Clustered Vector Textures

PEIHAN TU, University of Maryland, College Park, United States

LI-YI WEI, Adobe Research, United States

MATTHIAS ZWICKER, University of Maryland, College Park, United States

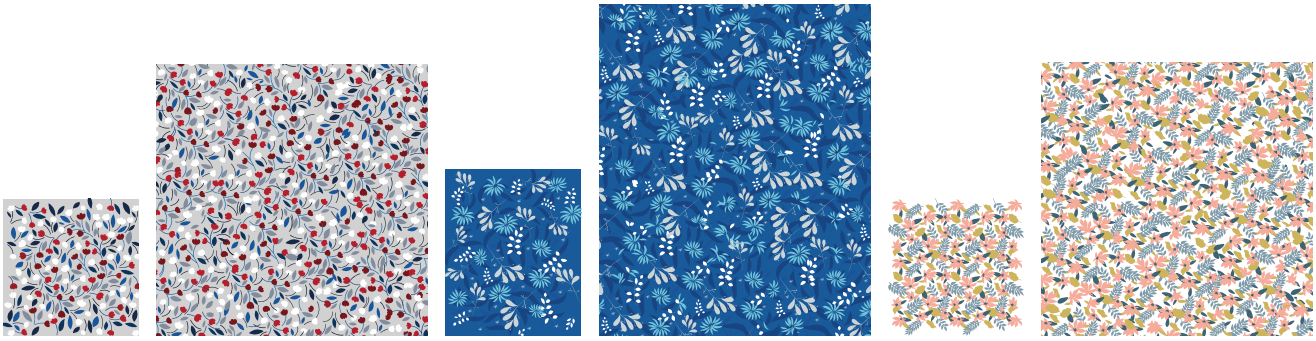


Fig. 1. *Example inputs and outputs of our method.* Given a small input exemplar, our algorithm can synthesize various types of structured vector patterns. All images are shown in vector format; please zoom in for details. Copyrights: (a) Snejana Sityaeva (c) Eva Kali (e) mspoint (stock.adobe.com)

Repetitive vector patterns are common in a variety of applications but can be challenging and tedious to create. Existing automatic synthesis methods target relatively simple, unstructured patterns such as discrete elements and continuous Bézier curves. This paper proposes an algorithm for generating vector patterns with diverse shapes and structured local interactions via a sample-based representation. Our main idea is adding explicit clustering as part of neighborhood similarity and iterative sample optimization for more robust sample synthesis and pattern reconstruction. The results indicate that our method can outperform existing methods on synthesizing a variety of structured vector textures. Our project page is available at https://phtu-cs.github.io/cvt-sig22/.

CCS Concepts: • **Computing methodologies** → **Texturing**.

Additional Key Words and Phrases: vector, pattern, texture, synthesis, cluster

ACM Reference Format:

Peihan Tu, Li-Yi Wei, and Matthias Zwicker. 2022. Clustered Vector Textures. *ACM Trans. Graph.* 41, 4, Article 159 (July 2022), 23 pages. https://doi.org/10.1145/3528223.3530062

1 INTRODUCTION

Vector patterns are common in design and engineering but can require high expertise and manual efforts to create. To address this issue, significant research has been devoted to automate the synthesis of vector patterns, either fully for batch generation or partially for interactive editing [Ganin et al. 2021; Guerrero et al. 2016; Guo et al. 2020; Hsu et al. 2020; Jacobs et al. 2018; Kazi et al. 2012; Kwan et al. 2016; Landes et al. 2013; Loi et al. 2017; Ma et al. 2011; Roveri et al. 2015; Santoni and Pellacini 2016; Tu et al. 2020; Wang et al. 2011; Zhou et al. 2014]. However, existing methods are more suitable for relatively simple patterns without diverse element shapes or complex, structured interactions (Figure 1). Another approach is to rasterize the vector pattern, apply image texture synthesis [Barnes and Zhang 2017; Wei et al. 2009], and vectorize the raster results. However, this process tends to lose the meaning of the original design components such as integral elements and overlapping shapes in different depth layers, often resulting in broken or merged elements.

This paper proposes an algorithm for generating vector patterns with diverse element shapes and structured local element interactions with potential overlaps (Figure 1). Instead of post-vectorizing

Authors' addresses: Peihan Tu, phtu@cs.umd.edu, University of Maryland, College Park, United States; Li-Yi Wei, review@liyiwei.org, Adobe Research, United States; Matthias Zwicker, zwicker@cs.umd.edu, University of Maryland, College Park, United States.

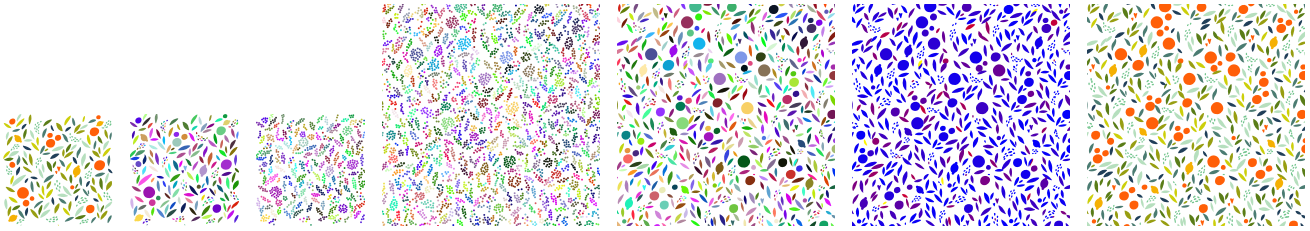


Fig. 2. *Pattern synthesis with clustered sample representation.* Given a vector input exemplar (a) with elements visualized in different colors in (b), our method first samples a clustered representation (c), synthesizes output sample clusters via *optimization* (d), reconstructs the output elements (e) from (d) with the reconstruction error (f), and generates the final output pattern (g) by filtering out elements in (e) with high reconstruction error (f). Corresponding clusters and elements are visualized in the same colors. In (f), red and blue indicate high and low reconstruction errors; yellow indicates filtered-out elements with errors surpassing a threshold. Copyright: (a) orangemilk (stock.adobe.com)

a synthesized raster output [Favreau et al. 2017; Price and Barrett 2006], our method directly produces vector patterns by optimizing a sample-based representation [Ma et al. 2011; Tu et al. 2020]. Our main idea is to explicitly optimize *clusters* of samples during the synthesis process. Similar to elements [Ma et al. 2011] and graphs [Tu et al. 2020], our clusters help to identify distinguishable pattern components that should remain integral (e.g., not merged, broken apart, or deformed) during systhesis. However, unlike these prior works that just copy element identifications [Ma et al. 2011] or compute curve identifications (reconstruct curves) from synthesized graphs as post-processing [Tu et al. 2020], our method optimizes clusters of samples as variables along with other pattern attributes such as sample positions, to facilitate robust optimization and reconstruction of the output vector patterns. The extra optimization of cluster configurations leads to better optimized patterns by expanding the feasible region - the set of all possible outcomes of pattern optimization.

Given a vector input exemplar (Figures 2a and 2b), our method samples an intermediate clustered representation where each vector element is represented as a cluster of samples (Figure 2c), synthesizes the output sample clusters (Figure 2d) via *optimization*, and finally reconstructs the output pattern (Figures 2e to 2g) from the output clusters (Figure 2d). Our algorithm *jointly* optimizes the spatial sample distributions and cluster configurations by adding a new clustering step into conventional search-and-assign based pattern synthesis [Ma et al. 2011; Tu et al. 2020]. We design an objective function to measure the quality of clusters by taking into account both input-output correspondences and cluster (element) shapes, and an optimization scheme to minimize the clustering objective.

We analyze our algorithms via robustness and ablation studies and comparisons with prior art, which show that our method is robust and can significantly outperform existing methods on synthesizing a variety of structured vector patterns.

2 RELATED WORK

Our work is mainly inspired by previous works on example-based and procedural pattern synthesis and authoring. We summarize the most related works below.

Example-based pattern synthesis approaches have been applied to generate geometric patterns with optional control, such as discrete elements [Barla et al. 2006; Hsu et al. 2018; Hurtut et al. 2009; Ijiri et al. 2008; Landes et al. 2013; Ma et al. 2013, 2011; Zhou et al. 2006], continuous structures [Roveri et al. 2015; Tu et al. 2020], and their combinations [Ganin et al. 2021; Roveri et al. 2015; Tu et al. 2020]. Existing methods can well handle relatively simple patterns but not complex ones with diverse element shapes and structured local interactions, as compared in Figures 3 and 13. Our cluster-based representation is inspired by Ma et al. [2011] where each vector element is represented by a set of samples with associated "element ids". The multi-sample representation is later augmented with graph-based representations [Hsu et al. 2020; Tu et al. 2020]. However, the "element ids" in [Hsu et al. 2020; Ma et al. 2011] are pre-defined and unoptimized; as each element is associated with a fixed set of samples, these prior methods may not be able to handle complex, structured patterns that require changing sample relationships during synthesis (Figures 3c and 3d). Tu et al. [2020] synthesize graphs and reconstruct curves from graphs by post-processing. However, graphs could not be applied for representing complex shapes other than curves, such as identifiable elements. Instead, our method *explicitly optimizes sample clusters* (e.g. their sizes and shapes), which can represent diverse element shapes, and cluster configurations (e.g. number of clusters and spatial distribution of clusters).

Procedural approaches can produce a variety of distributions and patterns including point distributions [Guehl et al. 2020; Wei 2010], packed elements [Hausner 2001; Hsu et al. 2020; Kwan et al. 2016; Saputra et al. 2020; Zou et al. 2016] or structured patterns [Guo et al. 2020; Loi et al. 2017; Nazzaro et al. 2020, 2021; Pedersen and Singh 2006; Santoni and Pellacini 2016; Šťava et al. 2010; Wong et al. 1998; Yeh and Měch 2009]. One key advantage of procedural methods is user control [Gieseke et al. 2021], even though the range of outputs can be limited by the underlying grammars or procedures.

Components of these automatic methods have been integrated with interactive interfaces to facilitate user control in the form of custom brushes or widgets, via exemplars [Fish et al. 2020; Kazi et al. 2012; Lu et al. 2013, 2014, 2012] or procedures [Jacobs et al. 2018]. However, existing methods work best for patterns that can be easily supplied

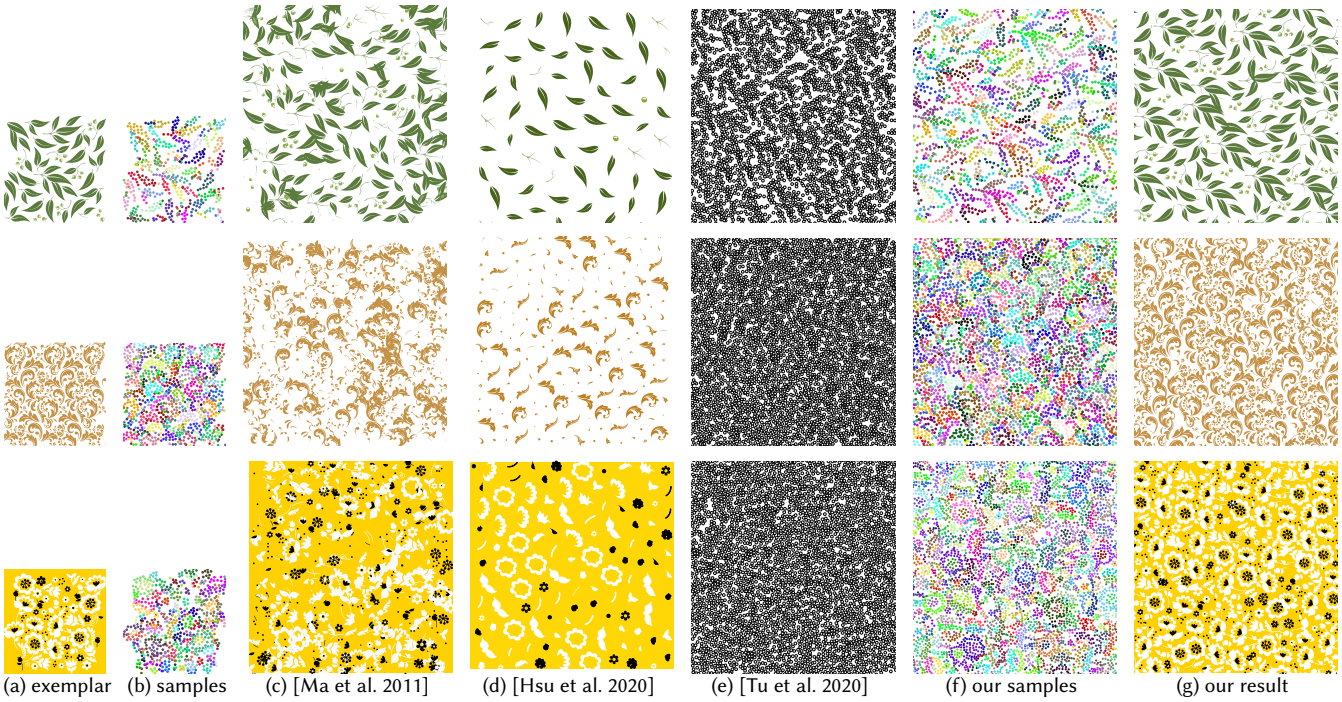


Fig. 3. *Comparison with previous methods.* Given an input exemplar (a), our method samples a clustered sample distribution (b), synthesizes clustered output samples (f), and reconstructs the final patterns (g). Clusters are visualized in colors. We compare our method against prior vector pattern synthesis methods [Ma et al. 2011] (c), [Hsu et al. 2020] (d), and [Tu et al. 2020] (e). As shown, while Ma et al. [2011] can generate broken structures and Hsu et al. [2020] can only place elements uniformly, our method can preserve diverse structures from the exemplars. Since [Tu et al. 2020] can only handle Bézier curve patterns via graph synthesis which could not be reconstructed as general shapes, we only show their sample synthesis results here. See Figures 13 and 24 for more comparisons. (a) top ©photo-nuke, middle ©ELENA, bottom ©galyna_p (stock.adobe.com).

by users, and more complex ones still require significant manual authoring. Our algorithm aims to narrow this gap with the ability to generate a variety of structured vector patterns from exemplars, as compared in Figure 3.

