A Randomized Approach for Patch-based Texture Synthesis using Wavelets

L. Tonietto, M. Walter and C. R. Jung
Masters in Applied Computing, Unisinos, Brazil
marcelow@cin.ufpe.br

Abstract
We present a wavelet-based approach for selecting patches in patch-based texture synthesis. We randomly select the first block that satisfies a minimum error criterion, computed from the wavelet coefficients (using 1D or 2D wavelets) for the overlapping region. We show that our wavelet-based approach improves texture synthesis for samples where previous work fails, mainly textures with prominent aligned features. Also, it generates similar quality textures when compared against texture synthesis using feature maps with the advantage that our proposed method uses implicit edge information (since it is embedded in the wavelet coefficients) whereas feature maps rely explicitly on edge features. In previous work, the best patches are selected among all possible using a $L_2$ norm on the RGB or grayscale pixel values of boundary zones. The $L_2$ metric provides the raw pixel-to-pixel difference, disregarding relevant image structures — such as edges — that are relevant in the human visual system and therefore on synthesis of new textures.

Keywords: texture synthesis, wavelets, textures from samples


1. Introduction

Recently, a great deal of attention has been given to synthesis of textures from samples. For many tasks in texturemapping [1], real textures, as opposed to procedurally generated ones [2], are the best option for achieving high realism. One of the main problems with real images is their usually small resolution. Texture synthesis from samples is an excellent solution for building textures that are not only visually similar to the given sample but also can be built at user-defined resolutions. The advances in this area grew from pixel-based techniques [3, 4], to more recent patch-based techniques [5–10], where the final texture is formed by joining together pieces or blocks of the original sample, with an RGB metric for selecting the best matches.

We propose here the selection of the best patches using a wavelet-based metric. Due to the wavelet properties we show that we can satisfactorily synthesize arbitrarily large textures where previous solutions fail, mainly maintaining spatial prominent features, as illustrated in Figure 1. In the image quilting result [6] and hybrid texture synthesis [7] there are edge discontinuities that are not present in our result.

2. Previous Work

The importance of texture synthesis has always been acknowledged in both Image Processing and Computer Graphics fields, with research going back as far as the late seventies [11]. There was a lot of research in late seventies and early eighties which have tried to generate textures either to validate texture models — mostly in Image Processing tasks — or simply to use the result in some application. Many approaches have used the idea of a sample as input information to create the result (see for instance [11–14]). Until recently, despite progress, such techniques were either too slow to be of practical use or the results were not general enough [15–17]. The most recent techniques on this subject synthesize a new, arbitrarily-sized texture, by copying either pixels or
blocks of pixels into the final texture. We divided this review section into four parts: one pixel at a time synthesis, patch-based synthesis, wavelets and approaches that use wavelets for patch-based synthesis.

2.1. One pixel at a time synthesis

The work of Efros and Leung presented in 1999 [3] introduced a new simple way of looking at the texture synthesis problem by “growing” a texture one pixel at a time from an initial seed. The color of a given pixel is determined by scanning over square patches of the sample texture that are similar to the patch on the texture being generated. A random patch in the sample is selected among the few satisfying the similarity criterion. The similarity is measured with a $L^2$ norm (sum of squared differences) on the RGB colors weighted by a Gaussian kernel. The original Efros and Leung’s algorithm is slow and later extensions have improved its performance, particularly the work of Wei and Levoy [4]. They have used a raster scan ordering to transform noise pixels into the result texture and have also improved the performance of the algorithm by using a multiscale framework and tree-structured vector quantization. Their approach also minimizes the $L^2$ norm in RGB space but without any weighting. Both approaches do not allow any kind of control of results. In 2001 Ashikhmin [18] improves on the Wey and Levoy (WL) technique by introducing two main contributions over prior work: first, his solution improves results for natural textures which mostly fail with the WL algorithm. Second, he introduces a second image, called the target image in the synthesis process. The final synthesis combines pixels from both the sample and the target, allowing nice effects such as writing words with the texture elements. A second approach towards local control for texture synthesis from samples was introduced by Tonietto and Walter [14] where the final texture is built from a collection of the same sample at different resolutions. This allowed synthesis of textures with local control over the size of texture elements.

