Inverse Texture Synthesis

Li-Yi Wei[1] Jianwei Han[2*] Kun Zhou[1] Hujun Bao[2] Baining Guo[1] Heung-Yeung Shum[1]

[1]Microsoft Research Asia [2]State Key Lab of CAD&CG, Zhejiang University

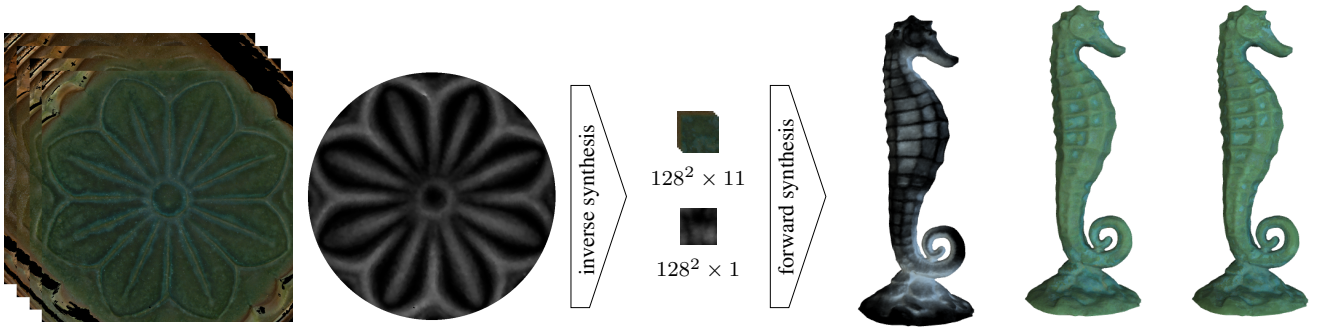


Figure 1: Inverse texture synthesis. Given a large globally-varying texture with an auxiliary control map (patina sequence from [Lu et al. 2007] in this case), our algorithm automatically computes a small texture compaction that best summarizes the original, including both texture and control. This small texture compaction can be used to reconstruct the original texture from its original control map, or to re-synthesize a new texture under a user-supplied control map. Due to the reduced data size, re-synthesis from our compaction is much faster than from the original without compromising image quality (right two images). In this example we use [Kwatra et al. 2005] for forward synthesis, but other algorithms can also be used since our compactions are just ordinary images.

Abstract

The quality and speed of most texture synthesis algorithms depend on a 2D input sample that is small and contains enough texture variations. However, little research exists on how to acquire such a sample. For homogeneous patterns this can be achieved via manual cropping, but no adequate solution exists for inhomogeneous or *globally varying* textures, i.e. patterns that are local but not stationary, such as rusting over an iron statue with appearance conditioned on varying moisture levels.

We present *inverse texture synthesis* to address this issue. Our inverse synthesis runs in the opposite direction with respect to traditional forward synthesis: given a large globally varying texture, our algorithm automatically produces a small texture compaction that best summarizes the original. This small compaction can be used to reconstruct the original texture or to re-synthesize new textures under user-supplied controls. More important, our technique allows real-time synthesis of globally varying textures on a GPU, where the texture memory is usually too small for large textures. We propose an optimization framework for inverse texture synthesis, ensuring that each input region is properly encoded in the output compaction. Our optimization process also automatically computes orientation fields for anisotropic textures containing both low- and high-frequency regions, a situation difficult to handle via existing techniques.

Keywords: texture synthesis, texture mapping, GPU techniques

*This research was conducted during Jianwei Han's internship at MSRA.

1 Introduction

Recent years have witnessed significant progress of example-based texture synthesis algorithms. Most of these techniques rely on the assumption that the input texture be homogeneous, i.e. local and stationary in the Markov-Random-Field definition [Efros and Leung 1999; Wei and Levoy 2000]. However, not all textures are homogeneous, as natural patterns often exhibit global variations conditioned on environment factors such as iron rusting following the moisture level over a statue. We term such textures that are local but not stationary *globally varying* textures, and the environment factors that determine the global texture distribution the *control map*. Examples of control map include the context information [Lu et al. 2007], the spatial-varying parameters [Gu et al. 2006], and the degree map [Wang et al. 2006]. With the advance of data capturing technologies as well achievable synthesis effects, globally varying textures are becoming more and more important [Wang et al. 2006; Gu et al. 2006; Lu et al. 2007].

One major problem for globally varying textures is their data size, as they need to cover sufficiently large surface area for capturing global variation. For example, a drying-cloth BRDF texture from [Gu et al. 2006] can be as large as 512 (width) $\times$ 512 (height) $\times$ 33 (time) with storage size 288 MB and a crackling-paint color texture from [Lu et al. 2007] can be 1226 $\times$ 978 $\times$ 50 with storage size 35 MB. Such large textures can cause problems for storage, computation, and transmission. These problems can be alleviated if we could find a small texture sample from which the original texture can be represented. This small sample can be easily obtained via manual cropping for homogeneous textures. Unfortunately, manual cropping does not work for globally varying textures as no small window can cover all relevant information. An example is demonstrated in Figure 2.

We present a new technique termed *inverse texture synthesis* to address this issue. Our inverse synthesis runs in the opposite direction with respect to traditional texture synthesis: given a large globally-varying texture with an associated control map, our algorithm automatically computes a small *texture compaction* that encodes both the texture details and the control map of the original

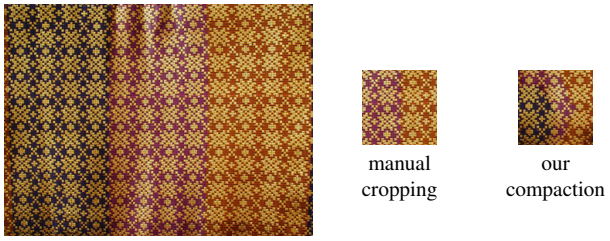


Figure 2: Comparison of cropping and our technique. Note that due to the globally varying nature of the original, it is impossible to perform a manual cropping that captures the major characteristics of the original texture (no matter where you put the window). Our method, in contrast, encodes the major features of the original into a small compaction via inverse synthesis.

(Figure 1). This small texture compaction can be used to reconstruct the original texture from its original control map (Figure 8) or to re-synthesize new textures under a user-supplied control map (Figure 11). Our technique is inspired by the pioneering work of epitome [Jojic et al. 2003] (and follow-ups such as jigsaw [Kannan et al. 2007]) and in terms of representation our compaction is just a variation of epitome.[1] However, so far these techniques have yet to provide satisfactory quality for graphics applications. We present a different algorithm that targets only textures and is thus less general than [Jojic et al. 2003] but produces better results for textures; see Figure 3 and Section 2&3 for more details.

More importantly, our small compaction allows real-time synthesis of globally varying textures on a GPU, where the texture memory is usually too small for large textures. Furthermore, in contrast to previous GPU techniques designed primarily for homogeneous input textures (e.g. [Lefebvre and Hoppe 2005; Lefebvre and Hoppe 2006]), our technique allows interactive user painting of globally varying textures, combining the generality of example-based texture synthesis as well as the controllability and flexibility of WYSIWYG-style user painting [Hanrahan and Haeberli 1990; Ashikhmin 2001; Hertzmann et al. 2001; Ritter et al. 2006].

We propose an optimization framework for inverse texture synthesis. Optimization has been utilized for forward texture synthesis as in [Kwatra et al. 2005] which casts the neighborhood search process in [Efros and Leung 1999; Wei and Levoy 2000] as optimizing an energy function. We employ optimization for a completely different purpose: to ensure that each neighborhood of the input texture has a presence in the output compaction. We achieve this by minimizing an *inverse energy function* that measures the similarity between each input neighborhood and its best match from the output compaction.

Optimization with the inverse energy function is a major technical challenge. Since our similarity measurement is performed in the reverse direction with respect to traditional texture synthesis, existing acceleration techniques such as tree-structure [Wei and Levoy 2000; Kwatra et al. 2005] are not applicable as they are designed for static images, whereas our compaction changes dynamically along with the optimization process. We address this issue by proposing a novel optimization solver that pre-processes the input neighborhoods into a small number of clusters [Wei and Levoy 2000] and performs measurement only for the center of each cluster. Measurements for the rest of input neighborhoods are then conducted through their cluster centers in constant time, thanks to the transitive property of neighborhood similarity,

Another non-trivial issue is dealing with anisotropic textures with non-uniform orientations. A good orientation field not only yields compaction with higher quality/size ratio but also provides better user control for orienting synthesized textures. One possible method to compute this orientation field is via manual specification at a few key locations followed by interpolation (e.g. [Turk 2001]), but this can be tedious and inaccurate. Another common approach is to rely on high frequency details (e.g. [Perona and Malik 1990; Ziou and Tabbone 1998; Paris et al. 2004]), but this is not applicable to low frequency regions. Our energy minimization framework automatically computes this orientation field, as a good orientation field usually yields lower energy value. As our approach utilizes spatial neighborhoods as the only metric, it works well for textures with both low and high frequency regions.

[1] Since *epitome* may refer either to the representation or the original algorithm [Jojic et al. 2003], we use the term *compaction* to avoid confusion.

2 Previous Work

Forward texture synthesis has made significant progress recently, with applications ranging from surface texturing [Turk 2001; Wei and Levoy 2001], animation [Bargteil et al. 2006; Kwatra et al. 2007], image editing [Efros and Freeman 2001; Drori et al. 2003; Fang and Hart 2004; Liu et al. 2004], and time varying phenomena [Wang et al. 2006; Gu et al. 2006; Lu et al. 2007]. The core algorithms of these techniques can be classified as being either local [Efros and Leung 1999; Wei and Levoy 2000; Kwatra et al. 2005] or global [Heeger and Bergen 1995; Portilla and Simoncelli 2000; Qin and Yang 2005] depending on the texture statistics/characteristics used. Both categories of forward synthesis techniques cannot be directly used for our inverse synthesis, as local techniques might not retain all the original features (e.g. Figure 6), whereas a global technique often has problem retaining local texture details.

