1. Let B_1 and B_2 be two blocks that overlap,
2. Get error surface using L2 norm error measure between the overlapping regions B_{1ov} and B_{2ov}

Forward Pass (constructing cost paths table)
3. Initialize dynamic programming table of the same size as the error surface with zeros. Set the first row with the first row of the error surface(initializing starting points).
4. For each starting point in the first row, move onto next row and check 3 neighbors that are directly connected with 8-connectivity to the previous point (bottom-left, bottom, bottom-right).

a. Select the minimal-error neighbor and add it along the path. Update dynamic programming table with cumulative error value.

Backward Pass (selecting the minimum cost path)
5. Initialize the minimum cost path starting point with the minimum value in the last row (we are using a cumulative cost path table, so the point with the minimum value in the last row represents minimum cost path). 6. Visit each row in descending order, checking 3 direct neighbors connected with 8-connectivity from the previous point (top-left, top, top-right).

a. Select the neighbor with minimum cost(error), and label it 0.
b. In the same row, label all pixels on the left of the 0-labeled pixel with -1 (this will mark the part in the overlap region that corresponds to the previously synthesized texture block, maintaining its textural properties).
c. Similarly, label all pixels on the right of the 0-labeled pixel with 1 (this will mark the part in the overlap region that corresponds to the newly matched texture block, bringing its textural properties).

4.3 Image Quilting

Integrating the previous overlap-region-matching method and the minimum error boundary cut algorithm, we can

