Eurographics Symposium on Rendering 2003
Per Christensen and Daniel Cohen-Or (Editors)

Hybrid Texture Synthesis

Andrew Nealen[†] and Marc Alexa[‡]

Technische Universität Darmstadt, Germany

Abstract

Patch-based texture synthesis algorithms produce reasonable results for a wide variety of texture classes. They preserve global structure, but often introduce unwanted visual artifacts along patch boundaries. Pixel-based synthesis algorithms, on the other hand, tend to blur out small objects while maintaining a consistent texture impression, which in return doesn't necessarily resemble the input texture. In this paper, we propose an adaptive and hybrid algorithm. Our algorithm adaptively splits patches so as to use as large as possible patches while staying within a user-defined error tolerance for the mismatch in the overlap region. Using large patches improves the reproduction of global structure. The remaining errors in the overlap regions are eliminated using pixel-based re-synthesis. We introduce an optimized ordering for the re-synthesis of these erroneous pixels using morphological operators, which ensures that every pixel has enough valid (i.e., error-free) neighboring pixels. Examples and comparisons with existing techniques demonstrate that our approach improves over previous texture synthesis algorithms, especially for textures with well-visible, possibly anisotropic structure, such as natural stone wall or scales.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture I.4.7 [Image Processing and Computer Vision]: Feature Measurement–*Texture*

1. Introduction

Most textures used in 3D computer graphics applications are produced either by manipulating digital images, by creating procedural [20] or hand-drawn textures, or by a combination of these techniques. Given the age of texture mapping [3], the notion of *texture synthesis*, where a sample is used to generate a similar, arbitrarily sized texture, is still a fairly new and not yet fully understood process. A large body of work has been dedicated to the topic in the past decade and the current results are very convincing for a large class of textures. Still, existing methods may generate artifacts in the presence of high frequency features, such as small scale structure or object boundaries, or simply fail to preserve global structure.

In this paper we present a simple, yet effective adaptive and hybrid texture synthesis method that tries to combine the best aspects from several successful approaches. Our core algorithm (described more formally in Section 2) is similar to other patch-based texture synthesis approaches [10, 17], however, uses the Fourier domain for finding the best match [23], that is, the patch from the input texture which minimizes overlap error with the existing synthesis result (Section 2.1). In addition to this general framework, the main ideas and contributions of our approach are:

- **Overlap re-synthesis:** Each new patch overlaps already synthesized regions. In these overlap regions an error is computed for each pixel (Section 2.3) and mismatched pixels are re-synthesized using a per-pixel texture synthesis strategy (Section 2.5). To ensure sufficient valid neighborhoods for these pixels they are ordered using morphological dilation of the valid regions (Section 2.4).
- **Adaptive patch sampling:** We adapt the patch size so that the error in overlap regions is bounded (Section 2.2). The error bound allows a trade-off between preserving global structure and avoiding detail artifacts: Increasing the error threshold leads to generally larger patches and a higher probability that large structures are preserved, however, at the cost of having more invalid pixels in the overlap region that need to be fixed, which is not always possible.

† andy@nealen.com
‡ alexa@informatik.tu-darmstadt.de

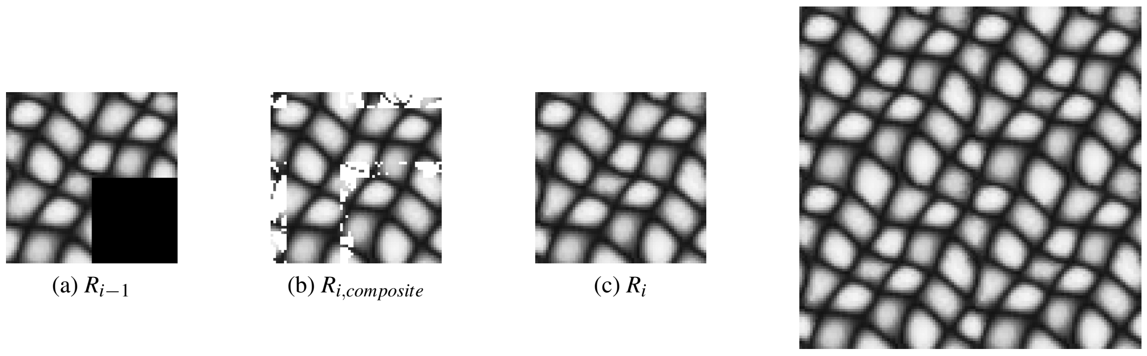
(a) R_{i-1} (b) $R_{i,composite}$ (c) R_i (d) $4 \times R_i$

Figure 1: *In algorithm step i (with the black area in (a) to be synthesized) we select a new patch from the input texture, constrained by the overlap with the already existing synthesis result R_{i-1} (a). Here, the algorithm finds a good match (with $\Delta_i < \Delta_{max}$) and does not adaptively split the patch. Pixels in the overlap region, which exceed a user-defined pixel error tolerance ($\delta_{max} \in [0,1]$) are marked as invalid (white) and the new patch P_i is composited into R_{i-1} (b). Then the invalid pixels are re-synthesized (c). The final patch P_i is constrained on its entire boundary, ensuring that the resulting texture will tile seamlessly (d). $\delta_{max} = 0.03$, $\Delta_{max} = 0.05$, $ov = 6$, initial patch size 32×32.*

With these extensions we achieve visually pleasing synthesis of most structured textures.

1.1. Previous Work

We list the work most relevant to ours in the following loose classification.

Pixel-Based Texture Synthesis: Efros and Leung's [9] Non-Parametric Sampling synthesizes a texture by repeatedly matching the neighborhood around the target pixel in the synthesis result with the input texture. They perform an exhaustive search for each synthesized pixel. Wei and Levoy's [26] algorithm is based on Efros/leung [9], extending it to a synthesis pyramid, which allows the use of smaller neighborhoods at possibly improved quality. They also apply tree structured vector quantization (TSVQ) to accelerate the algorithm by two orders of magnitude. Ashikhmin's [1] intelligent modification significantly reduces search space and achieves interactive framerates. His paper also thoroughly discusses drawbacks of previous, pixel-based methods, such as blurring. Merging both Ashikhmin and Wei/Levoy synthesis into a framework, Hertzmann et al. [14] produce interesting results and open new areas of application, such as *texture-by-numbers*. Zelinka and Garland [29] demonstrate an alternative preprocess to synthesize texture in real-time.

Patch-Based Texture Synthesis: These methods preserve global structure by generating the texture on a per-patch basis. Efros and Freeman's Image Quilting [10] algorithm aligns adjacent patch boundaries, constrained by overlap, and then performs a minimum-error-boundary-cut (MEBC) within the overlap region to reduce overlap artifacts. Liang et al.'s Patch-Based Sampling [17] uses the same technique for patch placement, but simply alpha-blends the overlap regions (feathering). Their implementation uses various search data-structures which lead to real-time acceleration.