Despite their success, most existing forward synthesis algorithms have limited pattern variation and computation speed as they have been primarily concerned with synthesizing homogeneous textures on a CPU. The speed issue has been addressed by parallel GPU texture synthesis [Lefebvre and Hoppe 2005; Lefebvre and Hoppe 2006] which runs much faster than CPU-based algorithms. A further advantage of GPU synthesis is reduced storage; this is very important for real-time applications as a commodity GPU often has limited texture memory. However, no GPU algorithms so far could support globally-varying synthesis as demonstrated in [Wang et al. 2006; Gu et al. 2006; Lu et al. 2007].

The pattern variation issue has been addressed by recent advances in globally varying texture synthesis. These methods either use stationary inputs and establish artificial correspondence for synthesis [Matusik et al. 2005; Zhang et al. 2003] or synthesize directly from captured patterns along with control maps [Wang et al. 2006; Gu et al. 2006; Lu et al. 2007]. Although the former method could produce interesting morphing or transition patterns, they often lack the ground truth which can be provided by the later approach. However, a disadvantage of captured globally varying textures is their large size, causing memory and speed problems for synthesis. This is particularly harmful for GPU performance.

We address both the quality and speed issues by our inverse texture synthesis framework as well as an accompanying GPU forward synthesis algorithm that is applicable to globally varying textures.

One possible alternative to reduce texture size is texture analysis, extracting a set of parameters characterizing a given texture [Heeger and Bergen 1995; Portilla and Simoncelli 2000; Qin and Yang 2005]. Such a set of parameters, if small enough, could serve as the ultimate compaction. However, so far these techniques have been designed for stationary, but not globally varying, textures. Another possibility is to summarize a texture into a set of textons (e.g. [Popat and Picard 1997; Leung and Malik 2001]) and use that for synthesis. However, we prefer to use a compacted image form since it is more general (i.e. can feed as input to any forward texture synthesis algorithm). This is particularly important for GPU im-

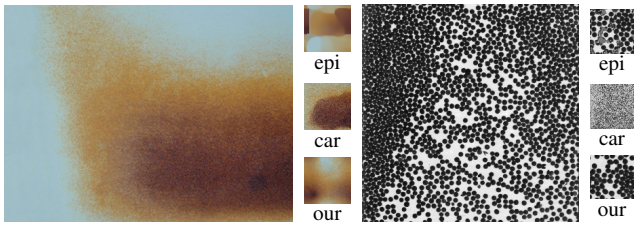


Figure 3: Comparison of our technique with epitome and seam carving. For each group of images, the original is shown on the left, the result by epitome [Jojic et al. 2003] on upper right, the result by seam carving [Avidan and Shamir 2007] on middle right, and our result on lower right. The epitome results are produced with parameters yielding the best quality. Note that epitome can produce either blur (right case) or discontinuity (left case). The seam carving results are produced with the e_1 energy function as recommended in [Avidan and Shamir 2007]. They tend to be noisy.

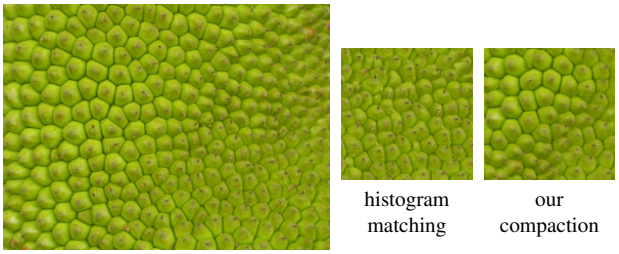


Figure 4: Comparison of histogram matching and our technique. Histogram matching as utilized in [Kopf et al. 2007] can preserve color histograms but not structures with similar colors. Our method preserves both.

plementations such as [Lefebvre and Hoppe 2005; Lefebvre and Hoppe 2006] where an image input is required.

[Jojic et al. 2003; Kannan et al. 2007] summarize local patch properties of a given image into an epitome or jigsaw, a process that has significantly inspired our research. However, these techniques use patches to summarize general images, which is different from our method of using neighborhoods for texture images. Consequently, their results are often not good enough for reconstruction as shown in Figure 3 and [Kannan et al. 2007](Figure 3). Furthermore, since these techniques possess no equivalent notions for global variance and control maps, there is no direct applicability for re-synthesis. (Their reconstruction is achieved by simply recording the explicit patch mappings between the original and the epitome/jigsaw.) We provide a more detailed discussion in Section 3 after introducing the main ideas of our algorithm. As a concurrent work, [Wang et al. 2008] factors repeated image content via epitome. This technique is targeted for compression whereas ours for synthesis.

Our technique is also related to seam carving [Avidan and Shamir 2007] for content-aware image resizing. But seam carving is optimized for preserving general image saliency, not textures which may contain non-salient features. Consequently compactions computed by seam carving tend to be noisy. [Kopf et al. 2007] preserves global statistics by integrating histogram matching [Heeger and Bergen 1995] with optimization [Kwatra et al. 2005], but this works only for color but not general structure information. Our technique preserves general neighborhood information (including both color and structure). See Figure 4 for comparison.

3 Overview

We cast our inverse synthesis process as an optimization problem. Specifically, given an original texture X, our goal is to calculate a small texture compaction Z with user-specified size, minimizing

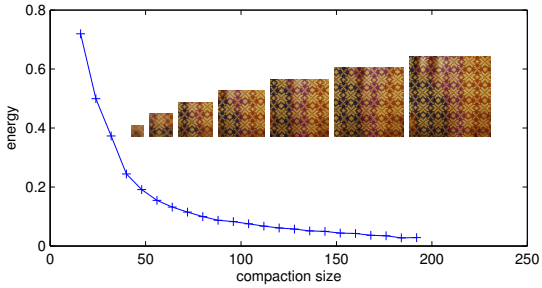


Figure 5: Energy function plot. The horizontal axis is compaction size (in pixels), whereas the vertical axis is energy value normalized with the maximum possible value.

the following energy function:

$$\Phi(\mathbf{x};\mathbf{z};\mathbf{w}) = \frac{1}{|X^\dagger|}\sum_{p\in X^\dagger}|\mathbf{x}_p(\mathbf{w}_p)-\mathbf{z}_p|^2 + \frac{\alpha}{|Z^\dagger|}\sum_{q\in Z^\dagger}|\mathbf{x}_q(\mathbf{w}_q)-\mathbf{z}_q|^2 \quad (1)$$

where $\mathbf{z}/\mathbf{x}$ represents the sample values for Z/X, $\mathbf{w}$ is the orientation field for X, q/p runs through a subset $Z^\dagger/X^\dagger$ of Z/X, $\mathbf{x}_p/\mathbf{z}_q$ indicates the spatial neighborhood around p/q, $\mathbf{w}_p/\mathbf{w}_q$ the local orientations at p/q from which $\mathbf{x}_p/\mathbf{x}_q$ is sampled from, $\mathbf{z}_p/\mathbf{x}_q$ is the most similar neighborhood in Z/X with respect to $\mathbf{x}_p/\mathbf{z}_q$, and α is a user tunable weighting. In our experiments, we have found that $\alpha = 0.01$ works well for most textures we have tried.

The energy function value varies depending on the compaction size; as expected, the larger the size of the compaction, the lower the error and therefore the energy value. An example is illustrated in Figure 5. The compaction size can either be picked by the user according to her specific application needs, or automatically determined by our simple heuristic as discussed in Section 5.

As shown in Equation 1 the energy function consists of two terms. Although having similar forms, they serve completely different purposes. We illustrate the necessity of both terms in Figure 6.

The first term measures the local similarity for a set of samples in X with respect to Z. By calculating an Z that minimizes this energy term, we attempt to ensure that for every input neighborhood $\mathbf{x}_p$, we could find a corresponding compaction neighborhood $\mathbf{z}_p$ that is similar to $\mathbf{x}_p$. We name the first term the *inverse* term due to its inverse synthesis nature. Without this inverse term, the resulting compaction might miss important features from the original, as shown in Figure 6.

The second term measures the local similarity for a set of samples in Z with respect to X. It is similar to the energy function in [Kwatra et al. 2005] for forward synthesis. The reason for incorporating this *forward* term is that Z computed from the inverse term alone may contain problems for re-synthesis. For example, it might happen that all the original sample neighborhoods $\{\mathbf{x}_p\}$ map to a corner of Z, causing garbage in other regions. $\{\mathbf{x}_p\}$ may also map to disjoint regions of Z causing discontinuity across region boundaries. Both problems cannot be detected by the inverse term but can be eliminated by the forward term. See Figure 6 for examples.

Using this energy function framework, the methods by epitome [Jojic et al. 2003] and jigsaw [Kannan et al. 2007] can be related to our technique as follows. First, their framework does not have an explicit forward energy term. Second, they optimize for patches instead of neighborhoods as in our approach. Due to these two reasons, epitome and jigsaw can produce blur and discontinuity artifacts (Figure 3) similar to the ones produced by our solver with only inverse energy term (Figure 6). In Figure 3, we have strived

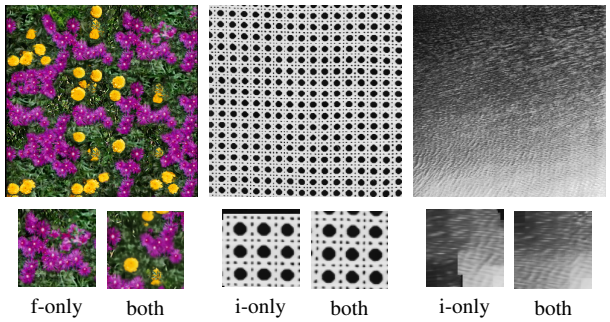


Figure 6: Why we need both forward and inverse terms in our energy function. With only the forward term the compaction will not provide sufficient coverage of the original (left case). With only the inverse term the compaction may contain garbage (middle case) or discontinuity (right case). All compactions are drawn in larger scale than the originals for clarity.

to optimize the parameters for epitome to produce the best results. But in general we have found it difficult to achieve our results due to the nature of epitome which uses a blend kernel to control the reconstruction: when the kernel is too large, the epitome tends to be too blurry. (This blur problem is also described in [Kannan et al. 2007].) When the kernel is too small, the epitome may contain holes or discontinuities.

