Image Quilting for Texture Synthesis

Implementing Techniques from (Efros & Freeman, 2001)

Sam Griesemer SAMGRIESEMER@WUSTL.EDU
Washington University in St. Louis

Abstract

Texture synthesis is a process by which large images are constructed from smaller texture samples. In this project we implement multiple techniques used in (Efros & Freeman, 2001) as simple yet effective improvements over basic tiling techniques. We outline these methods in detail and took a closer look at how and why they work. Finally, we analyze these methods and their performance on textures of varying complexity, as well as their sensitivity across a number of settings.

1. Introduction

Image synthesis encompasses a broad class of algorithms that are capable of creating new images. Many techniques address specific, constrained scenarios (e.g. texture synthesis), while others address larger, more general problems (e.g. synthesizing high-res natural images). These approaches vary wildly, from simple random tiling to sophisticated generative models like GANs. In this project we focus on texture synthesis, the process of producing from a small texture sample a larger, perceptually similar image. There are many techniques for approaching such a problem, each with varying degrees of success in terms of output quality and efficiency. For example, simpler methods are often more efficient but suffer from poor output quality due to strong assumptions about the nature of textures. More complex methods often do a better job at producing high-quality outputs (thanks to more flexible assumptions and model fitting), but can be very inefficient and thus difficult to use in practice. As a result, the primary challenge is finding a texture synthesis algorithm that makes an effective tradeoff between efficiency and output quality.

Texture synthesis is an important problem with a number of practical applications across computer vision. These include texture mapping, inpainting, image denoising, and more. Texture mapping may be considered the canonical application of texture synthesis, whereby a texture sample is mapped to a much larger area while retaining similar visual qualities. Such a process could also be used to "cover up" or remove objects from images using the surrounding texture context. This example overlaps with the goal of image inpainting, where damaged or missing parts of an image are restored or "filled in" using nearby texture context.

For this project we explore three texture synthesis methods discussed in (Efros & Freeman, 2001). These methods include simple random patch selection, a constrained overlapping patch search, and the so-called "minimum error boundary cut" method introduced by the authors. We seek to implement each of these methods and replicate the paper's texture synthesis results. We then describe each of the methods in detail, and explore their performance in a variety of situations. As a result, the primary goal of this project is to provide an efficient, working implementation of the paper's proposed methods, expand on the details of these algorithms in an intuitive way, and better understand the situations in which these methods perform successfully.

2. Related Work

Due to its implications and practical uses, texture synthesis has received a fair amount of attention over the past few decades. As a result, a wide variety of algorithms and approaches have been published in this time. There are far too many to address here, but a few notable classes of approaches such as physical simulations are worth mentioning. It is no surprise that surface textures can be synthesized using physical simulations. Although somewhat limited in the kinds of textures that can be represented, patterns such as fur, skin, and scales can be modeled using methods like cellular texturing (Worley, 1996). Addi-

tionally, the common weathering effects often seen on real life materials can be reproduced using computer simulation as seen in (J. Dorsey & Pedersen, 1999).

Additionally, we observe work related to the main paper of interest. This includes efficiency improvements introduced after the papers release such as (Wei & Levoy, 2000) and (Liang et al., 2001), along with the related prior work from the authors in (Efros & Leung., 1999).

3. Quilting

The process of stitching together texture patches to form an output image is referred to as "image quilting" in the original paper. The image quilting algorithm introduced in (Efros & Freeman, 2001) is the result of multiple successive improvements to a very simple stochastic tiling method. Here we build up from this simple technique and explore the results of the increasingly sophisticated modifications as introduced by the authors.

