Texture Representations for Image and Video Synthesis

Georgios Georgiadis
UCLA Vision Lab
University of California
Los Angeles, CA 90095
giorgos@ucla.edu

Alessandro Chiuso
Dept. of Information Eng.
University of Padova
Padova 35131, Italy
chiuso@dei.unipd.it

Stefano Soatto
UCLA Vision Lab
University of California
Los Angeles, CA 90095
soatto@ucla.edu

Abstract

In texture synthesis and classification, algorithms require a small texture to be provided as an input, which is assumed to be representative of a larger region to be re-synthesized or categorized. We focus on how to characterize such textures and automatically retrieve them. Most works generate these small input textures manually by cropping, which does not ensure maximal compression, nor that the selection is the best representative of the original. We construct a new representation that compactly summarizes a texture, while using less storage, that can be used for texture compression and synthesis. We also demonstrate how the representation can be integrated in our proposed video texture synthesis algorithm to generate novel instances of textures and video hole-filling. Finally, we propose a novel criterion that measures structural and statistical dissimilarity between textures.

1. Introduction

"Visual textures" are regions of images that exhibit some form of spatial regularity. They include the so-called "regular" or "quasi-regular" textures (Fig. 2 top-left), "stochastic" textures (top-right), possibly deformed either in the domain (bottom-left) or range (bottom-right). Analysis of textures has been used for image and video representation [41, 14], while synthesis has proven useful for image super-resolution [11], hole-filling [9] and compression [35].

For such applications, large textures carry a high cost on storage and computation. State-of-the-art texture descriptors such as [7, 31] are computationally prohibitive to use on large textures. These issues are especially acute for video, where the amount of data is significantly larger.

Typically, these descriptors as well as texture synthesis algorithms assume that the size of the input texture is small, and yet large enough to compute statistics that are representative of the entire texture domain. Few works in the literature deal with how to infer automatically this smaller texture from an image and even fewer from a video. In most cases, it is assumed that the input texture is given, usually manually by cropping a larger one. Wei et. al. [37] propose an inverse texture synthesis algorithm, where given an input texture I, a compaction is synthesized that allows subsequent re-synthesis of a new instance $\hat{I}$. The method achieves good results, but it is semi-automatic, since it relies on external information such as a control map (e.g. an orientation field or other contextual information) to synthesize time-varying textures and on manual adjustment of the scale of neighborhoods for sampling from the compaction.

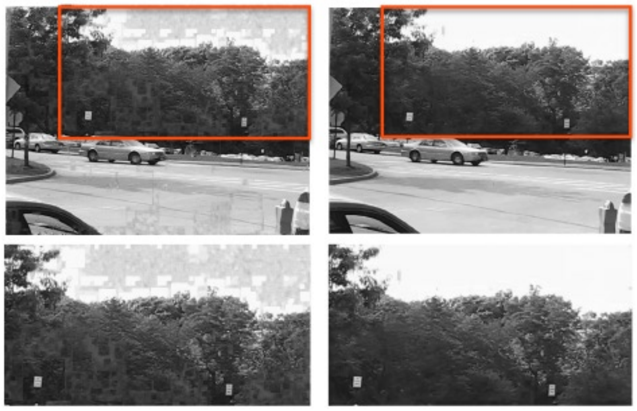


Figure 1. *Reconstructed frame in a video at 40% compression. Left: Video Epitome [6], Right: Our approach. Below: Zoomed-in view of the red box. Our method improves reconstruction of both homogeneous and textured areas.*

We propose an alternative scheme, which avoids using any external information by automatically inferring a compact time-varying texture representation. The algorithm also automatically determines the scale of local neighborhoods, which is necessary for texture synthesis [9]. Since our representation consists of samples from the input texture, for applications such as classification [7, 31], we are able to avoid synthesis biases that affect other methods [37].

Our contributions are to (i) summarize an image/video into a representation that requires significantly less storage than the input, (ii) use our representation for synthesis using the texture optimization technique [19], (iii) extend this framework to video using a causal scheme, similar to [9]

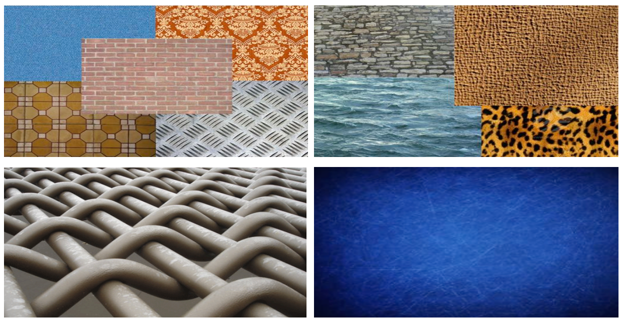


Figure 2. *Regular, stochastic, domain- and range-deformed textures.*

and show results for multiple time-varying textures, (iv) synthesize multiple textures simultaneously on video without explicitly computing a segmentation map, unlike [40], which is useful for hole-filling and video compression, (see Fig. 1), and (v) propose a criterion ("Texture Qualitative Criterion" (*TQC*)) that measures structural and statistical dissimilarity between textures.

2. Related work

Our work relates to texture analysis ([26, 21, 18, 23] and references therein), perception ([25] and refs.), synthesis and mapping ([9] and refs.). We do not address phenomenology (*e.g.* "3-D textures" generated by the interplay of shape and illumination [32] vs. "decal textures" due to radiance) and the algorithms used to pool statistics (*e.g.* patch-based [34], statistical [12], geometric [42] methods). We refer the reader to [15] for a more extensive survey.

Some work has been done in summarizing images (epitome [16] and jigsaw [17]) and video [6, 39]. These methods do not handle textures explicitly and as a result, their reconstructed textures suffer. [16, 17, 6] are also not able to extend textures to larger domains, since they rely on an explicit mapping between the input image/video and the summarization. Other schemes aim to compact the spectral energy into few coefficients [2, 10, 27]. [35] compresses regular textures, but fails with stochastic ones. Our work models textures explicitly and hence achieves both high compression rates and high quality synthesis results.

In video texture synthesis, there are several works ranging from pixel-based ([1], [38]), to patch-based [20], to parametric [8]. Our work falls in the patch-based category.

3. Textures

"Visual textures" are regions of images that exhibit spatial regularity. To characterize them, we use the notions of Markovianity, stationarity and ergodicity. We denote an image by $I : \Delta \to \mathbb{R}^+$, where $\Delta = (1,1) : (X,Y)$ is the pixel lattice. Statistics, or "features", map the image onto some vector space and *local* features operate on a subset of the image domain. Formally, a local statistic is defined on $\omega \subset \Delta$ as a function $\theta_\omega : \{I : \Delta \to \mathbb{R}^+\} \to \mathbb{R}^K$; $I \mapsto \theta_\omega(I)$ that only depends on the values of the image $I(x,y)$ for $(x,y) \in \omega$. A distribution on I, $dP(I)$ induces a distribution on θ_ω via $dP(\theta_\omega) \doteq dP(\theta_\omega(I))$.