More recently, Zhang and colleagues introduced an image-based texture synthesis method for rendering of Progressively Variant Textures (PVTs) [20] still on a pixel-at-a-time basis. Although not formally defined in the paper, the concept of PVT is an important one for texture synthesis, since it captures the class of textures where the texture elements vary in a progressive fashion, typical examples being mammalian fur patterns such as leopard skin. From an homogeneous texture sample they were able to synthesize a PVT. One main idea in their work was the notion of texton masks. A texton mask is a binary image marking prominent features or texture elements in the texture. In their synthesis algorithm, the texton masks prevent the disintegration of texture elements during synthesis.

2.2. Patch-based synthesis

Efros and Freeman introduced in 2001 another way of synthesizing image-based textures by stitching together random blocks of the sample and modifying them in a consistent way [6]. They call the technique ‘image quilting’. The idea improves dramatically on the one-pixel-at-a-time approach since it builds the texture at a much coarser scale while being able to keep high frequencies of the sample.

The same idea of using patches from the sample to synthesize the result was explored by Liang et al [5]. They were able to achieve real-time generation of large textures using special data structures and optimization techniques. A more detailed explanation of the Patch-Based Texture Synthesis (PBTS) is given in Section 3, since it is the basis of our solution.

More recently, three papers presented improvements and new approaches for PBTS. Nealen and Alexa [7] proposed an adaptive and hybrid technique for texture synthesis. They called it adaptive because it uses different patch sizes to perform block matching, according to a user-defined error tolerance, and it is hybrid because it may use a per-pixel texture...
synthesis strategy in the overlap regions, according to another user-defined tolerance. This method indeed improves the quality of the synthesized images for some textures when compared to [5, 6], but it has limitations for synthesizing other textures, as it will be discussed in Section 4. In [8] Wang Tiles are explored for texture synthesis. The textures are built as a collection of Wang Tiles, squares in which each edge is assigned a color. A valid tiling combines squares with the same color at the sharing edges. The interior of tiles can be filled in with textures, patterns or even 3D geometry. In [9] the problem of matching adjacent patches is reformulated as a minimum cost graph cut problem. This allowed for texture synthesis with arbitrarily-shaped patches, improving the results for some textures over previous work. This solution was also applied for spatio-temporal textures, allowing video synthesis as well. We used an approach similar to graph cuts in our work to compute the transitions at the patch boundaries.

In [10], Wu and Yu presented a method for texture synthesis from samples using feature maps to guide the process. This map is a binary image extracted from the sample that captures easy-to-detect features, such as the main edges and ridges of texture, which is then used in the synthesis process. The matching cost among patches is computed as a weighted average between the usual $L^2$ metric and a term computed from the features. In case of feature misalignments, they apply a small amount of deformation. The essence of Wu and Yu’s technique is similar to ours: perform edge matching as well as intensity matching. However, the technique in [10] relies explicitly on edge features, while our proposed method uses implicit edge information (since it is embedded in the wavelet coefficients). In the results section we compare our results with theirs.

2.3. Wavelets

Since wavelets were first introduced in the graphics community [21], they have come a long way and are an important tool in many graphics and image processing applications. In particular, some authors have used wavelets for texture synthesis and analysis [22, 23].

Portilla and Simoncelli [22] proposed a statistical model for texture representation using an overcomplete complex wavelet transform. In their approach, a set of statistics computed on pairs of coefficients corresponding to basis functions at adjacent spatial locations, orientations and scales are used to represent a texture. They are also able to synthesize textures by randomly reproducing such statistics. This technique tends to fail for structured textures.

Bar-Joseph and colleagues [23] also worked with texture synthesis in the wavelet domain. In their approach, input textures are treated as sample signals generated by a stochastic process. A tree that represents the multiscalar wavelet transform of the signal is computed, and new random trees, generated by learning and sampling the conditional probabilities of the paths in the original tree are used for texture synthesis.