Our clustering idea also relates to the use of guidance/control maps or channels in prior image texture synthesis methods [Fišer et al. 2016; Hertzmann et al. 2001; Kaspar et al. 2015; Lockerman et al. 2016; Rosenberger et al. 2009]. However, instead of providing coarse user control, the clusters in our method aim to preserve detailed vector structures. In addition, clustering constitutes a dedicated step in our optimization algorithm.

Unlike raster texture synthesis [Barnes and Zhang 2017; Wei et al. 2009], the output of our method is in vector format and thus can facilitate further editing (Figure 12).

3 OVERVIEW

Our method extends prior optimization-based pattern synthesis methods [Ma et al. 2011; Tu et al. 2020] to represent vector patterns via clusters of samples (Section 4 and Figure 2c). Analogous to elements in [Ma et al. 2011] and connected graph edges in [Tu et al. 2020], our clusters correspond to identifiable pattern components that should remain integral during synthesis, such as avoiding merging, splitting, or deformation. However, our method treats clusters as variables in our optimization, along with other attributes such as sample positions as in [Ma et al. 2011; Tu et al. 2020]. We aims to optimize the similarity between two patterns by maximizing the similarity between local neighborhoods. In Sections 4 and 5, we explicitly incorporate the cluster information into pattern neighborhoods and their similarity (Equation (2)) by associating each sample with a shape context feature [Belongie et al. 2001] computed using its cluster. In Section 6, we define the pattern optimization objective which is the sum of pairwise nearest neighorhood similarity (Section 6.1) and the optimization strategy including initialization (Section 6.2) and iterative search (Section 6.3), assignment (Section 6.4) and clustering (Section 6.5) steps. The search step finds nearest neighborhoods and the assignment and clustering steps update samples. The clustering step, which includes a clustering objective (Section 6.5.1) and a greedy optimization-based clustering algorithm (Section 6.5.2), is the key new component of our method to better handle complex patterns.

4 REPRESENTATION

4.1 Vector pattern

Our input vector pattern is in SVG format [W3 2020]. Vector elements are deduced from the SVG file, such as individual paths and

<g> tags that are used to group SVG elements [W3 2020]. Optionally, users can manually group or split some input paths to reflect their design intentions. Intuitively, elements are repetitive spatial geometric shapes.

4.2 Clustered pattern representation

For pattern synthesis, we use an intermediate clustered representation of vector patterns. We represent each element using a set (cluster) $\{s\}$ of samples s (Figures 2b and 2c). Specifically, we sample interior regions uniformly using a Poisson disk distribution [Cook 1986] with disk diameter δ. A smaller δ means a denser sample distribution that can capture more shape details. δ can be chosen proportional to the size of patterns or element shapes.

Similarly to hierarchical image texture synthesis [Kwatra et al. 2005; Lefebvre and Hoppe 2006], we further synthesize patterns with hierarchical sampling [Kwatra et al. 2005; Tu et al. 2020]. Increasing hierarchy levels use denser sample distributions (with decreasing δ).

Each sample records the following information:

Spatial parameters $\mathbf{S}$, which include sample position $\mathbf{p} \in \mathbb{R}^2$, cluster id $\mathbf{i} \in \mathbb{Z}^+$, which indicates the cluster C it belongs to, confidence of sample existence $\xi \in [0\ 1]$, which enables the optimization of the number of samples [Tu et al. 2020], z-index $z \in \mathbb{Z}^+$, which defines the relative layer order of associated elements appearing in the SVG file [W3 2020], and a feature vector $\mathbf{f} \in \mathbb{R}^{N_\mathbf{f}}$ ($N_\mathbf{f}$ is the feature length), which encodes its relative position within its cluster $C \ni s$. We use shape contexts [Belongie et al. 2001], which can be applied to point-based geometry, to compute $\mathbf{f}$ using the cluster C of samples. The output clusters are optimized automatically via our method, as detailed in Section 6.5.

Appearance attributes $\mathbf{A}$, such as color, opacity, and gradients.

Together, a sample is represented as

$$\mathbf{U}(s) = (\mathbf{S}(s), \mathbf{A}(s)) \tag{1}$$

5 NEIGHBORHOOD AND SIMILARITY

Our synthesis method follows the sample-based neigborhood optimization framework [Ma et al. 2011; Tu et al. 2020]. Therefore, one of the core parts of our method is to define neighborhoods and their pairwise similarity criterion.

We define $\mathbf{n}(s)$, the neighborhood of s, as a set of samples around s's spatial vicinity within a certain radius r, including s itself. The distance between an input $\mathbf{n}(s_i)$ (centered at s_i) and an output neighborhood $\mathbf{n}(s_o)$ (centered at s_o) is defined by matching samples $s'_i = m_s(s'_o)$ within the input and output neigborhoods $s'_i \in \mathbf{n}(s_i)$ and $s'_o \in \mathbf{n}(s_o)$, where m_s is the one-to-one sample matching function (i.e. $s'_i = m_s(s'_o)$ means that $s'_i \in \mathbf{n}(s_i)$ is the matched input sample for output sample $s'_o \in \mathbf{n}(s_o)$) defined between two neighborhoods $m_s : \mathbf{n}(s_o) \rightarrow \mathbf{n}(s_i)$. The distance is written as

$$d_\mathbf{n}\left(\mathbf{n}(s_o), \mathbf{n}(s_i)\right) = \min_{\substack{m_s \\ \text{s. t. } s_i = m_s(s_o)}} \sum_{\substack{s'_o \in \mathbf{n}(s_o) \\ s'_i = m_s(s'_o) \in \mathbf{n}(s_i)}} d_s(s'_o, s'_i), \tag{2}$$

where $d_s(s'_o, s'_i)$ is the sample similarity between $s'_o \in \mathbf{n}(s_o)$ and $s'_i \in \mathbf{n}(s_i)$. The distance in Equation (2) is defined as the minimum over all possible sample matchings m_s. The optimal matching m_s is computed using the Hungarian algorithm [Kuhn 1955], subject to that centers s_i and s_o are always matched $s_i = m_s(s_o)$.

The sample similarity $d_s(s'_o, s'_i)$ between $s'_o \in \mathbf{n}(s_o)$ and $s'_i \in \mathbf{n}(s_i)$ is defined as the weighted sum of differences between sample properties $\mathbf{U}(s'_i)$ and $\mathbf{U}(s'_o)$ (as in Equation (1)):

$$d_s(s'_o, s'_i) = \|\hat{\mathbf{p}}(s'_o, s_o) - \hat{\mathbf{p}}(s'_i, s_i)\| + \frac{w_\mathbf{f}}{2} \sum_{q=1}^{N_\mathbf{f}} \frac{\left(\mathbf{f}^q(s'_o) - \mathbf{f}^q(s'_i)\right)^2}{\mathbf{f}^q(s'_o) + \mathbf{f}^q(s'_i)} \tag{3}$$

The first term measures the difference on sample positions relative to centers s_o, s_i; $\hat{\mathbf{p}}(s', s) = \mathbf{p}(s') - \mathbf{p}(s)$. The second term, where $\mathbf{f}^q$ is the q^{th} entry in $\mathbf{f}$, measures the difference of shape context features, defined using χ^2 test statistic as in [Belongie et al. 2001]. $w_\mathbf{f} = 10$ is the weight for the second term. Since the shape context feature is computed using clusters, *the sample similarity effectively compares both individual samples and their associated clusters.* We do not consider appearance attributes $\mathbf{A}$ in our experiments, as different shapes often have unique colors and thus considering the differences between spatial $\mathbf{S}$ information is sufficient to evaluate the neighborhood similarity. One can define appearance similarity as the weighted sum of differences between the appearance vector $\mathbf{A}(s'_o)$ and $\mathbf{A}(s'_i)$ when necessary.

6 PATTERN SYNTHESIS

6.1 Optimization objective

We formulate the output pattern synthesis as an optimization problem [Kwatra et al. 2005; Ma et al. 2011; Tu et al. 2020] between input $\mathcal{I} = \{s_i\}$ and output $\mathcal{O} = \{s_o\}$ sample sets:

$$\mathcal{O}^\star = \arg\min_{\mathcal{O}, \mu_s} \sum_{\substack{s_o \in \mathcal{O} \\ s_i = \mu_s(s_o)}} d_\mathbf{n}\left(\mathbf{n}(s_o), \mathbf{n}(s_i)\right) \tag{4}$$

The sum loops over all the local output neighborhoods over $\mathcal{O}$. $s_i = \mu_s(s_o)$ indicates that $\mathbf{n}(s_i)$ is the most similar input neighborhood to $\mathbf{n}(s_o)$. In other words, μ_s is the sample matching function which defines the nearest-neighbor field [Barnes et al. 2009] between input $\mathcal{I}$ and output $\mathcal{O}$. Note μ_s is different from m_s in Equation (2), which is the sample matching function within two neighborhoods. Equation (4) is minimized by alternating the optimization over $\mathcal{O}$ and μ_s through iterative search-assignment-clustering steps (Sections 6.3 to 6.5). Essentially, the search step minimizes Equation (4) over μ_s by finding the most similar input neighborhood for each output neighborhood. The assignment and clustering steps minimize Equation (4) over $\mathcal{O}$ by modifying output sample parameters $\mathbf{U}(s_o)$ ($s_o \in \mathcal{O}$) using μ_s computed during the search step in the current iteration.

6.2 Initialization

Similar to [Ma et al. 2011; Tu et al. 2020], we randomly copy input patches into the output domain. A patch is a square block within a pattern domain, which can consist of several clusters, and cluster samples outside of a patch boundary are ignored. The patch size is chosen as twice the neighborhood radius $2r$. Our algorithm can also generate similar results with random sample initialization, but patch-based initialization can reach lower optimized objective values using the same number of iterations [Ma et al. 2011]. Please see Figure 16. All attributes $\mathbf{U}(s_o)$ of an output sample s_o are copied from the corresponding input sample s_i attributes $\mathbf{U}(s_i)$ except for 1) the position $\mathbf{p}(s_o)$ which is shifted from $\mathbf{p}(s_i)$ during patch copying, 2) the cluster id $\mathbf{i}(s_o)$ for which i) samples within the same copied patch will receive unique values different from those in other copied patches and ii) samples in a copied patch will share the same $\mathbf{i}(s_o)$ if their source input ids $\mathbf{i}(s_i)$ are the same, and 3) the sample feature $\mathbf{f}$ which is recomputed for broken clusters on input patch boundaries.

6.3 Search step

In the search step, for each output sample, we find the input sample with the most similar neighorhood evaluated by Equation (2), which is accelerated with the patch match algorithm [Barnes et al. 2009; Tu et al. 2020].