Pyramid-Based Sampling / Feature Matching: Some early texture synthesis algorithms model texture as a set of features and then generate new images by matching these features [2, 4, 13, 22]. These algorithms are efficient, yet possibly fail to preserve global [2, 4, 13] or local [22] patterns.

Texture Synthesis over Surfaces: Most surface texture synthesis methods are direct extensions of pixel-based [25, 27, 28] or patch-based [21, 23] algorithms. Turk [25] and Wei/Levoy [27] densely tessellate the input mesh and then perform a per-vertex color synthesis. Ying et al. [28] synthesize per-texel using a texture atlas of the polygonal mesh. Praun et al. [21] extend the chaos mosaic [12] to surfaces with a pre-computed vector field to direct anisotropy. Recently, Soler et al. [23] demonstrated how a mesh can be seamlessly textured with only the input texture and a set of texture coordinates for each vertex.

Bidirectional Texture Function Synthesis: In Liu et al.'s work [19], geometry is recovered, synthesized and used to generate templates for each viewing/lighting setting. These templates then guide the actual BTF synthesis, thereby preserving global mesostructure. Tong et al. [24] extend Ashikhmin's [1] algorithm by adding the k-nearest neighbors to each candidate pixel of the Ashikhmin-set (k-coherence search).

Geometrically Motivated Texture Synthesis: Dischler and Ghanzfarpour have published many geometrically and structurally motivated algorithms which are of semi-procedural nature, yet also resemble the input so closely that they could be classified as texture synthesis algorithms. In their work [6] a highly structured texture is analyzed with some user intervention, from which the algorithm generates seemingly random structured texture. Texture Particles [7] are gathered by segmentation and the analysis of spatial arrangements using morphological operations. These are then procedurally assembled in the synthesis stage.

There are numerous other examples closely related to texture synthesis and a complete survey is beyond the scope of this paper. The inclined reader could follow general work in the computer vision literature [15, 16, 30, 31, 32].

1.2. Overview

Variable	Meaning
T	2D input texture
R, R_i	resulting 2D texture (final/in step i)
$\mathcal{P}, \mathcal{P}_s$	list of non-overlapping pixel-patches p_i which, when combined, resemble a tiling of the resulting texture area $R = \bigcup_i p_i$ (initial/split)
ov, ov_s	patch overlap in pixels (initial/split)
Δ_{max}	user defined patch overlap error tolerance in [0,1]
δ_{max}	user defined pixel error tolerance in [0,1]
i	integer algorithm step, starting at 1
p_i	a patch of connected pixels with index i in list $\mathcal{P}$. p_i defines a region in R with $\text{SIZE}(p_i) > 0$
$R_{i,composite}$	intermediate 2D result after Compositing (Fig. 1)
I_i	2D image mask extracted from R_{i-1}, using p_i (Fig. 4)
J_i	2D binary support function for I_i (Fig. 4)
$E_i, E_{i,trim}$	2D error image computed from T, I_i, J_i (Fig. 5)
P_i	2D texture patch picked from T in step i (Fig. 5)
Δ_i	patch overlap error in [0,1] for P_i
S_i	2D error surface between R_{i-1} and P_i (Fig. 8)
M_i	2D pixel traversal map, defining the order in which erroneous overlap pixels are re-synthesized (Fig. 10)
$\mathcal{W}_c$	a three-element, color channel weighting vector ($c = \{R,G,B\}$) with $\sum_c \mathcal{W}_c = 1$
$\mathbf{x}, \mathbf{x_0}$	a 2D vector/coordinate (x,y)

Table 1: *List of used variables/terms*

Our goal was to improve upon existing texture synthesis algorithms by combining their strengths. The improvement is two-fold (Fig. 1): first, we allow the algorithm to adaptively split the sampling grid if the current best match exceeds a user-defined overlap error tolerance $\Delta_{max} \in [0,1]$, similar to Soler et al. [23] and Drori et al. [8]. Second, our overlap artifact minimizing procedure re-synthesizes invalid pixels above a user-defined pixel error tolerance $\delta_{max} \in [0,1]$, using a pre-computed pixel traversal map to order the pixels for the *overlap re-synthesis*. This results, as we find, in boundary regions which display a negligible number of noticeable artifacts when compared to feathering [17] or MEBC [10].

2. Our Algorithm

Now we give a description of our (recursive) algorithm for Hybrid Texture Synthesis (HTS). For variables/terms and their meanings, see Table 1. In algorithm step i, a patch of connected pixels is selected from an example texture T which best fits the target region p_i in the intermediate result R_{i-1}. This selection/search procedure is constrained by overlap of ov pixels in each direction with R_{i-1}. A candidate patch P_i is used either if the overall error in the overlap region is below the maximum overlap error Δ_{max}, or the patch cannot be further split – otherwise the search process is repeated using smaller patches. If a patch satisfying the error bound has been found, every pixel in the overlap region exceeding the pixel error threshold δ_{max} is marked as invalid and the valid region is dilated to form a pixel traversal map M_i, thereby defining the order in which erroneous overlap pixels are re-synthesized on a per-pixel basis. This ensures sufficient valid neighborhoods for each pixel. We repeat this process until each patch p_i in the initial patch list $\mathcal{P}$ has been assigned pixel values. Formally, the algorithm could be stated by the pseudocode in Figure 2.

```
1:  HYBRIDSYNTHESIZE($T, \mathcal{P}, ov, \Delta_{max}, \delta_{max}, R$) : $R$
2:  for all patches $p_i \in \mathcal{P}$ do
3:      $[P_i, \Delta_i] \leftarrow$ FINDBESTPATCH($T, R_{i-1}, p_i, ov$)
4:      if ($\Delta_i < \Delta_{max}$ or ISSINGLEPIXEL($p_i$)) then
5:          $S_i \leftarrow$ ERRORSURFACE($P_i, R_{i-1}$)
6:          $P_i \leftarrow$ MARKINVALIDPIXELS($P_i, S_i, \delta_{max}$)
7:          $M_i \leftarrow$ BUILDTRAVERSALMAP($P_i$)
8:          $R_{i,composite} \leftarrow$ COMPOSE($P_i, R_{i-1}$)
9:          $R_i \leftarrow$ OVERLAPRESYNTHESIS($T, R_{i,composite},$
                                              $M_i$)
10:     else
11:         $\mathcal{P}_s \leftarrow$ SPLITPATCH($p_i$)
12:         $ov_s \leftarrow max(3, \lceil ov/2 \rceil)$
13:         $R_i \leftarrow$ HYBRIDSYNTHESIZE($T, \mathcal{P}_s, ov_s, \Delta_{max},$
                                              $\delta_{max}, R_{i-1}$)
14:     end if
15: end for
16: return $R$
```

Figure 2: *The* HYBRIDSYNTHESIZE *Algorithm*

In our implementation, the patch list $\mathcal{P}$ initially consists of N×M quadrilateral patches with size 2^n by 2^m ($n, m \in \mathbb{N}$) in scanline order, however, our concepts generalize to arbitrarily shaped patches. Note that we have tried to use the notation given in Table 1 consistently throughout the text, in all pseudo-code examples, and all figures.