In order to exploit spatial predictability, we leverage on the existence of a statistic that is *sufficient* to perform the prediction. This is captured by the notion of Markovianity. We say that a process $\{I\}$ is Markovian if every set $\Pi \subset \Delta$ admits a neighborhood $N(\Pi)$ and a statistic $\theta_{N(\Pi)}$ that makes $I(\Pi)$ independent of the "outside" $I(\Pi^c)$. Π^c is the complement of Π in a region $\Omega \subset \Delta$, where Ω corresponds to the texture region (see Fig. 3):

$$I(\Pi) \perp I(\Pi^c) \mid \theta_{N(\Pi)}. \tag{1}$$

This condition makes the process $\{I\}$ with measure $dP(I)$ a Markov Random Field (MRF), and establishes $\theta_{N(\Pi)}$ as a *sufficient statistic* for $I(\Pi)$. In general, $N(\Pi)$ could correspond to many regions, for instance $\omega = \Omega \backslash \Pi$. In describing a texture, we seek the *smallest* ω, in the sense of minimum area ("scale") $|\omega| = r$, so the corresponding θ_ω is a *minimal (Markov) sufficient statistic*. Of particular interest is the case when such a neighborhood is spatially homogeneous[1], as is the case in a *stationary* MRF.

A process $\{I\}$ is *stationary* if statistics are translation invariant. More formally, $\{I\}$ is stationary in θ_ω if,

$$\mathbb{E}(\theta_\omega) = \mathbb{E}(\theta_{\omega+T}), \quad T \in \mathbb{R}^2 \mid \omega + T \subset \Omega. \tag{2}$$

Such condition may be satisfied only after transformations of the image domain and range (as in *deformed textures*, Fig. 2)[2]. Note that unless the process is defined on the entire plane, one has to *restrict* the set of T to *allowable translations* to ensure Eq. (2) is computed where $\theta_{\omega+T}$ is defined.

To *test* whether a process is stationary from just one sample, we have to assume it is ergodic, meaning that the sample statistics converge to the ensemble ones:

$$\frac{1}{N}\sum_{i=1}^{N} \theta_{\omega+T_i} \xrightarrow{\text{a. s.}} \mathbb{E}(\theta_\omega), \tag{4}$$

for all admissible T_i. Given Ω, we can test whether the process $\{I\}$ is stationary in this region, by approximating statistics θ_ω using samples in a neighborhood $\bar{\omega} \subset \Omega$. The larger the size of $\bar{\omega}$, the better the approximation of the statistics, but the lower the compression achieved. Therefore, we seek the *smallest* possible $\bar{\omega}$ that allows inferring the statistics "sufficiently well". Assuming that such $\bar{\omega}$ is found, we can test for stationarity by *translating* it and testing whether the resulting statistics remain "sufficiently constant". This, however, can only be tested for *admissible* translations that keep $\bar{\omega}$ within Ω.

[1]Spatially homogeneous means that $\omega = \omega(\Pi)$ can be written in terms of neighborhoods of each pixel $\boldsymbol{x} = (x,y)$ within Π, and the smallest Markov neighborhood of a pixel, $\omega(\boldsymbol{x})$, is translation invariant, i.e., its shape does not change as we translate $\boldsymbol{x}$: $\omega(\boldsymbol{x}+T) = \omega(\boldsymbol{x}) + T$.

[2]For a group G acting on the domain of I via $I(x,y) \mapsto I(g(x,y))$ and a group H acting on the range via $I(x,y) \mapsto h(I(x,y))$, we say that the process $\{I\}$ is $G-H$ stationary in $\theta_{g(\omega)}$ if,

$$\mathbb{E}(\theta_{g(\omega)}(h(I))) = \mathbb{E}(\theta_{g(\omega)+T}(h(I))), \quad T \in \mathbb{R}^2 \mid \omega + T \subset \Omega. \tag{3}$$

Such transformations can be inferred by a model selection criterion. Textures can be *rectified* by applying the inverse action g^{-1}, h^{-1} to the data.

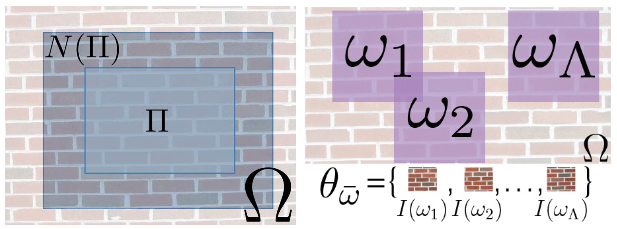

$N(\Pi)$
Π
Ω
ω_1
ω_Λ
ω_2
Ω
$\theta_{\bar\omega} = \{ \ldots \}$
$I(\omega_1)$ $I(\omega_2)$ $I(\omega_\Lambda)$

Figure 3. *Left: A subset of the image domain, Π, with its local neighborhood. Ω denotes the domain of the texture. Right: Texture representation $\theta_{\bar\omega}$ and samples drawn from Ω.*

We define a texture as *a region Ω of an image I that can be rectified into a sample of a stochastic process that is stationary, ergodic and Markovian.* A texture is parametrized by the following quantities: (a) The Markov neighborhood ω and its Markov scale $r = |\omega|$, (b) the stationarity region $\bar\omega$ and its stationarity scale $\sigma = |\bar\omega|$, (c) the minimal sufficient statistic θ_ω defined on ω, and (d) Ω, the texture region. Note that $\omega \subset \bar\omega \subset \Omega$.

3.1. Texture Representation

We initially assume (and later relax) that $\Omega = \Delta$. The representation should have smaller complexity than the collection of pixel values in Ω and allow extrapolation beyond Ω, traits not shared by [7, 31], but the latter can be used to further reduce complexity in classification applications.

In a non-parametric setting, θ_ω is a collection of sample image values. $\bar\omega \doteq \bigcup_{\lambda=1,\ldots,\Lambda} \omega_\lambda$ is the union of Λ sample regions ω_λ, each a Markov neighborhood of a pixel with coordinates (x_λ, y_λ). Collectively the neighborhoods capture the variability of the texture. Thus, a texture is represented by (a) ω_λ, chosen as a square for all λ with unknown area r, (b) $\bar\omega$, to be determined and (c) $\theta_{\bar\omega} \doteq \{\theta_{\omega_\lambda}\}_{\lambda=1}^{\Lambda} \doteq \{I(\omega_\lambda)\}_{\lambda=1}^{\Lambda}$ (where $I(\omega_\lambda) \doteq \{I(x,y),\ \forall (x,y) \in \omega_\lambda\}$) that is uniquely specified by the image given r and ω_λ (Fig. 3).

Complexity controls the cardinality of $\bar\omega$. The best we can do is to select all patches ω from Ω and store them as $\bar\omega$. When complexity is fixed, however, for instance via a compression parameter ξ, we can only store $\xi \times (X \times Y)$ values, rather than $(X \times Y)$. This determines the cardinality of $\bar\omega$. The larger $r = |\omega_\lambda|$, the fewer the patches that can be stored in a given $\bar\omega$. There is a natural tradeoff where too small an r fails to capture the local neighborhood of the texture (Markov sufficient statistic) and too large an r fails to capture the statistical variability, as too few patches ω_λ can be contained in a given $\bar\omega$. Both have detrimental effects on extrapolating a texture beyond Ω, which we discuss next.