6.4 Assignment step

In the assignment step, we optimize the output sample parameters by overlapping matched input neighborhoods over the output samples [Ma et al. 2011; Tu et al. 2020]. For each output sample, there is a set of input samples that are matched with it. For sample position $\mathbf{p}$, least squares is used to minimize the sum of differences between distances and expected distances between pairs of output samples. We also adaptively optimize the number of samples as in [Tu et al. 2020]. Please refer to Appendix A.2 for more details. The assignment of cluster id $\mathbf{i}$ is a clustering problem, which is significantly different from existing assignment algorithms [Ma et al. 2011; Tu et al. 2020]. We will discuss the id assignment as the clustering step in Section 6.5. The sample feature $\mathbf{f}$ is assigned using updated cluster configuration after the clustering step.

Depth z assignment. Vector patterns may consist of overlapping elements (e.g. Figure 5d), where one with a larger z-index z can cover another with a small z-index. To preserve layer relationship in pattern synthesis, we optimize the layer depth z of output samples as follows.

Since the exact value of z is not important but the ordering among z of different samples, we first compute a probabilistic pairwise ordering function $f_s(s_o, s'_o) \in [0, 1]$. The value of $f_s(s_o, s'_o)$ indicates the probability of s_o below s'_o, which is computed from matched input and output neighborhoods. Intuitively, in a pair of a matched input $\mathbf{n}_i$ and output neighborhood $\mathbf{n}_o$, $z(m_s(s_o)) < z(m_s(s'_o))$ suggests $z(s_o) < z(s'_o)$. Figure 4 illustrates the computation of $f_s(s_o, s'_o)$ using a toy example.

Based on the above intuition, we optimize depth as follows:

$$f_s(s_o, s'_o) = \begin{cases} \frac{|N_z(s_o, s'_o)|}{|N_n(s_o, s'_o)|}, & \|\mathbf{p}(s_o) - \mathbf{p}(s'_o)\| < 2\delta \\ \text{undefined}, & \|\mathbf{p}(s_o) - \mathbf{p}(s'_o)\| \geq 2\delta \end{cases} \tag{5}$$

$$N_n(s_o, s'_o) = \{(\mathbf{n}_i, \mathbf{n}_o) | (\mathbf{n}_i, \mathbf{n}_o) \in \mathcal{N}, s_o, s'_o \in \mathbf{n}_o\} \tag{6}$$

$$N_z(s_o, s'_o) = \{(\mathbf{n}_i, \mathbf{n}_o) | (\mathbf{n}_i, \mathbf{n}_o) \in N_n(s_o, s'_o), m_s : \mathbf{n}_o \rightarrow \mathbf{n}_i, z\left(m_s(s_o)\right) < z\left(m_s(s'_o)\right)\} \tag{7}$$

where $|\cdot|$ is the size of a set $\cdot$. $\mathcal{N} = \{(\mathbf{n}(s_i), \mathbf{n}(s_o)) | s_o \in O, s_i = \mu_s(s_o)\}$ is the set of matched input and output neighborhoods computed in the search step (Section 6.3). $N_n(s_o, s'_o)$ is the set that includes matched neighborhood pairs where the output neighborhoods contain both s_o and s'_o. $N_z(s_o, s'_o)$ is a subset of $N_n(s_o, s'_o)$where the matched input samples $m_s(s_o)$ is below $m_s(s'_o)$ ($z\left(m_s(s_o)\right) < z\left(m_s(s'_o)\right)$). Since we are only interested in pairs of output samples s_o and s'_o that are sufficiently close for potential overlap, $f_s(s_o, s'_o)$ is only defined when $\|\mathbf{p}(s_o) - \mathbf{p}(s'_o)\| < 2\delta$.

Since we order samples within the same clusters together instead of individually, we compute a pairwise ordering function $f_C(C_i, C_j)$ between sample clusters C_i, C_j:

$$f_C(C_i, C_j) = \frac{1}{Z} \sum_{s_o \in C_i} \sum_{\substack{s'_o \in C_j \\ \|\mathbf{p}(s_o) - \mathbf{p}(s'_o)\| < 2\delta}} f_s(s_o, s'_o) \tag{8}$$

where Z is the normalization factor that counts the number of entries within the double summation. Given the pairwise ordering function (Equation (8)), the depth order among clusters (or reconstructed elements) is computed using the "Order-By-Preferences" algorithm proposed in [Schapire and Singer 1998]. Figure 5 shows the effects with and without the z-index assignment.

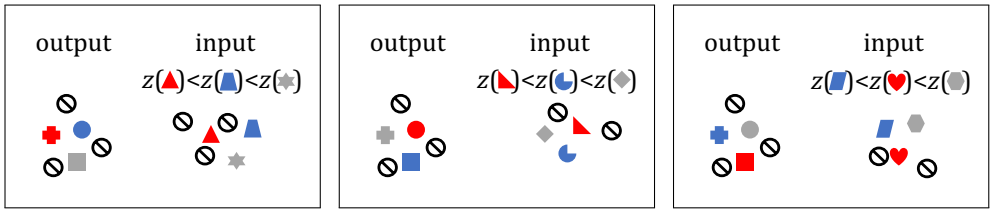
6.5 Clustering step

The clustering step updates sample cluster id $\mathbf{i}$. We address this clustering problem based on two observations:

Sample correlations In a pair of matched input and output neighborhoods $(\mathbf{n}_i, \mathbf{n}_o)$, if there are input samples within the same input cluster, their matched output samples should also be within the same output cluster, and vice versa.

Cluster shape similarity In pattern synthesis, the output element (and thus cluster) shape should be close to an input element (cluster) shape.

These two observations are incorporated into the optimization-based clustering step with objective E (Equation (9)), as two different energy terms: link energy E_l and shape energy E_s. We develop a greedy optimization algorithm to optimize Equation (9) via an iterative mechanism based on local operators including sample switching (from one cluster to another), and cluster merge and split (Section 6.5.2). While the sample switching gradually adjusts clusters, cluster merge and split update the clusters more drastically which not only optimizes the cluster shapes but also total number of clusters. Details are as follows.



(a) matched neighborhood pair 1 (b) matched neighborhood pair 2 (c) matched neighborhood pair 3

s_o \ s'_o	✚	●	■
✚	0	2/3	2/3
●	1/3	0	2/3
■	1/3	1/3	0

(d) $f_s(s_o, s'_o)$

Fig. 4. *A toy example of computing the probabilistic pairwise ordering function* $f_s(s_o, s'_o)$. (a), (b) and (c) show three pairs of input and output neighborhoods ($|N_n(s_o, s'_o)| = 3$) where the output neighorhoods are centered at different samples (in red) over the same set of output samples. The three input neighborhoods exhibit different layering relationships among samples. Different samples are visualized in different shapes. For example, output samples in all three pairs of neighborhoods are shown in the same shapes because they are the same set of samples. ⦸ indicates samples that are not used as examples in (d). Matched input and output samples are visualized in the same colors. (d) shows the computed $f_s(s_o, s'_o)$ among three output samples (in solid shape). . For example, $f_s(\bullet, \blacksquare)$ is $\frac{2}{3}$ because in the matched neighborhoods (a), (b) and (c) the matched input samples $m_s(\bullet)$ of $\bullet$ is below ($<$) $m_s(\blacksquare)$ in (a) and (b) but not in (c), so that $|N_z(s_o, s'_o)| = 2$. Note that we only consider the limited three pairs of neighborhoods (where the output contains s_o and s'_o) for illustration purposes.

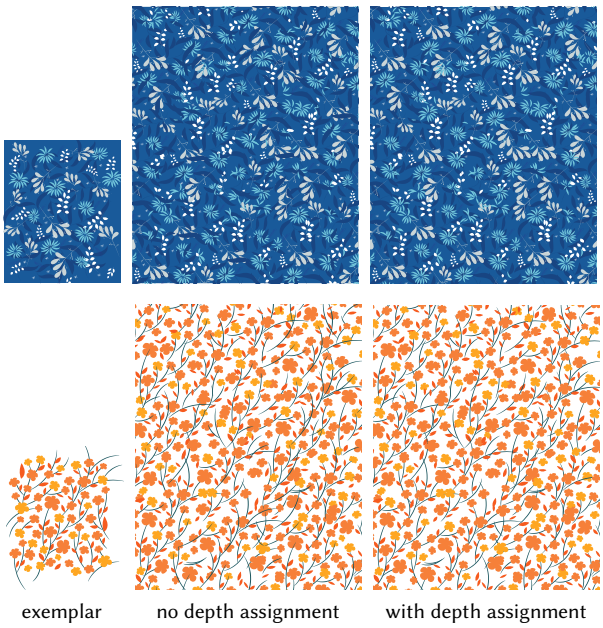


Fig. 5. *Z-index (depth) assignment.* In the input exemplars, the sky blue petals are above other elements (top) and the branches are below the leaves (bottom). Without depth assignment, this layer relationship is not preserved. With depth assignment, the relationship is mostly preserved. Copyright: exemplars from Eva Kali (top), Snejana Sityaeva (bottom) (stock.adobe.com).

6.5.1 Objective. Eventually, we hope to minimize the pattern synthesis objective (Equation (4)). Instead of minimizing Equation (4) with respect to cluster id $\mathbf{i}$ through brute-force search over all possible cluster configurations, we propose a separate objective defined over output cluster set $\{C_o^k\}$ (k is the index of clusters), for which we develop an efficient greedy optimization algorithm. The objective is as follows.

$$E(\{C_o^k\}) = w_l E_l(\{C_o^k\}) + w_s E_s(\{C_o^k\}) \tag{9}$$

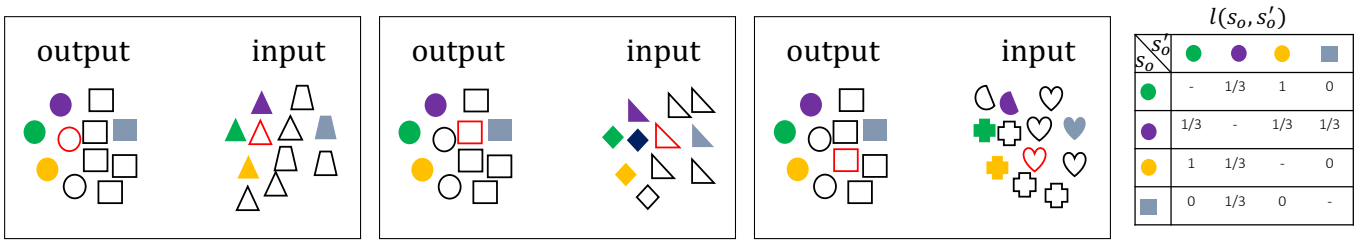
where E_l and E_s are the link and shape energies defined over a cluster configuration $\{C_o^k\}$, respectively. w_l, w_s are the weights for E_l and E_s. $w_l = 1$ and $w_s = 4$.

Link Energy. The link energy measures how compatible an output cluster configuration is with respect to input clusters based on *sample correlations*. Intuitively, in the set of matched pairs of input and output neighborhoods $\mathcal{N}$, if there are more pairs of output samples which are 1) in the same output cluster and matched input samples in the same input cluster or 2) in different output clusters and matched input samples in different input clusters, $E_l(\{C_o^k\})$ should be lower and indicates higher compatibility between input and output clusters. Before defining the link energy E_l, we compute the link confidence $l \in [0, 1]$ of two output samples belonging to the same output cluster from $\mathcal{N}$, which will be used to define E_l, as follows

$$l(s_o, s'_o) = \frac{|N_l(s_o, s'_o)|}{|N_n(s_o, s'_o)|} \tag{10}$$

$$N_l(s_o, s'_o) = \{(\mathbf{n}_i, \mathbf{n}_o) | (\mathbf{n}_i, \mathbf{n}_o) \in N_n(s_o, s'_o), m_s : \mathbf{n}_o \rightarrow \mathbf{n}_i, \mathbf{i}(m_s(s_o)) = \mathbf{i}(m_s(s'_o))\}. \tag{11}$$

Intuitively, we compute the confidence l of two output samples s_o, s'_o belonging to the same cluster based on the voting of all input neighborhoods overlapping them. Figure 6 illustrate the computation of link confidence l. $N_n(s_o, s'_o)$ is the set of matched input and output neighborhoods where the output neighborhood contain both of s_o and s'_o, as defined in Equation (6). $N_l(s_o, s'_o)$ is the subset of $N_n(s_o, s'_o)$ where the matched input samples $m_s(s_o)$ and $m_s(s'_o)$ have the same cluster id $\mathbf{i}(m_s(s_o)) = \mathbf{i}(m_s(s'_o))$. Note that $l(s_o, s'_o)$ is undefined when $|N_n(s_o, s'_o)| = 0$. The larger $l(s_o, s'_o)$ is, the more likely s_o, s'_o are within the same cluster.



