Multi-Scale Label-Map Extraction for Texture Synthesis

Yitzchak David Lockerman[1], Basile Sauvage[2], Rémi Allègre[2], Jean-Michel Dischler[2], Julie Dorsey[1], Holly Rushmeier[1]
[1] Yale University, Computer Graphics Group, New Haven, USA
[2] ICube, Université de Strasbourg, CNRS, France

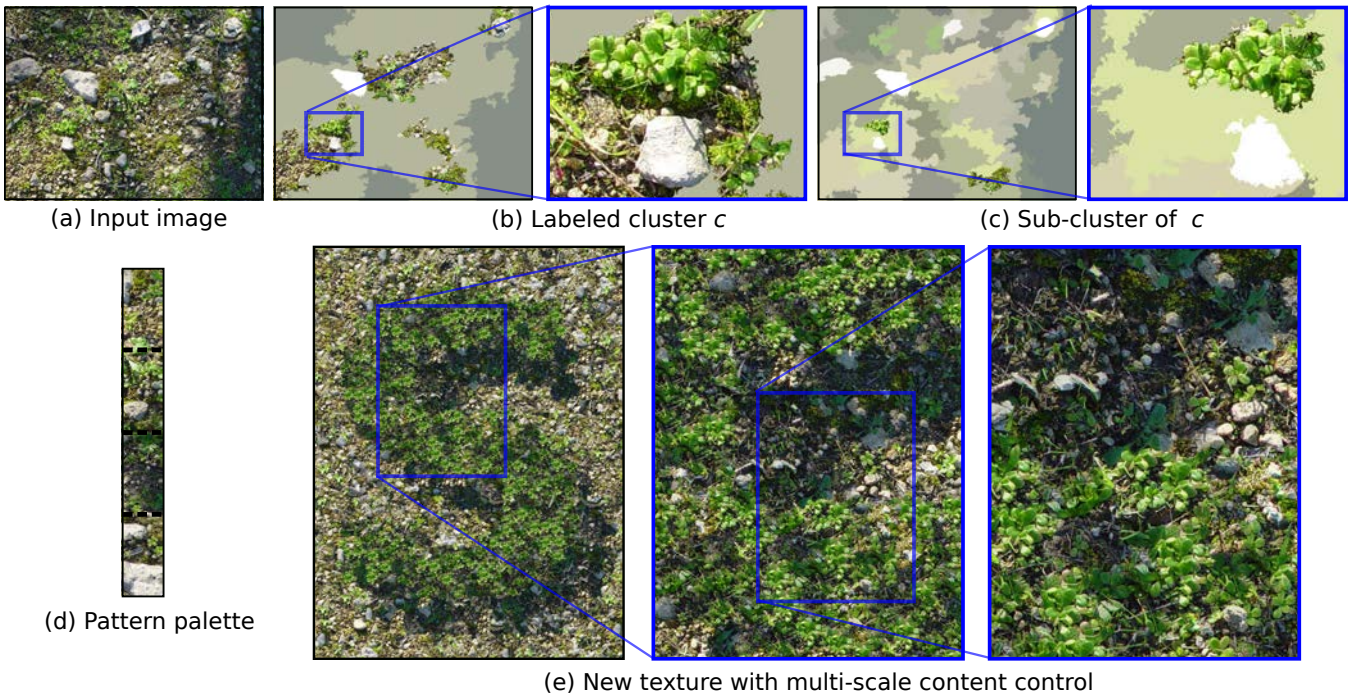


Figure 1: *The input image (a) is a non-stationary texture. Our multi-scale analysis provides a hierarchy of labeled clusters: one cluster at a coarse scale (b) includes sub-clusters at finer scales (c). It has several applications in texture synthesis, such as automatic pattern palettes for interactive texture editing (d) and content selection for creating new non-stationary textures (e).*

Keywords: texture analysis, pattern labeling, multi-scale clustering, superpixels, non-stationary textures, texture editing.

Concepts: •**Computing methodologies** → **Texturing;** *Image processing;*

Abstract

Texture synthesis is a well-established area, with many important applications in computer graphics and vision. However, despite their success, synthesis techniques are not used widely in practice because the creation of good exemplars remains challenging and extremely tedious. In this paper, we introduce an unsupervised method for analyzing texture content across multiple scales that automatically extracts good exemplars from natural images. Unlike existing methods, which require extensive manual tuning, our method is fully automatic. This allows the user to focus on using texture palettes derived from their own images, rather than on manual interactions dictated by the needs of an underlying algorithm.

Most natural textures exhibit patterns at multiple scales that may vary according to the location (non-stationarity). To handle such textures many synthesis algorithms rely on an analysis of the input and a guidance of the synthesis. Our new analysis is based on a labeling of texture patterns that is both (i) multi-scale and (ii) unsupervised – that is, patterns are labeled at multiple scales, and the scales and the number of labeled clusters are selected automatically. Our method works in two stages. The first builds a hierarchical extension of superpixels and the second labels the superpixels based on random walk in a graph of similarity between superpixels and a nonnegative matrix factorization. Our label-maps provide descriptors for pixels and regions that benefit state-of-the-art texture synthesis algorithms. We show several applications including guidance of non-stationary synthesis, content selection and texture painting. Our method is designed to treat large inputs and can scale to many megapixels. In addition to traditional exemplar inputs, our method can also handle natural images containing different textured regions.

SIGGRAPH '16 Technical Paper,, July 24-28, 2016, Anaheim, CA,
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925964

1 Introduction

Texture synthesis has received considerable attention and has now reached a certain degree of maturity. In spite of substantial

progress, automatic synthesis algorithms are still rarely used in practical content production applications. One reason is that high quality textures must be both multi-scale and non-stationary. Real-world textures indeed exhibit patterns at different visual scales. (By "pattern" we mean a distinctive shape and its content). For instance, a cluster in Figure 1 (large-scale pattern) contains many leaves (fine-scale, sub-patterns); textures contain repetitions of similar patterns at all these scales. Moreover natural textures are rarely stationary, as the distribution of patterns may vary across the texture (e.g. shady areas Figure 1). Despite this, non-stationarity is still an open and difficult problem for by-example texture synthesis. A recent study about "synthesizability" [Dai et al. 2014] shows that "homogeneous" (i.e. stationary) textures are easier to synthesize and the homogeneity criterion performs the best in predicting "how well present synthesis algorithms perform". Indeed, most tile, patch, pixel or optimization-based methods assume stationarity.

A popular approach to overcome this limitation is to spatially distribute patterns according to label-maps which encode non-stationarity, with each label representing a given pattern. The pipeline generally calls for, (i) defining a label-map for the input, (ii) synthesizing a label-map for the output, and (iii) synthesizing the output texture. Whereas a large number of algorithms are able to use label-maps for guiding texture synthesis, extraction and synthesis of the label-maps have been given very little attention. In this paper we tackle automatic extraction of label-maps from an input exemplar and demonstrate that our technique may greatly improve state-of-the-art synthesis algorithms.

The key idea of our approach is to consider stationarity as a scale-dependent and context-dependent property. Dai et al. [2014] noticed that scale matters; when zooming in, a repeated pattern becomes an "individual object" which is difficult to synthesize. In other words, a stationary distribution of similar patterns becomes non-stationary as the scale of the patterns reaches the size of the exemplar. We can also analyze stationarity in a given context. For instance, in Figure 1 the non-stationary distribution of leaves becomes stationary when regarding only clumps of leaves (context without stones, moss, etc.). Our idea is to look at large scale patterns as a "non-stationary context" for stationary distributions of sub-patterns. We exploit these two ideas in our label-map extraction algorithm where patterns are labeled not only at multiple scales but also in their context, namely as sub-patterns of larger scale patterns.

Our algorithm takes a large image as input and extracts a multi-scale label-map in two stages:

- *Partitioning* aims at *detecting* patterns (section 3). It works locally and produces connected components, namely superpixels. However, standard superpixels detect patterns with homogeneous color while we aim at detecting patterns containing sub-patterns, possibly with strong color variations. Our contribution is to extend SLIC superpixels hierarchically so as to detect patterns at multiple scales.
- *Labeling* aims at *describing* patterns (section 4). The issue is to identify similar patterns throughout the texture. It works globally and produces a label-map that describe the texture content at multiple scales. Our contribution is to state the problem as a random walk in a bipartite graph which leads to a nonnegative matrix factorization problem. Then we propose a new factorization algorithm adapted to our large and sparse matrix situation. This algorithm is unsupervised in the sense that no prior training is required and the number of labels is computed automatically.

Any pixel (section 5) and any region (section 6) of the texture can be described using the same labels. The main application in texture synthesis is that it allows for instant selection of contents at any scale, which is useful for guiding non-stationary synthesis and for interactive texture painting of extremely large outputs. We furthermore show benefits for extraction of stationary textures in arbitrary images.

2 Related work

To place our work in context, we review relevant texture extraction and synthesis methods.

2.1 Texture extraction

Texture extraction consists of identifying and labeling regions with similar textures in an image. This is different from the well studied problem of segmentation, which is comparable to the first stage of our algorithm (local partitioning). Indeed, extraction methods can afford to be extremely conservative when separating regions. If a segmentation algorithm were as conservative, it would produce an over-segmentation. Though it is possible to use segmentation algorithms to extract textures, it leads to poor results [Lu et al. 2009a]. In section 3.5 we provide a comparison between our partitioning algorithm and hierarchical segmentation [Arbeláez et al. 2011].