If X is defined over a 3D surface rather than a 2D grid, our algorithm samples $\mathbf{x}_p$ via local flattening as in [Wei and Levoy 2001]. As a result, our methodology does not require global or even large scale parameterization for surface textures, and can therefore produce a 2D compaction with little distortion.

For an anisotropic texture with non-uniform orientation our algorithm also computes the orientation field $\mathbf{w}$ automatically as part of the optimization process. We have observed that a $\mathbf{w}$ that results in lower energy values usually yields compaction with higher quality/size ratio. For isotropic textures we leave $\mathbf{w}$ as a constant; i.e. a regular grid for 2D textures and a smooth orientation field for surface textures [Turk 2001].

Since our output compactions are just ordinary images, they can be fed directly into any texture synthesis algorithm for re-synthesis. For homogeneous patterns all we need is the texture information. However, for globally varying textures, a control map is also required for user-controllable synthesis. The precise semantics and usage of the control map depend on the specific algorithms; some examples include the user specifications in [Hertzmann et al. 2001; Ashikhmin 2001], context information in [Lu et al. 2007], the spatially-varying parameters in [Gu et al. 2006], and the degree map in [Wang et al. 2006]. All such control maps are naturally handled by our algorithm, as we treat each pixel as a generic vector that may contain color, control, or any other auxiliary information.

4 Solver for Inverse Texture Synthesis

Our inverse texture synthesis is achieved by solving Equation 1. In Section 4.1, we first describe a straightforward solver. Although our basic solver is capable of achieving good quality for forward synthesis, it has both speed and quality issues for our inverse synthesis. We address these issues in Section 4.2 with our improved solver. For easy reference we have summarized our solver in Table 1.

Note that similar to previous texture synthesis work, our algorithm is multi-resolution and computes the output compaction from lower to higher resolutions. Since we apply identical algorithms for each resolution, below we only describe algorithms for computing one resolution.

Inverse Texture Synthesis

$\mathbf{z}_p^0 \leftarrow$ random neighborhood in $Z \; \forall$ p $\in X^\dagger$
$\mathbf{x}_q^0 \leftarrow$ random neighborhood in $X \; \forall$ q $\in Z^\dagger$
$\mathbf{w}^0 \leftarrow$ init from [Paris et al. 2004] and/or manual touch
for iteration m = 0:M // *for* **w**
  for resolution ℓ = 0:L
    if $\ell < $ L $\mathbf{w} \leftarrow$ downsample from **w** at L
    if $\ell > 0$ $\mathbf{z} \leftarrow$ upsample from **z** at ℓ-1
    for iteration n = 0:N
      $\mathbf{z}^{n+1} \leftarrow \text{argmin}_{\mathbf{z}} \; \Phi(\mathbf{x}; \mathbf{z}; \mathbf{w})$ // **z** *E-step*
      foreach p $\in X^\dagger$
        $\mathbf{z}_p^{n+1} \leftarrow \text{argmin}_{\mathbf{z}_p} \; |\mathbf{x}_p(\mathbf{w}_p) - \mathbf{z}_p|^2$ // *inverse M-step*
      foreach q $\in Z^\dagger$
        $\mathbf{x}_q^{n+1} \leftarrow \text{argmin}_{\mathbf{x}_q} \; |\mathbf{x}_q(\mathbf{w}_q) - \mathbf{z}_q|^2$ // *forward M-step*
      if $\mathbf{z}_p^{n+1} == \mathbf{z}_p^n \; \forall$ p $\in X^\dagger$
        $\mathbf{z} \leftarrow \mathbf{z}^{n+1}$
        if $\ell ==$ L // *highest resolution*
          $\mathbf{w}^{n+1} \leftarrow \text{argmin}_{\mathbf{w}} \; \Phi(\mathbf{x}; \mathbf{z}; \mathbf{w})$ // **w** *E-step*
          if $\mathbf{w}^{n+1} == \mathbf{w}^n$
            $\mathbf{w} \leftarrow \mathbf{w}^{n+1}$
            return
          end if
        end if
      end if
    end for
  end for
end for

Table 1: Pseudocode of our algorithm. The energy function Φ is defined in Equation 1. Solving for **w** is only necessary for anisotropic textures with non-uniform orientation. Note that Φ never increases at each E and M steps of our solver.

4.1 Basic Solver

Our basic solver is inspired by texture optimization [Kwatra et al. 2005], but since our energy function contains both forward and inverse terms we have to provide a different solver. Details are as follows.

The core part of the solver is marked as E-steps and M-steps in Table 1. (Note: our solver is not exactly expectation-maximization (EM), but we follow the usage of E/M-steps as in [Kwatra et al. 2005] for clarity.) At E-steps, we solve for **z**/**w** to minimize the energy function (covering both energy terms simultaneously). At M-steps, we search for each neighborhood $\mathbf{x}_p$/ $\mathbf{z}_q$ on X/Z the most similar neighborhood $\mathbf{z}_p$/$\mathbf{x}_q$ on the Z/X. The output of one step feeds as input to the other, and we iterate this process several times until convergence or a pre-determined number of iterations is reached.

The original optimization solver [Kwatra et al. 2005] utilized tree search for the M-step and least square for the **z** E-step. Unfortunately, both incur problems for our inverse synthesis. For the M-steps, we have found that tree search is too slow for large input textures. In addition, tree (or any other similar pre-processed acceleration data structures) is not applicable to the inverse M-step since the output Z is constantly changing. For the **z** E-step, least square solver could cause excessive blur as pointed out by [Han et al. 2006]. Furthermore, previous methods have no correspondence for the **w** E-step which involves finding optimal local orientations. We address these issues via our improved solver below.

4.2 Improved Solver

Our goal is to provide a solver that incurs no blur in the **z** E-step and consumes constant time per pixel search for both the forward and

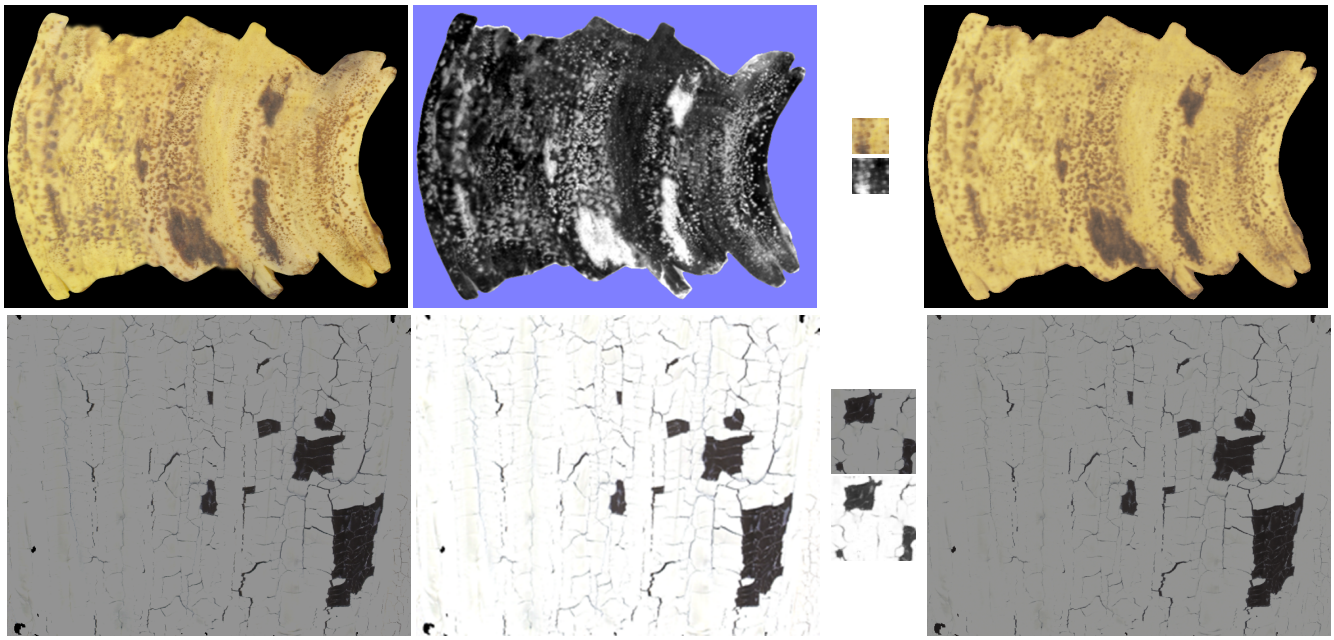


Figure 8: Reconstruction of the original from our compaction. Within each group of images, the original and control are on the left, the reconstruction is on the right, and our compaction (including both texture and control) is in the middle. Each reconstruction is computed from the compaction according to an original control map (with quarter resolution of the color) serving as constraints in [Kwatra et al. 2005]. From top to bottom: banana from [Wang et al. 2006] (original 720×540, compaction 64^2), paint peeling from [Lu et al. 2007] (original 615×488, compaction 128^2). In both cases the control map was originally in quarter resolution of the color texture but was up-sampled before feeding into our inverse synthesis algorithm. Note that our reconstruction results remain faithful to the original even with coarser control maps.

