Image Quilting for Texture Synthesis and Transfer

Alexei A. Efros[1,2] William T. Freeman[2]

[1]University of California, Berkeley [2]Mitsubishi Electric Research Laboratories

Abstract

We present a simple image-based method of generating novel visual appearance in which a new image is synthesized by stitching together small patches of existing images. We call this process *image quilting*. First, we use quilting as a fast and very simple texture synthesis algorithm which produces surprisingly good results for a wide range of textures. Second, we extend the algorithm to perform texture transfer – rendering an object with a texture taken from a different object. More generally, we demonstrate how an image can be re-rendered in the style of a different image. The method works directly on the images and does not require 3D information.

Keywords: Texture Synthesis, Texture Mapping, Image-based Rendering

1 Introduction

In the past decade computer graphics experienced a wave of activity in the area of image-based rendering as researchers explored the idea of capturing samples of the real world as images and using them to synthesize novel views rather than recreating the entire physical world from scratch. This, in turn, fueled interest in image-based texture synthesis algorithms. Such an algorithm should be able to take a sample of texture and generate an unlimited amount of image data which, while not exactly like the original, will be perceived by humans to be *the same texture*. Furthermore, it would be useful to be able to transfer texture from one object to anther (e.g. the ability to cut and paste material properties on arbitrary objects).

In this paper we present an extremely simple algorithm to address the texture synthesis problem. The main idea is to synthesize new texture by taking patches of existing texture and stitching them together in a consistent way. We then present a simple generalization of the method that can be used for texture transfer.

1.1 Previous Work

Texture analysis and synthesis has had a long history in psychology, statistics and computer vision. In 1950 Gibson pointed out the importance of texture for visual perception [8], but it was the pioneering work of Bela Julesz on texture discrimination [12] that paved the way for the development of the field. Julesz suggested that two texture images will be perceived by human observers to be the same if some appropriate statistics of these images match. This suggests that the two main tasks in statistical texture synthesis are (1) picking the right set of statistics to match, (2) finding an algorithm that matches them.

Motivated by psychophysical and computational models of human texture discrimination [2, 14], Heeger and Bergen [10] proposed to analyze texture in terms of histograms of filter responses at multiple scales and orientations. Matching these histograms iteratively was sufficient to produce impressive synthesis results for stochastic textures (see [22] for a theoretical justification). However, since the histograms measure marginal, not joint, statistics they do not capture important relationships across scales and orientations, thus the algorithm fails for more structured textures. By also matching these pairwise statistics, Portilla and Simoncelli [17] were able to substantially improve synthesis results for structured textures at the cost of a more complicated optimization procedure.

In the above approaches, texture is synthesized by taking a random noise image and *coercing* it to have the same relevant statistics as in the input image. An opposite approach is to start with an input image and *randomize* it in such a way that only the statistics to be matched are preserved. De Bonet [3] scrambles the input in a coarse-to-fine fashion, preserving the conditional distribution of filter outputs over multiple scales (jets). Xu el.al. [21], inspired by the Clone Tool in PHOTOSHOP, propose a much simpler approach yielding similar or better results. The idea is to take random square blocks from the input texture and place them randomly onto the synthesized texture (with alpha blending to avoid edge artifacts).