A first series of methods are "dominant texture extraction" in which pixels are labeled as inside/outside a region of interest. It is motivated by the observation that most natural images are inappropriate exemplars for synthesis algorithms because they are non-stationary or contain outliers. The goal is to select an appropriate region for synthesis. Lu et al. [2009a] extracted a dominant texture from kilopixel images using diffusion methods and performed a psychophysical study to demonstrate the performance of their method [Lu et al. 2009b]. Lockerman extended this approach [2013] so as to manage various inputs, larger images (megapixel), and allow user selection of scale and location of interest. Wang and Hua [2013] provided another method for dominant texture extraction which is orders of magnitude faster then Lu et al. Based on a prior supervised learning on a large set of exemplars, Dai et al. [2014] predicted a best candidate square region as input for synthesis. These existing "dominant texture extraction" methods differ from our approach in that they either require user input or make assumptions about the existence of one dominant texture. Our algorithm does not require either.

A second series of methods extract multiple classes. Layered Shape Synthesis [Rosenberger et al. 2009] considers a texture as an ordered set of homogeneous layers nested within each others. In a pre-processing stage they extract these layers from the input and generates a label-map with one label per layer. This is limited to specific types of natural textures that result from processes such as weathering or corrosion. In their Composite Texture Synthesis method, Zalesny et al. [2005] compute a mono-scale label-map using an unsupervised algorithm based on first and second image statistics. The authors point out the need for a multi-scale approach in order to detect sub-patterns in certain examples.

To our knowledge, no existing method is capable of extracting textures at multiple scales. Our method overcomes this limitation.

2.2 Texture synthesis using label-maps

A thorough review of texture synthesis methods is beyond the scope of this paper since synthesis is not a contribution here. We rather intend to bring out the broad usage of label-maps in various synthesis algorithms, as they have been used under different names (guidance-, control-, feature-, parameter-, correspondence-, or label-maps) to guide the synthesis. Different properties of the synthesized texture may be guided: the location of contents (non-stationarity, including so-called "composite textures"), the distribu-

tion of patterns (regularity), or the shape of patterns (e.g. sharp features).

Because of their very local nature, pixel-based methods struggle with the control of pattern's location. Thus label-maps are popular for guiding non-stationarity [Ashikhmin 2001; Hertzmann et al. 2001; Zalesny et al. 2005; Mertens et al. 2006; Rosenberger et al. 2009], pattern size and orientation [Zhang et al. 2003; Lefebvre and Hoppe 2006], or perspective distortions [Wu et al. 2016].

A series of methods processes a global optimization by iterative replacements of local neighborhoods. Label-maps have been used in this context to guide non-stationarity [Narain et al. 2007] or shape and distribution of patterns [Kaspar et al. 2015].

Patch-based approaches proceed by copying and pasting large pieces of textures [Efros and Freeman 2001; Cohen et al. 2003; Kwatra et al. 2003; Vanhoey et al. 2013]. Label-maps have been used for guiding non-stationarity [Efros and Freeman 2001; Diamanti et al. 2015]. These methods effectively preserve individual patterns, provided that the patches have an appropriate size and shape with respect to the content, but the transitions between patterns are difficult to hide. Our multi-scale labeling improves on several difficult stages of theses methods such as size and content selection.

Controlling the content is a challenging task for synthesis methods and thus many of them stumble over non-stationarity. Therefore, a common strategy uses label-maps to guide the synthesis but they often require a manual labeling of the input. Even those that do automatic analyses make strong assumptions about their exemplars, such as requiring the entire exemplar to contain a single texture [Kaspar et al. 2015]. A way to describe any content in terms of these labels is also needed. Our method provides fully automatic tools to solve these issues using arbitrary natural images as exemplars, as demonstrated in sections 5 and 6.

3 Hierarchy of superpixels

Superpixels (SP) are small connected components with homogeneous appearance that form a partition of an image. They are popular as pre-processing step for image analysis and computer vision applications. In this section we first summarize the original SLIC algorithm and comment on its limitations with respect to our setting. Then we describe our hierarchical extension that detects patterns with sub-patterns and color variations. The notation is summarized in Table 1.

3.1 Original SLIC algorithm

SLIC is a highly regarded algorithm for finding superpixels (SP) with a homogeneous color. It is similar to k-means clustering, with modifications to ensure both spatial and color coherence of the clusters. It operates on a 5 dimensional space (2 spatial and 3 color channels) along with the following distance metric:

$$\delta_{\sigma,m,I}\left(\mathbf{x},\mathbf{x}'\right) = \sqrt{\|\mathbf{x}-\mathbf{x}'\|^2_{color} + m^2\frac{\|\mathbf{x}-\mathbf{x}'\|^2_{spatial}}{\sigma^2}} \quad (1)$$

where sample vectors $\mathbf{x}$ contain the spatial position $\mathbf{p}$ and the color $I(\mathbf{p})$. The scale parameter $\sigma = \sqrt{I_x I_y/\#superpixels}$ tunes the scale of SPs or, equivalently, the approximate number of SPs. The compactness parameter m tunes between sensitivity to distance and sensitivity to color.

	Input
I	input image
I_x, I_y	input height and width
$\mathbf{p}$	pixel (position)
$I(\mathbf{p})$	pixel (color)
	Superpixels
$1 \leq \gamma \leq \Gamma$	levels of superpixels
σ_γ	scale of level γ
$\mathbb{S}^\gamma$	partition at level γ
S	superpixel (element of $\mathbb{S}^\gamma$)
$\mathbf{x}$	pixel + color vector (5D)
$\overline{\mathbf{x}}_S$	average of $\mathbf{x}$ over S
	Labels / clusters
$1 \leq l \leq L$	levels of labels
ρ_l	scale of level l
C^l	label-map at level l

Table 1: *Notations. The label-map C^l may be seen as a function assigning a label to each superpixel or pixel. From the "set theory" viewpoint it is a set partition, each subset being a cluster with a unique label.*

3.2 Limitations

SLIC superpixels' usefulness stems from their ability to fit the edges in the image. The algorithm works well for small superpixels with a homogeneous color. However, as shown in 3rd row Figure 2, when σ is large, SLIC is unable to distinguish between pattern and sub-pattern edges. Petals are merged with the wall behind and leaves are merged with concrete. The problem is partially solved by low-pass filtering of the image (Figure 2, 4th row). While removing the fine sub-patterns, this also blurs the pattern edges. Our solution (Figure 2, 2nd row) ignores edges *within* patterns while keeping edges *between* patterns precisely located.

The SLIC algorithm is also very sensitive to the scale parameter σ: a slight change in σ or initial seeds may result in a completely different set of SPs. This can be seen Figure 2 and more clearly in the accompanying video (poor temporal coherence while σ changes gradually). In contrast our decomposition is much more robust because large scale SPs are constrained by fine ones. This robustness w.r.t. σ allows us to select a small set of representative scales (section 4) without magnifying errors caused by slight perturbations in the selected scale.

Finally, the SLIC algorithm is not conducive to comparing SPs calculated at different scales. In general, any SP at any scale can partially overlap with any number of SPs at any other scale. Conversely, our decomposition produces a hierarchical structure where a fine scale SP will never be divided at larger scales. This greatly simplifies the use of our decomposition in hierarchical labeling.

3.3 Hierarchical partitioning

Our algorithm builds a hierarchy of superpixel partitions $\mathbb{S}^\gamma$ operating at scale σ_γ, $1 \leq \gamma \leq \Gamma$. By hierarchy we mean:

- The scales are increasing: $\sigma_{\gamma-1} < \sigma_\gamma$.
- $\mathbb{S}^{\gamma-1}$ is a sub-partition of $\mathbb{S}^\gamma$, i.e. every SP of $\mathbb{S}^{\gamma-1}$ is included in one SP of $\mathbb{S}^\gamma$.

As described in Algorithm 1 the partitioning starts with a fine scale partition $\mathbb{S}^1$. Then fine scale SPs are iteratively merged into larger scale SPs to get $\mathbb{S}^2$, $\mathbb{S}^3$, etc. The main advantage of this merging process is that large scale edges do respect fine scale patterns and