(a) matched neighborhood pair 1 (b) matched neighborhood pair 2 (c) matched neighborhood pair 3 (d) $l(s_o, s'_o)$

Fig. 6. *A toy example of link confidence computation.* (a), (b) and (c) show three pairs of matched input and output neighborhoods ($|N_n(s_o, s'_o)| = 3$) where the output neighborhoods are centered at different samples (shown in red). Matched samples are shown in the same colors. Clusters are shown in different sample shapes. For example, output samples/clusters in all three pairs of neighborhoods are shown in the same shapes because they are the same set of samples/clusters. Empty shapes indicate samples that are not used as examples in (d). The computed link confidences among four output samples (in solid shape) from the three neighborhood pairs above are shown in (d). For example, samples ● and ● are matched to the same cluster in (a) but different clusters in (b) and (c), thus $|N_l(s_o, s'_o)| = 1$ and the link l(●, ●) receive a 1/3 score. Note that we only consider the limited three pairs of neighborhoods (where the output contains s_o and s'_o) for illustration purposes.

The link energy $E_l(\{C_o^k\})$ is defined as

$$E_l(\{C_o^k\}) = \sum_{s_o \in O} \sum_{\substack{s'_o \in O \\ \|\mathbf{p}(s_o) - \mathbf{p}(s'_o)\| < \lambda\delta}} E'_l(s_o, s'_o) \tag{12}$$

where

$$E'_l(s_o, s'_o) = \begin{cases} 1 - 2l(s_o, s'_o) & \text{if } \mathbf{i}(s_o) = \mathbf{i}(s'_o) \\ 2l(s_o, s'_o) - 1 & \text{if } \mathbf{i}(s_o) \neq \mathbf{i}(s'_o) \end{cases} \tag{13}$$

$E'_l(s_o, s'_o) \in [-1, 1]$ is defined over pairs of samples that are spatially close ($\|\mathbf{p}(s_o) - \mathbf{p}(s'_o)\| < \lambda\delta$, λ is a hyperparameter) by two cases based on whether s_o and s'_o are in the same ($\mathbf{i}(s_o) = \mathbf{i}(s'_o)$) or different ($\mathbf{i}(s_o) \neq \mathbf{i}(s'_o)$) clusters. r is the neighborhood radius appearing in Section 5. By minimizing Equation (12) with respect to $\{C_o^k\}$, which is the sum of Equation (13) over pairs of output samples, we encourage s_o, s'_o to be in the same cluster if the link confidence $l(s_o, s'_o)$ is high and be in different clusters if $l(s_o, s'_o)$ is low.

Shape Energy. $E_s(\{C_o^k\})$ is the sum of cluster shape energies E'_s, which are defined on individual clusters, over all clusters

$$E_s(\{C_o^k\}) = \sum_k E'_s(C_o^k) \tag{14}$$

where $E'_s(C_o)$ measures the differences between an output cluster shape C_o and the corresponding input clusters $\{C_i\}$. A large $E'_s(C_o)$ indicates that the output cluster C_o is dissimilar to the shape of the corresponding input clusters $\{C_i\}$. As output elements should come from the input, an output cluster is of good shape (low $E'_s(C_o)$) if it is similar to the shape of one of the input clusters. The shape energy $E'_s(C_o)$ is defined as

$$E'_s(C_o) = \min_{C_i \in \mathfrak{C}_i} d_C(C_i, C_o), \tag{15}$$

where $\mathfrak{C}_i = \{C_i | s_i \in C_i, s_o \in C_o, s_i = \mu_s(s_o)\}$ is the set of candidate input clusters that have their samples s_i matched with the output samples s_o of the cluster C_o in μ_s computed in the search step (Section 6.3). $d_C(C_i, C_o)$ is the shape distance between an input cluster C_i and an output cluster C_o, which is defined as

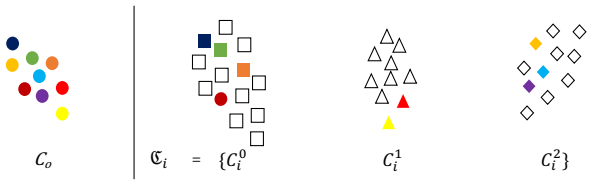
$$d_C(C_i, C_o) = \min_m \frac{1}{\epsilon} \left(\sum_{\substack{s_o \in C_o \\ s_i = m(s_o) \in C_i}} \|\mathbf{p}(s_o) - \mathcal{T}_r(\mathbf{p}(s_i), C_i, C_o)\| + \epsilon \cdot \text{abs}(|C_o| - |C_i|) \right) \tag{16}$$

The first term computes the sum of distances between the translated input cluster samples $s_i \in C_i$ by $\mathcal{T}_r$ and output cluster samples $s_o \in C_o$. $\epsilon = 2\delta$ is the extra cost induced by the difference between the numbers of input $|C_i|$ and output cluster samples $|C_o|$. m is the sample matching function in computing shape distance (Equation (16)) between two clusters of samples. The whole equation is normalized using $\epsilon = 2\delta$ to offset the scaling effect of δ (and thus the size of input exemplar/shapes) on the shape distance. $\mathcal{T}_r$ is used to translate the input cluster samples $s_i \in C_i$ to the position of the output cluster samples $s_o \in C_o$ so that we can evaluate the shape similarity based on distances between samples, which is defined as

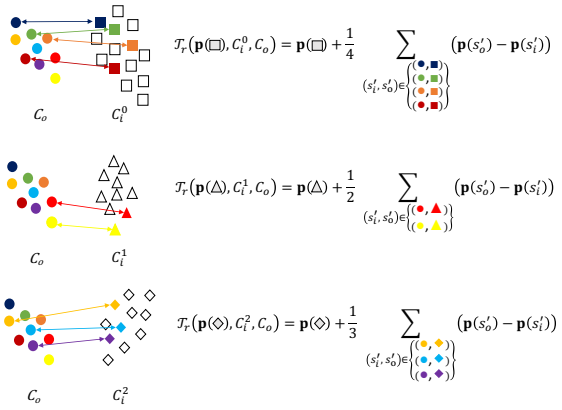
$$\mathcal{T}_r(\mathbf{p}(s_i), C_i, C_o) = \mathbf{p}(s_i) + \frac{1}{|\mathcal{S}(C_i, C_o)|} \sum_{(s'_i, s'_o) \in \mathcal{S}(C_i, C_o)} \left(\mathbf{p}(s'_o) - \mathbf{p}(s'_i)\right), \tag{17}$$

where $\mathcal{S}(C_i, C_o) = \{(s_i, s_o) | s_i \in C_i, s_o = \mu_s(s_i) \in C_o\}$ is a set of matched sample pairs where the input and output samples are associated with the input C_i and output clusters C_o respectively. Intuitively, the translation is the average of all translations between sample pairs within $\mathcal{S}$. Figure 7 illustrates the computation of Equations (15) to (17).

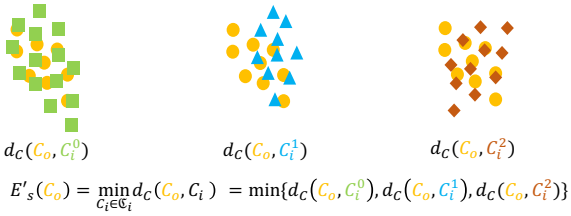
6.5.2 Optimization. To optimize Equation (9) with respect to $\{C_o^k\}$, inspired by previous image segmentation algorithms [Li et al. 2020; Liu and Sclaroff 2001] we develop a greedy algorithm that efficiently explores a large solution space via an iterative mechanism based on local operators, as shown in Figure 9, which 1) switch samples from one cluster to another nearby cluster, 2) split one cluster into two



(a) Output cluster C_o and its set of matched input clusters $\{C_i\}$



(b) Computing the translation $\mathcal{T}_r$ for each matched input cluster



(c) Shape energy as min. over matched and translated input clusters

Fig. 7. *Illustration of cluster shape energy computation.* (a) shows an output cluster C_o and its corresponding input clusters $\mathfrak{C}_i = \{C_i\}$. Matched samples visualized in the same colors are computed in the search step (μ_s). Empty shapes indicate input samples from C_i not matched with any samples from C_o. (b) shows the computation of $\mathcal{T}_r$ (Equation (17)) which will be used to translate and align input clusters C_i with the output cluster C_o for computing the shape distance $d_C(C_i, C_o)$ between them. In (c), $d_C(C_i, C_o)$ between each pair of input and output clusters is computed via Equation (16) using $\mathcal{T}_r$ computed in (b). The shape energy $E'_s(C_o)$ is defined as the minimum of shape distances between the output C_o and input $\{C_i\}$ clusters (Equation (15)).

clusters, and 3) merge two clusters into one. The energy variations induced by these operators are inserted and sorted into priority queues, with the front operator reducing the energy (Equation (9)) most. Since we will use different weights in computing sample switching and cluster operators, two queues P_s, P_C are used: P_s is for all sample switching operators and P_C is for all cluster splitting and merging operators. We first process the sample switching queue

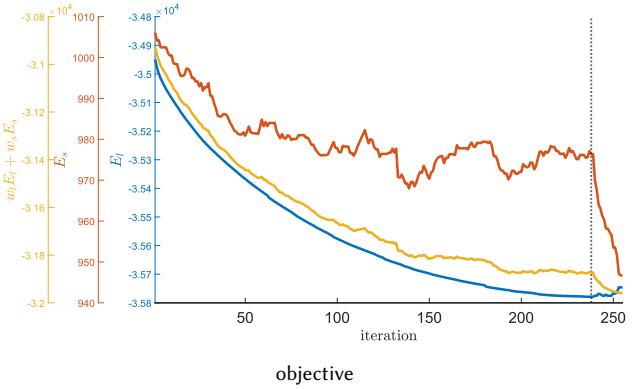
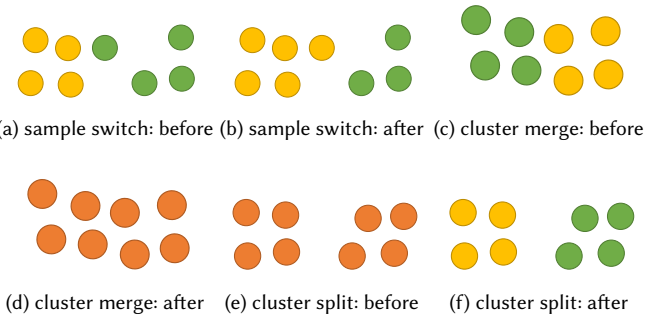


Fig. 8. *Clustering objective (Equation (9)) with respect to number of iterations.* The left side of the vertical dashed line shows the objective optimized by sample switch operators when processing P_s, and the right side shows the cluster split and merge when processing P_C. The blue, dark orange and yellow lines indicate link E_l, shape E_s and total energies $w_l E_l + w_s E_s$.



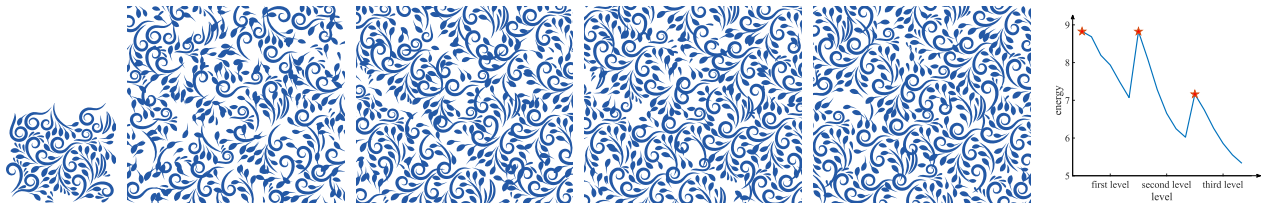
(a) sample switch: before (b) sample switch: after (c) cluster merge: before

(d) cluster merge: after (e) cluster split: before (f) cluster split: after

Fig. 9. *Clustering operators.* In each sub-figure, samples from the same cluster are shown in the same color. (a) and (b) show the sample switching operator where a sample in the green cluster is switched to the yellow cluster. (c) and (d) show the cluster merging operator where two clusters (green and yellow) are merged into one (orange). (e) and (f) show the cluster merging operator where one cluster (orange) is splitted into two (green and yellow).