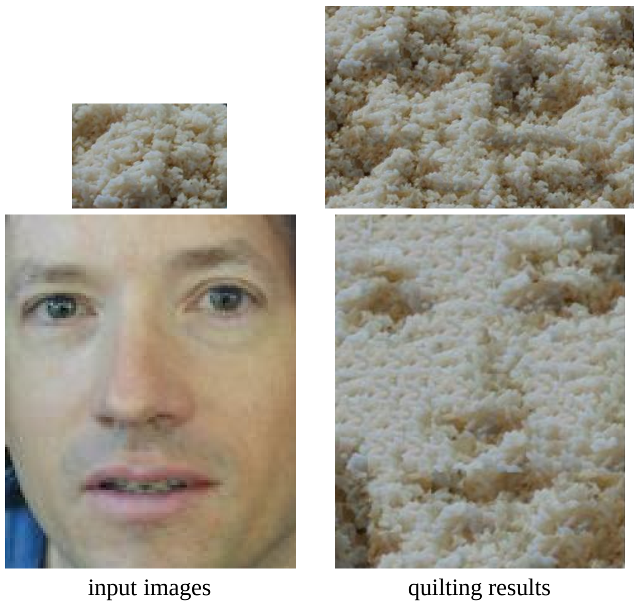
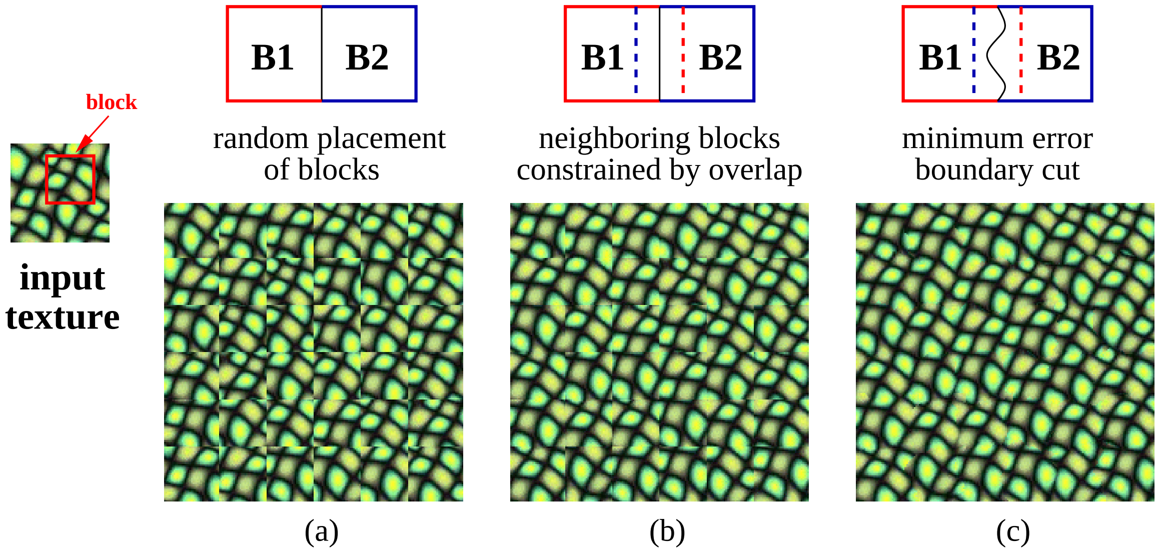


Figure 1: Demonstration of quilting for texture synthesis and texture transfer. Using the rice texture image (upper left), we can synthesize more such texture (upper right). We can also transfer the rice texture onto another image (lower left) for a strikingly different result.

[1]Computer Science Division, UC Berkeley, Berkeley, CA 94720, USA.
[2]MERL, 201 Broadway, Cambridge, MA 02139, USA.



B1
B2
block
input texture
random placement of blocks
neighboring blocks constrained by overlap
minimum error boundary cut
(a)
(b)
(c)

Figure 2: Quilting texture. Square blocks from the input texture are patched together to synthesize a new texture sample: (a) blocks are chosen randomly (similar to [21, 18]), (b) the blocks overlap and each new block is chosen so as to “agree” with its neighbors in the region of overlap, (c) to reduce blockiness the boundary between blocks is computed as a minimum cost path through the error surface at the overlap.

The statistics being preserved here are simply the arrangement of pixels within each block. While this technique will fail for highly structured patterns (e.g. a chess board) due to boundary inconsistencies, for many stochastic textures it works remarkably well. A related method was successfully used by Praun et.al. [18] for semi-automatic texturing of non-developable objects.

Enforcing statistics globally is a difficult task and none of the above algorithms provide a completely satisfactory solution. A easier problem is to enforce statistics locally, one pixel at a time. Efros and Leung [6] developed a simple method of “growing” texture using non-parametric sampling. The conditional distribution of each pixel given all its neighbors synthesized so far is estimated by searching the sample image and finding all similar neighborhoods. (We have recently learned that a nearly identical algorithm was proposed in 1981 by Garber [7] but discarded due to its then computational intractability.) The algorithm produces good results for a wide range of textures, but is excruciatingly slow (a full search of the input image is required to synthesize every pixel!). Several researchers have proposed optimizations to the basic method including Wei and Levoy [20] (based on earlier work by Popat and Picard [16]), Harrison [9], and Ashikhmin [1]. However, all these improvements still operate within the greedy single-pixel-at-a-time paradigm and as such are susceptible to falling into the wrong part of the search space and starting to “grow garbage” [6].