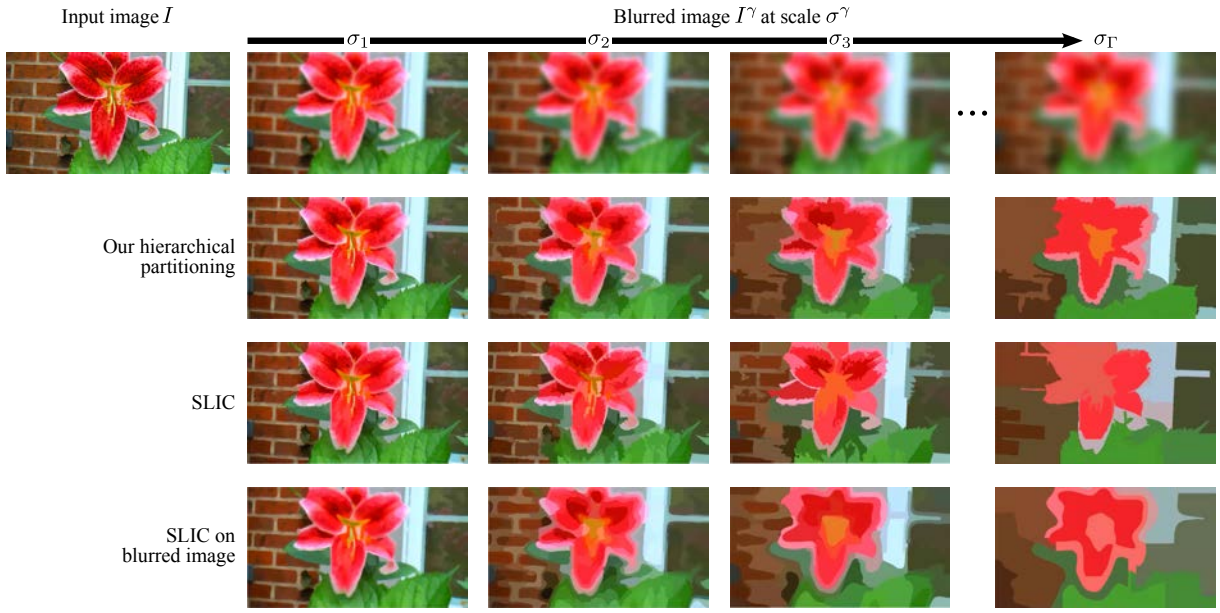


Figure 2: *Superpixel decomposition from our hierarchical algorithm (2nd row) compared to the SLIC algorithm on the original (3rd row) and on a blurred image (4th row). The average color of each superpixel is represented here.*

blurring is prevented. The inner loop stopping criterion ensures that σ_γ is the average scale of SPs in $\mathbb{S}^\gamma$.

The priority criterion for selecting the pair (R, S) to be merged is designed so that the SPs of $\mathbb{S}^\gamma$ match patterns at scale σ_γ. In other words we aim at merging patterns finer than σ_γ. We achieve this by guiding the process with the Gaussian stack of images defined by $I^\gamma = G_{\Delta\sigma} * I^{\gamma-1}$, where $I^1 = I$ and $G_{\Delta\sigma}$ is the Gaussian blur kernel of order $\Delta\sigma = \sigma_{\gamma+1} - \sigma_\gamma$. We then select the pairs (R, S) minimizing

$$|R|\, \delta^2_{\sigma_\gamma, m_\gamma, I^\gamma}\left(\bar{\mathbf{x}}_R, \bar{\mathbf{x}}_{R\cup S}\right) + |S|\, \delta^2_{\sigma_\gamma, m_\gamma, I^\gamma}\left(\bar{\mathbf{x}}_S, \bar{\mathbf{x}}_{R\cup S}\right). \quad (2)$$

As shown in Appendix A, this criterion locally minimizes the global error

$$\epsilon\left(\mathbb{S}^\gamma\right) = \sum_{S\in\mathbb{S}^\gamma} \sum_{\mathbf{p}\in S} \delta^2_{\sigma_\gamma, m_\gamma, I^\gamma}\left(\mathbf{x}, \bar{\mathbf{x}}_S\right), \quad (3)$$

where $\mathbf{x}$ is the 5D vector associated to $\mathbf{p}$ and $\bar{\mathbf{x}}_S$ denotes the average over S. It tends to produce compact SPs with respect to the distance metric (1): it balances between spatial coherence and color coherence.

To compute $\mathbb{S}^1$ we use a slightly modified form of the SLIC algorithm designed to work on a GPU so as to run faster. The most noticeable change from the original implementation is that we allow for a maximum of 1000 iterations and use a minimum superpixel size of 25 pixels. To improve result we also run the algorithm 10 times, and select the segmentation with the least global error as defined by equation (3).

3.4 Constants Selection

In theory we could examine all scales stepwise from a few pixels up to the full image. However, this would just provide a large amount of useless and redundant information. Instead, we examine scales regularly spaced by $\Delta\sigma = 3$ pixels from a minimum of $\sigma_1 = .02 \min(I_x, I_y)$ up to a maximum of $\sigma_\Gamma = .2 \min(I_x, I_y)$. This range allows us to keep most of the information needed for our application while still greatly accelerating the computation.

Input: image I.
Input: parameters $\{\sigma_1, \ldots, \sigma_\Gamma\}$ and $\{m_1, \ldots, m_\Gamma\}$.
Output: partitions $\{\mathbb{S}^1, \ldots, \mathbb{S}^\Gamma\}$.
$\mathbb{S} \leftarrow$ apply SLIC to I with parameters σ_1 and m_1
$\mathbb{S}^1 \leftarrow \mathbb{S}$
$I^1 \leftarrow I$
for $\gamma = 2 \ldots \Gamma$ **do**
  $I^\gamma \leftarrow G_{\Delta\sigma} * I^{\gamma-1}$
  for $S \in \mathbb{S}$ **do** recompute $\bar{\mathbf{x}}_S$ using I^γ
  while $\#superpixels > I_x I_y / \sigma_\gamma^2$ **do**
    Select the priority pair (R, S) according to Eq. (2)
    Merge superpixels R and S into $R \cup S$
  end
  $\mathbb{S}^\gamma \leftarrow \mathbb{S}$
end

Algorithm 1: Hierarchical partitioning.

We use $m_1 = 20$ for the initial segmentation to impose spatial regularity on the fine SPs. This compactness will be inherited by the higher level superpixels due to our hierarchical constraint. Thus, we don't need to give the same weight to compactness for higher scales. As such, we use a constant $m_\gamma = 1$ for $\gamma > 1$.

3.5 Comparison with segmentation

To validate our algorithms we prepared a benchmark provided in supplemental material. It consists of 15 large inputs with manually drawn ground truth labeling. We use the same ground truth to evaluate both our partitioning and labeling (section 4); connected components with the same label are considered as distinct regions in a segmentation point of view. Unlike segmentation data sets, which usually contain images with individual objects to be detected, we chose inputs more suitable for texture extraction: non-stationary textures (few perspective distortions and shadows) and natural images containing textures.

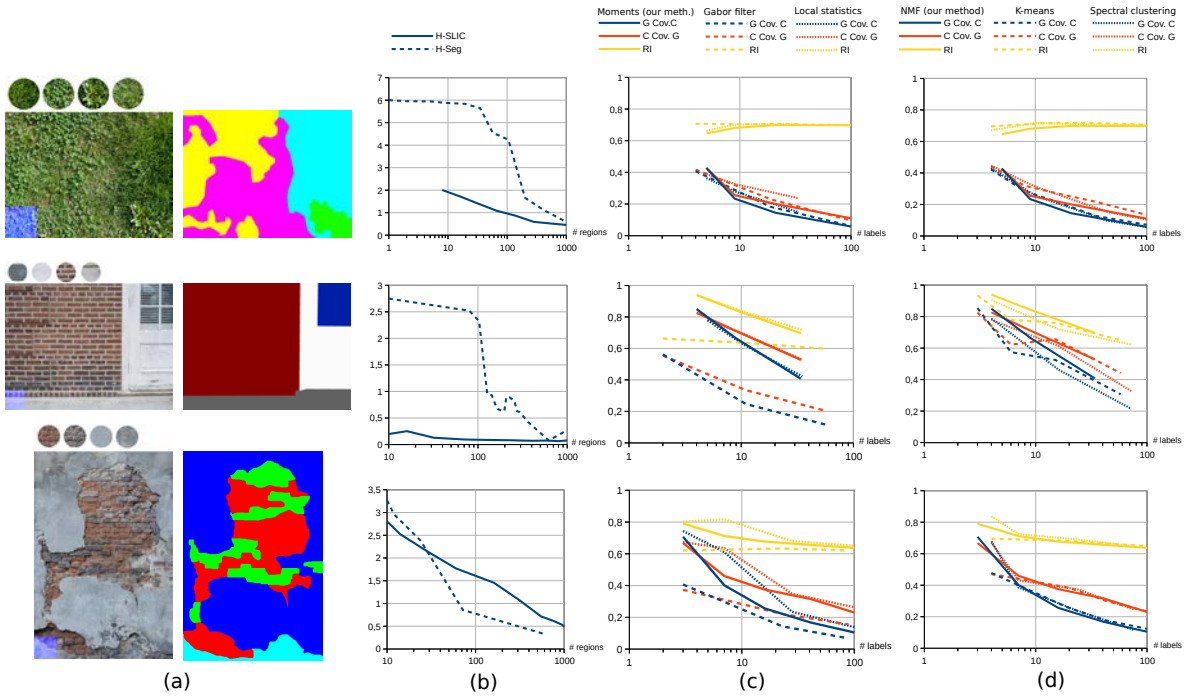


Figure 3: *Evaluation. (a) Input images with manual texture labeling taken as ground truth. (b) Our H-SLIC partitioning is compared to hierarchical segmentation using under-segmentation error (the lower the better). Our labeling is tested with alternative feature vectors (c) and compared to other clustering methods (d) using Rand index and covering measures (the higher the better).*

One issue is to compare multi-scale sequences $\{\mathbb{S}^1, \ldots, \mathbb{S}^\Gamma\}$ with a single scale ground truth $\mathbb{G}$. The idea is that we do not expect one single scale to perfectly match the ground truth but we want the hierarchy to potentially recover any ground truth. Thus we plot curves of the metrics on $(\mathbb{S}^\gamma, \mathbb{G})$ against $|\mathbb{S}^\gamma|$. So different sequences may be compared even if abscissas $|\mathbb{S}^\gamma|$ are not equal.