2.1. Finding the Best Patch

For each patch $p_i \in \mathcal{P}$ we must find the best possible patch P_i in the input texture T, constrained by overlap with the existing result R_{i-1}. We do this as described in Figure 3.

```
1: FINDBESTPATCH($T, R_{i-1}, p_i, ov$) : $[P_i, \Delta_i]$
2: $I_i, J_i \leftarrow$ BUILDIMAGEMASK($R_{i-1}, p_i, ov$)
3: $E_i \leftarrow$ ERRORIMAGE($T, I_i, J_i$)
4: $E_{i,trim} \leftarrow$ TRIMINVALIDREGIONS($E_i, J_i, p_i$)
5: $[P_i, \Delta_i] \leftarrow$ BESTPATCH($E_{i,trim}, T$)
6: return $[P_i, \Delta_i]$
```

Figure 3: *The* FINDBESTPATCH *Algorithm*

The image mask I_i and binary support function J_i (Fig. 4) are needed for the error image computation (Fig. 5). BUILDIMAGEMASK simply grows the current patch by the pixel overlap *ov* and checks the grown region in the current result R_{i-1} for already synthesized pixels: if there exists a valid pixel, J_i (initially all 0's) is set to 1 and the color value from R_{i-1} (Fig. 1(a)) is stored in I_i. Figures 1(a) and 4 show that for this patch we must match on its entire boundary to produce a tileable texture.

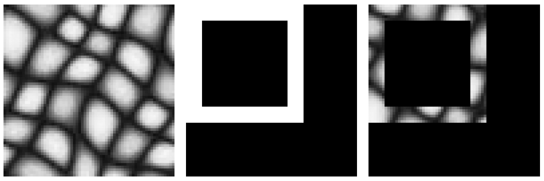
Figure 4: *The input texture T (left), the binary support function J_i (middle) and the image mask I_i for the example outlined in Figure 1*

Given I_i, the input texture T (both with RGB color values in [0,1]) and J_i , we can compute the weighted error $\Delta_i \in [0,1]$ between the mask I_i and a circular shift $\mathbf{x_0}$ of T (where the error image E_i stores Δ_i for each $\mathbf{x_0}$) as

$$E_i(\mathbf{x_0}) = \frac{1}{\kappa_i}\sum_{\mathbf{x}}\sum_{c}\left[J_i(\mathbf{x})\mathcal{W}_c(I_{i,c}(\mathbf{x}) - T_c(\mathbf{x}+\mathbf{x_0}))^2\right] \quad (1)$$

with $\kappa_i = \sum_{\mathbf{x}} J_i(\mathbf{x})$, $c = \{R,G,B\}$ (the set of color channels in RGB space) and $\sum_c \mathcal{W}_c = 1$. Given that $J_i(\mathbf{x})I_i(\mathbf{x}) = I_i(\mathbf{x})$, and the cross correlation between two images (functions) $f \diamond g$ is defined as $(f \diamond g)(\mathbf{x_0}) = \sum_{\mathbf{x}} f(\mathbf{x})g(\mathbf{x}+\mathbf{x_0})$, we can write Equation 1 as [23]

$$\begin{aligned} E_i(\mathbf{x_0}) \;=\; & \frac{1}{\kappa_i}\sum_c \mathcal{W}_c\Big[\sum_{\mathbf{x}} I_{i,c}(\mathbf{x})^2 - \\ & - 2(I_{i,c}\diamond T_c)(\mathbf{x_0}) + (J_i \diamond (T_c^2))(\mathbf{x_0})\Big] \quad (2)\end{aligned}$$

The correlation $f \diamond g$ between two functions can be computed in $O(N \log N)$ (N being the number of pixels in the input texture) in fourier space as $f \diamond g = F^{-1}(F(f) * \overline{F(g)})$. In our implementation we pre-compute the fourier transform for T and need only recompute the fourier transforms of I_i and J_i for each new patch p_i [23].

In our implementation, $\mathcal{W}_{R,G,B} = \{0.299, 0.587, 0.114\}$, analogous to the Y component (luminance) of the YIQ color model [11]. This accounts for the greater sensitivity of the human visual system to changes in luminance than changes in hue or saturation [11]. Note that this is not equivalent to matching on gray levels.

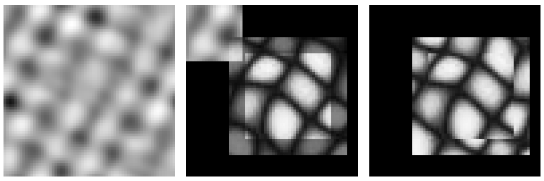
Figure 5: *The error Image E_i (left), the trimmed error image $E_{i,trim}$ with the selected patch P_i superimposed (middle, the overlap region is darkened), and P_i with I_i from Figure 4 superimposed to highlight the remaining artifacts before overlap re-synthesis (right). See Figure 12 for a comparison of overlap repair strategies using this example.*

The grayscale error image E_i, which in $\mathbf{x} = (x,y)$ stores Δ_i for the grown patch with these upper left bounding box coordinates, is computed under the assumption that the input texture wraps in both dimensions (the *green scales* texture used throughout this paper has this property). If T wraps and we were to allow all possible positions in E_i for the match, the algorithm would produce verbatim copies of T. Liang et al. [17] use a relative error value $\varepsilon \in [0,1]$, (where $\varepsilon = 0$ picks only the best match and $\varepsilon = 1$ picks at random) to control the allowed error, yet we still find that the results in their paper display a large amount of verbatim copying, even for values of $\varepsilon = 0.2$. Thus, in our implementation we always trim E_i such, that no part of P_i crosses the texture border (Fig. 5). As pointed out by Soler et al. [23], trimming the error image E_i significantly reduces possible matches, especially for small input textures (compare the images in Figure 5). We have experienced that this also results in more varied synthesis results with less large scale verbatim copying.

In our implementation, we search in pre-computed rotations and scales of T for isotropic textures. For anisotropic textures, the use of rotations and scales depends on the initial patch size, Δ_{max} and the feature size(s) within T. If these parameters are carefully balanced, using rotations and scales of T can produce interesting and correct results.