Methods have been developed in particular rendering domains which capture the spirit of our goals in texture transfer. Our goal is like that of work in non-photorealistic rendering (e.g. [4, 19, 15]). A key distinction is that we seek to characterize the output rendering style by sampling from the real world. This allows for a richness of rendering styles, characterized by samples from photographs or drawings.

A number of papers to be published this year, all developed independently, are closely related to our work. The idea of texture transfer based on variations of [6] has been proposed by several authors [9, 1, 11] (in particular, see the elegant paper by Hertzmann et.al. [11] in these proceedings). Liang et.al. [13] propose a real-time patch-based texture synthesis method very similar to ours. The reader is urged to review these works for a more complete picture of the field.

1.2 Motivation

One curious fact about one-pixel-at-a-time synthesis algorithms such as Efros and Leung [6] is that for most complex textures very few pixels actually have a choice of values that can be assigned to them. That is, during the synthesis process most pixels have their values totally determined by what has been synthesized so far. As a simple example, let us take a pattern of circles on a plane. Once the algorithm has started synthesizing a particular circle, all the remaining pixels of that circle (plus some surrounding ones) are completely determined! In this extreme case, the circle would be called the texture element (*texel*), but this same effect persists to a lesser extent even when the texture is more stochastic and there are no obvious texels. This means that a lot of searching work is waisted on pixels that already “know their fate”. It seems then, that the unit of synthesis should be something more than a single pixel, a “patch” perhaps. Then the process of texture synthesis would be akin to putting together a jigsaw puzzle, quilting together the patches, making sure they all fit together. Determining precisely what are the patches for a given texture and how they are put together is still an open problem. Here we will present an very naive version of stitching together patches of texture to form the output image. We call this method “image quilting”.

2 Quilting

In this section we will develop our patch-based texture synthesis procedure. Let us define the unit of synthesis B_i to be a square block of user-specified size from the set S_B of all such overlapping blocks in the input texture image. To synthesize a new texture image, as a first step let us simply tile it with blocks taken randomly from S_B. The result shown on Figure 2(a) already looks somewhat reasonable and for some textures will perform no worse than many previous complicated algorithms as demonstrated by [21, 18]. Still, the result is not satisfying, for no matter how much smoothing is done across the edges, for most structured textures it will be quite obvious that the blocks do not match.

As the next step, let us introduce some overlap in the placement of blocks onto the new image. Now, instead of picking a random block, we will search S_B for such a block that by some measure