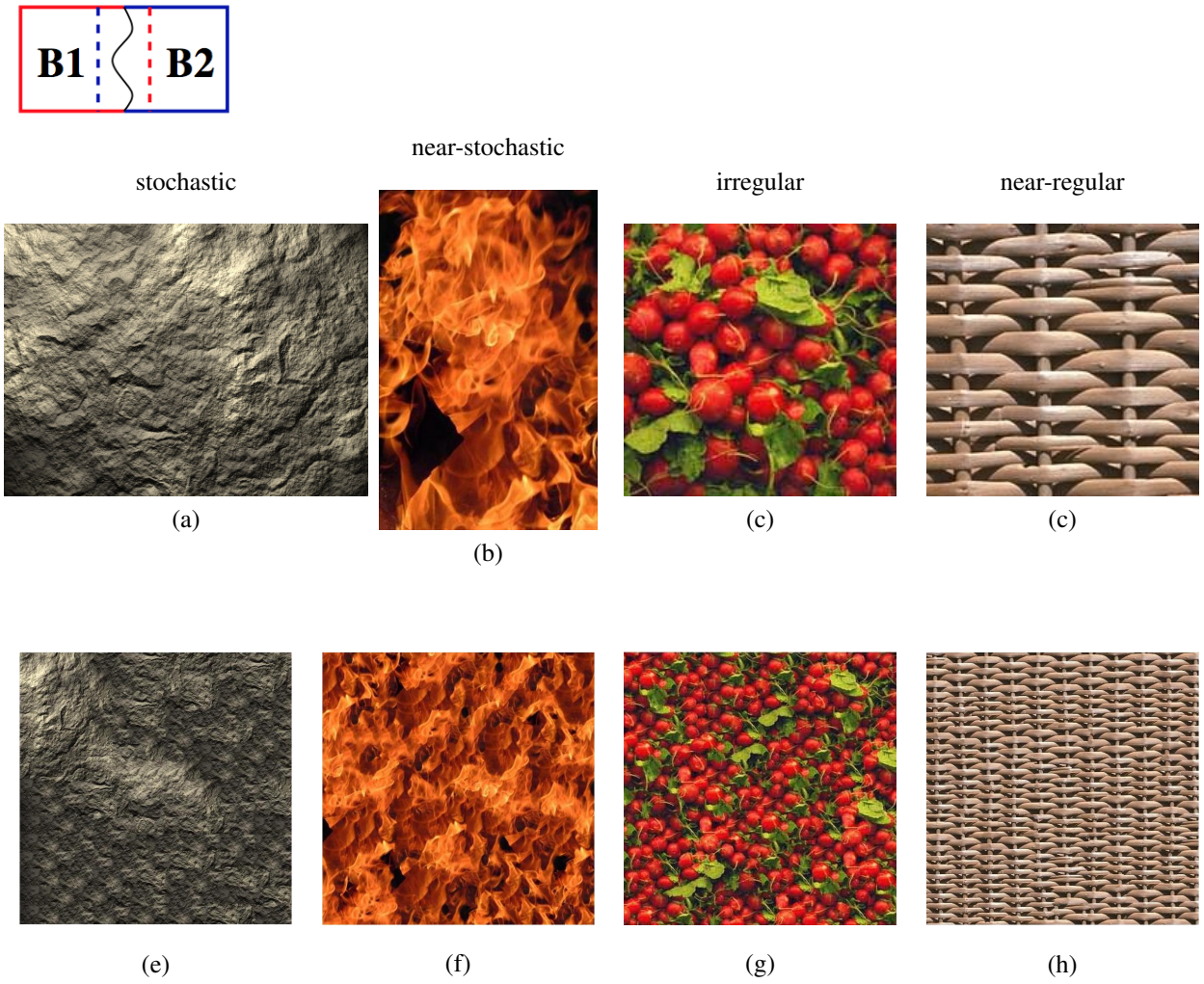


Figure 3: Image Quilting with Minimum Error Boundary Cut. Square blocks from the input texture are chosen to match and overlap with its neighbor, then cut along the minimum error boundary to synthesize new texture: (a) stochastic texture input, (b) near-stochastic texture input, (c) irregular texture input, (d) near-regular texture input, (e) stochastic texture synthesized; no edge artifact, (f) near-stochastic texture synthesized; no edge artifact, (g) irregular texture synthesized; no edge artifact, (h) near-regular texture synthesized; no edge artifact,

construct a general image quilting algorithm:
Image Quilting Algorithm:

1. Set block size, overlap size, and error tolerance.
2. For N Iterations:

    3. Initialize first block at the top-left corner with a randomly extracted source texture patch.

    4. Start placement in raster scan order in step size of (block size-overlap size)

        a. Search matching texture patches that satisfy the overlap error constraints(within error tolerance) in the top and left overlap region with the previously synthesized blocks. If multiple matches, choose one randomly.
        b. Perform Minimum Error Boundary Cut algorithm along the error surface constructed along the overlap region.
        c. Create a mask according to the labeled values(-1,0,1) from the minimum cut algorithm, apply mask to the match block, and paste onto the synthesized texture.

As given from Efros and Freeman[1], I have set the overlap region size to be 1/6 of the block size. L2 norm was used as the error measure, and the error tolerance was set to be less than 0.1 times the minimum error block(best matching block). Figure 3 shows texture synthesis results using the image quilting algorithm on various types of texture sources.

5 Texture Transfer

Texture Transferring is very similar to texture synthesis using image quilting algorithm. Image quilting only takes into account the overlap region matching error to find the best matching block because our objective is to generate texture, without any spatial target characteristics. However, if we want to *transfer* a source texture onto a target image, we must keep track the spatial properties of the target image, or the appearance information will be lost. To keep the spatial information, Efros and Freeman[1] integrates a concept of correspondence map with the image quilting algorithm. Because a correspondence map will contain 1-1 corresponding paired quantities between the source and the target image, we can use the correspondence as an additional matching constraint when computing overlap region errors. Then, the error will not only look for the best match between the textures but also look for the best match with the target appearance. The most obvious spatial information is the image intensity values (or luminance), and the luminance correspondence map of the source and the target can be easily obtained by converting our 3-channeled image into a grayscale image.

A modified error measuring term can be expressed as a weighted sum of the two matching constraints (textural, correspondence). If we have a weight $0 < \alpha < 1$, the modified error term is:
E_B: block overlap matching error (1st constraint),
E_C: correspondence map error (2nd constraint)

$$E = \alpha * E_B + (1 - \alpha) * E_C \tag{1}$$

By looking at this modified error measure term, we can see that as α increases, the algorithm matches with the textural property more, and as alpha decreases, the algorithm tries to match with the exact appearance of the target image. Because of the tradeoff in the error term, depending on the observation we cannot be sure that we will get a desired result in a single pass. Therefore, we take an iterative approach to satisfy a good textural result and appearance. For N iterations, In each iteration, we will decrease the block size by 1/3 and increase the textural weight α by re-assigning α at i^{th} iteration to be:

$$\alpha_i = 0.8 * (i - 1/N - 1) + 0.1 \tag{2}$$

However, my iterative approach was modified from Efros and Freemans, in that when I use $N = 2$ as the total number of iterations, I limit the α update to happen only after the first iteration, thereby preventing the α from increasing abruptly from 0.1 to 0.9, as if it were to be implemented as directed in the paper. In my approach, α is given a value of 0.6 in the first iteration, and then increases to 0.9 using the α update equation in the second iteration. Of course, if I were to use $N > 2$ as the number of iterations, then the initial α must be set to be a much lower value than 0.6, to ensure that α increases consistently through iterations.