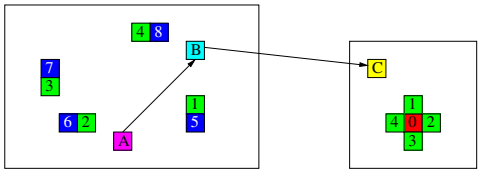


Figure 7: Illustrating of our improved solver. Here, we use a toy case with only four k-coherence neighbors as exemplified in pixels {**1, 2, 3, 4**} around **0** in the compaction. The sources of these four pixels are marked with the same numbers in the input. **z** E-step: the value of **0** is chosen from {**5, 6, 7, 8**} as determined by **0**'s neighbors {**1, 2, 3, 4**}. Forward M-step: the best match for **0** is also chosen from {**5, 6, 7, 8**}. Inverse M-step: **B** is a cluster center where **A** belongs to. So **B** first finds the best match **C** through exhaustive search, and the best match for **A** is determined through **B**.

case	basic solver				improved solver			
	z-E	f-M	i-M	all	**z**-E	f-M	i-M	all
banana	4	73	70	147	7	4	3	14
peeling	11	922	704	1637	47	48	167	263
rust	2	22	23	47	2	3	2	7
crack	5	135	200	340	16	17	17	50

Table 2: Timing information comparing our basic and improved solvers. The top 2 cases are shown in Figure 8 and the bottom 2 in Figure 11. All timing numbers are in units of seconds and are decomposed into **z** E-step, forward M-step, and inverse M-step. All measurements are performed on a PC with an Intel Xeon Dual-core 3.73GHz CPU and a 4GB RAM.

inverse M-steps. Our method is inspired by k-coherence [Tong et al. 2002] and clustering [Wei and Levoy 2000]; details are described below.

Preprocess During preprocess, we compute a k-coherence similarity-set **s**(p) for each input pixel p, where **s**(p) contains a list of other pixels with neighborhoods similar to p. The size of the similarity-set, K, is a user-controllable parameter that determines the overall speed/quality. **s**(p) will be utilized for our **z** E-step and forward M-step as detailed below. We also perform a clustering of the input neighborhoods via TSVQ [Wei and Levoy 2000] and collect the cluster centers into a set X_c. For each input pixel p, we find a subset **c**(p) of X_c with most similar neighborhood to p. X_c and **c**(p) will be utilized for the inverse M-step as detailed below.

We also initialize **w** using [Paris et al. 2004] and/or limited manual specifications. Even though these methods might not yield good orientations at every input pixel location, they provide better initial conditions to help our subsequent optimizations.

z E-step To address the blur issue, we adopt a discrete solver as inspired by [Han et al. 2006]. Instead of least square, we only allow direct copy of sample values from input X to compaction Z. During the copy operation, we not only copy the color plus control information, but also the source location of each copied pixel. Specifically, to compute $\mathbf{z}^{n+1}$ in E-step, each one of its values **z**(q) at pixel q is determined independently from each other. For each q, we first construct its k-coherence candidate set **k**(q) by taking the union of similarity sets $\{\mathbf{s}(q_i)\}$ from q's spatial neighbors $\{q_i\}$ (plus proper shifting as detailed in [Tong et al. 2002]). **z**(q) for the next iteration is then chosen from **k**(q) as the one that most reduces the energy function. Since now each **z**(q) is copied direct from some input pixel, we avoid the blur issue in least square.

Forward M-step Since pixels are directly copied in the **z** E-step, we can retain the input location information to conduct k-coherence search in forward M-step. Specifically, for each output neighborhood $\mathbf{z}_q$ at pixel q, we determine its best match $\mathbf{x}_q$ from the input as the one in **k**(q) (constructed in the same method as in **z** E-step) that is most similar to $\mathbf{z}_q$. Since this is a constant time operation, it is

much faster than tree search, a logarithmic operation as described in [Kwatra et al. 2005].

Unfortunately, this k-coherence acceleration cannot be applied to the inverse M-step, as the compaction Z is constantly changing. We address this issue below.

Inverse M-step At inverse M-step, we need to determine, for each input neighborhood $\mathbf{x}_p$, the best match $\mathbf{z}_p$ at compaction Z. One method is to perform an exhaustive search through Z for each $\mathbf{x}_p$, but this can be expensive if Z is sufficiently large. Unfortunately, as Z is constantly changing, this search cannot be accelerated via traditional methods that require preprocessing (such as kd-tree or TSVQ). Furthermore, even if Z is small enough to allow exhaustive search, repeat this process for all input neighborhoods is still computationally expensive due to the large input size.

We provide a new acceleration that can address both issues of a constantly changing output and a large input. The basic idea is as follows. Instead of search through Z for each $\mathbf{x}_p$ in X, we perform direct search for only a subset X_c of X. For each p not in X_c, instead of direct search, we find its best match in Z indirectly through a set of intermediaries in X_c. In this scheme, we achieve speedup by performing direct search for only a subset X_c of X, and guarantee quality via the transitive property of neighborhood similarity. Details are as follows.

X_c **construction** Intuitively, X_c should contain candidates that best represent the input neighborhoods. We achieve this by clustering the input neighborhoods via TSVQ [Wei and Levoy 2000], and construct X_c as the set of cluster centers. A minor difference between [Wei and Levoy 2000] and our approach is that we define the cluster centers via median, not average. This is to ensure good search results through Z. During run time, at the beginning of each inverse M-step, we perform an full search for each member of X_c.

Indirect search For each p not in X_c, we first find a subset $\mathbf{c}$(p) of X_c with most similar neighborhood to p. This is done only once as a pre-process. During run-time, after a full search is performed for X_c, we only search the possible output locations of $\mathbf{c}$(p) to find the best match for $\mathbf{x}_p$. Note that this is a constant time operation since $\mathbf{c}$(p) has a fixed size.

We can trade off quality/speed of this search algorithm by tuning its parameters, including the sizes of X_c and $\mathbf{c}$(p). Note that when $X_c \simeq X$, our method would reduce to full search. In our experiments, we choose the size of X_c to be roughly equivalent to the compaction size Z, with the rational being that Z should contain a good representation of input cluster centers. In our experiments, we have found this heuristic effective, and we usually set $\mathbf{c}(p) = 1$. To further facilitate varying sizes of Z, we use TSVQ to build a tree [Wei and Levoy 2000] and construct X_c via tree cut to achieve the optimal rate/distortion ratio [Gersho and Gray 1991]. In this scenario, the TVSQ tree is built only once for each input and can be repeatedly utilized for varying Z size.

w E-step Our **w** E-step refines the initial orientation field as part of our optimization process. As shown in Table 1, our solver updates the orientation field **w** only at the highest pyramid resolution after **z** stabilizes. There are two reasons for this. First, unlike **z** which starts with a totally random initialization, **w** starts with a reasonably good initial condition as described in the preprocess step. Consequently, our solver only needs to refine **w** instead of computing it from scratch. Second, empirically we have found that updating **w** only at the highest resolution after **z** stabilizes yields best results.

We now describe how we actually perform this refinement. A naive approach is to repeatedly rotate each $\mathbf{w}_p$ and resample $\mathbf{x}_p(\mathbf{w}_p)$ until $\text{argmin}_{\mathbf{w}}\ \Phi(\mathbf{x};\mathbf{z};\mathbf{w})$ is found, but this is not only computationally expensive but also prone to produce disoriented results. Instead, we compute $\mathbf{w}_p$ iteratively and within each iteration we only consider orientations within an interval $[\overline{\theta}\text{-}\Delta\theta, \overline{\theta}\text{+}\Delta\theta)$, where $\overline{\theta}$ is initialized as the average of p's spatial neighbors and updated as best orientation of $\mathbf{w}_p$ computed at the end of iteration, and $\Delta\theta$ is initialized as 90 degrees and halved at the end of each iteration. Within each iteration we sample 36 uniformly spaced samples from $[\overline{\theta}\text{-}\Delta\theta, \overline{\theta}\text{+}\Delta\theta)$, find out the three orientations yielding minimum energy function, and choose the one that is closest to $\overline{\theta}$. (Note that we do this instead of simply choosing the best one to avoid disoriented orientation fields; i.e. we introduce a sense of "regularization term" into our computation.) We perform three iterations per p. After **w** is updated for the entire input, we conduct a bilateral filtering similar to [Paris et al. 2004] for final smoothing.

Since input neighborhoods are resampled according to **w**, after **w** is changed at the end of each **w** E-step all our acceleration data structures such as X_c, $\mathbf{c}$(p) and $\mathbf{s}$(p) will be out of date. Although it is possible to incrementally update these data structures, in our current implementation we simply forgo all these accelerations and use a brute force optimization.

5 Results and Discussion

Quality and speed The first thing we need to verify is quality and speed of our approach. Quality-wise, our algorithm achieves high data size reduction while allows faithful reconstruction of the original, as shown in Figure 8. In addition to reconstruction of original textures, our algorithm can also be used to re-synthesize novel textures under user-supplied controls as demonstrated in Figure 11. As shown, re-synthesis from our compaction preserves visual quality while runs much faster than re-synthesis from the original. Note that our algorithm does not require a detailed control map for all textures (e.g. the control map is much coarser than the paint crack pattern in Figure 11). Also, even when the control map is in the same resolution as the color texture, our algorithm still provides significant data size reduction for time sequence textures such as those in [Lu et al. 2007]. Regarding inverse synthesis speed, our improved solver (Section 4.2) is more efficient than our basic solver (Section 4.1), as demonstrated in Table 2.

Limitations For globally varying textures, we have found that our algorithm works well when the original control map is reasonably correlated with the original texture pattern. If not, we might not be able to accurately recover the original from our compaction; an example is shown in Figure 9. Also, our algorithm cannot guarantee to compress all textures well. If a texture has little redundancy to begin with, our algorithm will not be able to produce a significantly smaller compaction without sacrificing re-synthesis quality. An example is shown for the paint crack texture in Figure 11 where re-synthesis from our compaction has a more accentuated appearance than re-synthesis from the original. The quality can be improved by using larger compactions.

Control map For source textures from [Gu et al. 2006] and [Lu et al. 2007], we use the original control maps (i.e. spatially varying parameters in [Gu et al. 2006] and context information in [Lu et al. 2007]) provided by the corresponding authors.[2] For the rest, we compute the control maps via the method described in [Wang et al. 2006].