2.2. Adaptive Patch Sampling

Existing patch-based synthesis algorithms operate on a fixed grid of uniform quadrilateral patches [10, 17]. Our adaptive approach is inspired by *Hierarchical Pattern Mapping* [23], and therefore very similar: after choosing a candidate patch P_i from the input texture T (line 3 in HYBRIDSYNTHESIZE), if the weighted error Δ_i between the overlapping pixels of P_i and the existing result R_{i-1} exceeds the user-defined Δ_{max}, we split the current patch p_i into four congruent patches (denoted by the set $\mathcal{P}_s$), and recurse (lines 3,4 and 11-13 in HYBRIDSYNTHESIZE and Figure 6). This leads to adaptive sampling grids such as the one shown in Figure 6, bottom right, and Figure 15, right column. Note that we halve the pixel overlap *ov* for recursive calls, choosing a minimal value of $ov = 3$ (line 12 in HYBRIDSYNTHESIZE).

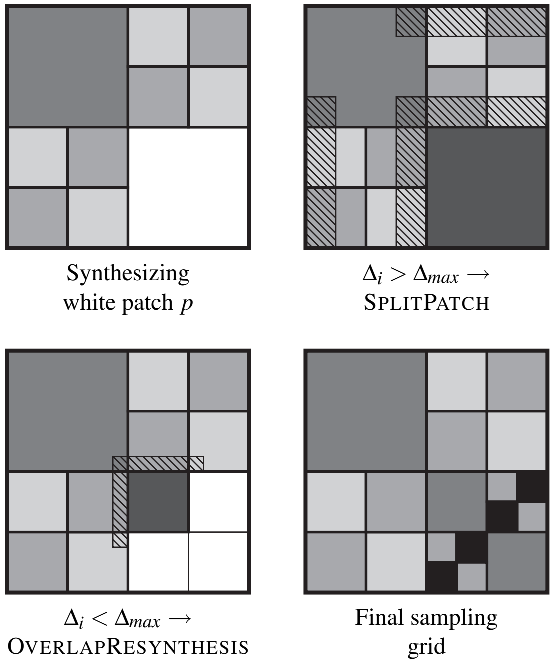
Synthesizing white patch p
$\Delta_i > \Delta_{max} \rightarrow$ SPLITPATCH
$\Delta_i < \Delta_{max} \rightarrow$ OVERLAPRESYNTHESIS
Final sampling grid

Figure 6: *Adaptive patch sampling. Shades of gray represent already synthesized patches, the hatched areas are the overlap regions used for best patch search.*

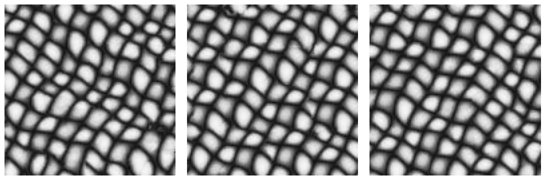

Figure 7: *Balancing* Δ_{max}. *Left:* $\Delta_{max} = 0.01$, *resulting in many small patches, comparable to pixel-based algorithms. Middle:* $\Delta_{max} = 1.0$, *which uses only the initial, uniform patch list and fails to eliminate all overlap artifacts. Right:* $\Delta_{max} = 0.04$, *resulting in a good trade-off between global and local structure. The images are all 128×128 results of Figure 13(a)*

The amount of patch splits depends entirely on the value for Δ_{max}: setting Δ_{max} to 0 always splits (yielding a traditional per-pixel synthesis method), whereas setting to 1 will only use the original patch list (as in typical patch-based methods). Using less extreme values allows trading structural inconsistencies for detail artifacts. If Δ_{max} has a large value (e.g. $\Delta_{max} \gtrsim 0.1$), large patches are used and global structures are preserved. On the downside, many invalid pixels in the overlap regions have to be re-synthesized, which might lead to artifacts. If Δ_{max} is chosen very small (e.g. $\Delta_{max} \lesssim 0.01$) only few pixels are erroneous and artifacts almost vanish, yet the use of small patches leads to some global structure problems. This trade-off is illustrated in Figure 7.

2.3. Overlap Error and Pixel Invalidation

After a patch P_i is picked from T (P_i includes the overlap pixels extracted from T, superimposed in Figure 5, middle), we compute the error surface S_i (Fig. 8, left) in the overlap region $J_i(\mathbf{x}) \neq 0$ (the darkened strip in Figure 5, middle) as

$$S_i(\mathbf{x}) = \sum_c J_i(\mathbf{x}) \mathcal{W}_c (I_{i,c}(\mathbf{x}) - P_{i,c}(\mathbf{x}))^2 \qquad (3)$$

with $\mathcal{W}$ and c as defined in Table 1 and Section 2.1.

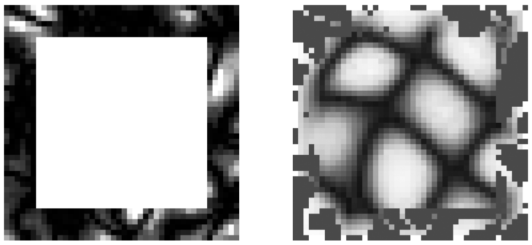

Figure 8: *The error in the overlap region* S_i, *defined in Equation 3 (left; the error-values* $S_i(\mathbf{x})$ *are normalized to the interval [0,1] for visualization purposes) and the feathered patch* P_i *with invalid pixels marked blue (dark gray).*

MARKINVALIDPIXELS (in HYBRIDSYNTHESIZE, line 6) takes the user-defined $\delta_{max} \in [0,1]$ to mark pixels with $J_i(\mathbf{x}) \neq 0$ and $S_i(\mathbf{x}) \geqslant \delta_{max}$ as *invalid* and in need of per-pixel re-synthesis. All pixels for which $J_i(\mathbf{x}) \neq 0$ and $S_i(\mathbf{x}) < \delta_{max}$ are preserved and a stepwise alpha mask (feathering) is applied during Compositing (Figures 8, right, and 1(b)). Note, that setting $\delta_{max} = 1$ results in pure feathering [17], whereas $\delta_{max} = 0$ re-synthesizes the entire overlap region. Typical values for δ_{max} used in this paper are between 0.02 and 0.05.