Other than the α update, the iterative approach follows Efros and Freemans, where our first matching constraint in our error computation is modified so that it also takes into the account of the overlap region from an already synthesized image at the particular position from the previous iteration, and computes a L2 norm between the current overlap region and the previous iterations overlap region. Then, finally, we can re-modify the error equation (1) to take into account the new error term into the 1st constraint:
E_{prev}: overlap matching error with previously synthesized image (1st constraint)

$$E = \alpha * (E_B + E_{prev}) + (1 - \alpha) * E_C \tag{3}$$

6 Experimental Results

6.1 Image Quilting

As seen in Figure 3, the *image quilting* algorithm performs remarkably well on wide range of input texture sources. In particular, it performs well on irregular to structured image without producing any edge artifacts, whereas previous works only performed well on stochastic texture input. Figure 4 shows the comparison of the synthesized texture output between previously proposed synthesis algorithms and the *image quilting* algorithm. The only shortcoming of the *image quilting* algorithm is that if the given source texture is too repetitive without much variability(such as the berry image), then it can produce a repetitive image.

6.2 Texture Transfer

Results of Texture Transfer can be seen in Figure 5 and Figure 6, where Figures 5(a) and 5(b) takes in a near-repetitive texture source input(wool fabric, and yogurt) and transfers it onto the famous David Bust sculptures. On the other hand, Figures 6(a) and 6(b) takes in a rather irregular texture source input(Vincent Van Gogh's *The*

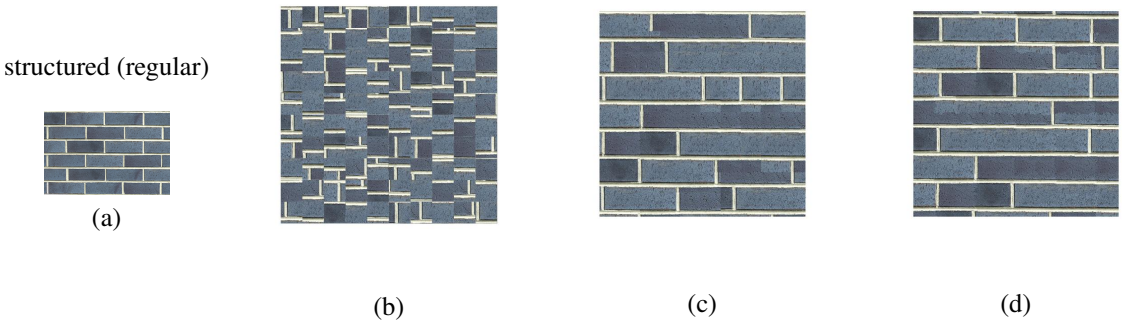


Figure 4: Comparison of 3 texture synthesis methods on structured textures: (a) structured texture input, (b) Random Placement (c) Matching Overlapping Regions Placement, (d) Image Quilting; match overlap regions and cut along minimum error boundary

Starry Night, and Claude Monet's *The Houses of Parliament*) and transfers the painting's texture onto a real-life scenery. Since the Texture Transfer algorithm only considers the luminance of both input texture source and the target image, the specific colors being transferred cannot be controlled and it only depends on the pixel intensity values. Using famous paintings as input texture sources does not produce as good of a transferred result as using near-repetitive texture sources as a texture source. This is because the textural properties of famous paintings lie within the brush strokes of acrylic painting, and a better contoured close-up image of those paintings would be needed in order to correctly extract textural properties.

7 Conclusion

In this report, I have investigated the paper "Image Quilting for Texture Synthesis and Transfer" by Efros and Freeman[1]. I have started out with a naive approach to synthesize texture by randomly sampling patches from source and placing them randomly onto the output image. Then, I studied the concept of overlapping blocks and instead of random placement, finding the best matching neighbor block under constraint through a search process. Finally, I've implemented the minimum error boundary cut algorithm using dynamic programming, and integrated with the previous best-match method to implement the *image quilting* algorithm as proposed in the paper. The algorithm was not conceptually difficult to implement, and it produced remarkable texture synthesis results very fast. Finally, I've extended the *image quilting* algorithm to a *texture transfer* algorithm as pointed by Efros and Freeman[1], and modified the initialization process slightly to produce better texture transfer results.