P_s and then cluster operator queue P_C. Given a priority queue P of operators, we update the cluster configurations $\{C_o^k\}$ with the front operator, update queues by removing the current operator and other affected operators, and insert new operators due to the update of $\{C_o^k\}$. This process iterates until a stopping criterion is met. Figure 8 shows an example of the objective function (Equation (9)) during optimization using the clustering operators.

Sample switching. Only boundary samples s_o of an output cluster can switch to its adjacent clusters $\mathcal{A}^C(s_o)$ (Figures 9a and 9b). We extract the boundary samples of a cluster using the concave hull. A cluster C_o is adjacent to a sample s_o ($C_o \in \mathcal{A}^C(s_o)$) if there is any samples in the cluster C_o that has distance to s_o less than 2δ. All the



(a) exemplar (b) initial step (c) first level (d) second level (e) third level (f)

Fig. 10. *Evolution of patterns in the optimization.* In this example, three hierarchies are used. We show the optimized patterns after the very first optimization step (initial step) and the last steps in each hierarchy. The energy plot is shown in the last column where red stars denote the start of each optimization hierarchy. Energies are normalized using the number of samples within an output neighborhood and the total number of matched neigborhood pairs. See Figures 17 to 20 in Appendix A as well as the supplementary video for more visualizations of pattern optimization. Copyright: (a) Meganathan (stock.adobe.com)

possible sample switching operators are collected and added into P_s.

Since there are a lot more candidate sample switches than cluster splits and merges (Table 1 in Appendix A), for which the shape energy E_s is expensive to compute, we only use the link energy E_l for computing the induced energy variation during sample switching.

Cluster merging. Similarly, for a pair of clusters C_o^i, C_o^j, if there is any pair of samples from the two clusters that are adjacent with distance less than 2δ, then $C_o^j \in \mathcal{A}^C(C_o^i)$ ($C_o^i \in \mathcal{A}^C(C_o^j)$). For a pair of adjacent clusters, we merge them into one cluster (Figures 9c and 9d) and collect such operators in the priority queue P_C.

Cluster splitting. For each cluster, we split it into two clusters (Figures 9e and 9f) using normalized cuts [Shi and Malik 2000], where the graph weights are the link confidence l. These operators are collected and added into P_C.

Update priority queue. After the cluster configuration $\{C_o^k\}$ is modified with an operator, we first remove the current operator and the affected operators related to the modified clusters from the priority queue. Then we add new operators related to new clusters and insert them into the corresponding sorted priority queue. Since all operations are local, the update of P is relatively efficient.

Stopping criterion. Once the front operator in P does not reduce the energy (Equation (9)) or the number of iterations reaches a pre-specified threshold $i_{\max}$ (which equals the number of output samples for P_s or initial number of clusters for P_C), the iteration is stopped.

6.6 Reconstruction

For each cluster, we reconstruct the final shape by translating the associated element of the candidate input cluster C_i that minimizes the shape energy in Equation (15). In addition, we filter out elements with high shape energy (error) ($E_s(C_o) > |C_o|$).

6.7 Hierarchical optimization

At each hierarchy level, the optimization iterates the search, assignment and clustering steps. The next level is initialized with samples from the optimized pattern from the previous level. With increasing hierarchy levels we use denser sample distributions with decreasing Poisson disk diameter δ. Hierarchical optimization allows the optimization to use less samples and larger neighborhood size, which can capture larger spatial structure with less computation. Figure 10 visualizes a hierarchical optimization process. See Figures 17 to 20 in Appendix A as well as the supplementary video for more visualizations of pattern optimization.

7 RESULTS

Our method can automatically synthesize satisfactory results for a variety of vector patterns, as exemplified in Figures 1, 3, 11 and 14. Since our synthesis outputs are in vector format, users can edit them in any vector graphics editor for further quality improvement and customization, as demonstrated in Figure 12.

7.1 Ablation studies

We analyze the components of our method via ablation studies. Without the clustering step (Section 6.5), the generated samples resemble these by [Tu et al. 2020] in Figure 3, which cannot robustly reconstruct shapes from unclustered samples. Without using cluster information in the search step (i.e., only the first term in Equation (3) is used), the result patterns tend to be more random as shown in the second column in Figure 14. Without the link/shape energy E_l/E_s in the clustering objective (Equation (9)), the results tend to have empty and broken regions as shown in the third/forth column in Figure 14. Our full results are shown in the last column in Figure 14. We include ablation studies on the clustering operators in Figure 23.

7.2 Comparison to previous methods

We compare our method against prior methods in vector element synthesis [Hsu et al. 2020; Ma et al. 2011] (Figure 3) and vector curve synthesis [Tu et al. 2020]. Tu et al. [2020] can generate Bézier curve patterns by reconstructing curves from graphs, which cannot be applied to the varied structured input patterns that have elements other than open Bézier curves. In order to compare with [Tu et al. 2020] in reconstructed patterns instead of just samples as in Figure 3, we add a basic clustering step [Von Luxburg 2007] to each iteration in the optimization process of [Tu et al. 2020]. We use spectral clustering [Von Luxburg 2007] as the basic clustering algorithm; each input element is represented as a fully connected graph, and the output edge confidences in [Tu et al. 2020] are used as the weights for

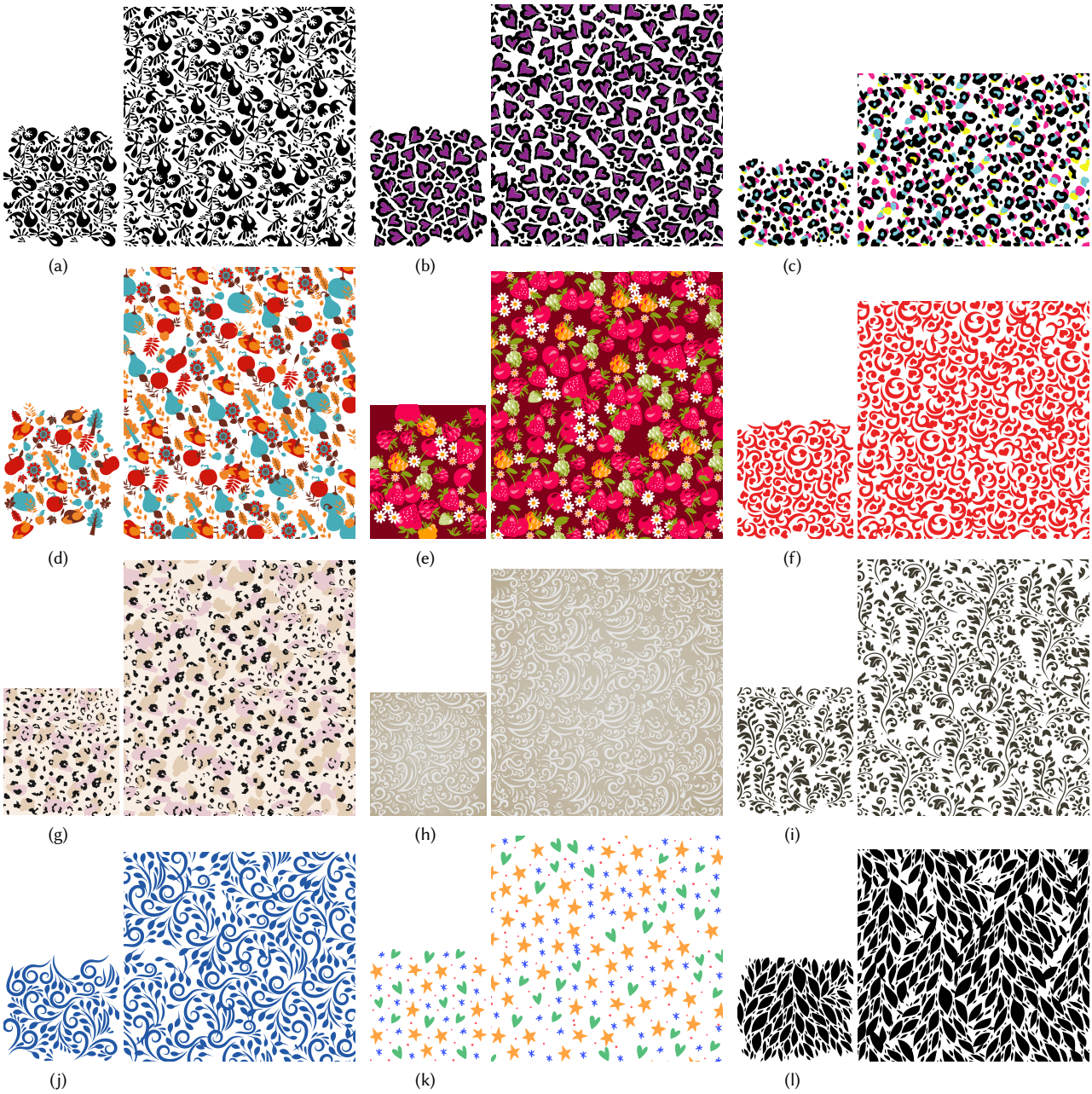


Fig. 11. *Automatic synthesis results with a variety of exemplar patterns.* Within each group, the input is on the left and our result is on the right. Copyrights: (a) galyna_p (b) leavector (c) Alona Khadzhyoglo (d) olga_milagros (e) galyna_p (f) hoverfly (g) Olgastocker (h) hoverfly (i) hoverfly (j) Meganathan (l) natalyon (stock.adobe.com), and (k) keeeny.am (vecteezy.com)

the weighted similarity graphs in [Von Luxburg 2007]; the number of output clusters is estimated as the product of the ratio between numbers of output and input samples and the number of input clusters (elements) $\frac{|O|}{|I|} \cdot |\{C_i\}|$, while our method can automatically determine the number of output clusters. In addition, the cluster informaton is used in the neighborhood similarity, as in Equations (2) and (3). The final shapes are reconstructed as described in Section 6.6.

As shown in Figures 3 and 13, our method can produce better results for a wider range of patterns than these existing methods. More comparisons are shown in Figure 24.

7.3 Parameters

Our method is robust to chosen parameters and initial conditions. Please see Appendix A.1 for a robustness analysis of our algorithm.

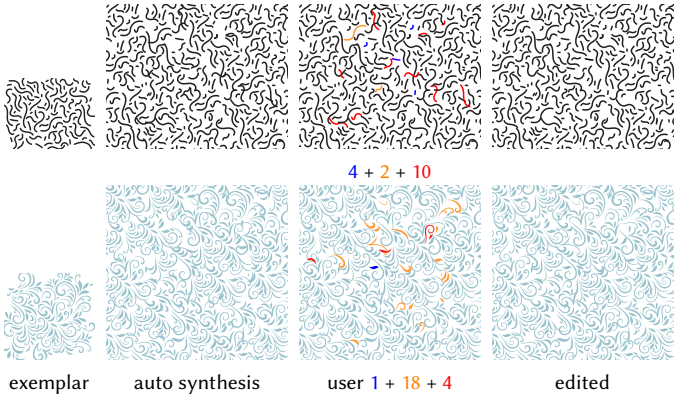


Fig. 12. *Automatic synthesis with user-editing.* Since the automatic synthesis results are in vector format, users can add (blue), modify (orange), or remove (red) elements from the output patterns. The user editing column also indicates the number of user operations for addition, modification, and removal. Top left ©Frogella.stock, bottom left ©Oleksandra (stock.adobe.com).

We use two or three levels of hierarchies. In each hierarchy, we use 7 iterations of search-assign-clustering steps. The neighborhood radius r is $\frac{1}{6}$, $\frac{1}{8}$, $\frac{1}{12}$ of the exemplar size (mean of width and height). The sampling distances δ are $\frac{1}{20}$, $\frac{1}{30}$, $\frac{1}{40}$ of the exemplar size. $\lambda = 4$ (in Equation (12) and $\lambda\delta \approx r$). All results shown in this paper are produced under a single parameter setting, except for Figure 1c, which uses $\lambda = 2$, Figure 11 (row 3, col 1), which uses one hierarchy (r, δ are $\frac{1}{12}$ and $\frac{1}{40}$ of the exemplar size), and Figure 2 where samples are denser for illustration purpose.

8 CONCLUSIONS, LIMITATIONS AND FUTURE WORK

This paper proposes a clustering method to synthesize vector patterns with complex structures that have yet to be well handled by existing techniques, as demonstrated in our results and analysis.