3.1. Naïve Tiling Method

The first and most basic quilting algorithm discussed in the paper establishes a performance baseline that we should hope to exceed with successive improvements. This naive method takes an input texture image, along with a block size B, and stitches together patches of size $B \times B$ drawn randomly from the source image. The diagram (Figure 1) below attempts to show this process visually:

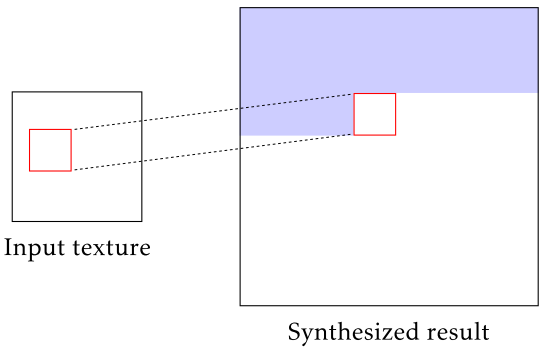


Figure 1. Diagram showing the random patch selection algorithm during synthesis. The red square represents a randomly selected patch of size $B \times B$ in the input texture on the left and its placement onto the synthesized output image on the right. The light blue region represents the portion of the output image that has already been synthesized.

We note from this figure that the sampled patch is placed directly into its own area in the output image; there is no overlap with the previously selected patches. This process continues in raster scan order until the output image has filled the desired size.

We now provide an example of the algorithm using one of the same textures seen in the paper (to allow for direct comparison):

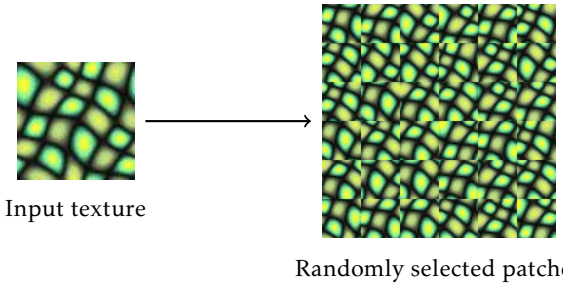


Figure 2. Random patch selection applied to the input texture (left) and the resulting image (right). The input texture is size 64×64, and the synthesized image is a 6×6 stitching of patches each sized 32×32.

Looking closely at the synthesized result, we can see distinct edges between most of the patches. As a result, the output is not a very natural or satisfying extension of the input texture. This, of course, is expected; there are no constraints in place to select "suitable" patches as the image is constructed. We now look to a slight modification of this method that yields significantly better results by enforcing a simple overlap constraint.

3.2. Overlapping Patches

We can significantly increase the quality of the synthesized image by searching the input texture for the "most suitable" patches during construction. The notion of a "suitable patch" refers to those patches which meet certain criteria according to some measure along a region of overlap between patches. What exactly is used as this measure is left undefined in the paper, however we believe the authors likely make use of squared error here as it's used elsewhere throughout the paper.

This simple "overlapping patches" method works as follows. Let S_B be the set of all possible patches of size $B \times B$ in the input texture, and let O be some fixed amount of pixel overlap to be used for selecting new patches. As we iterate over the output image in raster scan order, we fill it using the patch from S_B that minimizes the squared error along the region of overlap

at the current patch location. That is, if the overlap region in the output image is t_o, then we set the current output location to be the patch p such that

$$p = \underset{s \in S_B}{\operatorname{argmin}} (s_o - t_o)^2,$$

where s_o is the region of overlap in the patch $s \in S_B$. The original paper defines $e = (s_o - t_o)^2$ as the "error surface" of the patch s and the output image at the current location. Figure 3 below shows this process visually.

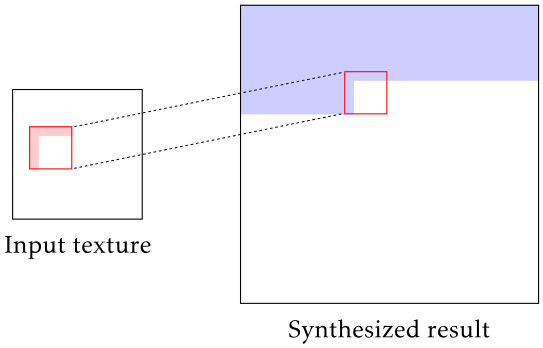
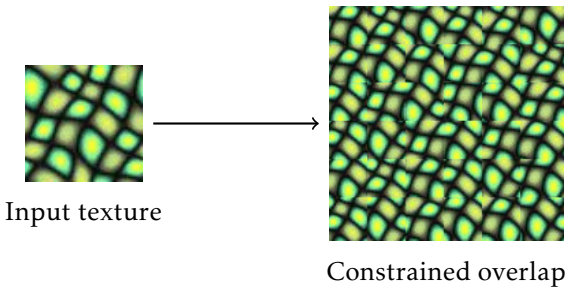