[2]The only exception is the peeling texture in Figure 8 where the original control map is problematic as demonstrated in Figure 9. Consequently we compute a better control map via [Wang et al. 2006].

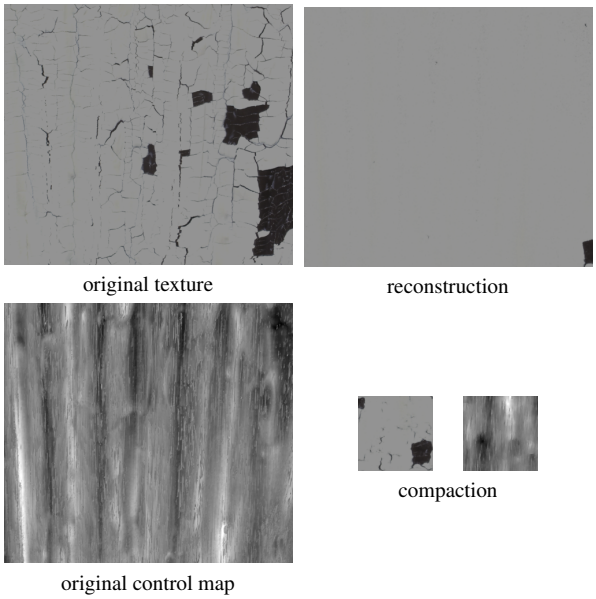


Figure 9: A failure case. The paint peeling texture and its original control map are both from [Lu et al. 2007]. Due to the low correlation between the actual texture pattern and the control map, our technique is unable to reconstruct the original texture at all.

Figure 10: GPU synthesis for globally varying textures. Top: cheese mold. Bottom: dirt accumulation. (Both from [Lu et al. 2007].) The left column shows the original textures (cheese: 1214×1212, dirt: 271×481) while the right columns show two frames of GPU renderings with user control maps for aging effects. Our compactions have size 128^2 and the synthesized textures have size 512^2. The synthesis speeds are 3.0/3.5 fps from the originals and 6/7 fps from our compactions. All frame rates are measured on a NVIDIA Quadro FX 4500 chip.

Parameters One of the key parameters of our inverse synthesis algorithm is the compaction size. In Figure 5, we plot the energy function value with respect to compaction size for a typical texture. Notice two things here. First, the energy function decreases with the increase of compaction size, as expected. Second, a good choice of compaction size lies somewhere around the knee point of the energy curve. However, determining the final compaction size by the aforementioned approach is time consuming as we need to plot the entire curve. In our experiments, we have been employing the following heuristic that estimates the optimal compaction size M (in pixel^2) from the number of input clusters N (computed by TSVQ as described in Section 4.2) via the following formula: $\mathrm{M} = 0.25 \times \mathrm{N}$. We build the tree with an error bound $\epsilon = 0.05 \times$ maximum neighborhood distance. We have found that this heuristic works well in practice.

Real-time GPU synthesis We have attempted two methods for real-time globally varying texture synthesis on a GPU. Our first method is an extension of image analogies [Hertzmann et al. 2001] for GPU synthesis [Lefebvre and Hoppe 2005]. Specifically, in the jargon of [Hertzmann et al. 2001], we treat the original control map as the unfiltered input, the original texture as filtered input, the output control map as the unfiltered target, and the output texture as the filtered target to be computed, and then perform all the synthesis computations on a GPU via [Lefebvre and Hoppe 2005] where we encode the control map as additional channels beyond the color texture. (The control map channels are read-only and not modified during synthesis.) In the second method, we utilize the discrete optimization in [Han et al. 2006] by encoding the control map as a constraint in the energy function. Since [Han et al. 2006] essentially replaces the least squares solver in [Kwatra et al. 2005] with a k-coherence solver, it can be efficiently implemented on GPU. In our experiments we have found that the second approach yields slightly better quality but the first approach is much faster. Consequently, we adopt the first approach for GPU synthesis of globally varying textures.

Our algorithm allows real-time, user controllable synthesis of globally varying patterns as shown in Figure 10. Rendering from our compaction produces similar visual quality as from the original. In addition, due to reduced GPU memory access and consumption, rendering from our compaction is more efficient.

Orientation field Figure 12 demonstrates our computation of orientation fields as well the impact on output compactions. As shown, orientation fields generated by our approach are better than those by edge-based techniques [Paris et al. 2004], which is less effective in low frequency regions, and manual specification (followed by interpolation), which is tedious and inaccurate. Furthermore, the quality of the orientation field has a direct impact on inverse synthesis quality. As shown in the figure, a good orientation field usually produces compactions with higher quality given a fixed compaction size.

6 Conclusions and Future Work

We present inverse texture synthesis to compute a small texture compaction from a large globally varying input texture. The small compaction can be used to reconstruct the original texture via the original control map or to re-synthesize novel textures under user-supplied controls. Due to the reduced size, it is more economic to store, transmit, and synthesize from our texture compaction than from the original. Our reduced compaction is particularly beneficial for GPU applications and we further propose a technique for real-time GPU synthesis of globally varying textures.

For future work, our inverse texture synthesis can be directly extended to the following applications. Instead of a single image, we could generate our compaction over a set of Wang tiles [Cohen et al. 2003]; all we need to is to properly handle boundary conditions across matching tile edges. Our methodology ought to produce a better tile set than previous methods. We can also apply our algorithm to spatiotemporal textures such as smoke and water waves and we expect this to yield even more data size reduction than 2D images. Another possibility is texture editing [Brooks and Dodgson 2002] as operations performed on our small compaction can be automatically propagated over a large original image. We are also interested in extending our work for texture analysis and classification. Finally, inverse texture synthesis could be utilized to compress texture regions in general images. Since current image compression techniques have not fully taken advantage of the spatial repetition nature of textures, we expect our technique to yield better quality to size ratio than previous methods.

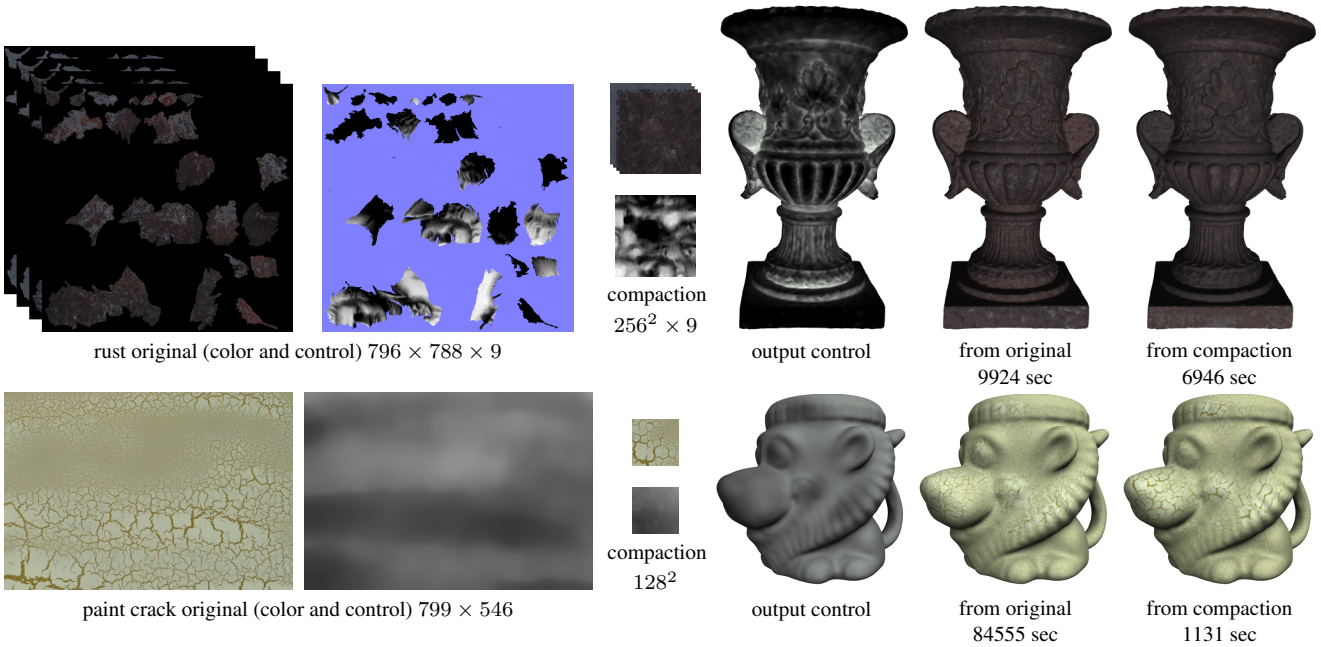


Figure 11: Resynthesis timing and quality comparison. Notice that synthesis from our compaction retains visual quality of synthesis from original, but runs much faster. In the rust case (from [Lu et al. 2007]) we use multiple frames, while in the paint crack case (also from [Lu et al. 2007]) we use only one frame. The synthesis algorithm we used is [Kwatra et al. 2005].

Acknowledgements We would like to thank Jian Sun for advising on orientation field methods, Frank Yu, Bennett Wilburn, and Eric Stollnitz for video dubbing, Dwight Daniels for help on writing, Yale University, Columbia University, Jiaping Wang + Xin Tong, and mayang.com for globally-varying textures, and reviewers for their valuable suggestions. Hujun Bao was partially supported by the NSF of China (No. 60633070) and the 973 Program of China (No. 2002CB312102).

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Symposium on Interactive 3D graphics*, 217–226.

AVIDAN, S., AND SHAMIR, A. 2007. Seam carving for content-aware image resizing. In *SIGGRAPH papers*, 10.

BARGTEIL, A. W., SIN, F., MICHAELS, J. E., GOKTEKIN, T. G., AND O'BRIEN, J. F. 2006. A texture synthesis method for liquid animations. In *Symposium on Computer animation*, 345–351.