4. Inference

In this section we discuss how to infer a (minimal) representation $\{\omega, \bar\omega, \theta_{\bar\omega}\}$ from a given texture image $\{\Omega, I\}$, and how to synthesize a novel texture image $\hat I$ from it. We start from the latter since the algorithm that infers the representation utilizes the synthesis procedure.

Algorithm 1: Texture Synthesis

```
Initialize $\hat I^{(0)}$ to a random texture;
Set $\nu_{\omega_s} = 1$ for $s = 1, \ldots, S$ and $j_{max} = 20, b = 0.7$ ;
for $j = 1, \ldots, j_{max}$ do
    for $s = 1, \ldots, S$ do
        $\omega_s^{(j)} = $ nrst_nghbr $(\theta_{\bar\omega}, \bar\omega, \hat I^{(j-1)}(\hat\omega_s))$ ;
    Let $\hat I^{(j)} = \arg\min_{\hat I} E(\hat I, \{\omega_s^{(j)}\}_{s=1}^{S})$;
    $\nu_{\hat\omega_s} = \|\hat I^{(j)}(\hat\omega_s) - I(\omega_s^{(j)})\|^{b-2}$ for $s = 1, \ldots, S$ ;
    if $(\forall \hat\omega_s \in \hat\Omega_S : \hat I^{(j)}(\hat\omega_s) = \hat I^{(j-1)}(\hat\omega_s))$ then
        break;

Function nrst_nghbr $(\theta_{\bar\omega}, \bar\omega, \hat I(\hat\omega))$
    Let $s$ be the index of the the nearest neighbor of
    $\hat I(\hat\omega)$ in $\theta_{\bar\omega}$ ;
    Retrieve $\omega_s$ within $\bar\omega$ ;
    return $\omega_s$ ;
```

4.1. Image Texture Synthesis

Given a representation $\{\omega, \bar\omega, \theta_{\bar\omega}\}$, we can synthesize novel instances of the texture by sampling from $dP(I(\omega))$ within $\bar\omega$. This is straightforward in a non-parametric setting, where the representation is itself a collection of samples. One can simply select neighborhoods ω_λ within $\bar\omega$, and populate a new lattice with patches $I(\omega_\lambda)$ ensuring compatibility along patch boundaries and intersections. Efros et. al. [9] proposed a *causal* sampling scheme that satisfies such compatibility conditions, but fails to respect the Markov structure of the underlying process (their $I(\omega_\lambda)$ are not a Markov sufficient statistic), which causes "blocky" artifacts and "drift." Instead, given $\{\omega, \bar\omega, \theta_{\bar\omega}\}$, we synthesize textures by choosing a subset of neighborhoods from $\bar\omega$ that satisfy the compatibility conditions and by construction also respect the Markov structure. We perform this selection and simultaneously also infer $\hat I$. We do so by first initializing $\hat I$ at random. We select neighborhoods $\hat\omega_s$ on a grid on the domain of the synthesized texture every $\frac{\sqrt{r}}{4}$. We let $\hat\Omega_S = \{\hat\omega_s\}_{s=1}^{S}$ denote the collection of the selected $\hat\omega_s$, $\Omega_S = \{\omega_s\}_{s=1}^{S}$ denote the chosen neighborhoods within $\bar\omega$ and $I(\omega_s) \in \theta_{\bar\omega}$ denote the nearest neighbor of $\hat I(\hat\omega_s)$. We minimize with respect to $\{\omega_s\}_{s=1}^{S}$ and $\hat I$ the function [19]:

$$E(\hat I, \{\omega_s\}_{s=1}^{S}) = \sum_{\hat\omega_s \in \hat\Omega_S} \nu_{\hat\omega_s} \|\hat I(\hat\omega_s) - I(\omega_s)\|^2. \quad (5)$$

The procedure to minimize the above energy function is given in Alg. 1. An illustration of the quantities involved is shown in Fig. 4. $\nu_{\hat\omega_s}$, defined in Alg. 1, is used to reduce the effect of outliers, as done typically in iteratively re-weighted least squares [19]. The process is performed in a multiscale fashion, by repeating the procedure over 3 neighborhood sizes: $|\hat\omega_s|, |\frac{\hat\omega_s}{2}|, |\frac{\hat\omega_s}{4}|$. By first synthesizing at scale $|\hat\omega_s| = r$, we capture the Markov structure of the texture. Subsequent repetitions refine the synthesized tex-

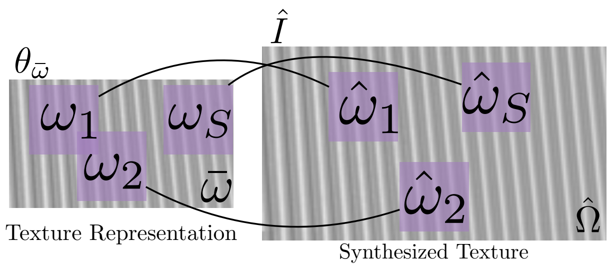
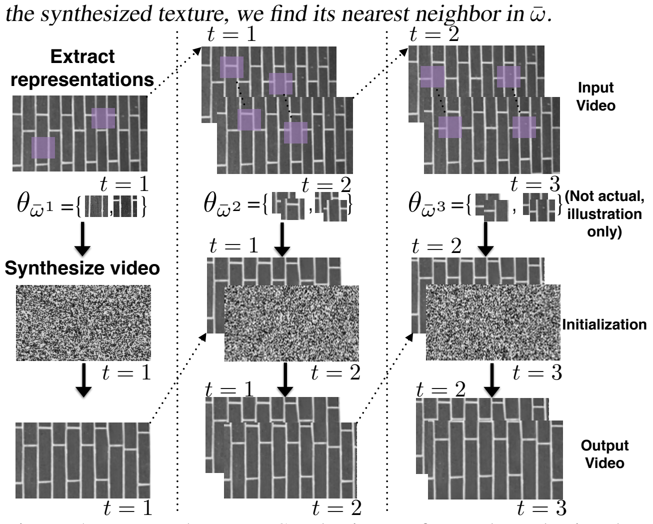


Figure 4. *Image Texture Synthesis. For each neighborhood $\hat\omega_s$ in the synthesized texture, we find its nearest neighbor in $\bar\omega$.*



Extract representations
$t = 1$
$t = 2$
Input Video
$\theta_{\bar\omega^1} = \{$
$\theta_{\bar\omega^2} = \{$
$\theta_{\bar\omega^3} = \{$
$t = 3$
(Not actual, illustration only)
Synthesize video
Initialization
Output Video

Figure 5. *Temporal Texture Synthesis. We forward-synthesize the video from the texture representations of each frame using the previously synthesized frame as a boundary condition.*

ture by adding finer details. We also repeat this process over a number of different output image sizes.

4.2. Video Texture Synthesis

The texture synthesis algorithm in [19] was extended to temporal textures, which however relied on the availability of optical flow. Unfortunately, optical flow is expensive to store, as encoding it is more costly than encoding the original images. We propose a temporal texture synthesis algorithm that relies on neighborhoods ω_λ that extend in time.