Figure 3. Diagram showing the overlapping patch selection during synthesis. The red square represents the patch $p \in S_B$ with the lowest overlap error (as defined above). On the right, we can see the location where the selected patch is to be placed in the synthesized image. The blue region inside of the red square here represents the overlap region t_o of the currently synthesized image. On the left, the light red region inside of the patch is the overlap region p_o for the patch p. The squared difference between the pixels in these two regions is defined to be the error surface, as seen above.

Now we run this algorithm on the same input texture as before:



Here we can easily see that the output quality has improved; the edges between patches are much more faint, and there are many places where the texture across the seams appears to line up. This is due to the fact we've explicitly chosen patches that work well within some overlapping region, and should expect that patches share some structure with the edges of neighboring patches. These results are very similar to those seen in the original paper for this simple overlap method. However the results still leave room for improvement; careful inspection makes it easy to spot separate patches in the image. The fact that patches must have straight edges prevents the successful alignment of features across seams. We will now look at the final improvement introduced by the authors that improves output quality even more.

3.3. Minimum Error Boundary Cut

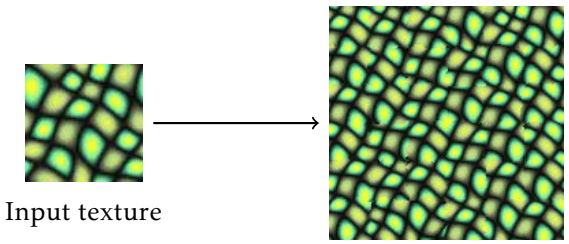
Making yet another simple modification to the previous algorithm, we arrive at the paper's primary texture synthesis contribution. This algorithm allows patches to be "cut" before they are stitched together, meaning patches are no longer required to have a straight boundary. As a result, patches can better align with each other and stitch features together in a more flexible and natural way.

The authors refer to the optimal boundary cut on a patch's edge as the "minimum error boundary cut". This cut is defined along the error surface e (presented in Section 3.2), and can be computed by finding the cumulative minimum error E for all possible paths through this surface:

$$E_{i,j} = e_{i,j} + \min(E_{i-1,j-1}, E_{i-1,j}, E_{i-1,j+1})$$

E is a matrix of values built up by traversing the error surface e and tracking the minimal error path at each location. In our implementation, we used dynamic programming to efficiently build up E as a memoization table (as is recommended in the paper). Once this computation is complete, the last row of E holds the cumulative costs for the minimal error paths ending at each location in the row. It is then trivial to trace back through E and recover the overall minimal error cut through the error surface.

Once again, we run this method on the same input texture for comparison: Here we observe improved output quality once again. The resulting texture appears to be quite consistent throughout the image, and there are no longer any sharp, salient patch edges. Because the algorithm is allowed to have curved patch edges, features are now able to fit together in a way that is more consistent with the structure present in the input texture. This isn't to say that the output is entirely free of artifacts; looking closely at the output image, we can spot regions where color or shape doesn't quite align properly. Nevertheless, for this



particular texture input, the image quilting method proposed by (Efros & Freeman, 2001) performs significantly better than the naive random approach introduced in Section 3.1.

4. Experiments

We now perform a number of experiments with the image quilting method introduced in Section 3.3. This includes further verification that our implementation matches that of the original paper, synthesis results on a number of different textures, and exploration into potentially troublesome situations for the algorithm.