Experimental results produced by these existing techniques indicate that wavelet analysis presents a large potential for texture synthesis and representation. However, approaches based on statistical simulation of wavelet coefficients typically have limitations for synthesizing more structured textures.

2.4. Wavelets and Patch-based Texture Synthesis

Recently, Wickramanayake et al. [24] proposed an approach for patch-based texture synthesis using wavelet coefficients in the similarity metric. In their approach, a 2D wavelet transform is computed for each candidate patch, and low-pass components (and only one of the three-detail images) in the lowest resolution are used to compare boundary zones. Using such reduced number of coefficients indeed increases the speed of the technique. However, the technique presents shortcomings for textures containing high-frequency details, because perceptually important wavelet coefficients in higher resolutions were disregarded in the matching process.

Wickramanayake et al. [25] proposed an improvement in block matching by using a zero-tree wavelet approach. In their method, no particular resolution is chosen; instead, relevant wavelet coefficients are retrieved through a thresholding technique, and block matching is performed only with such relevant coefficients. However, the threshold is based on an empirical formula (as stated by the authors), which may preserve irrelevant coefficients and/or discard perceptually significant coefficients.

Despite the good computational performance of the techniques described in [24, 25], the overall quality of synthesized textures is not impressive. Also, the idea of selecting random blocks within a certain error margin [5] instead of picking the minimum error block is ineffective when used in conjunction with the proposed multiresolution-based approaches, as stated by the authors. This leads to lack of variety in generated textures.

This paper presents a revised and significantly extended version of the work originally described in [26], improving the connection between patches and introducing a random selection of patches, that improves the computational burden of the synthesis procedure and also adds variety in generated textures. Next, we describe the proposed algorithm, which explores the wavelet transform to synthesize both ‘random’ and structured textures based on samples.

3. Our model

Our work replaces the distance block metric in the basic algorithm presented for PBTS and therefore we start this section by presenting an overview of [5].
3.1. Patch-based texture synthesis

In PBTS, patches of the original sample are combined to form the final texture. The algorithm starts by randomly picking a patch $B_k$ to start the process. This patch is positioned at the bottom left corner of the output texture (Figure 2 (left)). The size $w_B$ of the patches is user-defined and intuitively it should be the size of the main texture elements—or texels—present in the sample. For most textures using a patch of size between half and a quarter of the size of the original sample works well. For simplicity they are also restricted to square patches. The synthesis process follows by adding patches side-by-side and once a full row is completed the process continues for the row above and so on—Figure 2 (middle and right). There are three possible configurations for boundary zone matching, as illustrated in the same figure.

For each patch there is a boundary zone also with a user-defined width $w_E$. The optimal size of $w_E$ depends on the texture being generated. If it is too small, it will not capture enough details. If it is too large it will negatively impact the algorithm’s performance. As a balance they typically set $w_E$ as $\frac{1}{2}$ of $w_B$. The values set for $w_B$ and $w_E$ are very important, since they ultimately define the initial conditions for the synthesis process. As it stands, our work does not improve on the specific problem of defining the optimal $w_B$ and $w_E$. Later extensions on texture synthesis from patches have touched on this issue, using different patch sizes computed adaptively [7, 9]. Although we have not yet included this possibility in our work, the proposed wavelet metric for patch matching could be easily extended to account for different patch sizes.

The critical part of the algorithm is the selection of the next patch $B_k$ to be pasted onto the texture being constructed. As with many texture from sample techniques before [3, 4], they use an RGB metric to compare patches and build a list of candidate patches that satisfy an error criterion at the border area. From this list a random patch is selected. To build this list, the input sample is searched for all possible patches. If there is no patch satisfying the condition, the algorithm picks the patch with the smallest distance.