2.4. Pixel Traversal-Map

During our experiments, we have realized that simply re-synthesizing the invalid overlap pixels in scanline order often produces more artifacts than feathering. This is mainly due to the causal neighborhood of the invalid pixels: by re-synthesizing in scanline order, we do not actually re-synthesize invalid pixels based on the valid pixels of $R_{i,composite}$, but to a large extent on other re-synthesized pixels. To ensure that each re-synthesized pixel has a sufficient causal neighborhood in $R_{i,composite}$, we introduce the concept of a *traversal-map* (M_i). M_i is computed before overlap re-synthesis by repeated morphological dilation of the binary support for valid pixels with a euclidian disk of radius one (Fig. 10), which is, in essence, a city-block distance transform [5]. We build the traversal-map M_i as described in Figure 9.

```
1:  BUILDTRAVERSALMAP($P_i$) : $M_i$
2:  $level = 1$
3:  $D_{level} \leftarrow$ binary image of size $P_i$ initialized to 0
4:  $D_{level} \leftarrow$ set all valid $pixels \in P_i$ to 1
5:  $M_i \leftarrow D_{level}$
6:  while $(\exists pixel \in D_{level} \wedge pixel = 0)$ do
7:      $level = level + 1$
8:      $D_{level} \leftarrow$ DILATE$(D_{level-1}, Disk)$
9:      $M_i \leftarrow M_i + (D_{level} \wedge \overline{D_{level-1}}) * level$
10: end while
11: return $M_i$
```

Figure 9: *The* BUILDTRAVERSALMAP *Algorithm*

We then re-synthesize the pixels in each level of M_i sequentially in ascending order. This is analogous to the method applied by an art restorer: a stepwise restoration of the hole from the boundary of the existing image.

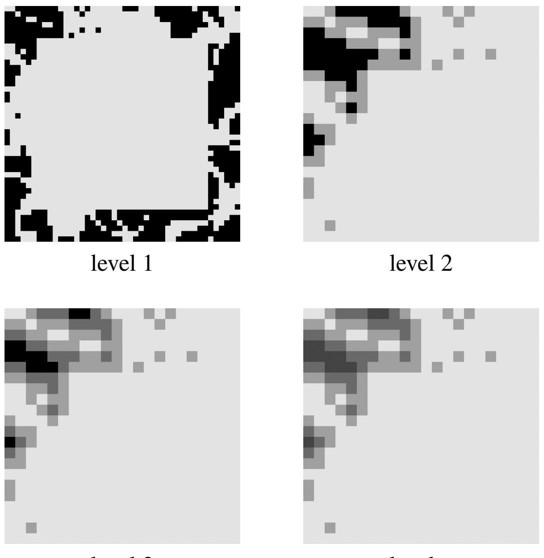

Figure 10: *Traversal-map (M_i) construction by repeated morphological dilation of valid pixels. Level 1 shows valid pixels in gray, invalid pixels in black. Levels $2, \ldots, n$ are the steps in creating M_i, each a zoom of the upper left corner of level 1.*

We have also experimented with dilating from both the new texture patch P_i and the existing result R_{i-1}, which intuitively seems like the straightforward solution, given the aforementioned analogy. Generally, more pleasing results are achieved when leaving the responsibility of fixing a bad overlap configuration to only the new patch. This does not always eliminate all overlap artifacts, but significantly reduces their frequency (Fig. 12, bottom).

2.5. Compositing and Overlap Re-synthesis

After Compositing P_i into R_{i-1}, resulting in $R_{i,composite}$ (Fig. 1(b)), we re-synthesize invalid pixels in traversal-map ordering. In this procedure, each re-synthesized pixel is assigned a square-shaped neighborhood (7×7 pixels in all of our examples) from which we gather already synthesized pixels into an image mask and a binary support function, identical to the best patch search of Section 2.1 (Fig. 11).

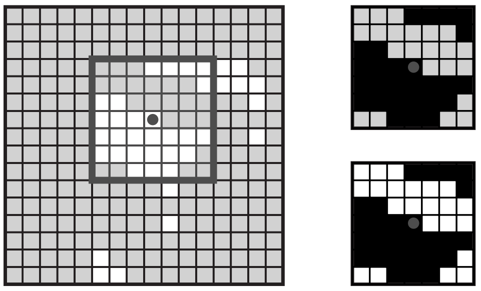

Figure 11: *Re-synthesizing a single pixel (circle) in the overlap region (with not yet synthesized pixels in white) using a 7×7 neighborhood. From the highlighted 7×7 area (left) we extract the image mask I (upper right) and the binary support function J (lower right) for best pixel search in T.*

We apply a gaussian falloff to the binary support function and furthermore blend the immediate four neighboring pixels of the selected pixel from T with $R_{i,composite}$ to reduce unwanted and, more importantly, non-existent noise.

3. Results

We were especially interested in improving the results of Wei/Levoy's *green scales* (Fig. 13), and Liang et al.'s *natural stone wall* (Fig. 14). Both display a significant amount of structure and anisotropy, which generally challenge existing synthesis algorithms. As pointed out by Liang et al. [17], Image Quilting has a tendency to produce abrupt color changes, termed *boundary mismatch*, which is why they prefer feathering over the MEBC. We find that feathering, although admittedly very fast and sufficient for a large class of textures, tends to blur high frequency features along patch boundaries.

In Figure 12 we compare a typical scenario in which our algorithm correctly fixes the artifacts in the overlap region: the results of synthesizing the final 32×32 patch of a small 64×64 synthesis result. We deliberately chose the final patch for these examples, as this is the most difficult match and very likely to introduce disturbing artifacts. We

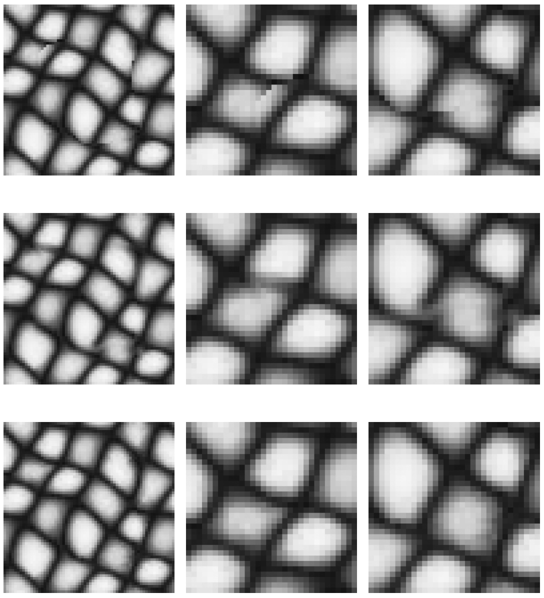

Figure 12: *Pasting the final 32×32 patch into a circularly shifted version of Figure 1(a). Top: MEBC* [10]*, middle: feathering* [17]*, bottom: overlap re-synthesis. See right image in Figure 5 for the above example before the overlap region is repaired.*

use a pixel overlap of six pixels and a pixel error tolerance of $\delta_{max} = 0.03$. The example is a circularly shifted version of Figure 1(a), with the black region centered. The results (each with two zooms by a factor of 2) show, that our overlap re-synthesis (bottom) significantly reduces boundary mismatch (top, MEBC) and blurring artifacts (middle, feathering).