4.1. Implementation Correctness

We first provide a few examples using the same textures as the original paper to verify our implementation's correctness. Although not definitive, this comparison allows us to check for possible inconsistencies across results. Note that our algorithm was run to produce output images of the same size as those seen in the original paper. We also use the same reported settings: an overlap O 1/6 the size of the block size B, and an error tolerance of 0.1 for selected matching patches.

4.2. Texture Complexity

An interesting factor of texture synthesis is the level of randomness present in the input texture. If the texture was perfectly deterministic and repetitive, then a successful image quilting algorithm would require no more than the equivalent of copy and paste where texture repeats. However, for more stochastic textures, there can be inconsistent variations throughout the image, making it difficult to scale in a natural way. Here we explore the degree to which the proposed method can handle varying degrees of stochasticity and/or complexity in the input texture. The results seen in Figure 4 are what we might classify as "medium" complexity; they undoubtedly have a de-

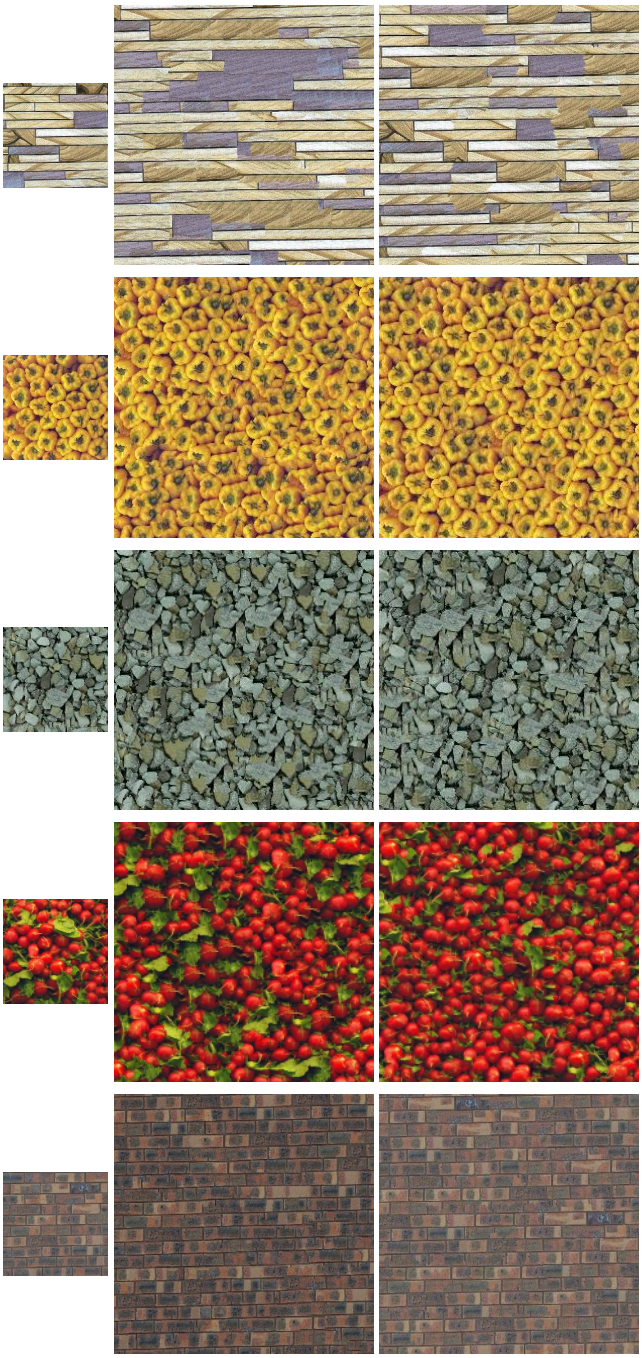


Figure 4. Result comparison between our implementation and the original paper using the image quilting algorithm. **Left**: input texture, **Center**: our implementation, **Right**: original paper

gree of repetition, but across the board all share some varying, non-trivial structure.