We take the input video $\{I^t\}_{t=1}^T$, and compute a compact representation $\theta_{\bar\omega^t}$, from which we synthesize $\{\hat I^t\}_{t=1}^T$. In this section we assume we have $\theta_{\bar\omega^t}$ and in Sec. 4.5 we explain how to infer it. We re-define all quantities to have domains that extend in time. To reduce computational complexity we fix the temporal extension of the neighborhoods to 2 frames, although longer choices are possible. Hence for $t > 1$, $\omega_\lambda^t \subset (1, 1, t-1) : (X, Y, t)$, which makes it a 3-D neighborhood and $\bar\omega$ becomes $\bar\omega^t \doteq \bigcup_{\lambda=1,\ldots,\Lambda} \omega_\lambda^t$, a union of 3-D neighborhoods. $I^t(\omega_\lambda^t)$ is therefore defined on the 3-D lattice and $\theta_{\bar\omega^t} \doteq \{I^t(\omega_1^t), \ldots, I^t(\omega_\Lambda^t)\}$. For $t = 1$, $\omega_\lambda^{t=1}, \bar\omega^{t=1}$ and $\theta_{\bar\omega^{t=1}}$ remain 2-D.

We initialize the output video, $\hat I^t$, to a random texture and first synthesize frame $\hat I^{t=1}$ using Alg.1. We then let $t \leftarrow t+1$ and synthesize frame t using a causal approach, similar to [9], by ensuring that the compatibility conditions with frame $t-1$ are satisfied: For each $\hat\omega_s^t \in \hat\Omega_S^t$ (where Ω_S^t corresponds to the set of selected $\hat\omega_s^t$ on the 3-D grid of the synthesized frames, extending temporally in $[t-1, t]$), we mask (*i.e.*, discount) the portion of the neighborhood that is in frame t (the unsynthesized part) and seek its nearest neighbor, ω_s^t, within $\theta_{\bar\omega^t}$ on *only* the unmasked region (*i.e.*, on only the region that has already been synthesized). Once we get the nearest neighbors of all $\hat\omega_s^t$, we fix them and do not allow them to change. This is done in order to achieve compatibility with the already synthesized textures. We then unmask all neighborhoods and use them to minimize the energy function in Eq. (5) (see Fig. 5).

4.3. Synthesizing Multiple Textures Simultaneously

We demonstrate how multiple textures can be synthesized simultaneously for video and images without computing a segmentation map. This is useful for applications such as video compression (where $\{\omega, \bar\omega, \theta_{\bar\omega}\}$ can be used to synthesize the textures of the input video) or for image processing tasks such as hole-filling and frame interpolation.

To place the textures in their corresponding locations in a video (or image) we implicitly define their boundaries by partitioning each frame into two types of regions: Textures and their complementary region type, *structures*. Structures are regions of images that trigger isolated responses of a feature detector. These include blobs, corners, edges, junctions and other sparse features. We determine which regions are structures, by using a feature point tracker such as [24].

Partitioning images or video into two types of regions has been previously proposed by several works ([13, 29, 3]) using a single image. In our framework, if a region with a scale ϵ triggers an *isolated* response of a feature detector (*i.e.*, it is a structure at scale ϵ), then the underlying process is, by definition, not stationary at the scale $|\bar\omega| = \epsilon$. Therefore, it is not a texture. It also implies that any region $\bar\omega$ of size $\epsilon = |\bar\omega|$ is not sufficient to predict the image outside that region. This of course does not prevent the region from being a texture at a scale $\sigma >> \epsilon$. Within a region σ there may be multiple frames of size ϵ, spatially distributed in a way that is stationary/Markovian. Vice-versa, if a region of an image is a texture with $\sigma = |\bar\omega|$, it cannot have a unique (isolated) extremum within $\bar\omega$. Of course, it could have multiple extrema, each isolated within a region of size $\epsilon << \sigma$. We conclude that, *for any given scale of observation σ, a region $\bar\omega$ with $|\bar\omega| = \sigma$ is either a structure or a texture.*

One must impose boundary conditions so that the texture regions fill around structure regions seamlessly. To perform texture extrapolation, we follow an approach similar to the one used for video texture synthesis. The video is initialized to a random texture. At locations where the structures were detected and tracked, we place the actual image (intensity) values. We select $\hat\omega_s^t \in \hat\Omega_S^t$ like before, on a 3-D grid of the synthesized frames, but with the added restriction that $\hat\omega_s^t$ needs to have at least one pixel in the texture domain (otherwise it is entirely determined *i.e.*, it is a structure). The patches that are entirely lying in the texture domain need to be synthesized. The patches that straddle the texture/structure partition are used as boundary conditions and are synthesized causally.

We mask as before the portion of each neighborhood, $\hat{\omega}_s^t$, that is in frame t and is part of the texture domain (since this is the only unknown, unsynthesized part of the neighborhood). We then proceed as in the temporal texture synthesis algorithm: We find the nearest neighbor of the unmasked region within $\theta_{\bar{\omega}^t}$, fix it and do not allow it to change. Finally we unmask all the neighborhoods and use them to minimize the energy function in Eq. (5). Note that for $t = 1$, if $\hat{\omega}_s^t$ lies entirely in the texture domain, its nearest neighbor is permitted to change through the iterations, similar to Alg.1.

Finally, in addition to the structures, we could also use $\bar{\omega}^t$ to provide additional boundary conditions. Placing $I(\omega_\lambda^t)$ for $\omega_\lambda^t \subset \bar{\omega}^t$ on the image domain allows us to avoid explicitly synthesizing on these locations. These are already stored in $\theta_{\bar{\omega}^t}$, so we only need to additionally store the location of their central pixels. Examples of synthesizing multiple textures simultaneously are shown in Fig. 13.

4.4. Texture Qualitative Criterion

To evaluate the quality of the texture synthesis algorithm, we need a criterion that measures the similarity of the input, I, and synthesized, $\hat{I}$, textures. The peak signal-to-noise ratio ($PSNR$) is typically used as the criterion for evaluating the quality of a reconstruction. However, when the final user is the human visual system, $PSNR$ is known to be a poor criterion, especially for textures, as imperceptible differences can cause large $PSNR$ changes. Works such as [28, 36, 33, 30] operate on general images and do not exploit properties of textures. To address this issue, we introduce the Texture Qualitative Criterion (*TQC*), represented by E_{TQC}, which is composed of two terms. The first one, $E_1(\hat{I}, I)$, penalizes structural dissimilarity, whereas $E_2(\hat{I}, I)$ penalizes statistical dissimilarity. We let $\hat{\omega}_s/\omega_i$ be patches within $\hat{\Omega}/\Omega$, the domains of $\hat{I}/I$, and their nearest neighbors be $\omega_s/\hat{\omega}_i$, which are selected within the domains of $I/\hat{I}$. $I/\hat{I}$ can correspond to the input/synthesized textures, or simply two textures, which we wish to compare.