More formally, given two texture patches $I_1$ and $I_2$ of the same size and shape, they match if

$$d(I_1, I_2) < \delta = \tau \left[ \frac{1}{A} \sum_{j=1}^{A} (p_{E_j}^I)^2 \right]^{1/2}$$

where $A$ is the number of pixels in the boundary zone, $p_{E_j}^I$ represent the values of the $j$th pixel in the $E$, boundary zone, $d(\cdot, \cdot)$ represents the distance between the two patches and $\tau$ is a defined constant (default value is $\tau = 0.2$). This distance is computed only for the boundary zone $E$ of patches as follows:

$$d(E_i, E_{i+1}) = \left[ \frac{1}{A} \sum_{j=1}^{A} (p_{E_j}^I - p_{E_j}^{I+1})^2 \right]^{1/2} \quad (1)$$

The pixel values can be either grayscale or RGB triplets, although in the paper there is no explicit reference to the metric used for their results. Once the patches are selected, there is a blending step to provide smooth transition among adjacent patches. This smoothing is performed with feathering as proposed by Szeliski and Shum [27].

The standard algorithm was optimized for real-time texture synthesis with an optimized kd-tree and a pyramid scheme, providing excellent timings.

3.2. Wavelet criterion

The wavelet transform (WT) is a mathematical tool that can be used to describe 1D or 2D signals (images) in multiple resolutions. A WT is obtained through a sequence of low-pass and high-pass filters, alternated with down-samplings [28]. The result of the WT is a downsampled smoothed signal and several detail coefficients obtained at each down-sampling, such that the resulting signal has the same size as the original one. In particular, detail coefficients are generated by signal transitions and can be used to obtain a multisolution representation of signal edges [27]. In other words, the WT produces a signal that encodes both information on the original signal values and its multiscale edges. Another interesting property of the WT is its space-frequency locality, that is, it has nice localization properties in both the spatial and the frequency domains.

We believe that using the $L2$ norm directly to compare pixels values, as proposed by some state-of-the-art techniques [5, 6], is not the best approach for patch-based texture synthesis. In fact, if we just compute raw pixel-to-pixel differences, we may disregard important image structures (such as edges) that are relevant in the human visual system. This was also acknowledged on the recent work by Wu and Lu [10]. On the other hand, wavelet coefficients encode both information on the original pixels (a smoothed and downsampled version) and multiscale edge information (detail coefficients). Furthermore, the pyramidal algorithm proposed by Mallat [28] allows a very fast computation of the WT.

Our main contribution in this paper is to use a wavelet-based metric to compute the distance among candidate
patches. Although wavelets have been employed in [25, 26], we propose several improvements in this work. First, we use 1D wavelets based on a new traversal scheme of the neighboring region between two adjacent patches (which is faster than 2D wavelets, with no perceptual loss of quality). Also, we do not disregard any wavelet coefficient in the matching procedure, opposed to the approaches described in [25, 26], which leads to a better adjustment among patches. Although the use of all coefficients may increase the execution time, we select the first random block that satisfies a minimum error criterion, which reduces the search space and adds variety to synthesized textures. In other words, we replace Equation 1 by the following:

$$d(E_i, E_{i+1}) = \left[ \frac{1}{A} \sum_{\Psi=K,G,B} \sum_{j=1}^{A} \left( c_{E_i}^j - c_{E_{i+1}}^j \right)^2 \right]^{1/2}$$ (2)

where $A$ is the number of pixels in the boundary zone and $c_{E_i}^j$ represent the values of the jth wavelet coefficient in the $E_i$ boundary zone at the color channel $\Psi$.

Similar to the approach of Liang et al. [5], we also use a normalized threshold $0 \leq \tau \leq 1$ to control the minimum acceptable error in the wavelet domain. Then, if $E_i$ is the last patch added to the texture, we search for patch candidates $E_{i+1}$ that satisfy

$$d(E_i, E_{i+1}) \leq \tau \left[ \frac{1}{A} \sum_{\Psi=K,G,B} \sum_{j=1}^{A} \left( c_{E_i}^j \right)^2 \right]^{1/2}$$ (3)

In [5], all blocks satisfying the minimum error criterion were retrieved and one block was randomly selected within this subset. In this work, instead of testing all possible blocks, we select blocks at random positions in the sample texture and retrieve the first block that satisfies Equation 3. If no block satisfies Equation 3, the patch with the minimum error is retrieved. In our approach, we can achieve texture variety as in [5], but using a cheaper algorithm (since we do not perform an exhaustive search of all blocks).