Our hierarchical SLIC superpixels algorithm (H-SLIC) is similar to segmentation though the regions we seek have different properties. SP partitions are a basis for later clustering (under-segmentation). In order to be useful each SP should ideally be contained in a single labeled region of the ground truth. This condition can be assessed by the under-segmentation error [Achanta et al. 2012]. We compare H-SLIC with the hierarchical segmentation (H-Seg) of Arbeláez et al. [2011]. Figure 3(b) show representative results (extensive results are provided in supplemental material). H-SLIC usually performs better, especially on non-stationary textures and on images with textures containing highly contrasted sub-patterns. H-Seg performs well when the labeled regions have sharp boundaries.

4 Multi-scale label-maps

In the previous section we *detected patterns* as superpixels (SP) with spatial coherence. Based on the resulting hierarchy of SPs, we now *label similar patterns* throughout the image, so as to be able to describe them and select them at different scales. At detection stage we used a dense set of scales in order to save maximum information about the content. Conversely the labeling stage identifies a small set of useful labels at a small set of scales. Patterns with different labels must be easy to distinguish and to describe, in order to be helpful during the synthesis process. Thus, our algorithm generates a sequence of scales $\rho_1 < \rho_2 < \ldots < \rho_L$ that are a subset of $\{\sigma_1, \ldots, \sigma_\Gamma\}$. At each level $1 \leq l \leq L$ the number of labels is automatically computed and then each SP at scale ρ_l is assigned a label, producing the label-map, C^l.

Our work is inspired by Lockerman et al. [2013] for measuring the similarity between patterns (section 4.1), with local scaling as proposed by Zelnik-Manor et al. [2004]. We formulate labeling as a random walk problem on a bipartite graph which we solve by Nonnegative Matrix Factorization (section 4.2). Then we propose a multi-scale clustering (section 4.3). Finally, we discuss our design choices and evaluate our algorithm (sections 4.4 to 4.7).

4.1 Similarity graph between patterns

Comparing patterns first requires describing them. As suggested by Lockerman et al. [2013] we use a 15 dimensional feature vector to represent each superpixel. The vector includes the first 5 centralized moments for each of the 3 Lab color channels. That is it includes the mean, standard deviation, and 3 higher moments. The kth moment is weighted by $\frac{1}{k!}$.

The distance d_{ij} between two SPs i and j is then defined as the L_2 norm of the difference between the corresponding feature vectors. We now need to turn the distance measure into a similarity measure suitable for later clustering. We define a similarity matrix $\hat{D}$ with entries

$$\hat{D}_{ij} = e^{\frac{-d_{ij}^2}{\hat{d}_i \hat{d}_j} e^{-r}}, \tag{4}$$

where $\hat{d}_i$ equals the distance between SP i and its 6-th closest neighbor as suggested by Zelnik-Manor et al. [2004]. It is a local scaling parameter that automatically adapts the measure to the local point density. e^{-r} is a global normalization parameter suggested by Lu. et. al. [2009a] where r maximizes

$$\frac{\partial}{\partial r} \log \sum_{i,j} \hat{D}_{ij} = \left(\sum_{i,j} \hat{D}_{ij} \right)^{-1} \sum_{ij} \frac{d_{ij}^2}{\hat{d}_i \hat{d}_j} e^{-r} \hat{D}_{ij} \tag{5}$$

The matrix is made sparse by saving the 256 largest entries in each

column (the remaining ones are set to zero) calculated by the ANN library [Mount 2010] with a $\epsilon = .001$. Then the columns are normalized, resulting in a sparse stochastic matrix D with entries

$$D_{ij} = \left(\sum_k \hat{D}_{kj} \right)^{-1} \hat{D}_{ij} \tag{6}$$

As illustrated in Figure 4 this can be interpreted as the transition matrix of a random walk on a directed k-nearest-neighbors graph whose nodes are SPs. Starting on any SP j the walker jumps to SP i with probability D_{ij}. A probability of presence is represented by a probability column vector p. Starting at p, one step leads to Dp and s steps lead to $D^s p$. This idea has been used on square tiles [Lockerman et al. 2013] as follows: starting at j, that is to say $p = \delta_j$ the Kronecker symbol, the probability $D^s \delta_j$ is interpreted as a map of similarity between all SPs and SP j (see the intensity of red dots in Figure 4). Inspired by this work, we go further by extracting clusters from the graph.

4.2 Mono-scale clustering

Our clustering assigns a label $C(i)$ to each SP i. We formulate the problem as a random walk on a bipartite graph. Then we propose a new algorithm that both (i) computes the number of clusters and (ii) clusters the SPs with maximum intra-cluster similarity.

In our case the similarity is described by a random walk on the graph of SP-nodes with transition probabilities D^s. Our key insight is treating a cluster as a latent variable embedded in the random walk: we assume that the graph has some hidden structure which the walker is discovering. We model this structure by a new graph, as illustrated in Figure 4: there is still a "SP-node" (in blue) for each superpixel, and there is a new "cluster-node" (in green) for each cluster. The graph is bipartite: edges are only permitted between the SP-nodes and the cluster-nodes. Each edge from a SP-node i to a cluster-node c is weighted by G_{cj}, which is intuitively the probability of the SP i to belong to cluster c. The reverse edge is weighted by F_{ic}: the intuition is that $F_{ic}/\max_j\{F_{jc}\}$ measures how well SP i fits in cluster c. In the end each SP i is labeled by the most probable cluster

$$C(i) = \underset{c}{\operatorname{argmax}}\, G_{ci} \tag{7}$$

The challenge consists in finding the number of cluster-nodes and the weights F and G. We formulate the problem as

$$\begin{aligned} \underset{F,G}{\operatorname{argmin}} \quad & \|D^s - FG\|^2 \\ s.t. \quad & F \text{ and } G \text{ are left stochastic} \\ & \text{(non-negative with all columns adding up to 1)} \end{aligned} \tag{8}$$

where matrix norm is Froebenius norm. It can be understood as follows: we search for a two-step random walk in the bipartite graph, from a SP-node to a cluster-node with probability G, and back to a SP-node with probability F. We want this process FG to be as equivalent as possible with the process D^s which describes the similarity between SPs (see section 4.1).

Equation (8) is a Nonnegative Matrix Factorization (NMF) problem. It is known for its ability to extract a small set of meaningful features [Gillis 2011; Gillis 2014] from large data: the number of representatives (i.e. columns in F, or rows in G, clusters in our case) is much smaller than the input data (D^s in our case).

Solving NMF is NP-hard in general [Gillis 2014] so many approximation heuristics have been created. However, our particular case's structure makes it hard to apply many existing methods. While D is a sparse matrix, D^s is, in general, an accountably large dense matrix which is prohibitively expensive to compute. Some existing algorithms are able to efficiently handle such matrices without explicit computation but are only to approximate our stochastic constraint [Berry et al. 2007]. As such, we have created our own family of algorithms specifically designed for our particular problem.

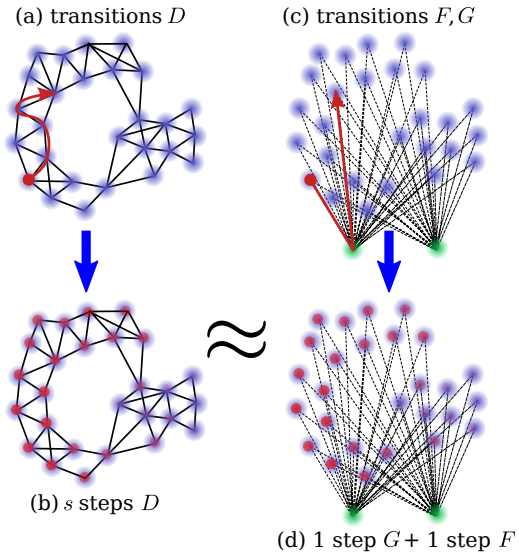


Figure 4: *Random walks on a bipartite graph. Each superpixel is represented by a $15D$ feature point which is a node of the directed k-nearest-neighbor graph (a) with transition probabilities based on the similarity between pairs of SPs. For any given starting node, random walks with several steps provide a similarity map represented by intensity of red dots in (b). Our algorithm approximates the similarity maps by a bipartite graph (c) with a small set of new "cluster-nodes" (green). The walker jumps only twice: from a SP-node to a cluster-node, and back to a SP-node. The goal is to construct such a graph that's destination probability (d) approximates the similarity maps (b).*

We developed a modified gradient descent solver, designed to factor powers of sparse matrices into statistic matrices (F, G). Choosing the number of representatives and the initial values is difficult because it depends on the context. One of our contributions is a heuristic for finding the number of labels and the initial (F, G).

We have also developed algorithms which make the factorization both faster and more reliable. When the number of superpixels is extremely large the NMF problem becomes overdetermined. To keep the problem tractable, we have developed a new projection algorithm which can relate our NMF problem to a simpler smaller one. Finally, due to the random nature of our starting values, it is possible that our algorithm might find a unwanted local minimum. We therefore introduce a novel boosting algorithm which combines multiple solutions to try to find a better final one.

For brevity we moved all the details to a supplemental document. A reader interested in understanding the overall flow of this algorithm can consider it a black box to solve equation (8). The viability of our approach relies on these algorithms. In particular the design for powers of large sparse matrices makes it possible to work on large images (many megapixels).