For $E_1(\hat{I}, I)$, we select N_S patches $\hat{\omega}_s \subset \hat{\Omega}$ and N_I patches $\omega_i \subset \Omega$ on a dense grid in the domain of the synthesized and input images respectively. We let $\hat{I}(\hat{\omega}_s)$ and $I(\omega_i)$ correspond to the intensity values in the synthesized and input neighborhoods respectively. We use the patches selected to compute the following cost function:

$$\begin{aligned} E_1(\hat{I}, I) = & \frac{1}{2N_I} \sum_{i=1}^{N_I} \frac{1}{|\omega_i|} \|\hat{I}(\hat{\omega}_i) - I(\omega_i)\|^2 + \\ & \frac{1}{2N_S} \sum_{s=1}^{N_S} \frac{1}{|\hat{\omega}_s|} \|\hat{I}(\hat{\omega}_s) - I(\omega_s)\|^2. \end{aligned} \tag{6}$$

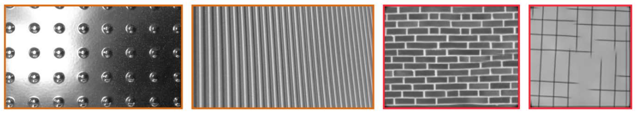


Figure 6. *The first term in Eq. (6) identifies global range/domain transformations of the input texture (left images). The second term identifies erroneous texture synthesis results (right images).*

Note that this expression resembles Eq. (5), with one change: There is an added summation in Eq. (6), which is over patches in the input image. The need of both of these terms has also been noted by others [37] and is illustrated in Fig. 6. The first term identifies domain/range deformations of the input texture, whereas the second term identifies artifacts in the synthesized texture. We compute this cost function over multiple scales (typically 3) and average over all scales. This makes the cost function more robust, as it is able to compute similarity of patches at multiple scales.

$E_2(\hat{I}, I)$ is based on a distance between histograms of filter responses, which allows us to capture the statistical differences between two textures:

$$E_2(\hat{I}, I) = \frac{1}{L} \sum_{l=1}^{L} \|\phi(g_l(I)) - \phi(g_l(\hat{I}))\|_{\chi^2}, \tag{7}$$

where $\|.\|_{\chi^2}$ is the χ^2 distance, $\phi(.)$ is a histogram of filter response values and $g_l(I), l = 1, \ldots, L$ are the responses of the L filters. We chose the filter bank of [22] [3]. Finally, *TQC* is given by:

$$E_{TQC}(\hat{I}, I) = E_1(\hat{I}, I) + E_2(\hat{I}, I). \tag{8}$$

4.5. Inference of Texture Representation

Given a complexity constraint ξ, we have a bound on the number, $\Lambda \doteq \Lambda(r)$, of samples, ω_λ, that can be stored in $\bar{\omega}$, which depends on r, the scale of ω_λ. To estimate r, the inference algorithm involves two levels of computation. The first level involves fixing r_{cand}, a candidate of r, and computing *which* samples $\omega_\lambda \subset \Omega$ should be stored in $\bar{\omega}$. This computation is repeated for a finite number of r_{cand}. To choose $\hat{r}$, an estimate of r, we use *TQC* to rank the representations and choose the best one according to this ranking. In this section, we describe this procedure in greater detail.

For each r_{cand}, we use Alg.1 (see Sec. 4.1) (or its variant if the input is a video) to synthesize a novel instance of the texture at just one scale, r_{cand}. Upon convergence, for each $\hat{\omega}_s$ there is an ω_s (its nearest neighbor), that is assigned to it. The set $\Omega_S = \{\omega_s\}_{s=1}^{S}$ denotes the collection of nearest neighbors within Ω and it is the entire data that the algorithm needs to synthesize the texture.

However, due to ξ, we need to choose which $\Lambda \leq S$ samples from Ω_S we store. To do so, we run the k-medians algorithm [4], with an ℓ^2 distance and Ω_S as the input. The algorithm chooses the Λ cluster centers, $\omega_{\lambda_{cand}}$, from the

[3]The filter bank consists of 48 filters: first and second derivates of Gaussians at 6 orientations and 3 scales, 8 Laplacian of Gaussian and 4 Gaussian filters. We used the default parameters of [22].

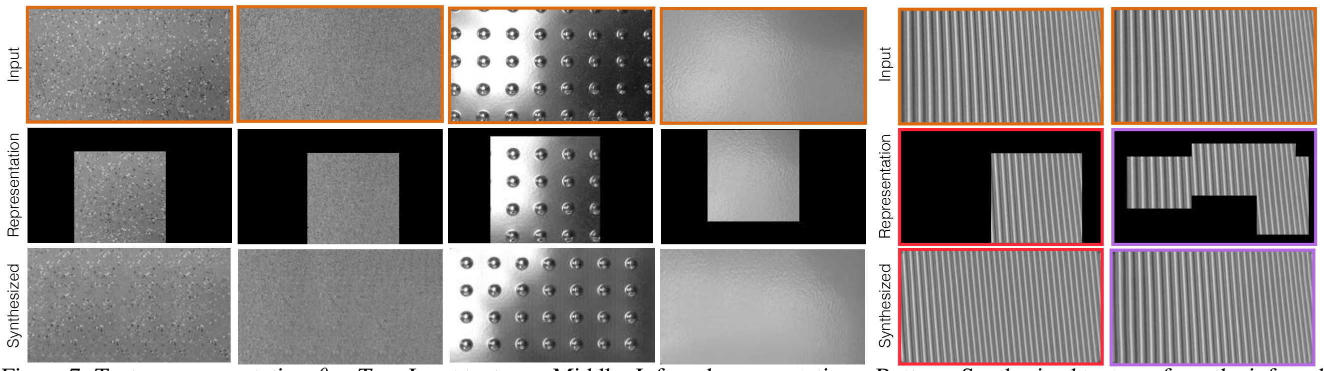


Figure 7. *Texture representation, $\theta_{\bar{\omega}}$. Top: Input textures. Middle: Inferred representations. Bottom: Synthesized textures from the inferred representation. Right pair of images: Complexity ξ determines the representational power of $\theta_{\bar{\omega}}$. Increasing the number of stored samples, allows the representation to capture the domain transformation.*