Figure 13 compares existing texture synthesis algorithms to Hybrid Texture Synthesis, using the benchmark *green scales* texture, which is compared in most other papers as well. (a) Shows the input 64×64 texture T. In (b) we see the results of Efros/leung's non-parametric sampling, which, as it applies no neighborhood blending or multiscale-synthesis, produces a significant amount of non-existent noise. This noise is successfully eliminated in Wei/Levoy's result, shown in (c). The question at hand is: do (b) or (c) resemble (a)? We find that while anisotropy is debateable, scale is not perceived as similar. (d) Is a texture generated by Image Quilting (IQ) and (e) by Patch-Based Sampling (PBS). Although these results preserve global structure and similarity much better than (b) and (c), the blurring and boundary mismatch artifacts are noticeable, thus revealing the regular patch grid. We find that while our algorithm (f) does not entirely nullify the existence of overlap artifacts, it does reduce their overall frequency, thereby eliminating visible patch boundaries. All synthesis results in Figure 13 are of size 128×128. Note that (b) was taken from Alexei Efros' website, (c) from Li-Yi Wei's website and (d) from Alexei Efros' *SIGGRAPH* presentation. We produced (e) by using our algorithm and setting $\Delta_{max} = \delta_{max} = 1$.

The *natural stone wall* texture used by Liang et al. [17] is compared in Figure 14. The PBS result shows the typical blurring of small scale structure. Our result tends to cut off structure where the overlap is too small to completely repair the damage done. It seems to be a question of personal flavor what type of artifact is less disturbing. The *mosaic* texture is presented as a failure [17] as the object boundaries are often not properly aligned and therefore blurred. Our algorithm manages to avoid such pitfalls in several cases.

More Hybrid Texture Synthesis results are shown in Figure 15 (see color page), with their respective sampling grids. The top three results are of good quality, whereas the result of the puzzle texture (bottom) exhibits the main limitation of our method: in the presence of high frequency structure the error metric given by Equation 1 simply fails.

4. Conclusions and Future Work

Our algorithm does a good job of eliminating overlap artifacts between adjacent patches, but also has some drawbacks in its current state which will be addressed in our future work.

Most importantly, both the adaptive patch sampling and the overlap re-synthesis stages suffer from their computational expense, as we perform multiple fourier transforms, each of $O(N \log N)$, for each sampled patch/pixel. A straightforward application of existing acceleration schemes (e.g. fixed neighborhood searching [26] or reduced set of overlap cases [17]) is not possible, as we would have to build multiple search structures on the patch level (due to varying patch and overlap sizes), and we do not *a priori* know the pixel neighborhood in the overlap region. The drawback on the patch scale is negligible when compared to the pixel scale, as we generally (depending on δ_{max}) synthesize much more pixels than patches. Our speedup work is therefore primarily focussed on the overlap re-synthesis stage. One approach we are currently examining is to fill invalid pixels in the overlap region from the valid part by scattered data interpolation (push-pull: downsampling and upsampling of $R_{i,composite}$), yielding a fixed, square-shaped neighborhood for each re-synthesized pixel, to which we can apply fixed neighborhood search methods such as Wei/Levoy's TSVQ. Other per-pixel acceleration possibilities include using Ashikhmin's [1] algorithm or it's extension, k-coherence search [24].

As in previous approaches, high frequency features, such as small scale structure or object boundaries, are given no preference in the distance metric used for overlap error computation. Even though overlap re-synthesis works well in many cases, it can't repair certain overlap configurations. Possible future research topics could include the development of a better distance metric, augmenting the

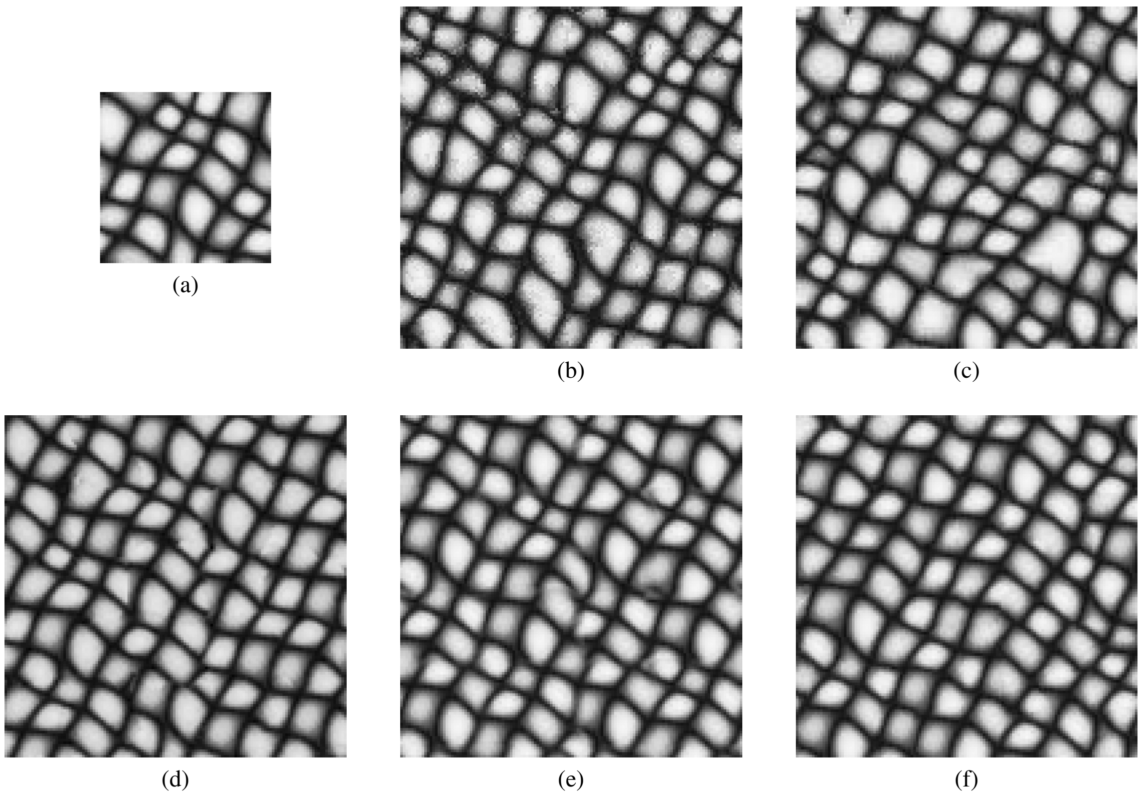

Figure 13: *A comparison of texture synthesis algorithms on the benchmark* green scales *texture: (a) the input texture sample (64×64), (b) Efros/Leung synthesis, (c) Wei/Levoy synthesis, (d) Image Quilting, (e) Patch-Based Sampling, and (f) Hybrid Texture Synthesis. Each of the above is a 128×128 crop of a larger synthesis result. The synthesized textures behave toroidally in all cases, excluding the Image Quilting result. Patch sizes in (e) are 32×32. Parameters for (f):* $\delta_{max} = 0.02$, $\Delta_{max} = 0.04$, *ov* $= 7$, *initial patch size 32×32.*

input texture with semantic/structural information, or pre-synthesizing a structural template [19].