To test simple, deterministic textures we run the algorithm on a repetitive checkerboard pattern. The results, unsurprisingly, look as follows:

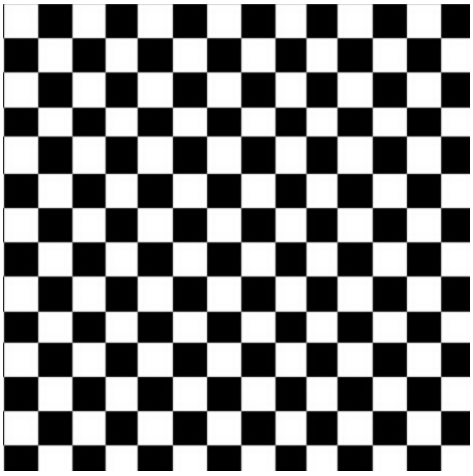


Figure 5. Simple checkered result

Here we see the algorithm succeeds in the expected way. It is a able to identify the simple, checkered pattern and repeat it consistently. This is the kind of behavior we would expect from even the most simple of texture synthesis algorithms.

For a more difficult pattern, we searched for textures that had a large degree of variability and inconsistent structure. We hoped this would reveal some potential weak points of the algorithm when faced with increasingly random textures. Our selected image and algorithm's performance look as follows:

Figure 6. Complex input texture

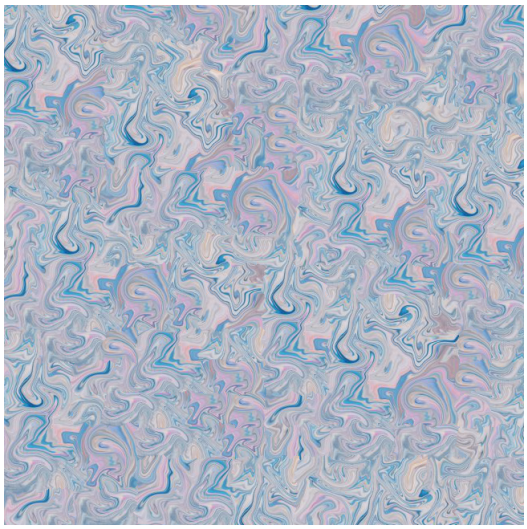


Figure 7. Complex texture synthesized result

Here we find the algorithm is ultimately able to come up with a reasonable quilting pattern for the seemingly complicated input texture. This is somewhat surprising, as there are a number of "medium-tier" complexity textures that appear to have some minor artifacts. We anticipated these inconsistenices would be magnified in this scenario. One possibility here is that the inherent curvy nature of the local structure in the image lends itself well to the algorithm's flexible boundaries, making it fairly easy to hide the "stitches" across patches.

5. Conclusion

In this project we implemented and explored three texture synthesis methods discussed in (Efros & Freeman, 2001). Among these approaches include the "minimum error boundary cut" method introduced by the authors, on which we focus most of our analyses. We first described this method in detail, as a result of multiple simple improvements to a naive random tiling method. We then implemented this algorithm and compared our results to those seen in the original paper. We then performed a few additional experiments to better understand how the algorithm performs on textures of varying complexity. Here we found that the algorithm did surprisingly well at what appeared to be a sufficiently stochastic and difficult texture to extend.

References

Efros, A. and Leung., T., 1999. Texture synthesis by non-parametric sampling.

Efros, Alexei A. and Freeman, William T., 2001. Image quilting for texture synthesis and transfer.

J. Dorsey, A. Edelman, J. Legakis H. W. Jensen and Pedersen, H. K., 1999. Modeling and rendering of weathered stone.

Liang, Lin, Liu, Ce, Xu, Ying-Qing, Guo, Baining, and Shum, Heung-Yeung, 2001. Real-time texture synthesis by patch-based sampling.

Wei, Li-Yi and Levoy, Marc, 2000. Fast texture synthesis using tree-structured vector quantization.

Worley, S. P., 1996. A cellular texture basis function.