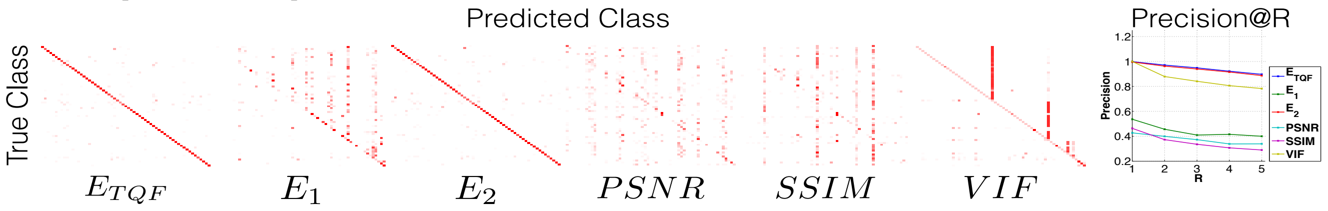


Figure 8. *Left: Confusion tables for six competing methods. Right: Precision of methods for various values of retrieved nearest neighbors.*

set Ω_S that minimize the distance of all samples with their clusters. In this sense, they are the most "representative" samples of the underlying process. Since the cluster centers are samples from the distribution, if r_{cand} is the true Markov neighborhood scale, then the cluster centers approximate the variability of the texture. The cluster centers form $\bar{\omega}_{cand} \doteq \bigcup_{\lambda=1,\ldots,\Lambda} \omega_{\lambda_{cand}}$, which we can use to compute $\theta_{\bar{\omega}_{cand}}$. Using $\theta_{\bar{\omega}_{cand}}$, we re-synthesize the textures at multiple scales according to Sec. 4.1. We repeat this procedure for all r_{cand}. We choose $\hat{r}$ (an estimate of r) to be the scale that minimizes TQC. This ensures that the chosen $\hat{r}$ synthesizes the most similar texture among all r_{cand} and hence captures best the Markov neighborhood structure. We discretize the space of $\hat{r}$ to $\left[4^2, 8^2\right] \cup \{k \times 16^2\}_{k=1}^{K}$, where K is bounded by the scale of Ω. When Ω is unknown, we can bound the space of $\hat{r}$ to the image size, although empirically it can be set to a much lower value (e.g. 128×128 for 640×480 images). Once $\hat{r}$ is selected, we retrieve the corresponding estimates of $\bar{\omega}$ and $\theta_{\bar{\omega}}$ computed at scale $\hat{r}$. For video, we repeat this process on each frame independently.

4.6. Extending the representation to multiple scales

To use the TQC, we need to assume that there is only one texture in the image domain. This poses issues when inferring the representation $\theta_{\bar{\omega}}$ and when evaluating the quality of reconstruction for images with more than one texture. To circumvent that, we partition the image domain into smaller regions and infer the texture representations on each one of these independently. We partitioned 640×480 frames into regions of 128×128, which in general gave us good results.

Since there is still no guarantee that there will not be more than one texture in each of these subsets, we allow the texture representation to be slightly more flexible: we allow ω_λ to take any of 3 different scales. Based on the restriction imposed by ξ, we distribute the available number of samples, Λ, to samples at 3 different scales *i.e.*, $(\Lambda_r, \Lambda_{r/2}, \Lambda_{r/4})$, computed so that they satisfy the same complexity constraint. Since multiple choices would achieve this condition, we try all of them sequentially.

The above approach leads to forming a representation that is multi-scale: $\theta_{\bar{\omega}} = \{\theta_{\bar{\omega}_r}, \theta_{\bar{\omega}_{r/2}}, \theta_{\bar{\omega}_{r/4}}\}$, where the subscript on $\bar{\omega}$ denotes the scales of the samples ω_λ that belong to that particular $\bar{\omega}$. Like before, we repeat this process for various candidates of r and choose the best one according to TQC. Note that synthesis can still take place on the whole image domain and it is independent of the partitioning done for the inference of the texture representation.

5. Experiments

Texture Representation Analysis. To qualitatively evaluate the representational power of our scheme, we show in Fig. 7 a number of examples. With the exception of the example on the right, the algorithm picked in all others to represent the texture with just one sample, $\Lambda = 1$. $r = 96 \times 96$ for textures 1, 2, 4 and $r = 112 \times 112$ for texture 3. This is due to two reasons: (i) by storing ω_λ with large r, we can select neighborhoods at smaller scales within the stored ω_λ in texture synthesis, (ii) the textures shown with the exception of the last one, do not exhibit significant variation to require more than one sample to be stored. The first two exhibit little variation. In the next two, the representation captures the photometric variation by simply selecting (correctly) an ω_λ that exhibits this variation. The texture on the right exhibits a domain deformation. The algorithm is unable to identify

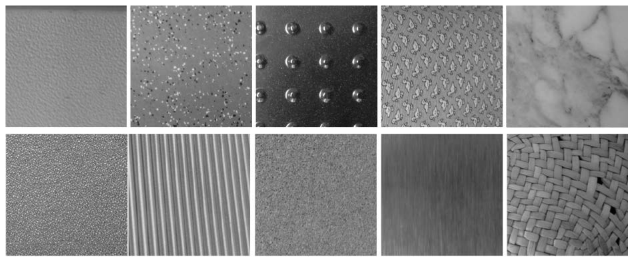


Figure 9. *Texture dataset: 10 randomly selected examples.*

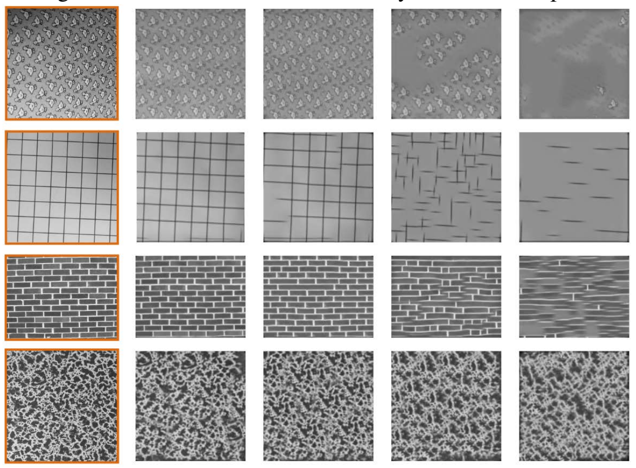


Figure 10. *Texture Qualitative Function (TQC): Ordered synthesized textures using TQC. Left: Original textures. Right: Synthesized textures, left being the most similar to the input texture.*

this variability with $\Lambda = 1$ and $r = 96 \times 96$ (second to last column), but with more samples stored (rightmost column), it is able to do so. In that case, $\Lambda = 5$ and $r = 64 \times 64$. In Fig. 13, we show inferred representations for video. In such video sequences the structures provide boundary conditions between textures and the stored samples allow localization of the various transformations occurring on the textures.

Texture Qualitative Criterion. To evaluate *TQC*, we have constructed a dataset[4], made out of 61 classes of textures, with 10 samples in each class (10 randomly selected examples of classes are shown in Fig. 9). Each sample is compared against the other 609 texture images using six different quantities: E_{TQC}, E_1, E_2, $PSNR$, $SSIM$ [36] and VIF [30]. For each image we retrieve R nearest neighbors using each of the six quantities and identify the class they belong to. We compute confusion matrices and show the result for $R = 5$ in Fig. 8, where the results are accumulated over all images. Furthermore, we plot the precision of each of the six methods also in Fig. 8, for $R = 1, \ldots, 5$. E_{TQC} performs slightly better than E_2 and significantly better than the other methods. Note that E_1 is equivalent to [37], without using a control map.