References

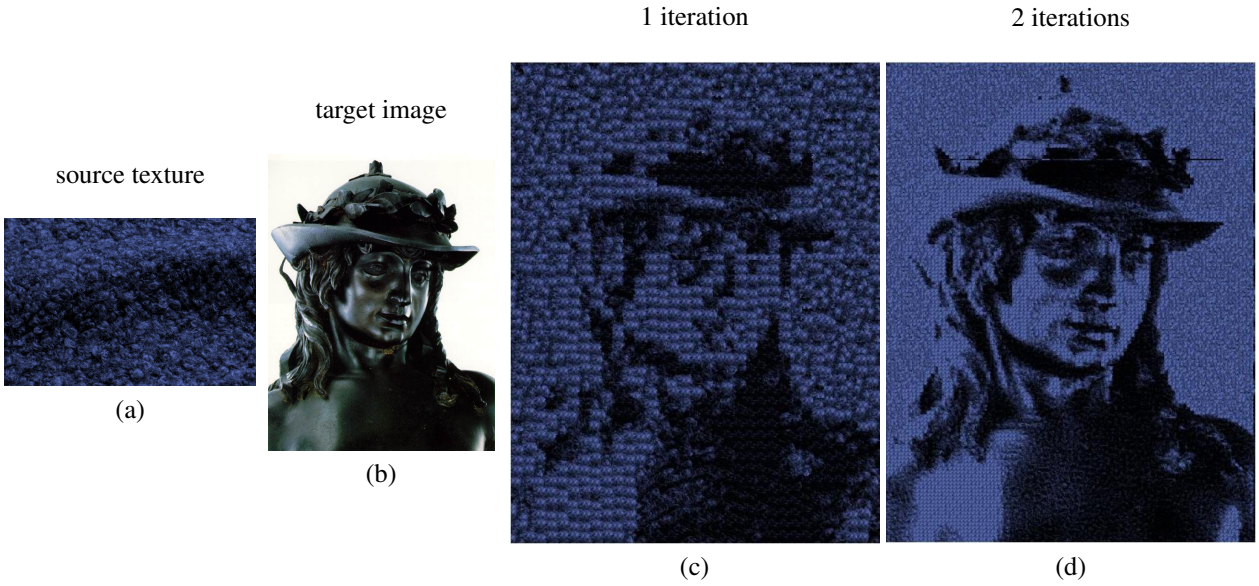
[1] Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. Proceedings of SIGGRAPH 2001. Computer Graphics Proceedings, Annual Conference Series. pages 341-346. August 2001.

[2] Bela Julesz. Visual pattern discrimination. *IRE Transactions on Information Theory*. 8(2):84-92, 1962.

[3] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH 95*, pages 229-238, 1995.

[4] Y. Xu, B. Guo, and H.Y. Shum Chaos mosaic: Fast and memory efficient texture synthesis. Technical Report MSR-TR-2000-32, Microsoft Research, April 2000.

[5] Alexei A. Efros and T. K. Leung Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision*, pages 1033-1038, Corfu, Greece, September 1999.