Vector patterns with regular or non-local structures or insufficient repetitions might not be well handled by our method (Figure 15) due to the inherent limitation of the neighborhood-based measures and local neighborhood optimization. Machine learning along with hierarchical synthesis or differentiability could be applied to address these issues [Liu et al. 2020; Mardani et al. 2020; Reddy et al. 2021; Zhou et al. 2018]. The proposed pattern optimization framework could incorporate further improvement developed in prior texture optimization literatures, such as bidirectional similarity [Simakov et al. 2008; Wei et al. 2008], spatial uniformality, or advanced initialization [Kaspar et al. 2015], for more robust pattern synthesis. For patterns with very elongated elements (e.g. the first pattern in Figure 12), our method may produce undesirable overlaps. A possible solution is to represent each elongated element using samples with central skeletal path, and synthesize them together like [Tu et al. 2020] followed by reconstruction guided by the synthesized skeletons [Hsu and Lee 1994]. We leave this as a future work.

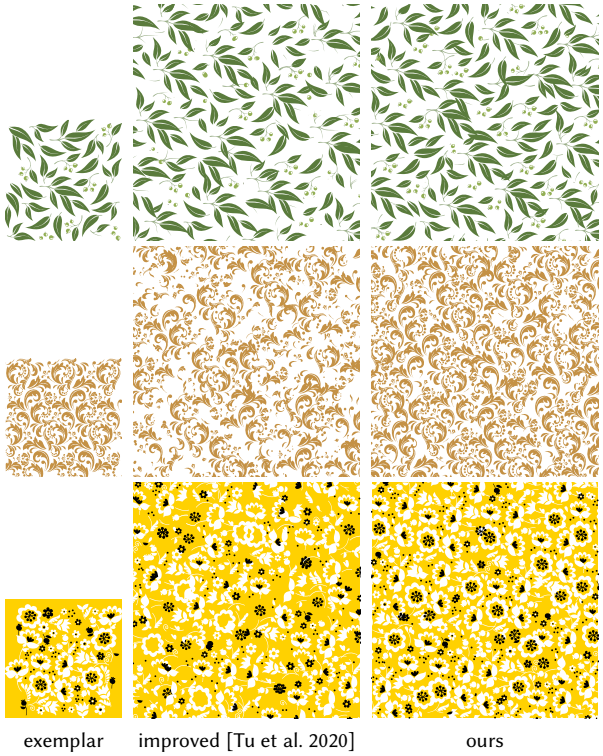


Fig. 13. *Comparisons against improved [Tu et al. 2020] with baseline clustering.* The improved version of [Tu et al. 2020] uses spectral clustering [Von Luxburg 2007] as the basic clustering algorithm; each input element is represented as a fully connected graph, and the output edge confidences in [Tu et al. 2020] are used as the weights for the weighted similarity graphs in [Von Luxburg 2007]. In addition, clusters are used during neighborhood search, as in Equations (2) and (3). The final shapes are reconstructed as described in Section 6.6.

The core of our proposed framework is explicit clustering, which is general and could be improved with more advanced clustering methods [Gan et al. 2020].

Our current method can be extended for pattern interpolation, animated patterns [Saputra et al. 2020], and 3D volumes [Takayama et al. 2010; Wang et al. 2011], as well as integrated with more user control [Kazi et al. 2012; Lu et al. 2014], such as synthesis over user-specified (irregular) domain and optional toroidal boundary condition (Figure 15e). Our unoptimized implementation is slow and only suitable for off-line synthesis. Please see the supplementary materiel for the running time of each algorithm component. Theoretically, the major computational bottleneck is in the search step (that computes nearest neighbor fields), which could be readily parallelized on a GPU [Huang et al. 2007]. For complexity analysis of the search step, please see [Tu et al. 2020]. We plan to further accelerate it to achieve interactive speed in the future, for example, by GPU parallelization, or by improving the sampling method [Wei et al. 2020] to reduce the number of required samples.

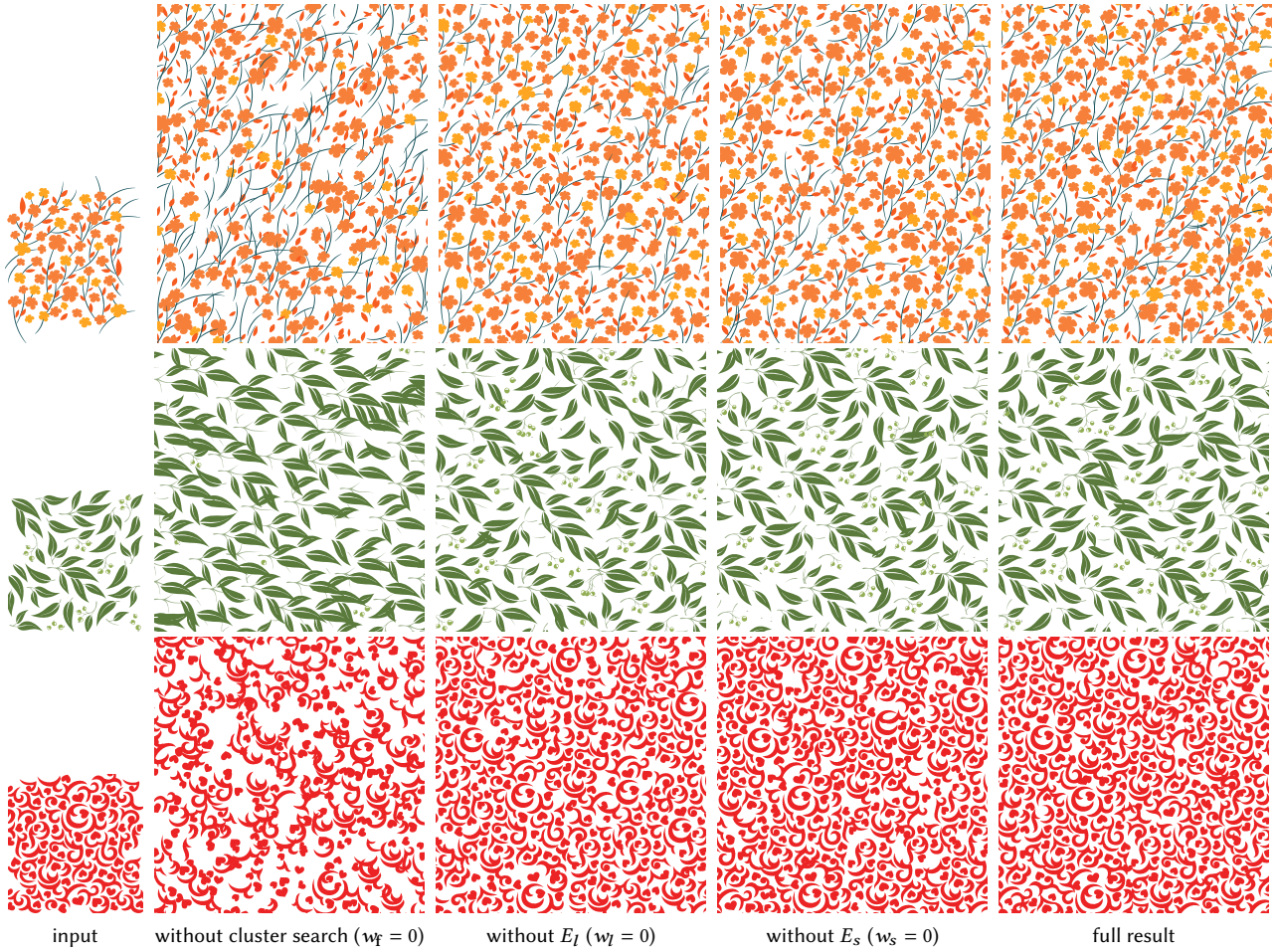


Fig. 14. *Ablation study.* Without using sample features which contain cluster information ($w_f = 0$) in the search step, the synthesis would generate results that tend to be more random. Without link/shape energy E_l/E_s in the clustering objective (Equation (9)), the results tend to have empty and broken regions as shown in the third/forth column. Our full results are shown in the last column.

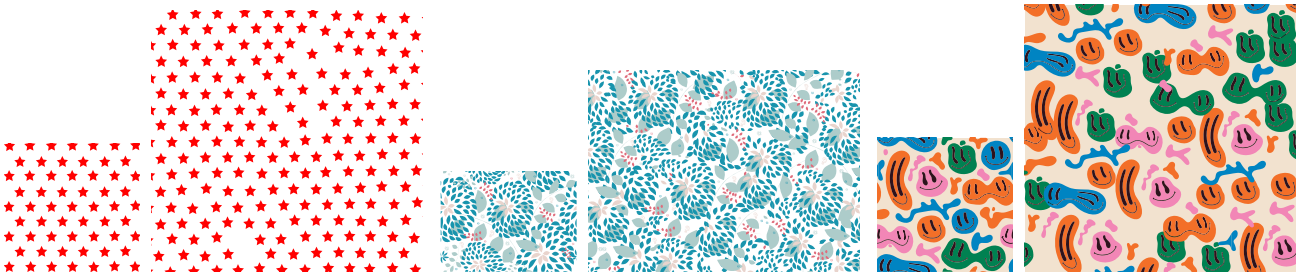


Fig. 15. *Limitations.* Our method is unable to handle patterns with (a) regular or (c) non-local structures due to the inherent limitations of patch-based synthesis method. Our method requires the input to contain sufficient spatial repetitions and could not handle small tiles with little spatial repetitions (e). Our current implementation has yet to consider toroidal boundary conditions and thus the outputs will not tile seamlessly. Copyrights: (c) Eva Kali, (e) Dariia (stock.adobe.com).

With the development of vector pattern synthesis algorithm which can generate more diverse patterns, it is desirable to use quantitative metrics to evaluate the synthesis quality in the future.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback, and Shally Kumar for reproducing Figure 3d. This work has been partially supported by an Adobe gift funding.

REFERENCES

Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion, and Lee Markosian. 2006. Stroke pattern analysis and synthesis. In *Computer Graphics Forum*, Vol. 25. Wiley Online Library, 663–671.

Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. 2009. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. *ACM Trans. Graph.* 28, 3, Article 24 (July 2009), 11 pages. https://doi.org/10.1145/1531326.1531330

Connelly Barnes and Fang-Lue Zhang. 2017. A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media* 3, 1 (2017), 3–20.

Serge Belongie, Jitendra Malik, and Jan Puzicha. 2001. Shape context: A new descriptor for shape matching and object recognition. In *Advances in neural information processing systems*. 831–837.

Robert L Cook. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)* 5, 1 (1986), 51–72.

Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. 2017. Photo2clipart: Image Abstraction and Vectorization Using Layered Linear Gradients. *ACM Trans. Graph.* 36, 6, Article 180 (Nov. 2017), 11 pages. https://doi.org/10.1145/3130800.3130888

Noa Fish, Lilach Perry, Amit Bermano, and Daniel Cohen-Or. 2020. SketchPatch: Sketch Stylization via Seamless Patch-level Synthesis. *ACM Trans. Graph.* 39, 6, Article 227 (12 2020). https://doi.org/10.1145/3414685.3417816

Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. 2016. StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings. *ACM Trans. Graph.* 35, 4, Article 92 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925948

Guojun Gan, Chaoqun Ma, and Jianhong Wu. 2020. *Data clustering: theory, algorithms, and applications*. SIAM.

Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. 2021. Computer-Aided Design as Language. arXiv:cs.CV/2105.02769

Lena Gieseke, Paul Asente, Radomir Mech, Bedrich Benes, and Martin Fuchs. 2021. A Survey of Control Mechanisms for Creative Pattern Generation. *Computer Graphics Forum* (2021). https://doi.org/10.1111/cgf.142658

Pascal Guehl, Rémi Allegre, J-M Dischler, Bedrich Benes, and Eric Galin. 2020. Semi-Procedural Textures Using Point Process Texture Basis Functions. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 159–171.