BROOKS, S., AND DODGSON, N. 2002. Self-similarity based texture editing. In *SIGGRAPH papers*, 653–656.

CABRAL, B., AND LEEDOM, L. C. 1993. Imaging vector fields using line integral convolution. In *SIGGRAPH papers*, 263–270.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. In *SIGGRAPH papers*, 287–294.

DRORI, I., COHEN-OR, D., AND YESHURUN, H. 2003. Fragment-based image completion. In *SIGGRAPH papers*, 303–312.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *SIGGRAPH papers*, 341–346.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *ICCV '99*, 1033.

FANG, H., AND HART, J. C. 2004. Textureshop: texture synthesis as a photograph editing tool. In *SIGGRAPH papers*, 354–359.

GERSHO, A., AND GRAY, R. M. 1991. *Vector quantization and signal compression*. Kluwer Academic Publishers.

GU, J., TU, C.-I., RAMAMOORTHI, R., BELHUMEUR, P., MATUSIK, W., AND NAYAR, S. 2006. Time-varying surface appearance: acquisition, modeling and rendering. In *SIGGRAPH papers*, 762–771.

HAN, J., ZHOU, K., WEI, L.-Y., GONG, M., BAO, H., ZHANG, X., AND GUO, B. 2006. Fast example-based surface texture synthesis via discrete optimization. *Vis. Comput. 22*, 9, 918–925.

HANRAHAN, P., AND HAEBERLI, P. 1990. Direct wysiwyg painting and texturing on 3d shapes. In *SIGGRAPH papers*, 215–223.

HEEGER, D. J., AND BERGEN, J. R. 1995. Pyramid-based texture analysis/synthesis. In *SIGGRAPH papers*, 229–238.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *SIGGRAPH papers*, 327–340.

JOJIC, N., FREY, B. J., AND KANNAN, A. 2003. Epitomic analysis of appearance and shape. In *ICCV '03*, 34.

KANNAN, A., WINN, J., AND ROTHER, C. 2007. Clustering appearance and shape by learning jigsaws. In *Advances in Neural Information Processing Systems 19*.

KOPF, J., FU, C.-W., COHEN-OR, D., DEUSSEN, O., LISCHINSKI, D., AND WONG, T.-T. 2007. Solid texture synthesis from 2d exemplars. In *SIGGRAPH papers*, 2.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. In *SIGGRAPH papers*, 795–802.

KWATRA, V., ADALSTEINSSON, D., KIM, T., KWATRA, N., CARLSON, M., AND LIN, M. 2007. Texturing fluids. *IEEE Trans. Visualization and Computer Graphics 13*, 5, 939–952.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. In *SIGGRAPH papers*, 777–786.

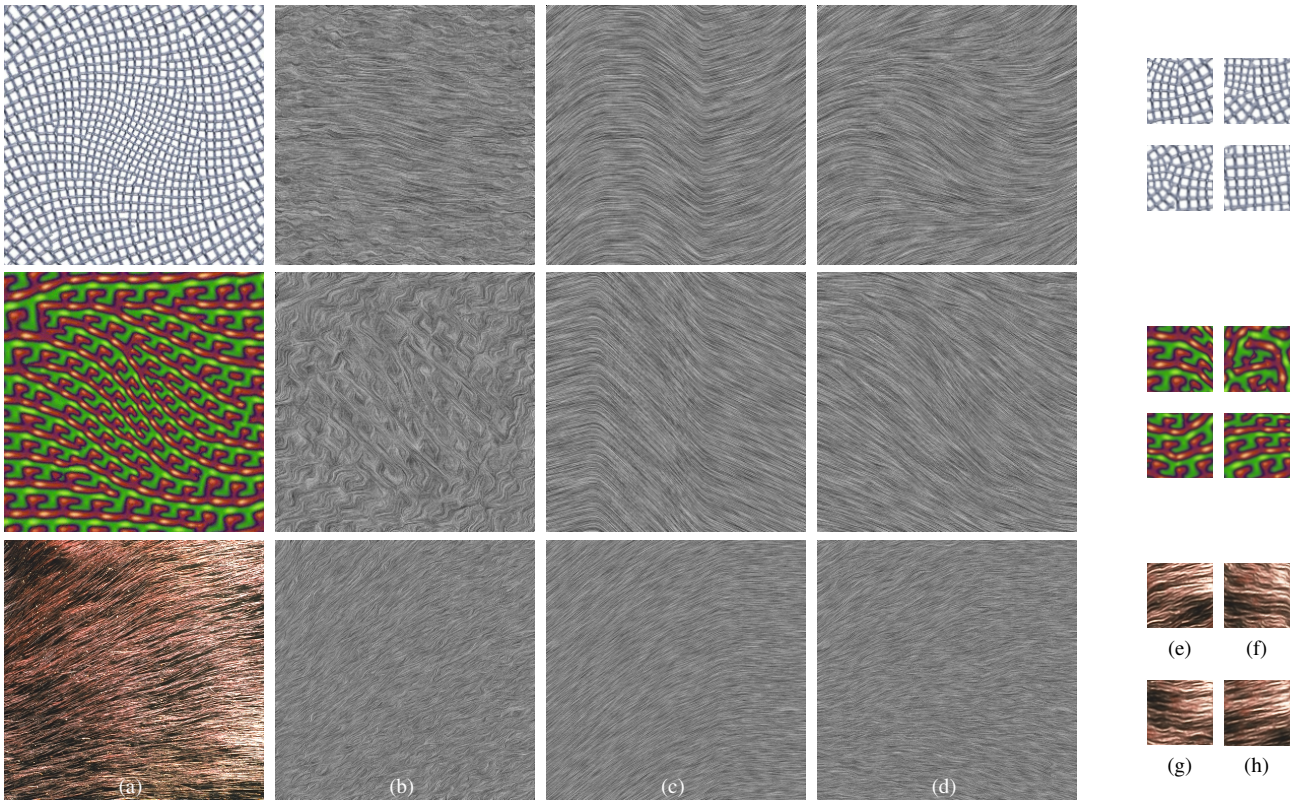


Figure 12: Computing orientation fields from anisotropic textures. Within each row are (a) original texture with uniform x-y orientation, (b) orientation field from [Paris et al. 2004], (c) orientation field via manual specification followed by interpolation, (d) orientation field by our method with initialization from (c), (e,f,g,h) compactions from (a,b,c,d). The top two cases are synthetic (from [Lefebvre and Hoppe 2006]) while the bottom case is real (VisTex Frabic.0004). The orientation fields are visualized via line integral convolution [Cabral and Leedom 1993].

LEFEBVRE, S., AND HOPPE, H. 2006. Appearance-space texture synthesis. In *SIGGRAPH papers*, 541–548.

LEUNG, T., AND MALIK, J. 2001. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. J. Comput. Vision 43*, 1, 29–44.

LIU, Y., LIN, W.-C., AND HAYS, J. 2004. Near-regular texture analysis and manipulation. In *SIGGRAPH papers*, 368–376.

LU, J., GEORGHIADES, A. S., GLASER, A., WU, H., WEI, L.-Y., GUO, B., DORSEY, J., AND RUSHMEIER, H. 2007. Context-aware textures. *ACM Trans. Graph. 26*, 1, 3.

MATUSIK, W., ZWICKER, M., AND DURAND, F. 2005. Texture design using a simplicial complex of morphable textures. In *SIGGRAPH papers*, 787–794.

PARIS, S., BRICENO, H. M., AND SILLION, F. X. 2004. Capture of hair geometry from multiple images. In *SIGGRAPH papers*, 712–719.

PERONA, P., AND MALIK, J. 1990. Detecting and localizing edges composed of steps, peaks and roofs. In *ICCV '90*, 52–57.

POPAT, K., AND PICARD, R. W. 1997. Cluster based probability model and its application to image and texture processing. *IEEE Trans. Image Processing 6*, 2, 268–284.

PORTILLA, J., AND SIMONCELLI, E. P. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comput. Vision 40*, 1, 49–70.

QIN, X., AND YANG, Y.-H. 2005. Basic gray level aura matrices: Theory and its application to texture synthesis. In *ICCV '05*, 128–135.

RITTER, L., LI, W., CURLESS, B., AGRAWALA, M., AND SALESIN, D. 2006. Painting with texture. In *Eurographics Symposium on Rendering*, 371–376.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH papers*, 665–672.

TURK, G. 2001. Texture synthesis on surfaces. In *SIGGRAPH papers*, 347–354.

WANG, J., TONG, X., LIN, S., PAN, M., WANG, C., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Appearance manifolds for modeling time-variant appearance of materials. In *SIGGRAPH papers*, 754–761.

WANG, H., WEXLER, Y., OFEK, E., AND HOPPE, H. 2008. Factoring repeated content within and among images. In *SIGGRAPH papers*.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH papers*, 479–488.

WEI, L.-Y., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH papers*, 355–360.

ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. In *SIGGRAPH papers*, 295–302.

ZIOU, D., AND TABBONE, S. 1998. Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis 8*, 537–559.

Supplementary Materials

The following pages are additional images and do not constitute as an official part of the final paper.

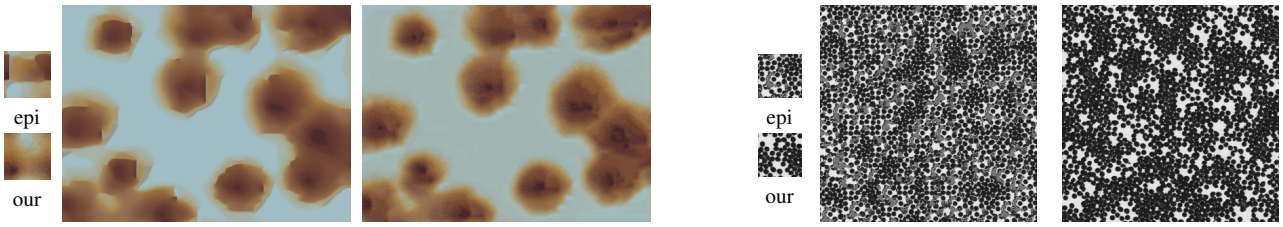


Figure 13: Resynthesis results from epitome and from our compaction. Notice the blur and discontinuity artifacts in the results produced from epitome. See Figure 3 for the source images.