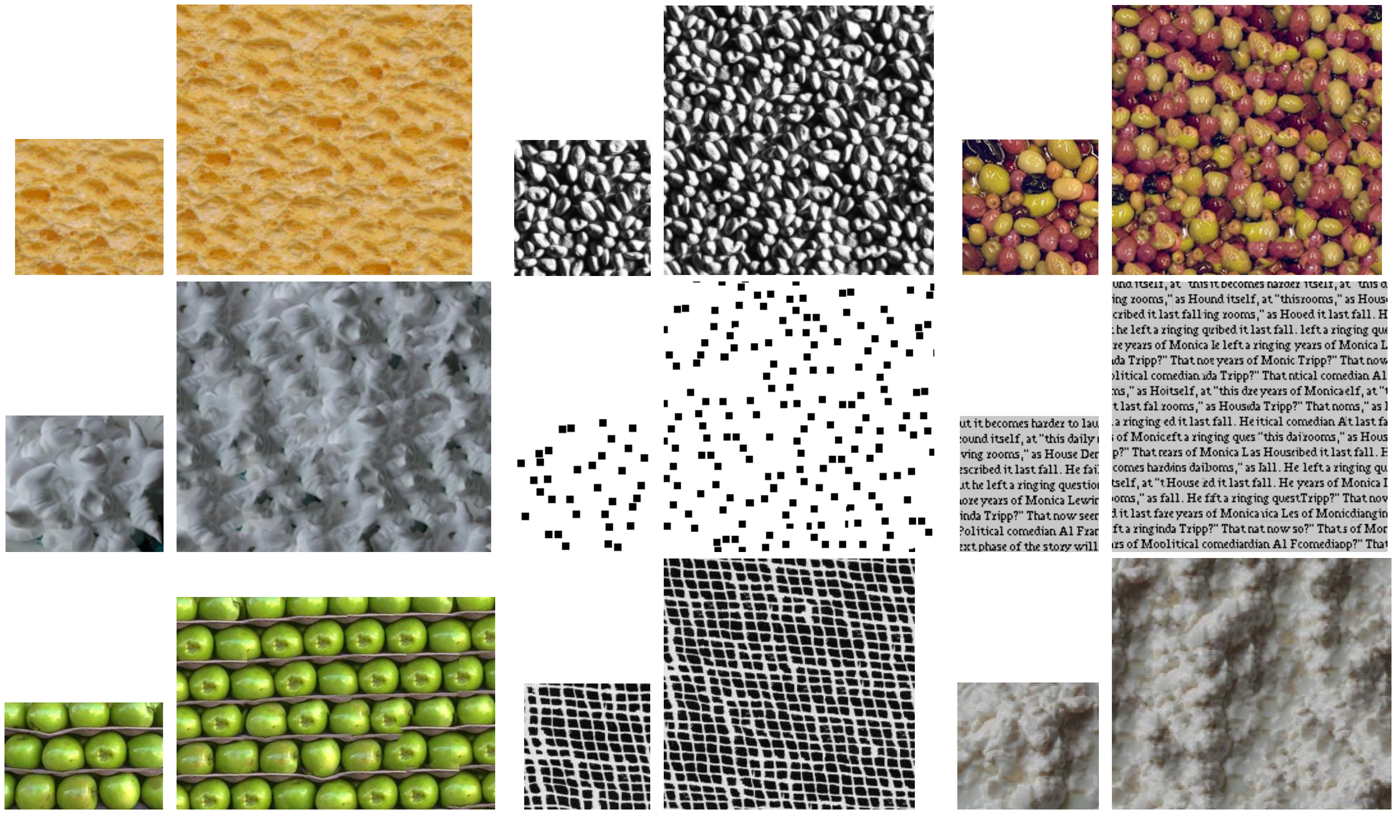


Figure 3: Image quilting synthesis results for a wide range of textures. The resulting texture (right side of each pair) is synthesized at twice the size of the original (left).

agrees with its neighbors along the region of overlap. Figure 2(b) shows a clear improvement in the structure of the resulting texture, however the edges between the blocks are still quite noticeable. Once again, smoothing across the edges will lessen this problem but we will attempt to solve it in a more principled way.

Finally, we will let the blocks have ragged edges which will allow them to better approximate the features in the texture. Now, before placing a chosen block into the texture we will look at the error in the overlap region between it and the other blocks. We find a minimum cost path through that error surface and declare that to be the boundary of the new block. Figure 2(c) shows the results of this simple modification.

2.1 Minimum Error Boundary Cut

We want to make the cut between two overlapping blocks on the pixels where the two textures match best (that is, where the overlap error is low). This can easily be done with dynamic programming (Dijkstra's algorithm can also be used [5]).

The minimal cost path through the error surface is computed in the following manner. If B_1 and B_2 are two blocks that overlap along their vertical edge (Figure 2c) with the regions of overlap B_1^{ov} and B_2^{ov}, respectively, then the error surface is defined as $e = (B_1^{ov} - B_2^{ov})^2$. To find the minimal vertical cut through this surface we traverse e ($i = 2..N$) and compute the cumulative minimum error E for all paths:

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

In the end, the minimum value of the last row in E will indicate the end of the minimal vertical path though the surface and one can trace back and find the path of the best cut. Similar procedure can be applied to horizontal overlaps. When there is both a vertical and a horizontal overlap, the minimal paths meet in the middle and the overall minimum is chosen for the cut.

2.2 The Image Quilting Algorithm

The complete quilting algorithm is as follows:

- Go through the image to be synthesized in raster scan order in steps of one block (minus the overlap).
- For every location, search the input texture for a set of blocks that satisfy the overlap constraints (above and left) within some error tolerance. Randomly pick one such block.
- Compute the error surface between the newly chosen block and the old blocks at the overlap region. Find the minimum cost path along this surface and make that the boundary of the new block. Paste the block onto the texture. Repeat.

The size of the block is the only parameter controlled by the user and it depends on the properties of a given texture; the block must be big enough to capture the relevant structures in the texture, but small enough so that the interaction between these structures is left up to the algorithm.

In all of our experiments the width of the overlap edge (on one side) was 1/6 of the size of the block. The error was computed using the $L2$ norm on pixel values. The error tolerance was set to be within 0.1 times the error of the best matching block.