For smoothing out the transition between adjacent patches we use an approach similar to graph cuts [9]. We initially compute the minimum cost boundary in the overlap region between adjacent patches. Then, we perform feathering with linear interpolation with respect to the mincut boundary (so that exactly at the mincut boundary we use the weight 0.5 for both patches). In [9], a weighted sum of pixel values based on a Gaussian kernel was used for feathering (but no details on the variance of this kernel are provided in the paper).

For the wavelet computation we implemented both 1D and 2D transforms for the Haar basis, because of its simplicity and computational efficiency. It should be emphasized that we adopted a non-normalized version of the Haar transform instead of the normalized orthogonal decomposition. The reason for such choice is that the normalized version tends to increase the weight of the low-pass signal as the signal is downsampled, while the non-normalized version keeps a better balance between low-pass and high-pass coefficients. The expressions for the low-pass $h[n]$ and high-pass $g[n]$ decomposition filters for the Haar wavelets without normalization are provided in Equation 4.

$$h[n] = \left[ \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right], g[n] = \left[ \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \right]$$ (4)

Our algorithm allows the computation of 1D or 2D wavelet transforms to perform block matching. To use the 2D WT, we need rectangular image domains. In the first and second configurations of block matching (i.e., block matching only horizontally or only vertically), the overlap region is already rectangular. However, for the L-shaped configuration, we need to rearrange pixels to a rectangular domain. For that purpose, we devised a pixel traversal scheme illustrated in Figure 3. The motivation for this traversal is to maintain a balance between the vertical and horizontal details of the boundary zone when using 2D WT. We have not investigated yet the effect on the results of other schemes for traversal, but we expect that for anisotropic textures traversal ordering may play a role on the final results.

When using the 1D WT, we need to transform the overlap region into a 1D signal. As described above, the first and second configurations lead naturally to rectangular overlap regions and the L-shaped regions can be transformed into a rectangular one using the proposed traversal scheme. Hence, in all block matching configurations we end up to rectangular overlap regions. To transform such regions into 1D signals, we perform a pixel scan along the largest dimension (e.g., if an overlap region is wider than taller, we perform a line scan). It is important to notice that both horizontal and vertical details are somehow embedded in the 1D signal, since discontinuities in the overlap region (edges) tend to be preserved during the 2D to 1D conversion.

As it will be further discussed in Section 4, three levels in the wavelet decomposition are typically enough to provide a good description of the image/signal and its edges. A brief...
description of our algorithm is provided below

1: select the first patch randomly. Call this patch \( E \)
2: for remainder positions in the output grid do
3: \( d_{best} = \infty \)
4: repeat
5: select a random patch in sample texture. Call this patch \( C \)
6: detect overlap region (horizontal, vertical or L-shaped) and apply traversal scheme
7: compute distance \( d \) for overlapping regions of existing patch \( E \) and candidate patch \( C \) according to equation 3
8: if \( d < d_{\text{best}} \) then
9: \( B = C \)
10: end if
11: until \( d <= \text{MAXERROR} \) or all possible patches examined
12: if \( d <= \text{MAXERROR} \) then
13: Output patch \( C \)
14: else
15: Output patch \( B \)
16: end if
17: \( E = C \)
18: end for