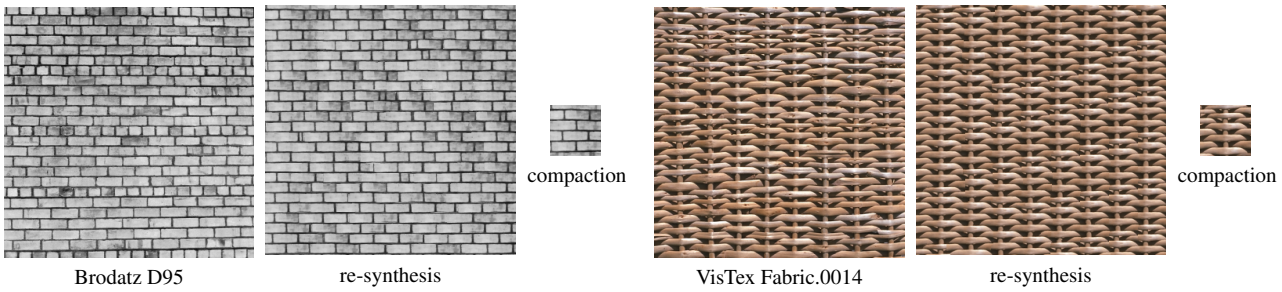


Figure 14: Inverse synthesis and re-synthesis for stationary textures. Even though our algorithm is designed primarily for globally varying textures, it can also be applied to stationary textures. Note that re-synthesis results from our compactions often look more homogeneous than the original. This indicates that even textures considered stationary are rarely perfectly so. The originals and reconstructions have sizes 320^2 and the compactions have sizes 64^2.

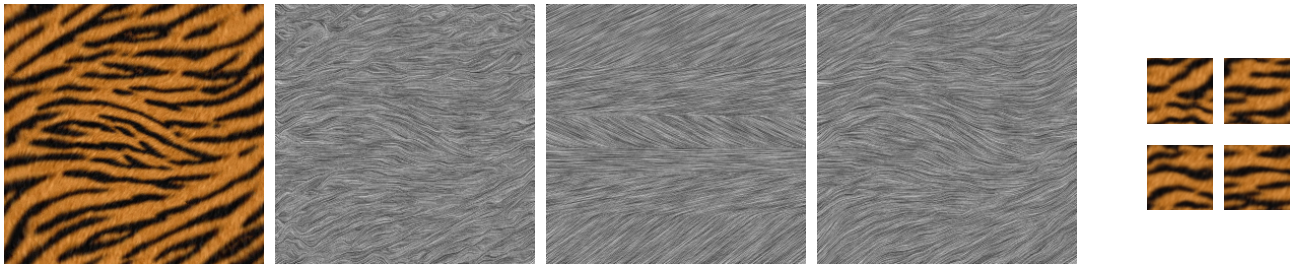
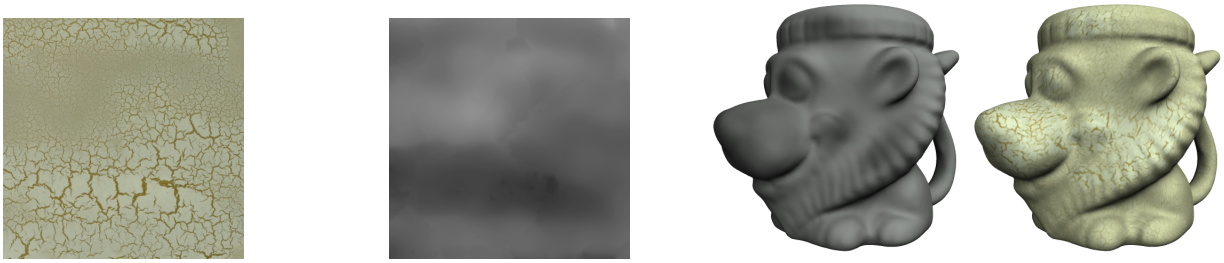


Figure 15: Additional examples for Figure 12

paint crack original (color and control) 799×546 | output control | from original



paint crack compaction (color and control) 512^2 | output control | from compaction

paint crack compaction (color and control) 256^2 | output control | from compaction

paint crack compaction (color and control) 128^2 | output control | from compaction

Figure 16: Resynthesis with different compaction size. Note that the larger the size of the compaction, the better the resynthesis result.

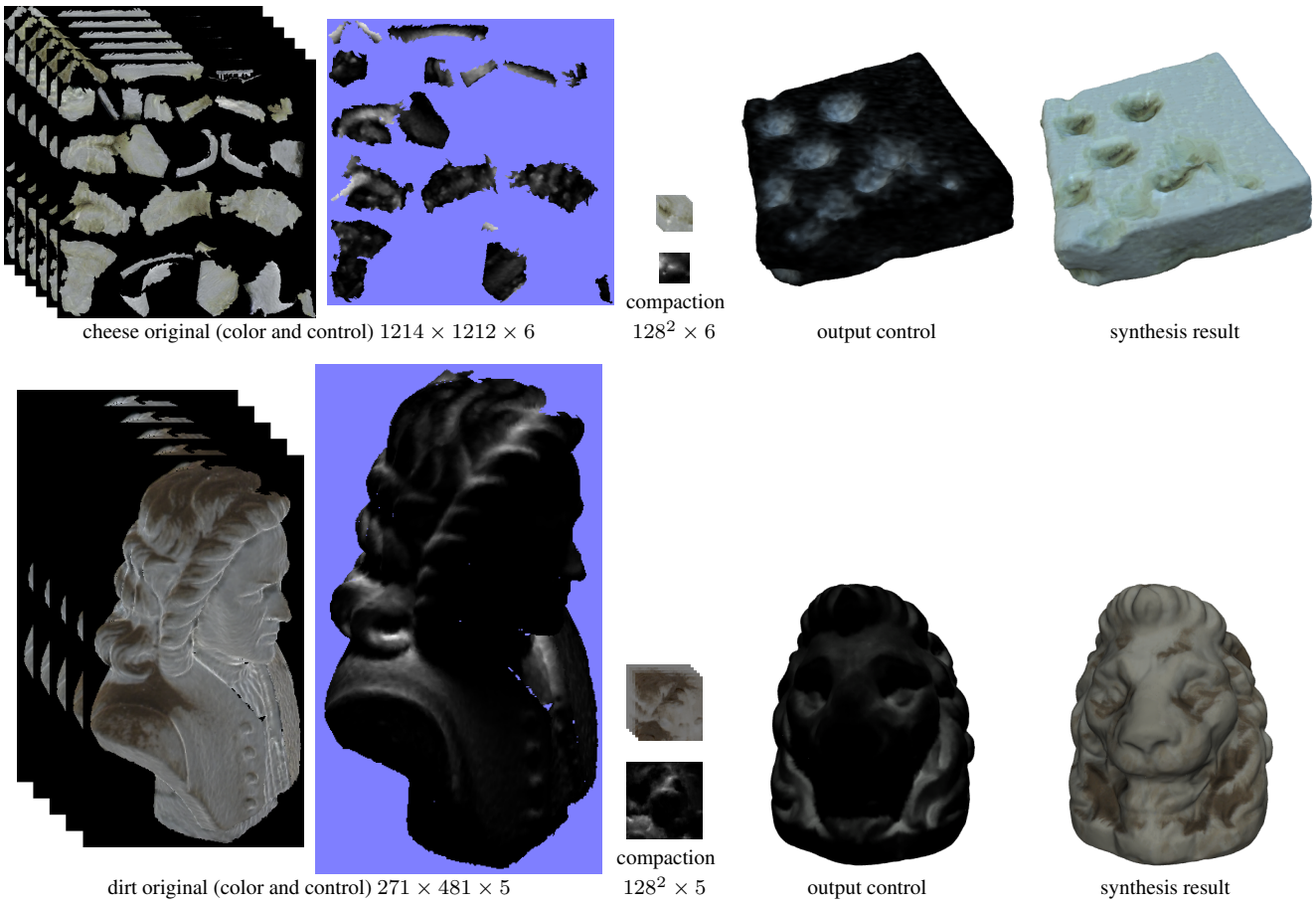


Figure 17: Control maps for Figure 10. For the output synthesis results we only show one time frame, even though the textures are time-varying. See our video for run-time demo of time-varying effects.