Our current implementation uses a geometrically defined quad mesh, which is rastered during the synthesis process. This is to allow an alignment of the patches to the main directions of texture anisotropy. We have started initial experiments on automating the initial quad mesh layout by autocorrelation [18] and first results look promising.

Acknowledgements

We acquired the original textures used in this paper from the websites of Song Chun Zhu, Li-Yi Wei and Alexei Efros. Image Quilting and Non-Parametric Sampling results were taken from Alexei Efros' website and *SIGGRAPH* 2001 presentation, Wei/Levoy results from Li-Yi Wei's website. We would like to thank the respective authors for making their results publicly available. We also thank the anonymous reviewers for their insightful comments, which led to the addition of Table 1.

References

1. M. Ashikhmin. Synthesizing Natural Textures. *Proc. ACM Symposium on Interactive 3D Graphics*, 217–226, 2001.
2. Z. Bar-Joseph, R. El-Yaniv, D. Lischinski and M. Werman. Texture Mixing and Texture Movie Synthesis Using Statistical Learning. *IEEE Transactions on Visualization and Computer Graphics*, 120–135, 2001.
3. J. Blinn and M. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, **19**(10): 542–547, October 1976.
4. J.S. De Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images. *SIGGRAPH 97 Conference Proceedings*, 361–368, 1997.
5. O. Cuisenaire. Region growing Euclidean distance transforms. *ICIAP'97, 9th International Conference on Image Analysis and Processing*, , 1997.
6. J.-M. Dischler and D. Ghazanfarpour. A Geometrical Based Method for Highly Complex Structured Textures Generation. *Computer Graphics Forum*, **14**(4): 203–215, October 1995.
7. J.-M. Dischler, K. Maritaud, B. Lévy and D. Ghazanfarpour. Texture Particles. *EUROGRAPHICS 2002 Conference Proceedings*, 2002.
8. I. Drori, D. Cohen-Or and H. Yeshurun. Example-Based Style Synthesis. to appear in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003.
9. A. Efros and T. Leung. Texture Synthesis by Non-parametric Sampling. *IEEE International Conference on Computer Vision*, 1033–1038, 1999.

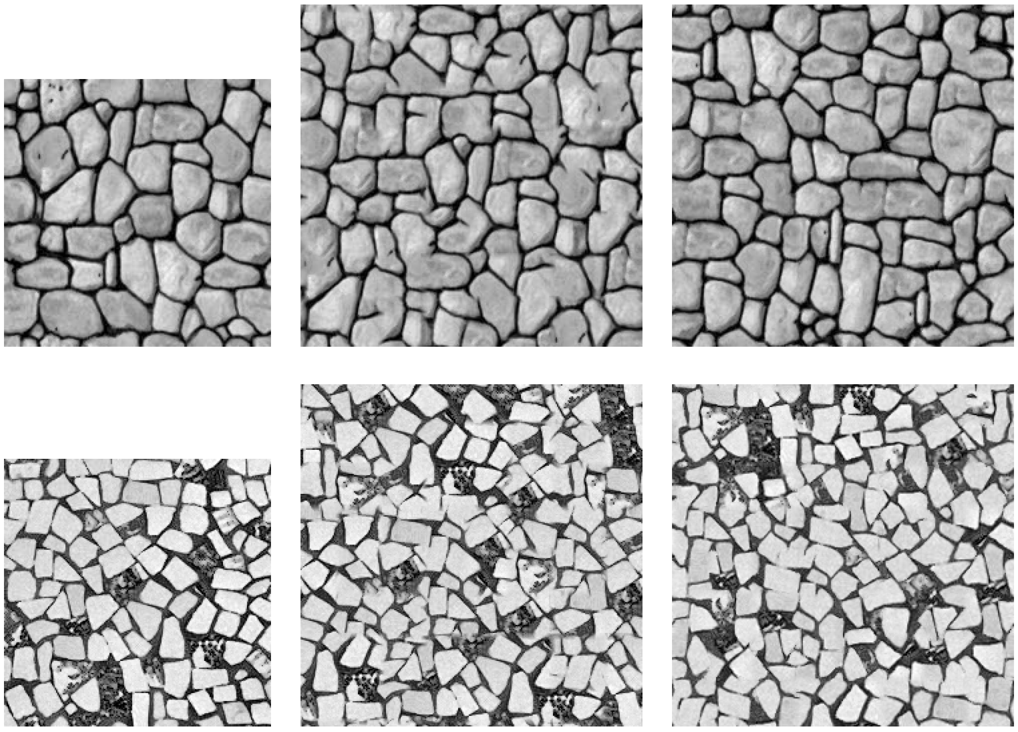

Figure 14: *Comparing synthesis results: the original texture (left), Patch-Based Sampling (32×32 patches, middle), and Hybrid Texture Synthesis. The inputs are 200×200, results are all 256×256. In our results,* $\delta_{max} = 0.02$, $\Delta_{max} = 0.02$, $ov = 12$ *and initial patch size 64×64.*

10. A.A. Efros and W.T. Freeman. Image Quilting for texture synthesis and transfer. *SIGGRAPH 2001 Conference Proceedings*, 341–346, 2001.

11. J.D. Foley, A. van Dam, S.K. Feiner and J.F. Hughes Computer Graphics. Principles and Practice. *Addison Wesley*, 1990.

12. B. Guo, H. Shum and Y.-Q. Xu. Chaos Mosaic: Fast and Memory Efficient Texture Synthesis. *Microsoft Technical Report MSR-TR-2000-32*, 2001.

13. D.J. Heeger and J.R. Bergen. Pyramid-Based Texture Analysis/Synthesis. *SIGGRAPH 95 Conference Proceedings*, 229–238, 1995.

14. A. Hertzmann, C.E. Jacobs, N. Oliver, B. Curless and D.H. Salesin. Image Analogies. *ACM Computer Graphics (Proc. of SIGGRAPH '01)*, 327–340, 2001.

15. T. Leung and J. Malik. Recognizing Surfaces using Three-Dimensional Textons. *Proc. International Conference in Computer Vision*, 1010–1017, 1999.

16. T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using 3d textons. *International Journal of Computer Vision*, **43**(1): 29–44, June 2001.

17. L. Liang. C. Liu, Y.-Q. Xu, B. Guo and H.-Y. Shum. Real-time Texture Synthesis by Patch-Based Sampling. *ACM Transactions on Graphics*, **20**(3): 127–150, July 2001.

18. Y. Liu and Y. Tsin. The Promise and Perils of Near-Regular Texture. *Texture 2002*, 2002.

19. X. Liu, Y. Yu and H.-Y. Shum. Synthesizing Bidirectional Texture Functions for Real-World Surfaces. *SIGGRAPH 2001, Computer Graphics Proceedings*, 97–106, 2001.