The use of a NMF algorithm has several advantages over other clustering methods. First, it utilizes and approximates the diffusion framework that past works have shown to work extraneously well

[Lu et al. 2009b; Lu et al. 2009a; Lockerman et al. 2013]. Second, a NMF algorithm requires a similarity map as input, allowing it to work with datasets where the creation of feature vectors is not possible. Third, we obtain information as to how well each superpixel fits into each cluster. While the last two properties are only indirectly used in this paper, they will allow for future works to use our algorithm for other graphical purposes. Thus, we hope that the introduction of NMF based methods will be useful beyond our particular application.

4.3 Multi-scale clustering

We propose a clustering algorithm which provides a multi-scale label-map as illustrated in Figure 5. Our algorithm takes as input the hierarchy of superpixels (section 3) and returns a small subset of scales $\rho_1 < \rho_2 < \ldots < \rho_L$ and the label-maps $\{C^1, C^2, \ldots, C^L\}$ at these scales. It works as follows:

- Bottom-up stage: compute the scales ρ_l and independent label-maps $\tilde{C}^l$. It first creates a label-map $\tilde{C}^1$ at scale $\rho_1 = \sigma_1$ as discussed in Section 4.2. It then does a binary search to find the minimum scale γ such that each $S \in \mathbb{S}^\gamma$ overlaps multiple clusters in $\tilde{C}^1$. This γ becomes ρ_2 and a label-map $\tilde{C}^2$ is computed. This process iterates to fill out $\rho_3 < \ldots < \rho_L$ until we can no longer find a suitable next level.

- Top-down stage: compute a hierarchy of label-maps C^l. The key idea is to intersect recursively any cluster $c \in \tilde{C}^l$ with C^{l+1} as described by Algorithm 2.

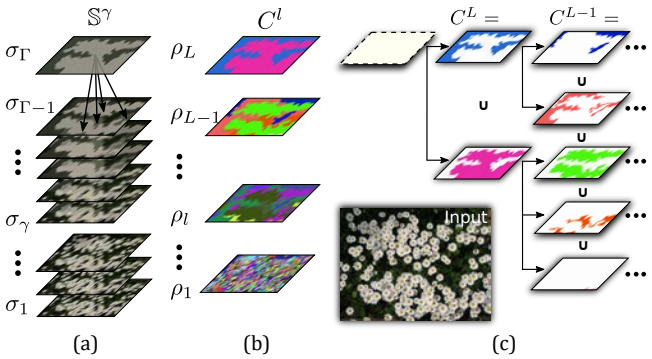


Figure 5: *Multi-scale labeling. Based on the dense hierarchy of superpixels (a), a few representative levels are selected to build a multi-scale label-map (b). It can be seen as a tree of clusters (c).*

Input: Independent label-maps $\{\tilde{C}^1, \tilde{C}^2, \ldots, \tilde{C}^L\}$.
Output: Hierarchy of label-maps $\{C^1, C^2, \ldots, C^L\}$.
$C^L \leftarrow \tilde{C}^L$
for *level* $l = L-1 \ldots 1$ **do**
  $C^l \leftarrow \emptyset$
  for *cluster* $c \in C^{l+1}$ **do**
    for *cluster* $d \in \tilde{C}^l$ **do**
      if $c \cap d \neq \emptyset$ **then** add $c \cap d$ to C^l
    end
  end
end

Algorithm 2: Multi-scale labeling (top-down stage).

4.4 Selection of step count

There have been a number of methods proposed for selecting the number s of diffusion steps [Zelnik-Manor and Perona 2004; Wang and Tu 2012; Lu et al. 2009a]. We have opted to use a simple method based on the eigenvectors despite its limitations [Zelnik-Manor and Perona 2004]. It is based on the well known property that the eigenvector of a matrix raised will change exponentially according to that power.

Specifically, we find the largest eigenvalue, λ, that is less than .9999. We then determine how many steps it would take for that eigenvalue to decay to 95% of its original power: $s = \left\lfloor \frac{\log(.95\lambda)}{\log(\lambda)} \right\rfloor$.

4.5 Performance

We implemented our method in Python and OpenCL (major libraries used are: Numpy, Scipy, scikit-image, scikit-learn, scikits.ann, bottleneck, pyopencl, and matplotlib). We tested our software on an Intel Core i7 5820K with 16 GB of RAM, and equiped with an AMD Radeon R9 280X with 3 GB of memory. Table 2 provides timings for producing the multi-scale label-maps of Figure 6. The timings mostly depend on the resolution of the input. Note also in Figure 6 how our method captures the different patterns at different scales. Additional results and timings for arbitrary natural images are given in the supplemental material.

Image name	Times in seconds		
	H-SLIC	Multi-scale labeling	Total
Flowered_wall (1233×924)	19.4	590.5	609.9
Ground (2560×1920)	66.2	2627.2	2693.4
River (1616×616)	19.5	1075.3	1106.8
Plaster_damaged (373×560)	7.8	702.1	709.9
Rust (2272×1704)	51.2	1486.6	1537.8
City (2016×703)	22.2	876.6	898.8

Table 2: *Time performance (in seconds) of our hierarchical SLIC method (H-SLIC) and of our multi-scale labeling method.*

4.6 Evaluation

To measure the quality of a label-map C with respect to the ground truth $\mathbb{G}$ we make use of two standard segmentation metrics [Arbeláez et al. 2011]: the Rand index and the covering (asymmetric metric evaluated in both directions). These measures give values between 0 (worse) and 1 (best) that we plot against the number of labels. More details, as well as extensive qualitative and quantitative results are provided in supplemental material.

To evaluate our NMF based mono-scale labeling algorithm we compare it with K-means and spectral clustering. As illustrated on the selected results of Figure 3(d), the different clustering methods perform similarly in many cases. However, unlike our NMF algorithm the other methods require the number of clusters to be supplied as input. As such, we used the initial part of our NMF algorithm to calculate the number of clusters before running them. As explained in section 4.2, the NMF algorithm also provides additional information encoded in the F and G matrices.

4.7 Choice of the feature vectors

The labeling relies on the feature vector used to define distances between nodes (section 4.2). As an alternative to the moments (5 moments per color channel) we tested Gabor filters (12 filter responses

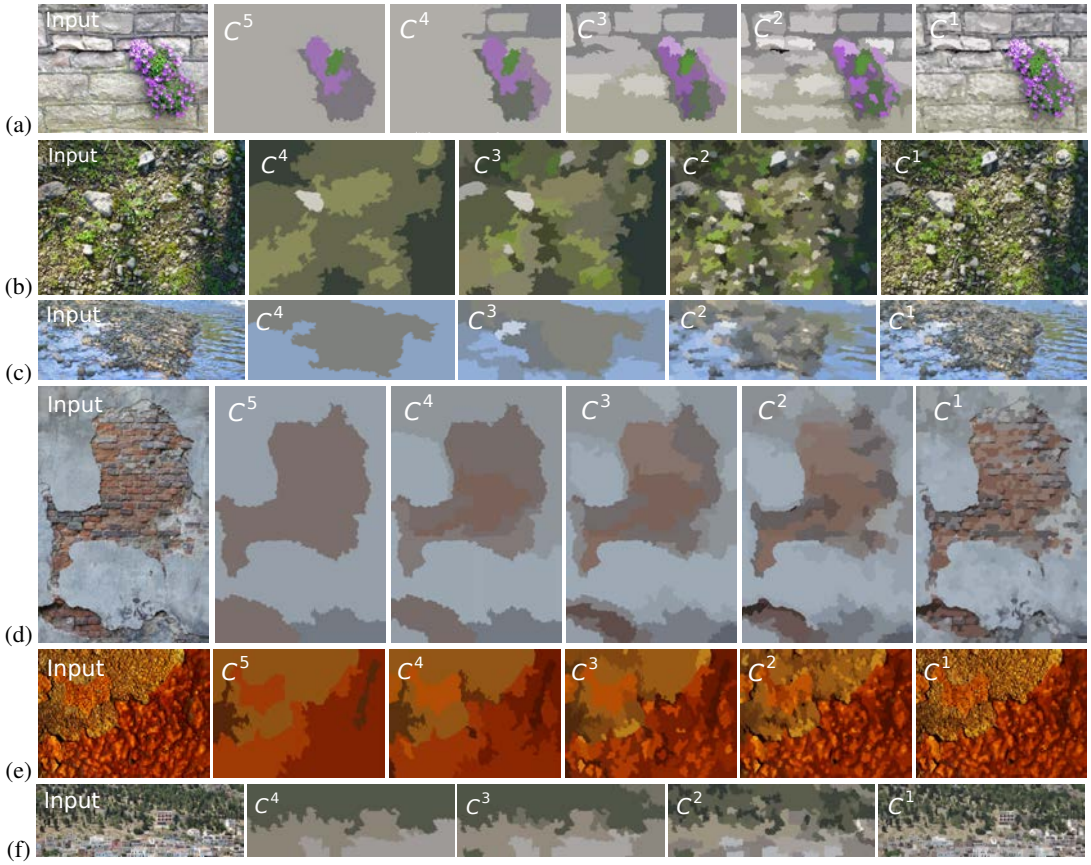


Figure 6: *Multi-scale label-maps. (a): Flowered_wall (1233×924); (b): Ground (2560×1920); (c): River (1616×616); (d): Plaster_damaged (373×560); (e): Rust (2272×1704); (f): City (2016×703).*

per channel) and local region statistic (mean values in 21 cells of a local quadtree per channel). All details and extensive results are in the supplemental material.