To qualitatively evaluate *TQC*, we synthesized a number of textures using a varying ξ and ω. We used *TQC* to order the synthesized textures and we show the ordered results in

[4] http://vision.ucla.edu/~giorgos/cvpr2015/

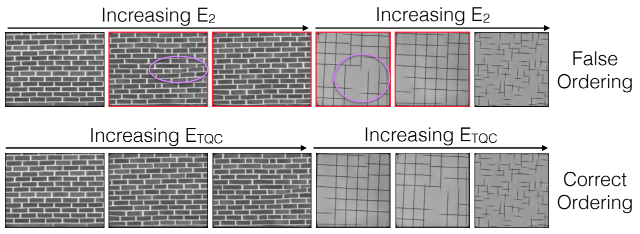


Figure 11. *Two examples where E_2 fails and E_{TQC} succeeds in ordering the synthesized textures correctly with respect to the input texture. Images with a red outline have been incorrectly ordered. The issue arises mainly in regular textures. The regions within the purple ellipses are major sources of error.*

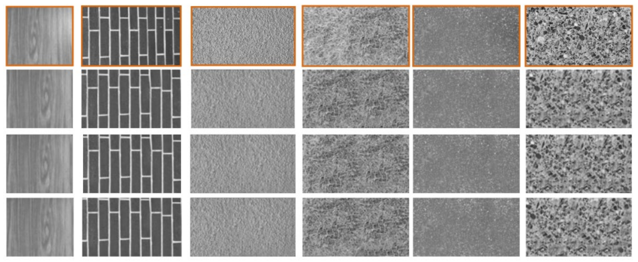


Figure 12. *Temporal texture synthesis. Synthesizing a novel instance of a texture in a video sequence. The first frame of a 20-frame long input video sequence is shown in the top row. 1st, 10th and 20th synthesized frames are shown below each input image.*

Seq-1	**Seq-2**	**Seq-3**	**Seq-4**	**Seq-5**	**Seq-6**
358(61%)	430(61%)	707(59%)	499(64%)	435(63%)	708(60%)
359(41%)	429(40%)	703(40%)	501(41%)	437(41%)	708(38%)
Seq-7	**Seq-8**	**Seq-9**	**Seq-10**	**Seq-11**	**Seq-12**
618(55%)	480(56%)	371(63%)	557(54%)	206(58%)	452(61%)
616(37%)	482(38%)	372(41%)	557(37%)	207(38%)	451(40%)

Table 1. *Video texture synthesis evaluation.* TQC *for each of the 12 sequences on the texture regions (smaller is better). In brackets, we show the corresponding compression percentage achieved.*

Fig. 10. The biggest benefit of E_{TQC} over E_2 is shown in Fig. 11. E_2 fails to detect artifacts in the synthesized textures, especially in regular ones, since the pooled statistics do not reveal any inconsistencies. On the other hand, E_{TQC} is able to identify these cases, since samples in the synthesized texture are artifacts of synthesis and hence are not present in the input texture.

Texture Synthesis. To evaluate the temporal extension of the synthesis algorithm, we decouple the representation from the synthesis procedure by letting $\xi = 1$. We initialize the output to a random video of the same length and size as the input. We use 20 frame long video input sequences to produce a novel instance of the texture (Fig. 12). Synthesized textures capture the statistics of the inputs and are temporally consistent.

Video Hole-Filling for Multiple Textures.

We synthesize multiple textures in the first 5 frames of 12 videos from the MOSEG dataset[5] at two different compression ratios using the inferred representations. We store the intensity values in 8-bit unsigned integers and the locations of the centers of ω in 16-bit unsigned in-

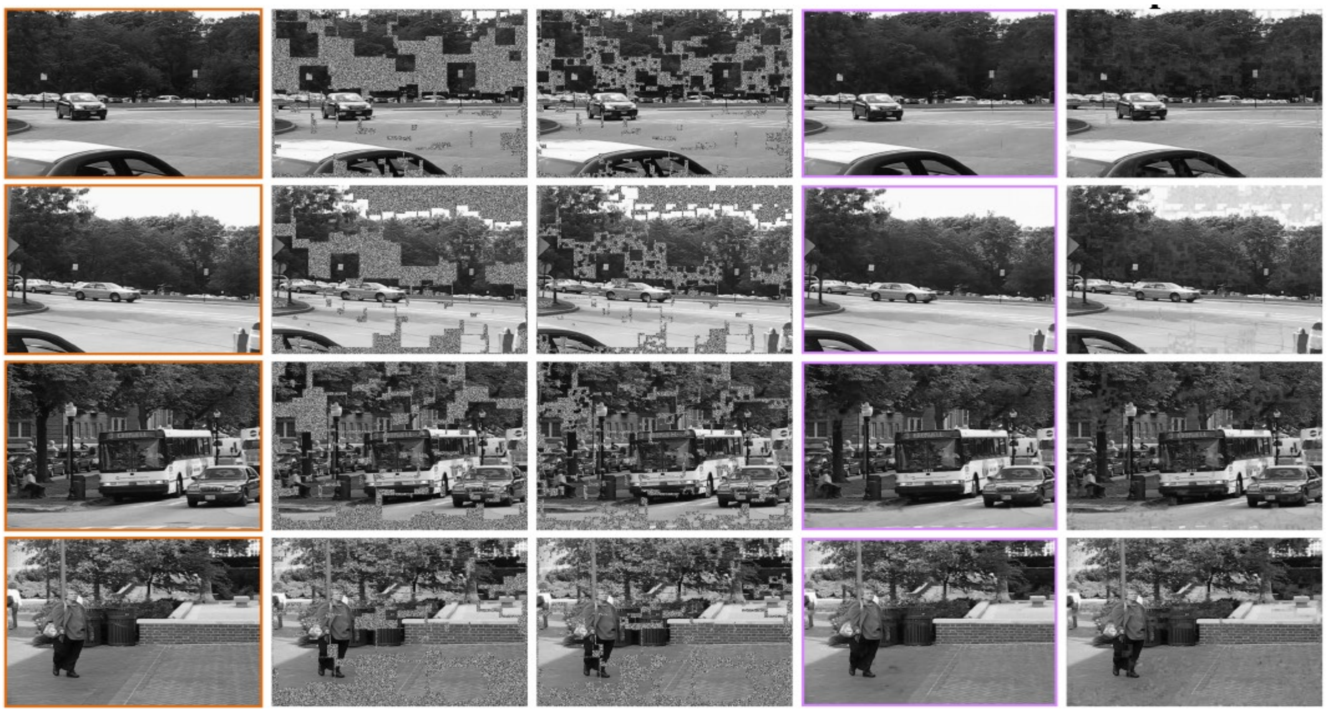


Figure 13. *Video texture synthesis in natural images (Hole-filling). From left to right: (i) Last frame (5th) of input video, (ii) Structure regions, (iii) Structure / Texture regions, (iv) Synthesized frame (our result), (v) Video Epitome [6] (zoom in to view details).*