2.3 Synthesis Results

The results of the synthesis process for a wide range of input textures are shown on Figures 3 and 4. While the algorithm is particularly effective for semi-structured textures (which were always the

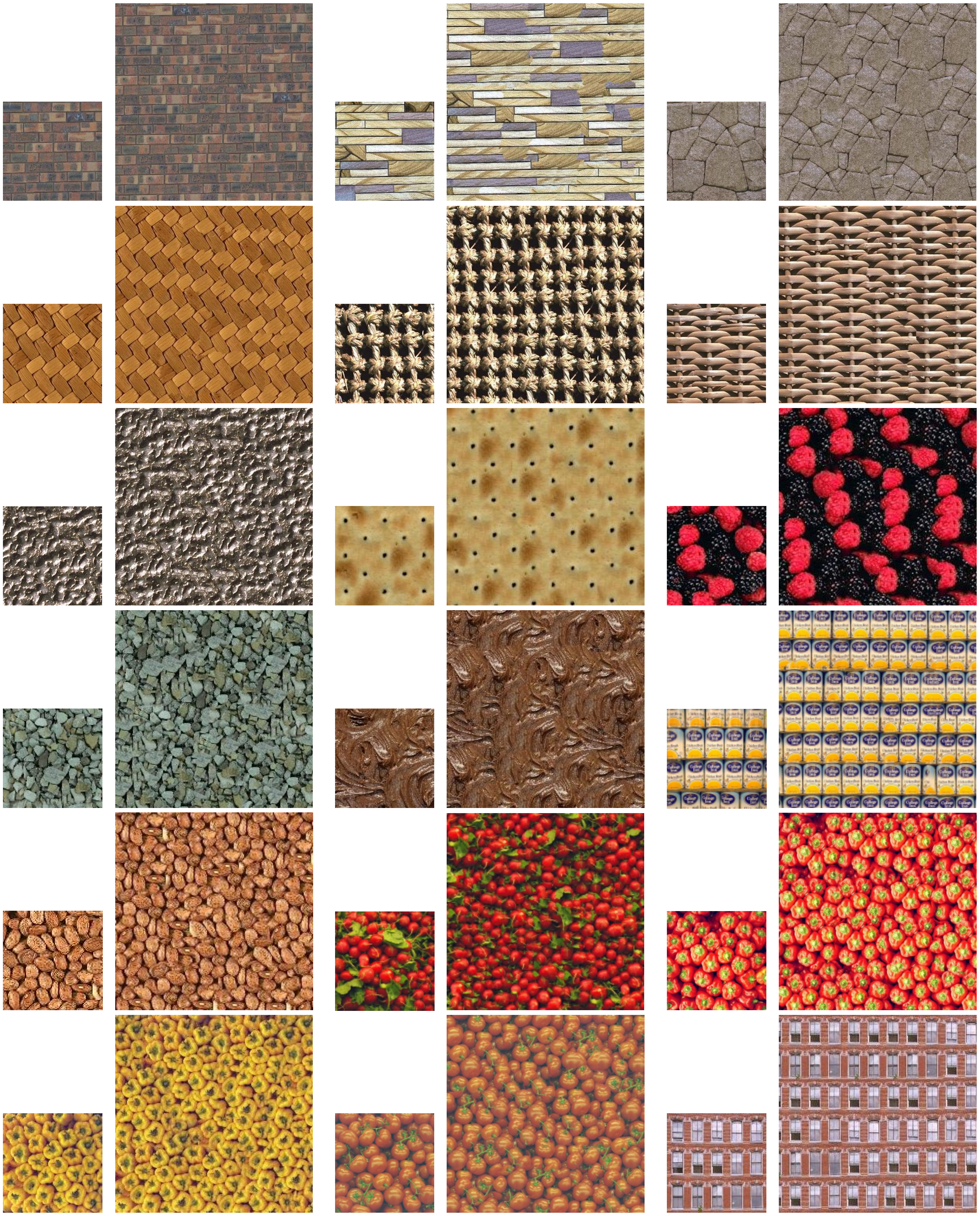


Figure 4: More image quilting synthesis results (for each pair, left is original, right is synthesized)