There are quantitative differences in the resulting label-maps. However none of the 3 feature vectors performs better universally (see Figure 3(c)). The moments vector has the advantage of being computationally the simplest and fastest. Alternatives are available and usable depending on the needs of an application.

5 Pixel description

Assume we are given a texture exemplar which undergoes the multi-scale labeling of section 4. Every pixel $\mathbf{p}$ inherits from the superpixels it belongs to L labels $\{C^1(\mathbf{p}), C^2(\mathbf{p}), \ldots, C^L(\mathbf{p})\}$, i.e. one at each scale ρ_l. In this section we show how these labels may be exploited as descriptors for $\mathbf{p}$ so as to improve texture synthesis algorithms.

5.1 Layered textures

In Figure 7 we compare our multi-scale labeling with the preprocessing of Layered-Shape Synthesis [Rosenberger et al. 2009]. Unlike our method, the number of labels must be given. Our large scales are quite similar, though no fixed number of labels has been preset. The main difference is that our result is much less noisy. This is due to the hierarchy of patterns, which may contain subpatterns with different colors.

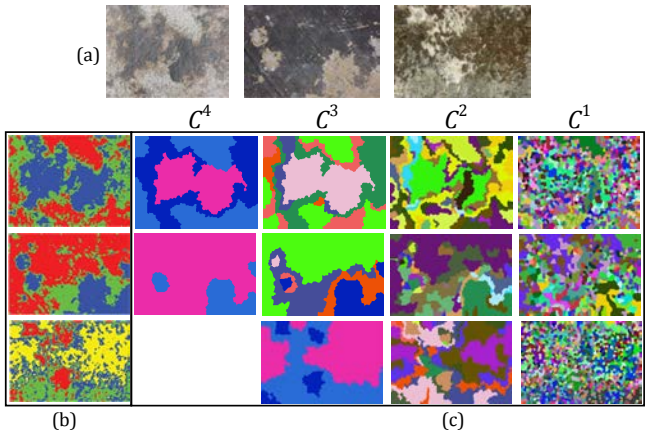


Figure 7: *Comparison of our label-maps to the labeling results of Layered-Shape Synthesis [Rosenberger et al. 2009]. (a): Input images (from left to right); (b): corresponding Layered-Shape Synthesis label-maps (top to bottom); (c): Our label-maps (4 levels for the two leftmost input images, and 3 levels for the rightmost one).*

5.2 Extraction of stationary textures in natural images

Most natural images contain completely different textures in several regions, possibly with outliers. This observation motivated work on the extraction of one dominant texture. Our label-maps allows us to

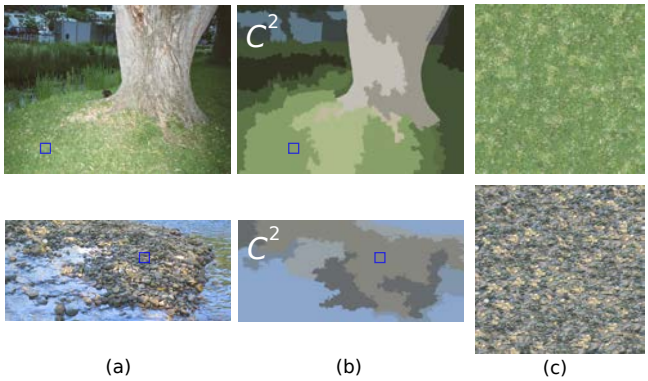


Figure 8: *Stationary texture extraction. Our labeling method is used to pre-process input natural images (a). In the resulting pattern label-map (b) at a given level ($l = 2$), a pattern (blue square) is chosen. Stationary synthesis (c) uses only similar patterns: graphcut synthesis using only patches with the same label.*

go further, as shown in Figure 8. Once the input has been labeled, the user selects a label to which the synthesis can be restricted. The goal is to give access to many more texture exemplars than carefully prepared libraries, including photographic collections. The multi-scale label-maps can then be used to synthesize textures with the rich variation at smaller scales of natural images.

6 Region description

Many texture synthesis algorithms process connected sets of pixels (such as patches, tiles, or neighborhoods). Let us call a *region* R any such set consisting of pixels $\mathbf{p}$. A standard descriptor for regions is the histogram of pixel values. In our case each pixel is described by one label $C^l(\mathbf{p})$ per level $0 \leq l \leq L$ so we can define L histograms h^l_R of labels:

$$\forall c \in C^l, \quad h^l_R(c) = \#\{\mathbf{p} \in R | C^l(\mathbf{p}) = c\} \tag{9}$$

Each histogram describes the content of region R when analyzed at scale ρ_l as shown in Figure 9. In the following we show some applications to the control of texture synthesis.

6.1 Content selection for patch-based synthesis

Many patch-based synthesis techniques, such as quilting [Efros and Freeman 2001] or graphcut [Kwatra et al. 2003], work by copy-paste of patches, i.e. large pieces of an input example texture. To control non-stationarity we need to be able to select the content of the patches. To do so we propose to describe the content by a simple key based on the histograms of labels. Although complete histograms describe the content, they are not easy to manipulate because the number of labels increases as l gets lower. The issue is to define a set of keys which are meaningful, distinguishable, and such that regions with similar contents have the same key. For instance in Figure 10 a key may signify "stones only" or "mainly flowers with little stone".

Before selecting the content one must decide the size of patches. This is a delicate problem as they must be large enough to caputure and faithfully reproduce patterns but as small as possible so as to avoid repetition artifacts. Our multi-scale labeling helps as follows. First the user chooses the relevant level l: for instance figure 10, $l = 5$ is appropriate to control wall versus flowers while $l = 3$ is appropriate to control flowers' hue variations. Then the

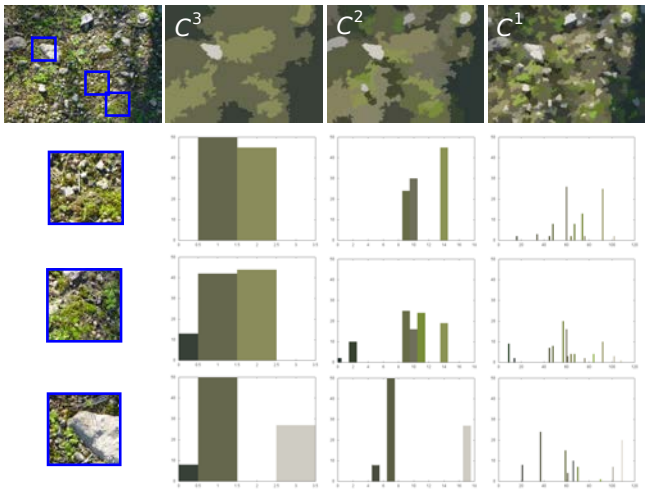


Figure 9: *Histograms of labels are statistical content descriptors for regions R (in blue). Regions with similar contents at level l have similar histograms h^l_R (see column 2, $l = 3$). As l gets lower, for the same regions, content differences get captured in corresponding finer histograms.*

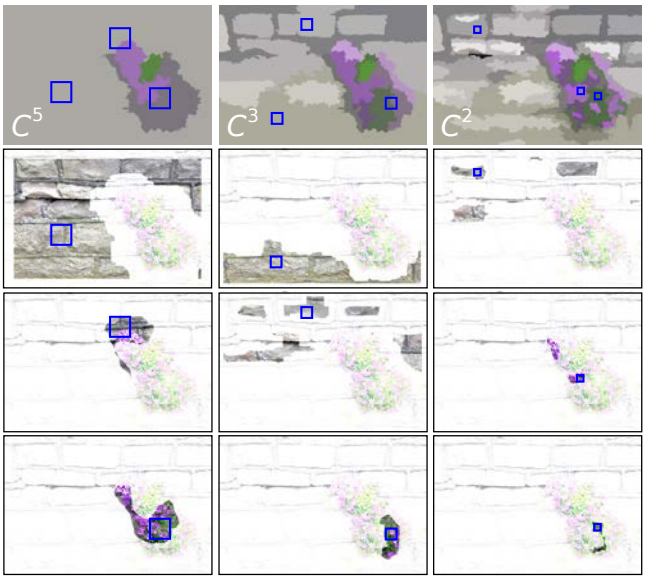


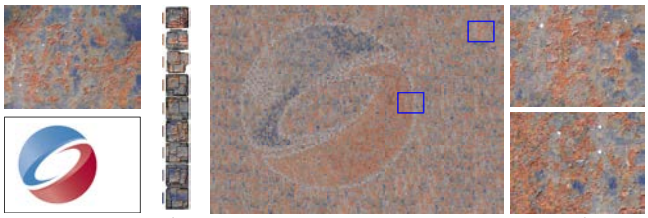
Figure 10: *Content similarity. For each level l of the label-map (3 levels, top row) we show three patches R outlined in blue. ρ_l decreases from left to right, and the size of the patches decreases accordingly. Rows two to four highlight subsets of pixels that are centers of patches of the same size, so that the histogram is "similar" to h^l_R (in this case, we consider histograms as similar if their two most frequent labels are matching).*

corresponding scale ρ_l gives the size of patterns to be captured. Finally the patch size can be chosen proportional to the scales ρ_l. In practice we round ρ_l to the closest power of two.