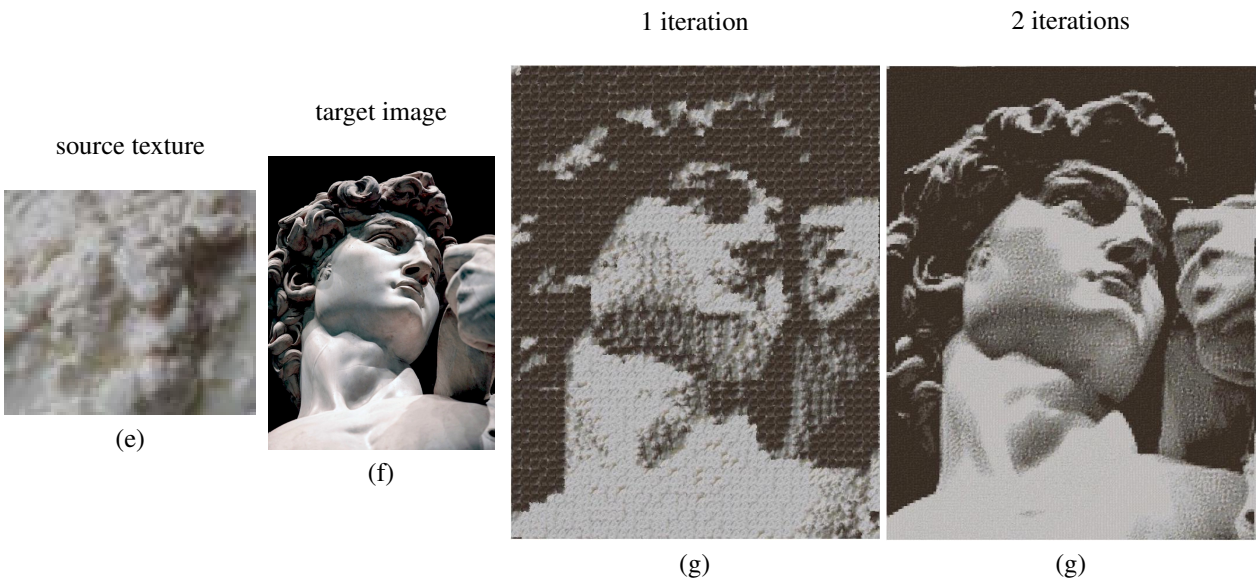
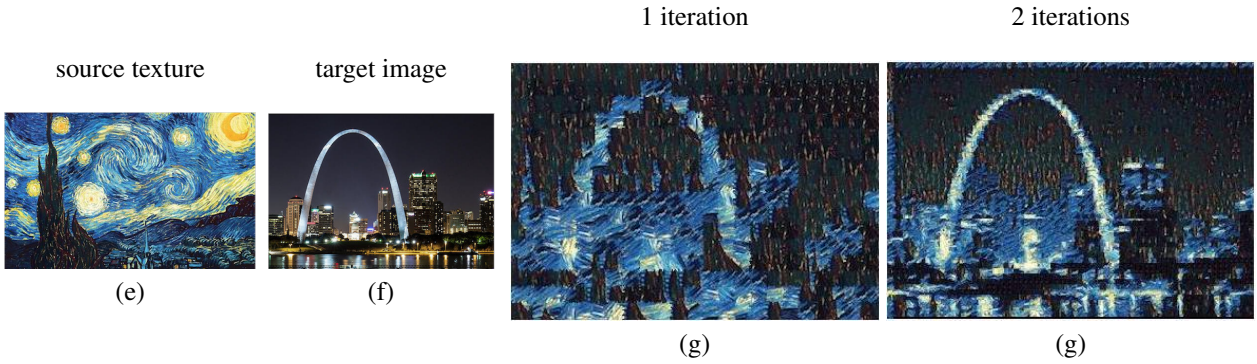


Figure 5: Near-repetitive texture source input(wool fabric, and yogurt) and transfers it onto the famous David Bust sculptures.



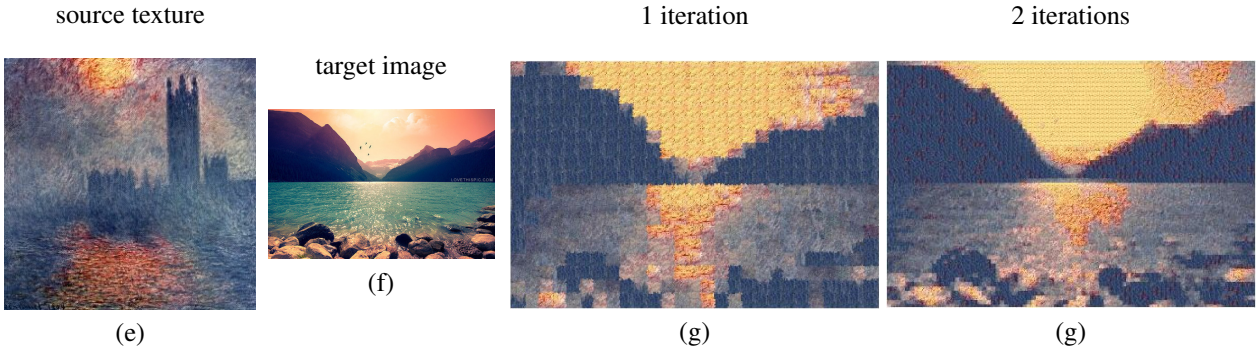


Figure 6: Irregular texture source input(Vincent Van Gogh's *The Starry Night*, and Claude Monet's *The Houses of Parliament*). Paintings' textures are transferred onto a real-life scenery.