Paul Guerrero, Gilbert Bernstein, Wilmot Li, and Niloy J. Mitra. 2016. PATEX: Exploring Pattern Variations. *ACM Trans. Graph.* 35, 4, Article 48 (July 2016), 13 pages. https://doi.org/10.1145/2897824.2925950

Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. 2020. Inverse Procedural Modeling of Branching Structures by Inferring L-Systems. *ACM Trans. Graph.* 39, 5, Article 155 (June 2020), 13 pages. https://doi.org/10.1145/3394105

Alejo Hausner. 2001. Simulating Decorative Mosaics. In *SIGGRAPH '01*. 573–580. https://doi.org/10.1145/383259.383327

Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. 2001. Image Analogies. In *SIGGRAPH '01*. Association for Computing Machinery, New York, NY, USA, 327–340. https://doi.org/10.1145/383259.383295

Chen-Yuan Hsu, Li-Yi Wei, Lihua You, and Jian Jun Zhang. 2018. Brushing Element Fields. In *SIGGRAPH Asia 2018 Technical Briefs (SA '18)*. Article 6, 4 pages. https://doi.org/10.1145/3283254.3283274

Chen-Yuan Hsu, Li-Yi Wei, Lihua You, and Jian Jun Zhang. 2020. Autocomplete Element Fields. In *CHI '20*. 1–13. https://doi.org/10.1145/3313831.3376248

Siu Chi Hsu and Irene H. H. Lee. 1994. Drawing and Animation Using Skeletal Strokes. In *SIGGRAPH '94*. 109–118. https://doi.org/10.1145/192161.192186

Hao-Da Huang, Xin Tong, and Wen-Cheng Wang. 2007. Accelerated parallel texture optimization. *Journal of Computer Science and Technology* 22, 5 (2007), 761–769.

T. Hurtut, P.-E. Landes, J. Thollot, Y. Gousseau, R. Drouillhet, and J.-F. Coeurjolly. 2009. Appearance-guided Synthesis of Element Arrangements by Example. In *NPAR '09*. 51–60. https://doi.org/10.1145/1572614.1572623

Takashi Ijiri, Radomír Mech, Takeo Igarashi, and Gavin Miller. 2008. An Example-based Procedural System for Element Arrangement. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 429–436.

Jennifer Jacobs, Joel Brandt, Radomír Mech, and Mitchel Resnick. 2018. Extending Manual Drawing Practices with Artist-Centric Programming Tools. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 590, 13 pages. https://doi.org/10.1145/3173574.3174164

Alexandre Kaspar, Boris Neubert, Dani Lischinski, Mark Pauly, and Johannes Kopf. 2015. Self Tuning Texture Optimization. *Comput. Graph. Forum* 34, 2 (May 2015), 349–359. https://doi.org/10.1111/cgf.12565

Rubaiat Habib Kazi, Takeo Igarashi, Shengdong Zhao, and Richard Davis. 2012. Vignette: Interactive Texture Design and Manipulation with Freeform Gestures for Pen-and-ink Illustration. In *CHI '12*. 1727–1736. https://doi.org/10.1145/2207676.2208302

Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

Kin Chung Kwan, Lok Tsun Sinn, Chu Han, Tien-Tsin Wong, and Chi-Wing Fu. 2016. Pyramid of Arclength Descriptor for Generating Collage of Shapes. *ACM Trans. Graph.* 35, 6, Article 229 (Nov. 2016), 12 pages. https://doi.org/10.1145/2980179.2980234

Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. 2005. Texture Optimization for Example-based Synthesis. *ACM Trans. Graph.* 24, 3 (July 2005), 795–802. https://doi.org/10.1145/1073204.1073263

Pierre-Edouard Landes, Bruno Galerne, and Thomas Hurtut. 2013. A Shape-Aware Model for Discrete Texture Synthesis. *Computer Graphics Forum* 32, 4 (2013), 67–76.

Sylvain Lefebvre and Hugues Hoppe. 2006. Appearance-space Texture Synthesis. In *ACM SIGGRAPH 2006 Papers (SIGGRAPH '06)*. ACM, New York, NY, USA, 541–548. https://doi.org/10.1145/1179352.1141921

Muxingzi Li, Florent Lafarge, and Renaud Marlet. 2020. Approximating shapes in images with low-complexity polygons. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Guilin Liu, Rohan Taori, Ting-Chun Wang, Zhiding Yu, Shiqiu Liu, Fitsum A. Reda, Karan Sapra, Andrew Tao, and Bryan Catanzaro. 2020. Transposer: Universal Texture Synthesis Using Feature Maps as Transposed Convolution Filter. arXiv:cs.CV/2007.07243

Lifeng Liu and Stan Sclaroff. 2001. Region segmentation via deformable model-guided split and merge. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, Vol. 1. IEEE, 98–104.

Yitzchak David Lockerman, Basile Sauvage, Rémi Allègre, Jean-Michel Dischler, Julie Dorsey, and Holly Rushmeier. 2016. Multi-scale Label-map Extraction for Texture Synthesis. *ACM Trans. Graph.* 35, 4, Article 140 (July 2016), 12 pages. https://doi.org/10.1145/2897824.2925964

Hugo Loi, Thomas Hurtut, Romain Vergne, and Joelle Thollot. 2017. Programmable 2D Arrangements for Element Texture Design. *ACM Trans. Graph.* 36, 4, Article 105a (May 2017). https://doi.org/10.1145/3072959.2983617

Jingwan Lu, Connelly Barnes, Stephen DiVerdi, and Adam Finkelstein. 2013. RealBrush: Painting with Examples of Physical Media. *ACM Trans. Graph.* 32, 4, Article 117 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461998

Jingwan Lu, Connelly Barnes, Connie Wan, Paul Asente, Radomir Mech, and Adam Finkelstein. 2014. DecoBrush: Drawing Structured Decorative Patterns by Example. *ACM Trans. Graph.* 33, 4, Article 90 (July 2014), 9 pages. https://doi.org/10.1145/2601097.2601190

Jingwan Lu, Fisher Yu, Adam Finkelstein, and Stephen DiVerdi. 2012. HelpingHand: Example-based Stroke Stylization. *ACM Trans. Graph.* 31, 4, Article 46 (July 2012), 10 pages. https://doi.org/10.1145/2185520.2185542

Chongyang Ma, Li-Yi Wei, Sylvain Lefebvre, and Xin Tong. 2013. Dynamic Element Textures. *ACM Trans. Graph.* 32, 4, Article 90 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2461921

Chongyang Ma, Li-Yi Wei, and Xin Tong. 2011. Discrete Element Textures. *ACM Trans. Graph.* 30, 4, Article 62 (July 2011), 10 pages. https://doi.org/10.1145/2010324.1964957

Morteza Mardani, Guilin Liu, Aysegul Dundar, Shiqiu Liu, Andrew Tao, and Bryan Catanzaro. 2020. Neural FFTs for Universal Texture Image Synthesis. In *NeurIPS '20*.

Giacomo Nazzaro, Enrico Puppo, and Fabio Pellacini. 2020. DecoSurf: Recursive Geodesic Patterns on Triangle Meshes. arXiv:cs.GR/2007.10918

Giacomo Nazzaro, Enrico Puppo, and Fabio Pellacini. 2021. GeoTangle: Interactive Design of Geodesic Tangle Patterns on Surfaces. *ACM Trans. Graph.* 41, 2, Article 12 (nov 2021), 17 pages. https://doi.org/10.1145/3487909

Hans Pedersen and Karan Singh. 2006. Organic Labyrinths and Mazes. In *NPAR '06*. 79–86. https://doi.org/10.1145/1124728.1124742

Brian Price and William Barrett. 2006. Object-based vectorization for interactive image editing. *The Visual Computer* 22, 9 (2006), 661–670.

Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. 2021. Im2vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7342–7351.

Amir Rosenberger, Daniel Cohen-Or, and Dani Lischinski. 2009. Layered Shape Synthesis: Automatic Generation of Control Maps for Non-Stationary Textures. *ACM Trans. Graph.* 28, 5 (Dec. 2009), 1–9. https://doi.org/10.1145/1618452.1618453

Riccardo Roveri, A Cengiz Öztireli, Sebastian Martin, Barbara Solenthaler, and Markus Gross. 2015. Example based repetitive structure synthesis. *Computer Graphics Forum* 34, 5 (2015), 39–52.

Christian Santoni and Fabio Pellacini. 2016. gTangle: A Grammar for the Procedural Generation of Tangle Patterns. *ACM Trans. Graph.* 35, 6, Article 182 (Nov. 2016), 11 pages. https://doi.org/10.1145/2980179.2982417

Reza Adhitya Saputra, Craig S. Kaplan, and Paul Asente. 2020. AnimationPak: Packing Elements with Scripted Animations. In *Graphics Interface '20*. https://openreview.net/forum?id=sr89orrDo-o.

WWCRE Schapire and Yoram Singer. 1998. Learning to order things. *Advances in Neural Information Processing Systems* 10 (1998), 451.

Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.

Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani. 2008. Summarizing visual data using bidirectional similarity. In *CVPR 2008*. 1–8. https://doi.org/10.1109/CVPR.2008.4587842

Ondrej Šťava, Bedrich Beneš, Radomir Měch, Daniel G Aliaga, and Peter Krištof. 2010. Inverse procedural modeling by automatic generation of L-systems. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 665–674.

Kenshi Takayama, Olga Sorkine, Andrew Nealen, and Takeo Igarashi. 2010. Volumetric Modeling with Diffusion Surfaces. In *SIGGRAPH ASIA '10*. Article Article 180, 8 pages. https://doi.org/10.1145/1866158.1866202

Peihan Tu, Li-Yi Wei, Koji Yatani, Takeo Igarashi, and Matthias Zwicker. 2020. Continuous Curve Textures. *ACM Trans. Graph.* 39, 6, Article 168 (12 2020). https://doi.org/10.1145/3414685.3417780

Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17, 4 (2007), 395–416.

W3. 2020. SVG Rendering Model. https://www.w3.org/TR/SVG/render.html.

Lvdi Wang, Yizhou Yu, Kun Zhou, and Baining Guo. 2011. Multiscale vector volumes. *ACM Transactions on Graphics (TOG)* 30, 6 (2011), 1–8.

Li-Yi Wei. 2010. Multi-class Blue Noise Sampling. *ACM Trans. Graph.* 29, 4, Article 79 (July 2010), 8 pages. https://doi.org/10.1145/1778765.1778816

Li-Yi Wei, Arjun V Anand, Shally Kumar, and Tarun Beri. 2020. Simple Methods to Represent Shapes with Sample Spheres. In *SA '20 Technical Communications*. Article 3, 4 pages. https://doi.org/10.1145/3410700.3425424

Li-Yi Wei, Jianwei Han, Kun Zhou, Hujun Bao, Baining Guo, and Heung-Yeung Shum. 2008. Inverse Texture Synthesis. *ACM Trans. Graph.* 27, 3, Article 52 (Aug. 2008), 9 pages. https://doi.org/10.1145/1360612.1360651

Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. 2009. State of the Art in Example-based Texture Synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association. http://www-sop.inria.fr/reves/Basilic/2009/WLKT09

Michael T Wong, Douglas E Zongker, and David H Salesin. 1998. Computer-generated floral ornament. In *SIGGRAPH '98*. 423–434.

Yi-Ting Yeh and Radomír Měch. 2009. Detecting symmetries and curvilinear arrangements in vector art. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 707–716.

Kun Zhou, Xin Huang, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. 2006. Mesh Quilting for Geometric Texture Synthesis. *ACM Trans. Graph.* 25, 3 (July 2006), 690–697. https://doi.org/10.1145/1141911.1141942

Shizhe Zhou, Changyun Jiang, and Sylvain Lefebvre. 2014. Topology-constrained Synthesis of Vector Patterns. *ACM Trans. Graph.* 33, 6, Article 215 (Nov. 2014), 11 pages. https://doi.org/10.1145/2661229.2661238

Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. 2018. Non-Stationary Texture Synthesis by Adversarial Expansion. *ACM Trans. Graph.* 37, 4, Article 49 (July 2018), 13 pages. https://doi.org/10.1145/3197517.3201285

Changqing Zou, Junjie Cao, Warunika Ranaweera, Ibraheem Alhashim, Ping Tan, Alla Sheffer, and Hao Zhang. 2016. Legible Compact Calligrams. *ACM Trans. Graph.* 35, 4, Article 122 (July 2016), 12 pages. https://doi.org/10.1145/2897824.2925887

A APPENDIX

A.1 Robustness analysis

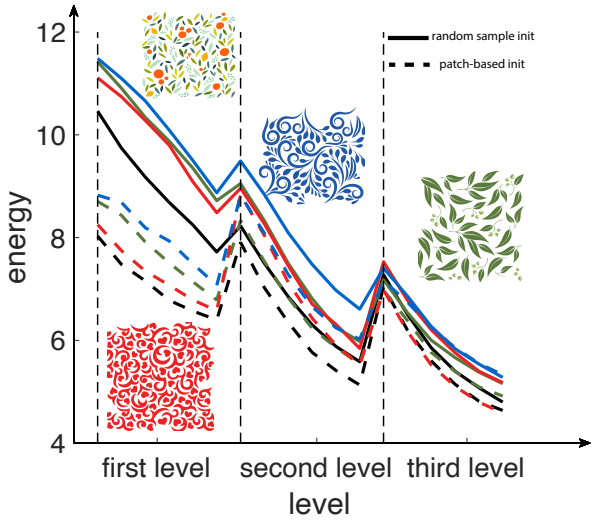


Fig. 16. *Energy plots.* We visualize all energy plots in a single figure for direct visual comparison. Energies are normalized using the number of samples within an output neighborhood and the total number of matched neigborhood pairs. Solid and dashed curves represent plots produced with random sample and patch-based initializations, respectively. Corresponding exemplars and energy curves are shown in the same colors, except for the colorful pattern (shown on the top left) which corresponds to the black curves. Vertical dashed lines indicate the start of each optimization hierarchy.