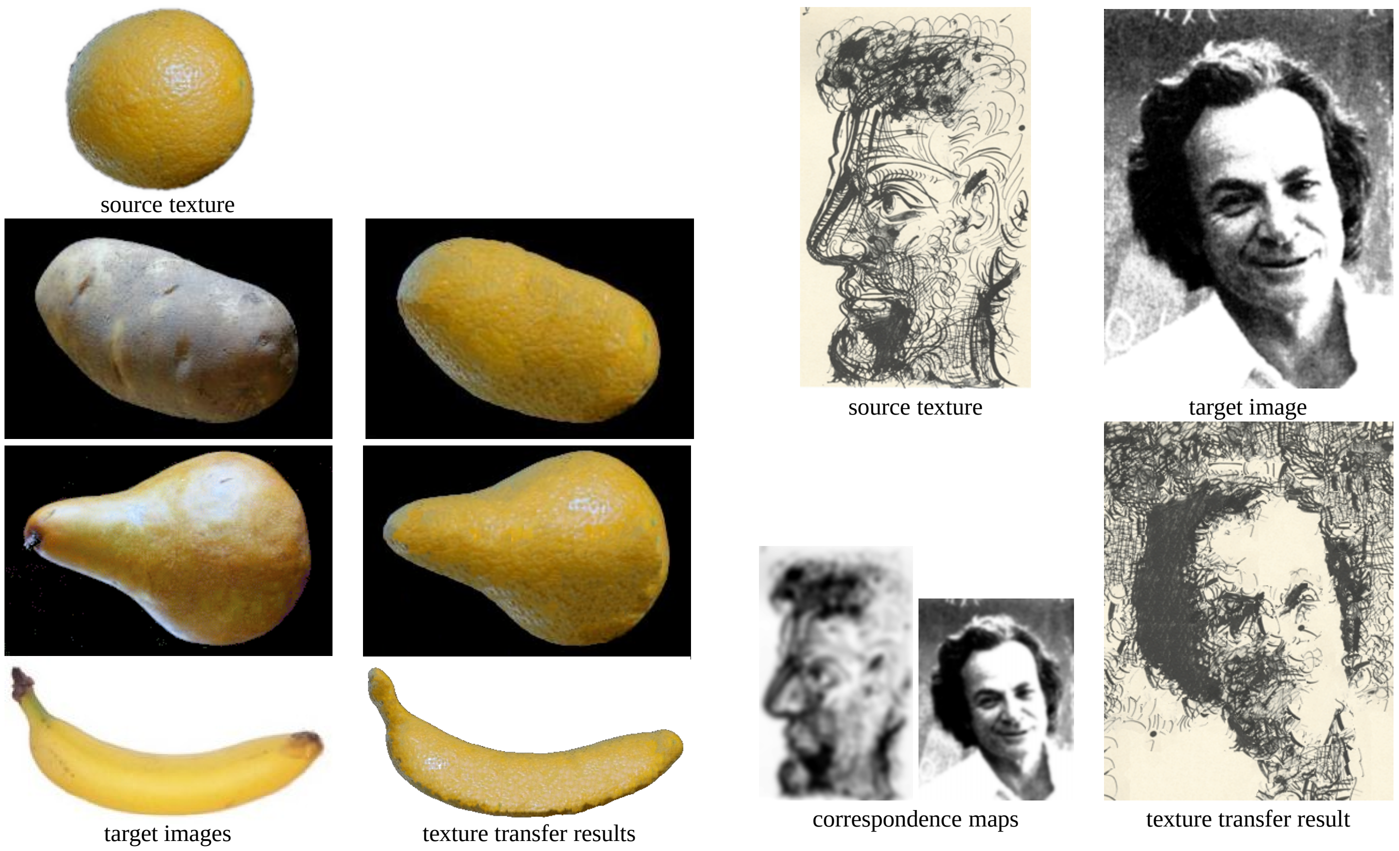


Figure 5: Texture transfer: here, we take the texture from the orange and the Picasso drawing and transfer it onto different objects. The result has the texture of the source image and the correspondence map values of the target image.

hardest for statistical texture synthesis), the performance is quite good on stochastic textures as well. The two most typical problems are excessive repetition (e.g. the berries image), and mismatched or distorted boundaries (e.g. the mutant olives image). Both are mostly due to the input texture not containing enough variability. Figure 6 shows a comparison of quilting with other texture synthesis algorithms.

The algorithm is not only trivial to implement but is also quite fast: the unoptimized MATLAB code used to generate these results ran for between 15 seconds and several minutes per image depending on the sizes of the input and output and the block size used. Because the constraint region is always the same it's very easy to optimize the search process without compromising the quality of the results (see also Liang et.al. [13] who report real-time performance using a very similar approach).