4. RESULTS

In this section we discuss the effect of selected parameters in the quality of the synthesized textures. In particular, we compare 1D versus 2D wavelets, analyse the consequence of changing the number of scales used in the wavelet decomposition, compare block matching using color information (RGB channels) and only luminance (intensity) and evaluate the influence of the threshold \( \tau \). Since the choice of the initial block may affect significantly the synthesized texture, we used the same initial block to guarantee that only the change of the analysed parameter is affecting the result. We also present some results of textures generated with our approach, comparing them with other state-of-the-art algorithms. When comparing to block-based approaches [5–7], we tried to use the same block parameters \( w_B \) and \( w_E \) for all methods. The choice of \( w_B \) and \( w_E \) for each synthesized texture is provided in Table 1. For comparisons with the technique described in [7], we used the ‘MATLAB’ code provided by the authors in their web site, using \( \delta_{\text{max}} = \Delta_{\text{max}} = 0.02 \), as in most examples provided in [7] (block sizes and overlap regions were exactly the same as used in our wavelet-based method). Other relevant parameters are described along the section.

We have noticed that, in general, 1D wavelet computations showed to be good enough to generate our results. We illustrate in Figure 4 that there is no noticeable difference between the two synthesized textures when using 1D and 2D wavelet computation. In fact, this result was somewhat expected, since generally the 1D signal encodes information on 2D edges. Furthermore, boundary zones are rectangular regions with one dimension typically much larger than the other (i.e., rectangular regions much taller than wider or vice-versa). Consequently, 2D features are not very noticeable in the boundary zones, and 1D wavelets showed to be efficient for detecting relevant structures.

We have also observed that, in general, two or three levels in the wavelet decomposition are enough to provide a good matching of details. Indeed, raw pixel-to-pixel comparisons do not take into account high-frequency components and the first level of wavelet decomposition only considers very sharp transitions (in particular, Haar wavelets have a very small support). This behavior is illustrated in Figure 5, that shows different synthesized apple textures using 1D wavelets with 1, 2, 3 and 4 levels, respectively. Except for the first results (1 level), all synthesized textures presented a good preservation of detail continuity.

Another parameter that can be adjusted in the proposed algorithm is to either use only the intensity or all three color components to perform block matching. The first option has the clear advantage of being much faster than the second (approximately one third), while the second tends to produce a more accurate texture quilting. In general, using only pixel intensity is enough for texture samples presenting a low variety of chromatic components, as illustrated in Figure 6. On the other hand, the texture shown in Figure 7 presents a larger diversity of colors and results obtained using color

<table>
<thead>
<tr>
<th>Figure</th>
<th>1</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_B )</td>
<td>48</td>
<td>36</td>
<td>48</td>
<td>28</td>
<td>32</td>
<td>36</td>
<td>24</td>
</tr>
<tr>
<td>( w_E )</td>
<td>16</td>
<td>6</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>12</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 4: 1D versus 2D wavelets.
Figure 5: Comparison of synthesized textures using different levels in the wavelet decomposition. From left to right, we used 1, 2, 3 and 4 levels.

Figure 6: Gray versus RGB (no significant difference). Sample, result computed with gray values and result computed with the RGB values.

Figure 7: Gray versus RGB (significant difference). Sample, result computed with gray values and result computed with the RGB values.

information are considerably superior to the result using only luminance.

The threshold $\tau$ used in Equation 3 for block matching defines a minimum acceptable error with respect to the energy of the transition region to be matched, similarly to the approach described in [5]. It is important to notice that a certain value for $\tau$ (say, $\tau = 0.2$) may work fine for some textures, but may also present very bad results for another texture. This behavior is due to the fact that determining a threshold with respect to the signal energy does not relate directly to the root mean squared (RMS) error defined by Equation 2. Figures 8 and 9 illustrate the effect of changing the parameter $\tau$ for different textures. In Figure 8, $\tau = 0.1$ produced a nice result,

Figure 8: Effect of varying parameter $\tau$. Sample, result with $\tau = 0.1$, result with $\tau = 0.2$.

Figure 9: Effect of varying parameter $\tau$. Sample, result with $\tau = 0.3$, result with $\tau = 0.5$.