Figure 10 illustrates our content descriptor for square patches. The selected size (i.e. proportional to ρ_l) makes that two labels at most cover the great majority of the pixels in patches. This is why a key $k^l(R) = (k_1, k_2)$ where k_1 and k_2 are the most represented (frequent) labels in h^l_R yields an efficient patch content descriptor. Using such a key, all possible patches of an input can be put in a

(a) *input* **(b)** *hash table*

Figure 11: *Description of regions' content. All tiles of an input are given keys (k_1, k_2) (row,column) stored in a hash table. The keys are obtained from our label-maps. The hash table allows tiles to be instantly selected according to their content.*

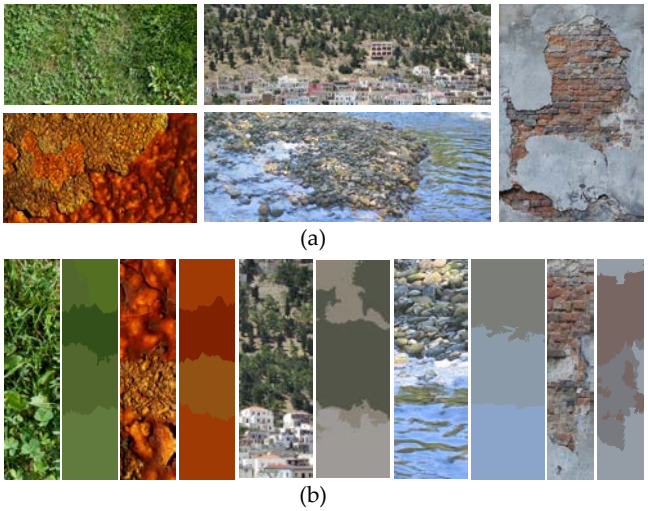


(a) (b) (c) (d)

Figure 12: *Non-stationary texture synthesis is guided by region description. An example (2560×1920) and a guidance map are given as input (a). Our label-map is used to describe tiles by keys (b) which are linked to the guidance map. The result of quilting (16638×11776) is shown (c) with close-ups (d).*

hash table (Figure 11). The patches need not be stored but only their offset so the hash table has a size proportional to the size of the input image. It can then be used for controlling non-stationarity during texture synthesis, by instant patch access according to the desired content using the corresponding key. In Figure 12 keys are provided in a guidance map and only patches with this key are candidates. Here, we used texture synthesis based on quilting (regular arrangement of square patches) because it scales very well and allows us to produce huge texture maps that can be rendered at fast framerates at a low memory cost. Only the input, as well as patch indices and some stitching information need to be stored on the GPU.

The use of a hash table significantly speeds up patch selection for texture synthesis. At a given level, we instantly get a list of potential patch candidates, by getting rid of an explicit search [Barnes et al. 2009]. We can then select among these, the "best candidate" according to some criteria depending on the applied synthesis algorithm. The fact that we have a multi-scale content description allows us to further discriminate candidates according to various sub-level contents, which is useful for the evaluation of the similarity of neighboring or overlapping regions. Interactive patch-based texture editing becomes possible even for high resolution inputs. The quilting result of Figure 12 took less than five minutes, while conventional quilting takes more than an hour and optimization-based synthesis techniques are not practicable. Tiling methods such as Wang tiles [Cohen et al. 2003] or irregularly shaped tiles [Vanhoey et al. 2013] are also not suitable in this case, because they assume uniformity of contents and cannot well handle local content variations, unless a very large number of adequate well-connecting tiles are specifically pre-computed.



(a)

(b)

Figure 13: *Texture palettes. For each input (a), patterns at the coarsest scale level are organized in palettes. (b) shows synthesized palettes (left) with corresponding label-map (right). These palettes can be used for selecting patterns in texture painting applications.*

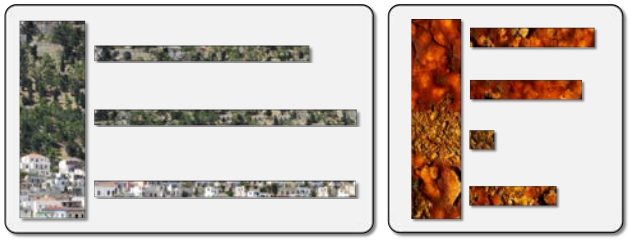


Figure 14: *Multi-scale palettes. Each pattern in level l (vertical palette) can be refined into a level $l-1$ sub-palette (horizontal).*

6.2 Multi-scale patterns palettes for texture painting

Tasks such as browsing texture examples or interactive texture painting require quickly seeing the patterns in the texture. The user experience might be improved by a clear exhibition of these patterns. As shown in Figure 13 we can build palettes of patterns by synthesizing a band of texture guided by our label-maps. The patterns may be organized as one pleases so as to provide a user-friendly palette in which the user can pick up a "pattern brush" for texture painting. The accompanying video shows interactive texture editing. One can also take advantage of the multiple scales by computing sub-palettes, as shown in Figure 14. This allows interactive editing not only at one scale, but at multiple different scales. In Figure 15 the textures are edited at two different scales. It greatly extends user interaction and control compared to conventional composite texturing approaches which are generally limited to one label-map scale. The use of multiple scales for texture editing has an additional advantage as it reduces repetition artefacts because non-stationarity is possible at all scales.

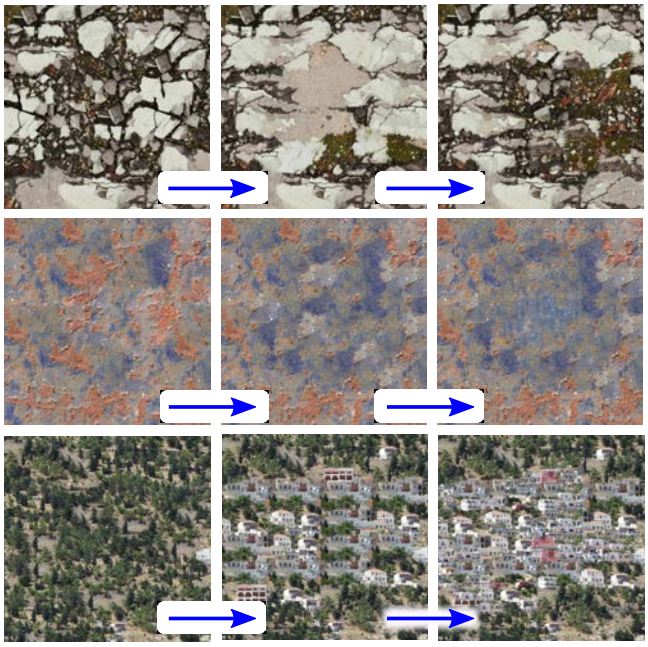


Figure 15: *Multi-scale editing. Textures are first edited at large scale and then at a fine scale so as to modify local details.*

7 Conclusions

We proposed an algorithm for the analysis of large input textures. It builds a multi-scale label-map of texture patterns. In a first stage we compute a hierarchy of superpixels which is able to detect patterns with sub-patterns precisely. In a second stage an unsupervised algorithm provides a label-map that describes the texture content at multiple scales. We propose a novel formulation of the labeling problem as a random walk on a bipartite graph of superpixels and clusters, which lead to a nonnegative matrix factorization. We provide a series of algorithms for factorizing powers of large sparse stochastic matrices which allow to process large inputs. We show the benefits of our analysis in several texture synthesis applications including content selection, non-stationary synthesis, interactive painting at multiple scales, and choice of tile size.

In our future work, we are looking for more applications to further improve texture synthesis. Pixel description may improve the computation of optimal cuts and blending in patch-based synthesis, as well as the comparison and search of similar local neighborhoods in optimization-based synthesis. We used content selection for interactive texture painting, but it could also lead to synthesis from multiple inputs as well as automatic synthesis of non-stationary textures. Our hash coding gives us instant access to patch candidates for texture synthesis. We also want to experiment with the use of our label-maps to further improve the search of patch candidates in structured image editing using random algorithms like PatchMatch.

Some pixel-based techniques [Rosenberger et al. 2009] or optimization-based techniques [Kaspar et al. 2015] have used distance maps to sharp features so as to guide individual patterns. The first stage of our algorithm (pattern detection) could be combined with distance transforms in this context.

Beyond texture synthesis, we believe that automatic multi-scale labeling in images can be useful for many other applications, such as texture and color transfer between images, as well as the synthesis of image appearance from exemplars and retexturing. Indeed, most of these methods need a prior labeling of regions with similar patterns, which is mostly done manually. Our approach provides not only an automatic and robust solution to do so, but it also provides an efficient tool to handle multiple scales instead of a single one.

Acknowledgements

We would like to thank Sylvain Thery for data processing, as well as Steven Zucker, Dana Angluin, Stanley Eisenstat, Noam Tanner, and anonymous reviewers for their insightful comments. We also thank Frédéric Larue, Samuel Brenner and Joseph Lanzone who helped proofread this paper. Images courtesy: Eli Lockerman (Tower), Mayang's Textures (Bubbling_rusty_metal, Daisylike_flowers, Rust), www.public-domain-image.com (Ground), www.textures.com (Plaster_damaged), Rosenberger et al. [2009] (images of figure 7 [1]), Flowered_wall: "A True Wall Flower" by Alan Levine, resized, licensed under CC BY 2.0, other images taken by authors. This research has been partially supported by NSF grants IIS-1064412 and IIS-1218515.