3 Texture Transfer

Because the image quilting algorithm selects output patches based on local image information, it is particularly well suited for *texture transfer*. We augment the synthesis algorithm by requiring that each patch satisfy a desired *correspondence map*, $\vec{C}$, as well as satisfy the texture synthesis requirements. The correspondence map is a spatial map of some corresponding quantity over both the texture source image and a controlling target image. That quantity could include image intensity, blurred image intensity, local image orientation angles, or other derived quantities.

An example of texture transfer is shown in Figure 1. Here, the correspondence map are the (luminance) image intensities of the man's face. That is, bright patches of face and bright patches of rice are defined to have a low correspondence error. The synthesized rice texture conforms to this second constraint, yielding a rendered image where the face image appears to be rendered in rice.

For texture transfer, image being synthesized must respect two independent constraints: (a) the output are legitimate, synthesized examples of the source texture, and (b) that the correspondence image mapping is respected. We modify the error term of the image quilting algorithm to be the weighted sum, α times the block overlap matching error plus $(1 - \alpha)$ times the squared error between the correspondence map pixels within the source texture block and those at the current target image position. The parameter α determines the tradeoff between the texture synthesis and the fidelity to the target image correspondence map.

Because of the added constraint, sometimes one synthesis pass through the image is not enough to produce a visually pleasing result. In such cases, we iterate over the synthesized image several times, reducing the block size with each iteration. The only change from the non-iterative version is that in satisfying the local texture constraint the blocks are matched not just with their neighbor blocks on the overlap regions, but also with whatever was synthesized at this block in the previous iteration. This iterative scheme works surprisingly well: it starts out using large blocks to roughly assign where everything will go and then uses smaller blocks to make sure the different textures fit well together. In our tests, we used $N = 3$ to $N = 5$ iterations, reducing the block size by a third each time, and setting α at the ith iteration to be $\alpha_i = 0.8 * \frac{i-1}{N-1} + 0.1$.

Our texture transfer method can be applied to render a photograph using the line drawing texture of a particular source drawing; or to transfer material surface texture onto a new image (see Figure 5). For the orange texture the correspondence maps are the source and target image luminance values; for Picasso the correspondence maps are the blurred luminance values.