Figure 10: Example of texture diversity using the same sample texture and same initial block.
but increasing \( \tau \) to 0.2 leads to several block mismatches. On the other hand, Figure 9 shows a texture for which \( \tau = 0.3 \) produces a very good synthesized texture and even \( \tau = 0.5 \) produces only a few incorrect block matches. It is also important to notice that selecting larger values of \( \tau \) decreases the computational cost. For example, the same result presented in Figure 8 using \( \tau = 0 \) (minimum error) doubled the running time when compared to the results using \( \tau = 0.1 \).

An example of the texture variation obtained using \( \tau \neq 0 \) is illustrated in Figure 10. In this figure, we show two synthesized textures using the same initial block and exactly the same parameters, with \( \tau = 0.5 \). Since random blocks are used, there is a noticeable difference between the two generated results and both appear to be reasonably natural. It should be pointed out that wavelet-based approaches described in [25, 26] are deterministic, in the sense that the same initial block always leads to the same generated texture.

Although the proposed method presents several adjustable parameters, our theoretical and experimental analysis indicated that good results are typically obtained using 1D wavelets with three decomposition levels and performing block matching in all three color channels. Next, we present some results obtained with these default parameters (and \( \tau = 0 \)). In Figure 11 we present our result for one standard texture that has been used as a test case for many texture from sample synthesis, as well as synthesis results with competitive approaches (for the method described in [7], we used \( \delta_{\text{mut}} = 0.01 \) and \( \Delta_{\text{mut}} = 0.05 \), as in the original paper). We can see that the algorithm works well for non-structured textures such as this. In Figure 12 we can see that our solution correctly generates the cracker holes evenly spaced, whereas in the results by Liang et al. [5] and Alexa and Nealen [7] these are not preserved. We also show in this figure the effect of changing the size of the boundary zone \( w_E \). We can see that the wavelet transform needs a boundary zone large enough to be able to capture the features of the texture. We have
not investigated an automatic way of determining the proper size for $w_E$. Another comparison of our technique against three other state-of-the-art algorithms is given in Figure 13. As it can be observed, our wavelet-based method provides a similar quality matching between adjacent blocks when compared with [10] and a better matching than the approaches described in [7] and [9].

The synthesis time for our results vary according to the selected parameters, ranging from a few seconds to about 20 minutes on a Pentium IV 1.8GHz machine with 512MB of RAM. In general, our running times are larger than those presented in the original patch-based work [5], since we have not yet implemented any optimizations as presented in [5]. This is left for future work.

5. Conclusions

We presented a wavelet-based approach for selecting blocks in patch-based texture synthesis. The final output texture is constructed from randomly selected blocks—patches—of the original sample that satisfy a minimum error criterion, computed from the wavelet coefficients (using 1D or 2D wavelets) for the overlapping region. Our results show that this criterion improves on previous results for textures that have regular noticeable features and brings similar quality results when compared against texture synthesis using feature maps, without the overhead of computing the maps.

There are many avenues left for further research on this topic. Although we have implemented the Haar wavelet basis, we believe that further investigations could explore other wavelet basis. Also, for the wavelet computation we could explore alternative ways of visiting the pixels in the boundary zones (i.e., alternative traversal schemes). The usage of random blocks combined with the selection of parameter $\tau$ brings two advantages: the introduction of more variety in the synthesized textures and decreasing running times. However, the selection of $\tau$ is tricky, since the same value for $\tau$ may produce good results for some textures and bad results for others. As future work, we would like to determine a threshold that is not computed with respect to the energy of the boundary zone; instead, it should be computed based on the expected distribution of block errors, which would be adaptively computed for each sample. We would like also to develop an automatic way for deciding whether to use color or grayscale information for texture synthesis, based on the chromaticity variance of the sample texture.

Acknowledgments

We acknowledge the partial financial support from agencies CAPES and CNPq-Brazil. We thank Baining Guo and Ying-Qing Xu for kindly running their patch-based sampling algorithm on the cracker texture (Figure 12 (middle-bottom)). We also thank the anonymous reviewers, for their valuable comments and suggestions.

References