tegers. The space required to store the representation is $\Xi(\bar{\omega}) = \Lambda \times r \times 8 + \Lambda \times 2 \times 16$. The space required to store the texture regions without using the representation is $\Xi(\Omega) = 8 \times |\Omega|$. We let the compression ratio be $\xi = \frac{\Xi(\bar{\omega})}{\Xi(\Omega)}$. In Fig. 13, we show the last frame of the synthesized videos at $\xi = 40\%$. The results show that we can successfully compress textures in video using our representation; during decoding, the original video can be recovered by hole-filling. Note that we have also overlaid the detected structures on each frame. We control the number of feature points detected and hence the structure/texture partition, by empirically adjusting the detection threshold to retain in general long and stable tracks. In the same figure, we also show the results from [6], the most relevant work to ours, for the same target compression. We use their publicly available code and we have done our best to optimize the parameters to get the best results possible. Since their approach does not explicitly model textures, their method fails in synthesizing them well. For example, in the second sequence, the sky is only barely part of the "known" region and while our method is able to extrapolate and fill in the rest of it by synthesis, [6] is not able to do so from their summarization (epitome). In Table 1 we show the value of *TQC* for two target compression levels used in the experiments ($\xi = 40\%$ and $\xi = 60\%$) for each of the 12 video sequences. Examining the synthesized video sequences, it can be observed that the spatial artifacts for both methods are comparable (hence the values of *TQC* are approximately the same), but the temporal artifacts are significantly more for the higher compressed video, which *TQC* does not capture. An extension of this work would be to compute *TQC* on 3D samples, rather than 2D. For additional results, please refer to the project website[4].

Future challenges. When the system is used at high compression ratios, synthesizing large holes becomes challenging. Extending our coarse-to-fine approach to hole-filling could improve the results. In addition, to infer the representations in these videos, our algorithm requires an amount of time that is in the order of hours. Synthesis of frames typically takes 5-10 minutes. These times are comparable to similar methods [6, 20]. The reported time refers to a MATLAB implementation on an Intel 2.4 GHz dual core processor machine. In future work, we plan to find efficient approximations to speed up inference.

6. Conclusion

A texture region exhibits a form of spatial regularity, hence it lends itself to spatial image compression techniques. We presented a new texture representation that requires considerably less space to store than the input texture, while at the same time it retains many important characteristics: it can still be used as input to texture classification tasks and image and video texture synthesis. We presented a causal video compression scheme that utilizes the proposed representation and demonstrated how this can be used for hole-filling and texture compression in video. Finally, we proposed a new criterion to compare two textures that measures structural and statistical dissimilarity.

Acknowledgments. We would like to thank Virginia Estellers and Vasiliy Karasev for valuable discussions. Research supported by ONR N00014-13-1-034, ARO W911NF-11-1-0391 and AFOSR.

References

[1] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman. Texture mixing and texture movie synthesis using statistical learning. *TVCG*, 2001. 2

[2] A. C. Beers, M. Agrawala, and N. Chaddha. Rendering from compressed textures. In *ACM SIGGRAPH*, 1996. 2

[3] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher. Simultaneous structure and texture image inpainting. *TIP*, 12(8):882–889, 2003. 4

[4] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Clustering via concave minimization. *NIPS*, 1997. 5

[5] T. Brox and J. Malik. Object segmentation by long term analysis of point trajectories. *ECCV*, 2010. 7

[6] V. Cheung, B. J. Frey, and N. Jojic. Video epitomes. *IJCV*, 2008. 1, 2, 8

[7] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. *CVPR*, 2014. 1, 3

[8] G. Doretto, A. Chiuso, Y. Wu, and S. Soatto. Dynamic textures. *IJCV*, 2003. 2

[9] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. *ICCV*, 1999. 1, 2, 3, 4

[10] S. Fenney. Texture compression using low-frequency signal modulation. In *ACM SIGGRAPH*, 2003. 2

[11] W. Freeman and C. Liu. Markov random fields for super-resolution and texture synthesis. *Advances in Markov Random Fields for Vision and Image Processing*, 2011. 1

[12] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *TPAMI*, 1984. 2

[13] C. Guo, S. Zhu, and Y. N. Wu. Toward a mathematical theory of primal sketch and sketchability. *ICCV*, 2003. 4

[14] C. Guo, S. C. Zhu, and Y. Wu. Primal sketch: Integrating structure and texture. *CVIU*, 2007. 1

[15] R. M. Haralick. Statistical and structural approaches to texture. *Proceedings of the IEEE*, 2005. 2

[16] N. Jojic, B. J. Frey, and A. Kannan. Epitomic analysis of appearance and shape. *ICCV*, 2003. 2

[17] A. Kannan, J. Winn, and C. Rother. Clustering appearance and shape by learning jigsaws. *NIPS*, 2006. 2

[18] I. Kokkinos, G. Evangelopoulos, and P. Maragos. Texture analysis and segmentation using modulation features, generative models, and weighted curve evolution. *PAMI*, 31(1):142–157, 2009. 2

[19] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *Proc. of ACM SIGGRAPH*, 2005. 1, 3, 4

[20] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graph.*, 2003. 2, 8

[21] S. Lazebnik, C. Schmid, and J. Ponce. A sparse texture representation using local affine regions. *PAMI*, 2005. 2

[22] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *IJCV*, 2001. 5

[23] Y. Liu, W.-C. Lin, and J. H. Hays. Near-regular texture analysis and manipulation. *SIGGRAPH*, 2004. 2

[24] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *IJCAI*, 1981. 4

[25] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *JOSAA*, 1990. 2

[26] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *PAMI*, 2002. 2

[27] P. Mavridis and G. Papaioannou. Texture compression using wavelet decomposition. *Proceedings of Pacific Graphics*, 2012. 2

[28] M. Pedersen and J. Hardeberg. Full-reference image quality metrics: Classification and evaluation. *Found. Trends. Comp. Graphics and Vision.*, 2012. 5

[29] B. Sandberg, T. Chan, and L. Vese. A level-set and Gabor-based active contour algorithm for segmenting textured images. *UCLA CAM report*, 2002. 4

[30] H. Sheikh and A. Bovik. Image information and visual quality. *Image Processing, IEEE Transactions on*, 2006. 5, 7

[31] L. Sifre and S. Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. *CVPR*, 2013. 1, 3

[32] P. H. Suen and G. Healey. Analyzing the bidirectional texture function. *CVPR*, 1998. 2

[33] P. Teo and D. Heeger. Perceptual image distortion. *ICIP*, 1994. 5

[34] M. Varma and A. Zisserman. Texture classification: Are filter banks necessary? *CVPR*, 2003. 2

[35] H. Wang, Y. Wexler, E. Ofek, and H. Hoppe. Factoring repeated content within and among images. In *ACM TOG*, 2008. 1, 2

[36] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE TIP*, 2004. 5, 7

[37] L. Wei, J. Han, K. Zhou, H. Bao, B. Guo, and H. Shum. Inverse texture synthesis. *ACM Trans. Graph.*, 2008. 1, 5, 7

[38] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. *SIGGRAPH*, 2000. 2

[39] Y. Wexler, E. Shechtman, and M. Irani. Space-time completion of video. *PAMI*, 2007. 2

[40] S. Xiaowei, Y. Baocai, and S. Yunhui. A low cost video coding scheme using texture synthesis. In *ISP*, Oct 2009. 2

[41] H. Zhi, Z. Xu, and S.-C. Zhu. Video primal sketch: A generic middle-level representation of video. *ICCV*, 2011. 1

[42] S. W. Zucker. Toward a model of texture. *CGIP*, 1976. 2