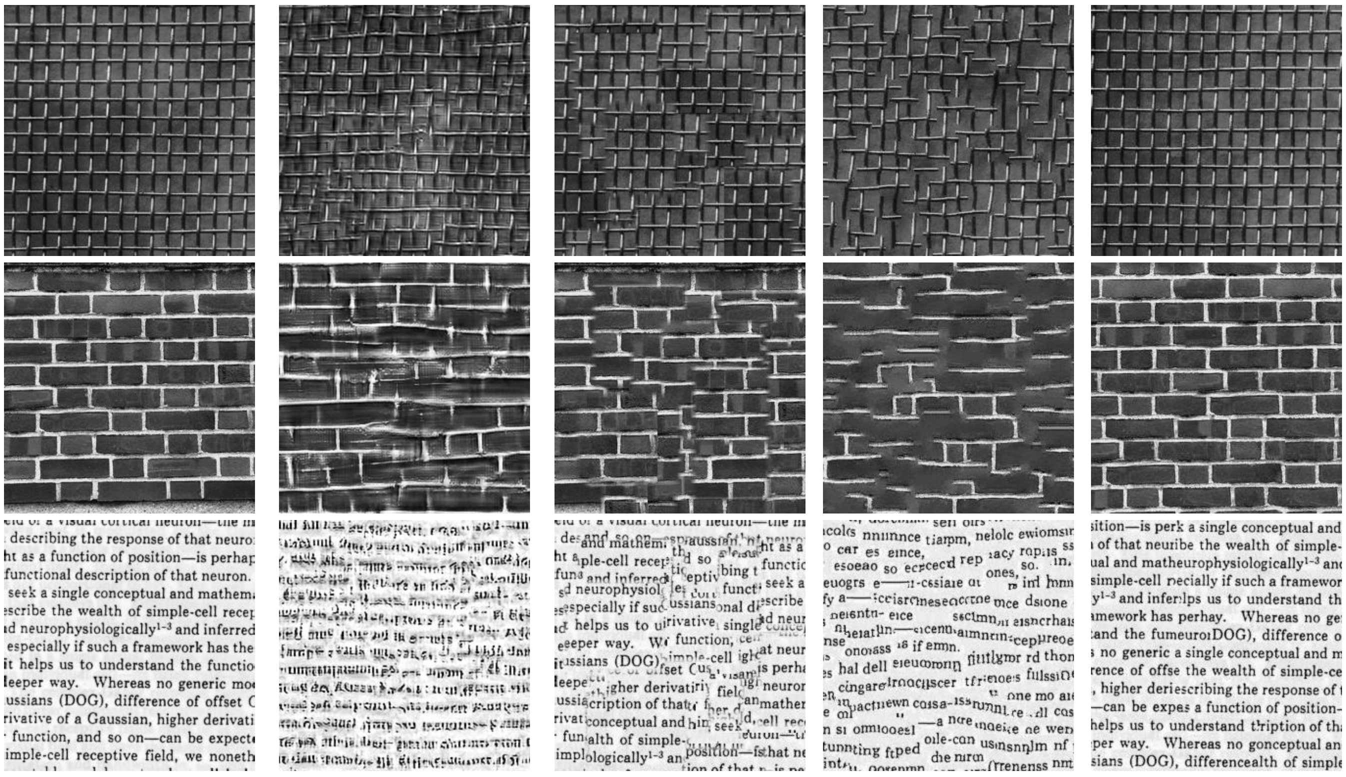


Figure 6: Comparison of various texture synthesis methods on structured textures. Our results are virtually the same as Efros & Leung [6] (not shown) but at a much smaller computational cost.

4 Conclusion

In this paper we have introduced *image quilting*, a method of synthesizing a new image by stitching together small patches of existing images. Despite its simplicity, this method works remarkably well when applied to texture synthesis, producing results that are equal or better than the Efros & Leung family of algorithms but with improved stability (less chance of "growing garbage") and at a fraction of the computational cost. We have also extended our method to texture transfer in a general setting with some very promising results.

Acknowledgements: We would like to thank Alex Berg, Rahul Bhotika and Jitendra Malik for their help and encouragement. This research was conducted while AE was a summer Research Intern at MERL. The photograph of Richard Feynman is courtesy of The Archives, California Institute of Technology. The Picasso drawing is Copyright 2001 Estate of Pablo Picasso / Artists Rights Society (ARS), New York.

References

[1] M. Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, 2001.

[2] J. Bergen and E. Adelson. Early vision and texture perception. *Nature*, 333:363–364, 1988.

[3] J. S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH 97*, pages 361–368, 1997.

[4] C. J. Curtis, S. E. Anderson, J. E. Seims, Kurt W. Fleisher, and D. H. Salsin. Computer-generated watercolor. In *SIGGRAPH 97*, pages 421–430, 1997.

[5] J. Davis. Mosaics of scenes with moving objects. In *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.*, 1998.

[6] A. 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.

[7] D. D. Garber. *Computational Models for Texture Analysis and Texture Synthesis*. PhD thesis, University of Southern California, Image Processing Institute, 1981.

[8] J. J. Gibson. *The Perception of the Visual World*. Houghton Mifflin, Boston, Massachusetts, 1950.

[9] P. Harrison. A non-hierarchical procedure for re-synthesis of complex textures. In *WSCG '2001 Conference proceedings*, pages 190–197, 2001. See also http://www.csse.monash.edu.au/~pfh/resynthesizer/.

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

[11] A. Hertzmann, C.E. Jacobs, N. Oliver, B. Curless, and D.H. Salesin. Image analogies. In *SIGGRAPH 01*, 2001.

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

[13] L. Liang, C. Liu, , Y. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. Technical Report MSR-TR-2001-40, Microsoft Research, March 2001.

[14] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanism. *JOSA-A*, 5(5):923–932, May 1990.

[15] V. Ostromoukhov and R. D. Hersch. Multi-color and artistic dithering. In *SIGGRAPH 99*, pages 425–432, 1999.

[16] Kris Popat and Rosalind W. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. In *Proc. SPIE Visual Comm. and Image Processing*, 1993.

[17] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40(1):49–71, December 2000.

[18] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *SIGGRAPH 00*, pages 465–470, 2000.

[19] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97*, 1997.

[20] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH 00*, pages 479–488, 2000.

[21] 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.

[22] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame). *International Journal of Computer Vision*, 27(2):1–20, March/April 1998.