References

ACHANTA, R., SHAJI, A., SMITH, K., LUCCHI, A., FUA, P., AND SUSSTRUNK, S. 2012. SLIC superpixels compared to state-of-the-art superpixel methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 34*, 11 (November), 2274–2282.

ARBELÁEZ, P., MAIRE, M., FOWLKES, C., AND MALIK, J. 2011. Contour detection and hierarchical image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 33*, 5 (May), 898–916.

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, ACM, I3D '01, 217–226.

BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. 28*, 3 (July), 24:1–24:11.

BERRY, M. W., BROWNE, M., LANGVILLE, A. N., PAUCA, V. P., AND PLEMMONS, R. J. 2007. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics & Data Analysis 52*, 1, 155 – 173.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Trans. Graph. 22*, 3 (July), 287–294.

DAI, D., RIEMENSCHNEIDER, H., AND VAN GOOL, L. 2014. The synthesizability of texture examples. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3027–3034.

DIAMANTI, O., BARNES, C., PARIS, S., SHECHTMAN, E., AND SORKINE-HORNUNG, O. 2015. Synthesis of complex image appearance from limited exemplars. *ACM Trans. Graph. 34*, 2 (Mar.), 22:1–22:14.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, SIGGRAPH '01, 341–346.

[1] "Layered shape synthesis: automatic generation of control maps for non-stationary textures", ACM Transactions on Graphics, Vol. 28:5, http://doi.acm.org/10.1145/1661412.1618453

GILLIS, N. 2011. *Nonnegative Matrix Factorization: Complexity, Algorithms and Applications*. PhD thesis, Université Catholique de Louvain, École Polytechnique de Louvain.

GILLIS, N. 2014. The why and how of nonnegative matrix factorization. In *Regularization, Optimization, Kernels, and Support Vector Machines*. Chapman & Hall/CRC, 257–291.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, SIGGRAPH '01, 327–340.

KASPAR, A., NEUBERT, B., LISCHINSKI, D., PAULY, M., AND KOPF, J. 2015. Self tuning texture optimization. *Computer Graphics Forum 34*, 2, 349–359.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. In *ACM SIGGRAPH 2003 Papers*, ACM, SIGGRAPH '03, 277–286.

LEFEBVRE, S., AND HOPPE, H. 2006. Appearance-space texture synthesis. In *ACM SIGGRAPH 2006 Papers*, ACM, SIGGRAPH '06, 541–548.

LOCKERMAN, Y. D., XUE, S., DORSEY, J., AND RUSHMEIER, H. 2013. Creating texture exemplars from unconstrained images. In *Proceedings of the 2013 International Conference on Computer-Aided Design and Computer Graphics*, IEEE Computer Society, CADGRAPHICS '13, 397–398.

LU, J., DORSEY, J., AND RUSHMEIER, H. 2009. Dominant texture and diffusion distance manifolds. *Computer Graphics Forum 28*, 2, 667–676.

LU, J., GARR-SCHULTZ, A., DORSEY, J., AND RUSHMEIER, H. 2009. A psychophysical study of dominant texture detection. In *Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization*, ACM, APGV '09, 133–133.

MERTENS, T., KAUTZ, J., CHEN, J., BEKAERT, P., AND DURAND, F. 2006. Texture transfer using geometry correlation. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, Eurographics Association, EGSR '06, 273–284.

MOUNT, D. M. 2010. ANN programming manual. Tech. rep., University of Maryland, College Park, Maryland.

NARAIN, R., KWATRA, V., LEE, H.-P., KIM, T., CARLSON, M., AND LIN, M. C. 2007. Feature-guided dynamic texture synthesis on continuous flows. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, Eurographics Association, EGSR'07, 361–370.

ROSENBERGER, A., COHEN-OR, D., AND LISCHINSKI, D. 2009. Layered shape synthesis: Automatic generation of control maps for non-stationary textures. *ACM Trans. Graph. 28*, 5 (dec), 107:1–107:9.

VANHOEY, K., SAUVAGE, B., LARUE, F., AND DISCHLER, J.-M. 2013. On-the-fly multi-scale infinite texturing from example. *ACM Trans. Graph. 32*, 6, 208.

WANG, W., AND HUA, M. 2013. Extracting dominant textures in real time with multi-scale hue-saturation-intensity histograms. *Image Processing, IEEE Transactions on 22*, 11, 4237–4248.

WANG, B., AND TU, Z. 2012. Affinity learning via self-diffusion for image segmentation and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2312–2319.

WU, F., DONG, W., KONG, Y., MEI, X., YAN, D.-M., ZHANG, X., AND PAUL, J.-C. 2016. Feature-aware natural texture synthesis. *The Visual Computer 32*, 1, 43–55.

ZALESNY, A., FERRARI, V., CAENEN, G., AND VAN GOOL, L. 2005. Composite texture synthesis. *International Journal of Computer Vision 62*, 1-2, 161–176.

ZELNIK-MANOR, L., AND PERONA, P. 2004. Self-tuning spectral clustering. In *Advances in neural information processing systems*, 1601–1608.

ZHANG, J., ZHOU, K., VELHO, L., GUO, B., AND SHUM, H.-Y. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph. 22*, 3 (July), 295–302.

A Partition error minimization

For clarity we remove subscripts m and γ an I. Equation (3) becomes

$$\epsilon\left(\mathbb{S}\right) = \sum_{S\in\mathbb{S}}\sum_{\mathbf{p}\in S}\delta^2\left(\mathbf{x},\bar{\mathbf{x}}_S\right) \tag{10}$$

and equation (2) becomes

$$\Delta\epsilon = |R|\,\delta^2\left(\bar{\mathbf{x}}_R,\bar{\mathbf{x}}_{R\cup S}\right) + |S|\,\delta^2\left(\bar{\mathbf{x}}_S,\bar{\mathbf{x}}_{R\cup S}\right) \tag{11}$$

where $\bar{\mathbf{x}}_{R\cup S} = \dfrac{|R|\,\bar{\mathbf{x}}_R + |S|\,\bar{\mathbf{x}}_S}{|R|+|S|}$

Lemma 1. *When merging the pair of superpixels (R,S) into $R\cup S$ the global error* (10) *increases by* (11)*.*

Proof.

$$\begin{aligned}\Delta\epsilon &= \sum_{\mathbf{p}\in R\cup S}\delta^2\left(\mathbf{x},\bar{\mathbf{x}}_{R\cup S}\right) - \sum_{\mathbf{p}\in R}\delta^2\left(\mathbf{x},\bar{\mathbf{x}}_R\right) - \sum_{\mathbf{p}\in S}\delta^2\left(\mathbf{x},\bar{\mathbf{x}}_S\right)\\ &= \left(\sum_{\mathbf{p}\in R}\delta^2\left(\mathbf{x},\bar{\mathbf{x}}_{R\cup S}\right) - \sum_{\mathbf{p}\in R}\delta^2\left(\mathbf{x},\bar{\mathbf{x}}_R\right)\right)\\ &+ \left(\sum_{\mathbf{p}\in S}\delta^2\left(\mathbf{x},\bar{\mathbf{x}}_{R\cup S}\right) - \sum_{\mathbf{p}\in S}\delta^2\left(\mathbf{x},\bar{\mathbf{x}}_S\right)\right)\end{aligned}$$

The second equality is obtained by splitting the first sum over $R\cup S$ into two sums over disjoint sets R and S. Then we apply lemma 2 twice with $\mathbf{y} = \bar{\mathbf{x}}_{R\cup S}$ to complete the proof. □

Lemma 2. *Let X be a set of points $\mathbf{x}$ in an inner product space. For any point $\mathbf{y}$*

$$\sum_{\mathbf{x}\in X}\|\mathbf{x}-\mathbf{y}\|^2 - \sum_{\mathbf{x}\in X}\|\mathbf{x}-\bar{\mathbf{x}}\|^2 = |X|\,\|\mathbf{y}-\bar{\mathbf{x}}\|^2$$

where and $\bar{\mathbf{x}}$ is the average over X, which cardinal is $|X|$.

Proof.

$$\begin{aligned}\sum_{\mathbf{x}\in X}\|\mathbf{x}-\mathbf{y}\|^2 &= \sum_{\mathbf{x}\in X}\|\mathbf{x}-\bar{\mathbf{x}}+\bar{\mathbf{x}}-\mathbf{y}\|^2\\ &= \sum_{\mathbf{x}\in X}\|\mathbf{x}-\bar{\mathbf{x}}\|^2 + \sum_{\mathbf{x}\in X}\|\bar{\mathbf{x}}-\mathbf{y}\|^2\\ &\quad - 2\sum_{\mathbf{x}\in X}\langle\mathbf{x}-\bar{\mathbf{x}};\bar{\mathbf{x}}-\mathbf{y}\rangle\\ &= \sum_{\mathbf{x}\in X}\|\mathbf{x}-\bar{\mathbf{x}}\|^2 + |X|\,\|\bar{\mathbf{x}}-\mathbf{y}\|^2 + \mathbf{0}\end{aligned}$$

□