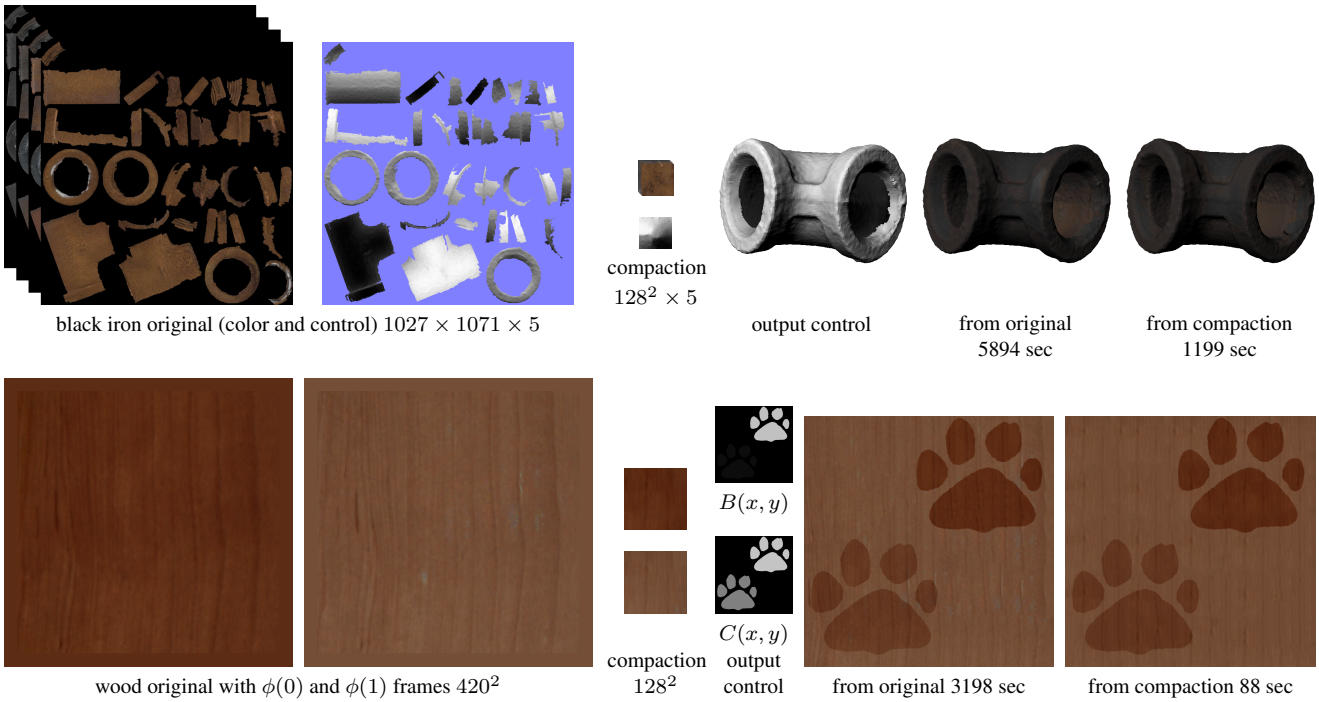


Figure 18: Additional examples for Figure 11.

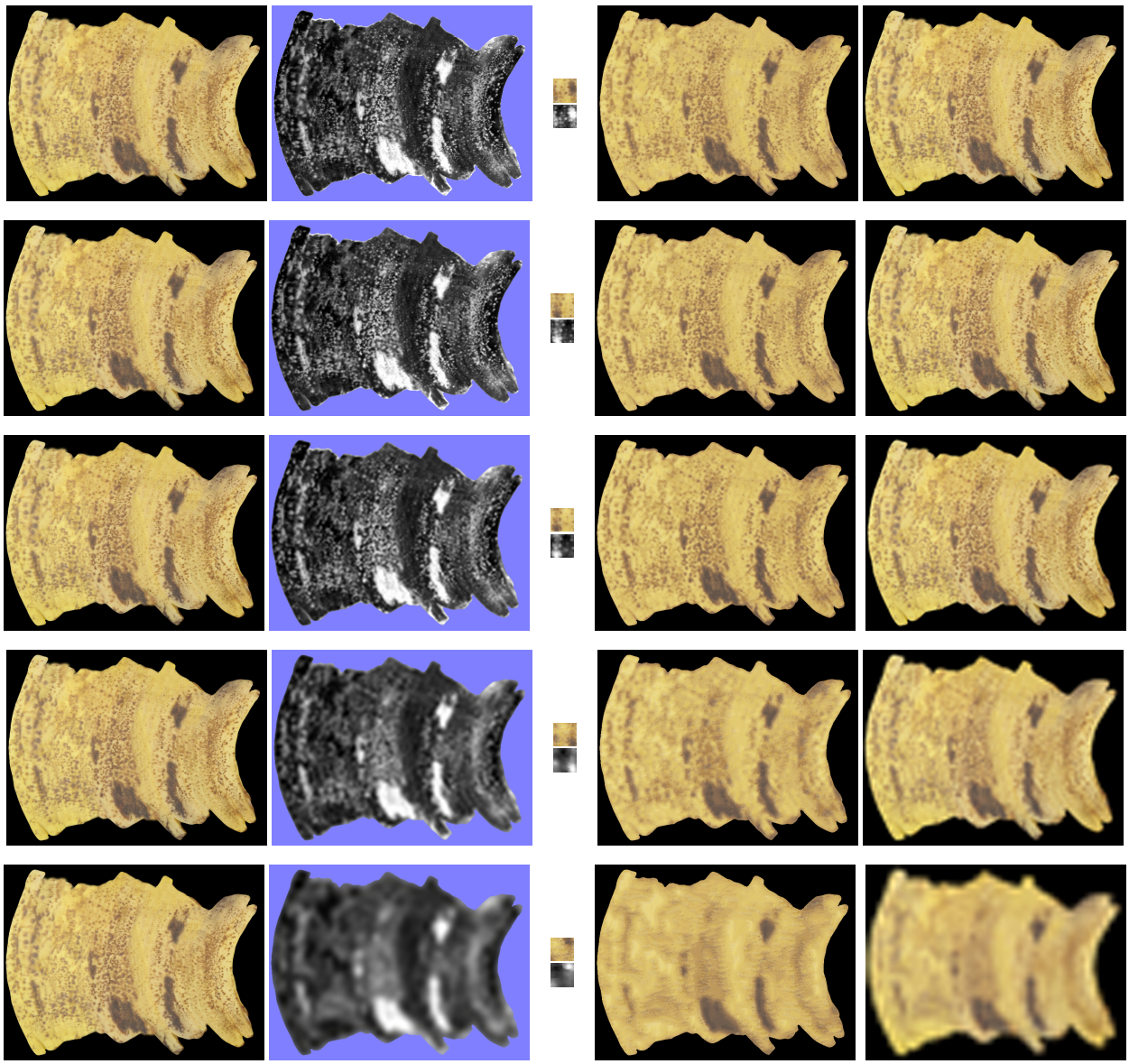


Figure 19: Banana reconstruction with different control map resolutions. From left to right: original texture, original control map, compaction and compaction control map, reconstructed texture and blurred original (produced by downsampling the original followed by upsampling). The original control maps are in successively lower resolutions from top to bottom. Note that even with excessively low-resolution control maps, our algorithm still produces results with crispier texture details than the blurred original.

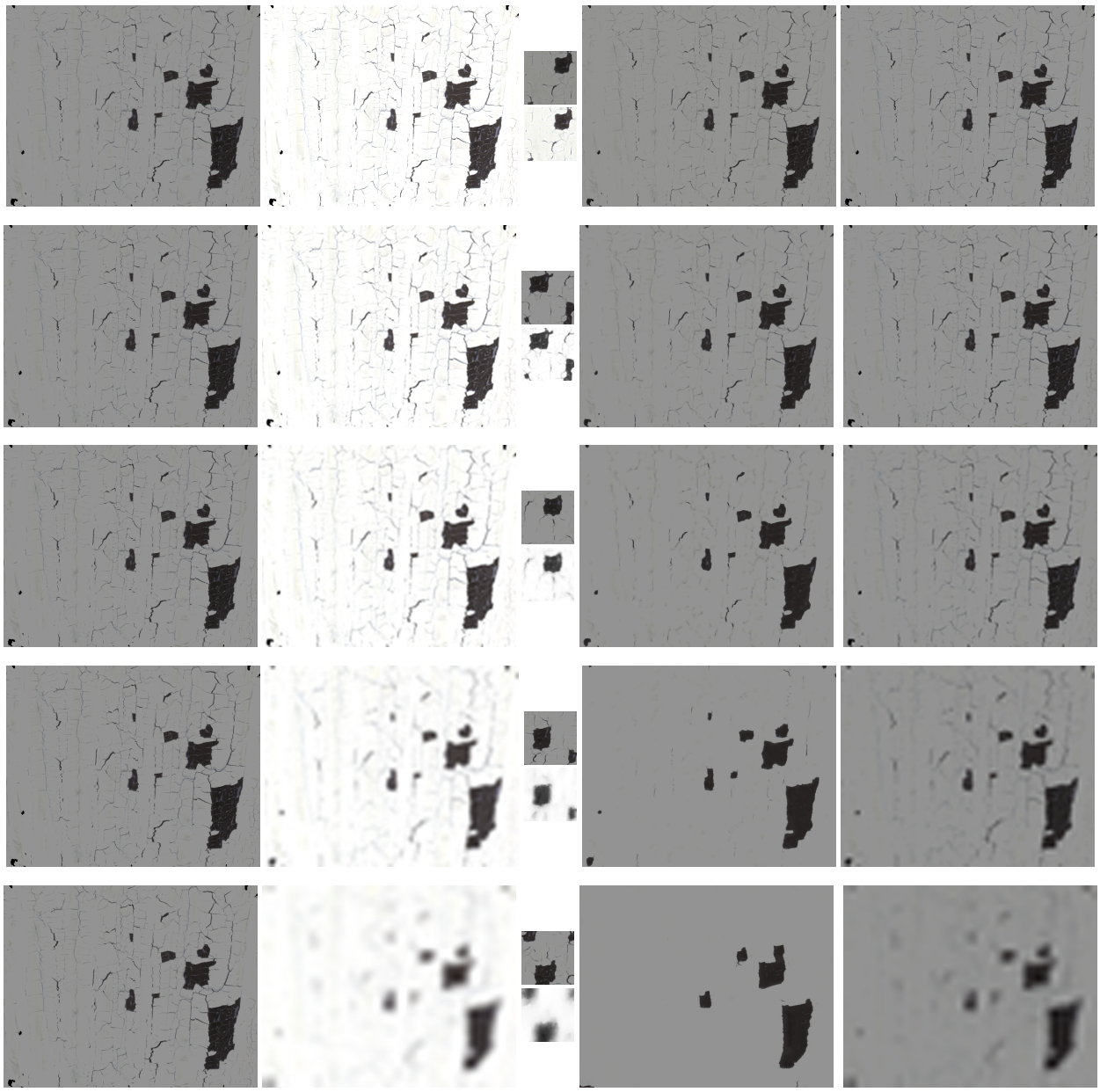


Figure 20: Paint-peeling reconstruction with different control map resolutions. See caption for Figure 19.