20. K. Perlin. An Image Synthesizer. *SIGGRAPH 1985, Computer Graphics Proceedings*, 287–296, 1985.

21. E. Praun, A. Finkelstein and H. Hoppe. Lapped Textures. *Siggraph 2000, Computer Graphics Proceedings*, 465–470, 2000.

22. J. Portilla and E.P. Simoncelli. A Parametric Texture Model based on Joint Statistics of Complex Wavelet Coefficients. *Int'l Journal of Computer Vision*, **40**(1): 49–71, 2000.

23. C. Soler, M.-P. Cani and A. Angelidis. Hierarchical Pattern Mapping. *SIGGRAPH 2002 Conference Proceedings*, 673–680, 2002.

24. X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo and H.-Y. Shum. Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces. *SIGGRAPH 2002 Conference Proceedings*, 665–672, 2002.

25. G. Turk. Texture Synthesis on Surfaces. *SIGGRAPH 2001, Computer Graphics Proceedings*, 347–354, 2001.

26. L.-Y. Wei and M. Levoy. Fast Texture Synthesis Using Tree-Structured Vector Quantization. *Siggraph 2000, Computer Graphics Proceedings*, 479–488, 2000.

27. L.-Y. Wei and M. Levoy. Texture Synthesis Over Arbitrary Manifold Surfaces. *SIGGRAPH 2001, Computer Graphics Proceedings*, 355–360, 2001.

28. L. Ying, A. Hertzmann, H. Biermann and D. Zorin. Texture and Shape Synthesis on Surfaces. *Proc. 12th Eurographics Workshop on Rendering*, 301–312, 2001.

29. S. Zelinka and M. Garland. Towards Real-Time Texture Synthesis with the Jump Map. *Proc. 13th Eurographics Workshop on Rendering*, 2002.

30. S.C. Zhu, Y. Wu and D. Mumford. Minimax Entropy Principles and Its Application to Texture Modeling. *NeurComp*, **9**(8): 1627-1660, 1997.

31. S.C. Zhu, Y. Wu and D. Mumford. FRAME: Towards a Unified Theory for Texture Modeling. *IJCV*, **27**(2): 107-126, March 1998.

32. S.C. Zhu, C.E. Guo, Y. Wu and Y. Wang. What Are Textons?. *ECCV02*, 793 ff., 2002.

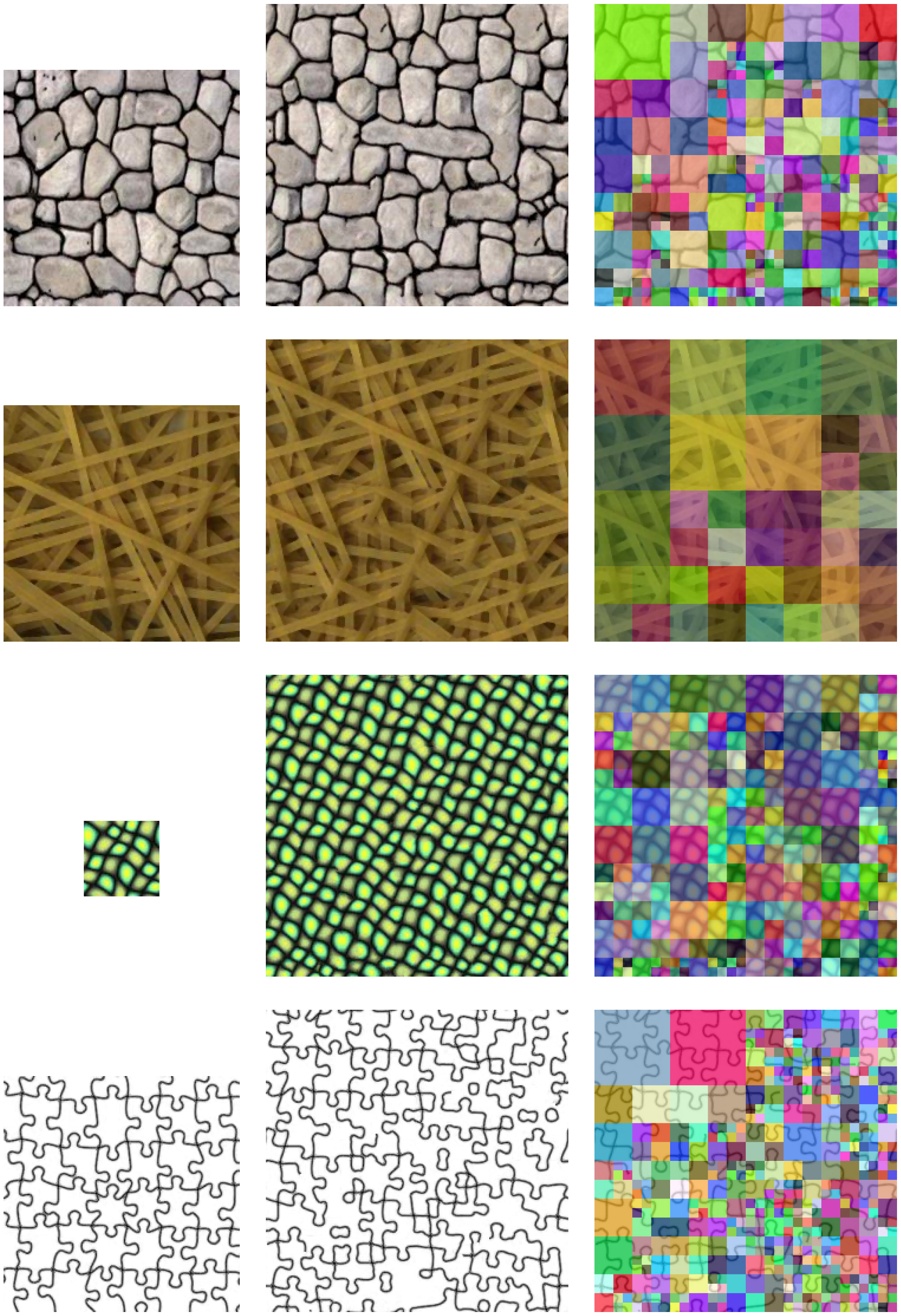

Figure 15: *Hybrid Texture Synthesis results: the original texture (left), the tileable result (middle), and the result with the adaptive sampling grid overlaid (right). The inputs are 200×200 and 64×64, results are all 256×256,* $[\delta_{max}, \Delta_{max}, ov, initial patchsize]$ *from top to bottom:* $[0.02, 0.02, 14, 64 \times 64]$, $[0.02, 0.02, 14, 64 \times 64]$, $[0.01, 0.05, 7, 32 \times 32]$ *and* $[0.04, 0.02, 12, 64 \times 64]$. *The bottom row shows an example where our algorithm fails to produce a satisfying result.*