Table 1. *Ratios between numbers of clustering operations and samples/clusters in each hierarchy.* For sample switch, the ratio is computed between the numbers of sample switches and the numbers of samples. For cluster merge/split, the ratios are computed between the numbers of cluster merges/splits and the numbers of clusters. Since a cluster usually contain many samples, there are many more sample switches than cluster splits/merges.

		Ratios		
Exemplar	Levels	Switch	Merge	Split
	1	0.1420	0.1047	0.0793
	2	0.1197	0.0526	0.0788
	3	0.0756	0.0323	0.0532
	1	0.1030	0.0782	0.0556
	2	0.0664	0.0384	0.0549
	3	0.0417	0.0319	0.0486
	1	0.1048	0.0994	0.0525
	2	0.0939	0.0358	0.0612
	3	0.0383	0.0095	0.0215

A.1.1 Initial conditions and optimization. Figures 16 to 20 analyze the robustness of our method with different initial conditions and visualize the optimization process. Side-by-side comparisons of pattern optimization process using patch-based and random sample initialization are shown in the supplementary video.

A.1.2 Hyperparameters. We also show the influence of hyperparameters w_f (in Equation (3)) and w_s (in Equation (9)) on the synthesis results. It is shown that our method is robust to moderate parameter variations.

A.1.3 Clustering operators. We study the effects of operators in the clustering step (Section 6.5.2). Figure 23 shows the effects of using different combinations of clustering operators. Table 1 shows the ratios between the numbers of sample switch operations and samples as well as the ratios between the numbers of cluster merge and split operations and clusters.

A.1.4 More Comparisons with prior art. We show more comparisons with prior methods in Figure 24.

A.1.5 A large synthesis result. We enlarge the exemplar (Figure 11j) 4×4 times, and the result is shown in Figure 25.

A.2 Implementation details

A.2.1 Representation. To compute the shape context feature vector $\mathbf{f}$, we use default hyperparameters (5 bins for logarithmic distances and 12 bins for angles) provided in [Belongie et al. 2001].

A.2.2 Assignment step.

Position assignment. The sample position is computed using least squares [Ma et al. 2011; Tu et al. 2020]

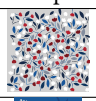
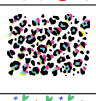
$$\underset{\{\mathbf{p}(s_o)\}}{\arg\min} \sum_{\substack{s_o \in O \\ s_i = \mu_s(s_o)}} \sum_{\substack{m_s : \mathbf{n}(s_o) \to \mathbf{n}(s_i) \\ s'_o \in \mathbf{n}(s_o) \\ s'_i = m_s(s'_o)}} \left\| \hat{\mathbf{p}}(s_o, s'_o) - \left(\mathbf{p}(s_i) - \mathbf{p}(s'_i)\right) \right\|^2 \tag{18}$$

Sample existence assignment. We follow [Tu et al. 2020] on sample existence assignment, which adaptively adjust the number of total samples by adding or removing samples based on confidence of existence ξ. Since the element filtering in Section 6.6 plays similar role to sample removal in getting rid of redundant elements (and thus redundant samples), which is complemented with sample addition, we only add samples during existence assignment.

A.3 Performance

Our implementation has not yet been optimized for interactive applications. Our algorithm is run on a workstation with AMD Ryzen 9 3950X 16-Core 3.49 GHz CPU and 32 GB RAM. The algorithm is run on a single thread, except that the patch match algorithm [Barnes et al. 2009] in the search step is parallelized using four threads on the CPU. We show the running time for each algorithm component in Table 2. The major computational bottleneck is in the search step. Our method can be accelerated by further parallelizing the nearest neighborhood search, assignment step with voting scheme and the queue updates of clustering operators with hardware acceleration; faster linear assignment problems (LAP) for the computation of neighborhood metric and shape energy; and sparser sampling of patterns while maintaining representative quality.

Table 2. *Running time of each algorithm component at different hierarchy (unit: second).* Note there are seven search/assignment/clustering steps in a hierarchy, and what we show is the total time consumed by the seven steps. Since we adapt number of samples (# samples) during optimization, # samples is the average number of samples in a hierarchy of optimization.

					Assignment		Clustering				
Exemplar	Level	# Samples	Initialization	Search	Position	Attribute	Switch	Merge&Split	Recon	Total	
	1	1816	1.1	254.5	0.8	24.8	4.1	4.8	25.7	315.6	
	2	3032	5.2	291.4	1.4	36.9	19.9	5.7	37.3	397.8	1002.1
	3	4591	6.1	144.6	2.5	32.6	44.9	7.7	50.4	288.7	
	1	2845	2.4	1603.6	2.1	89.6	30.2	17.9	53.8	1799.5	
	2	4751	10.3	1829.4	4.4	134	98.5	25	77.7	2179.2	5513.5
	3	7370	12.3	913.6	9.5	189	237.6	44.3	128.5	1534.8	
	1	2388	3.1	814.4	1.4	57.2	12.0	6.1	22.3	916.5	
	2	4629	10.5	1706.8	3.8	145.2	101.1	16.9	37.2	2021.5	5036.5
	3	7626	13	1482.7	9.7	280.5	227.2	30.3	55.1	2098.5	
	1	1641	2.6	317.6	0.9	24.1	15.1	5.1	7.4	372.8	
	2	3057	4.9	460.5	1.8	57.9	37.4	6.8	11.7	581	1290.1
	3	4510	5.9	149.5	2.8	72	80.0	8.7	17.4	336.3	
	1	1200	0.6	61.7	0.4	7.2	2.7	1.3	3.4	77.2	
	2	2104	2.6	84.3	0.7	11.1	6.1	2.3	4.5	111.7	320.5
	3	3357	3.0	87.1	1.6	17.1	12.5	3.6	6.6	131.6	
	1	1309	0.7	75.3	0.5	9.4	3.1	2.3	2.4	93.6	
	2	2417	2.7	134.5	0.9	18.6	9.9	3.5	3.6	173.8	508.8
	3	3888	3.3	174.7	1.9	33.6	18.7	4.6	4.8	241.5	
	1	1900	1.3	350.2	0.9	29.4	11.3	4.6	16.2	413.9	
	2	3879	4.6	849.3	2.8	79.2	76.6	12.4	24.9	1049.8	3592.1
	3	6965	6.1	1589	9.1	196.1	264.5	27.8	35.8	2128.4	
	1	573	4.5	5.8	0.1	1.1	4.5	0.4	1.6	18.1	
	2	913	19.7	6.5	0.3	3.2	26.2	0.3	2.5	58.7	182.0
	3	1505	22.1	22	0.7	12.2	45.3	0.4	2.5	105.2	
	1	1707	4.7	269.1	1	22.9	11.4	3.7	12	324.9	
	2	2953	6.8	361.9	1.7	47.6	29.1	7.7	16.5	471.4	1060.9
	3	4688	7.4	80.5	2.6	62.1	72.6	11.8	27.7	264.6	
	1	2075	4.1	531.6	1.1	38	9.4	5.6	27.1	617	
	2	3311	7.1	472.2	1.9	58.9	31.1	8.9	38.8	618.9	1615.5
	3	5259	8.3	147.9	3.7	84.6	68.6	15.2	51.3	379.6	

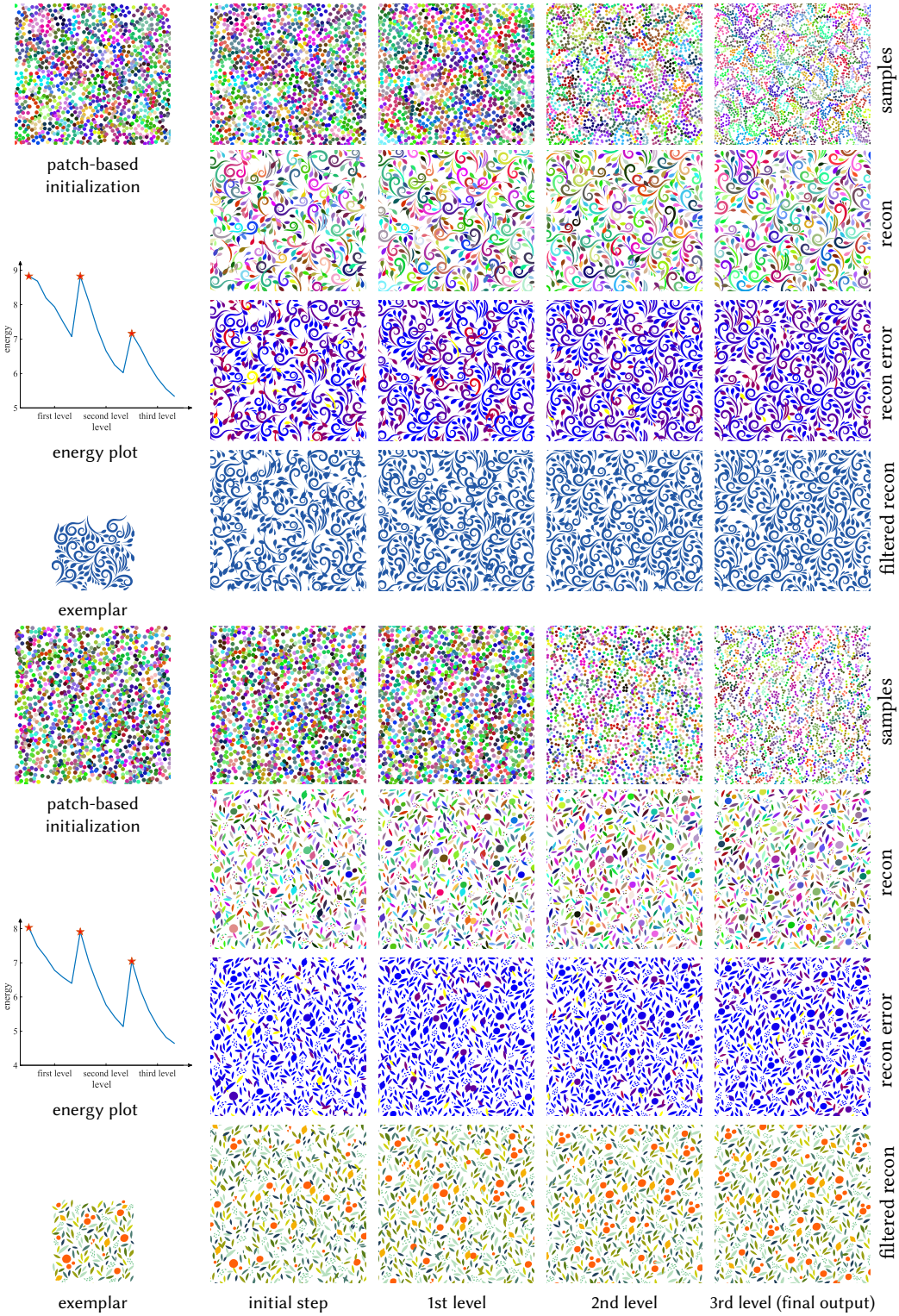


Fig. 17. *Optimization process with patch-based initialization.* We show the intermediate samples (first/fifth rows, where the first column shows initialization), reconstructed patterns (second/sixth rows), reconstruction errors (third/seventh rows) and filtered reconstructions (forth/eighth rows) after the very first optimization step (initial step) and the last steps in each hierarchical levels. The energy plots are shown in the first column where red stars annotate the start of each optimization hierarchy. Corresponding clusters and elements are visualized in the same colors. In third/seventh rows, red and blue indicate high and low reconstruction errors; yellow indicates filtered-out elements with errors surpassing a threshold.

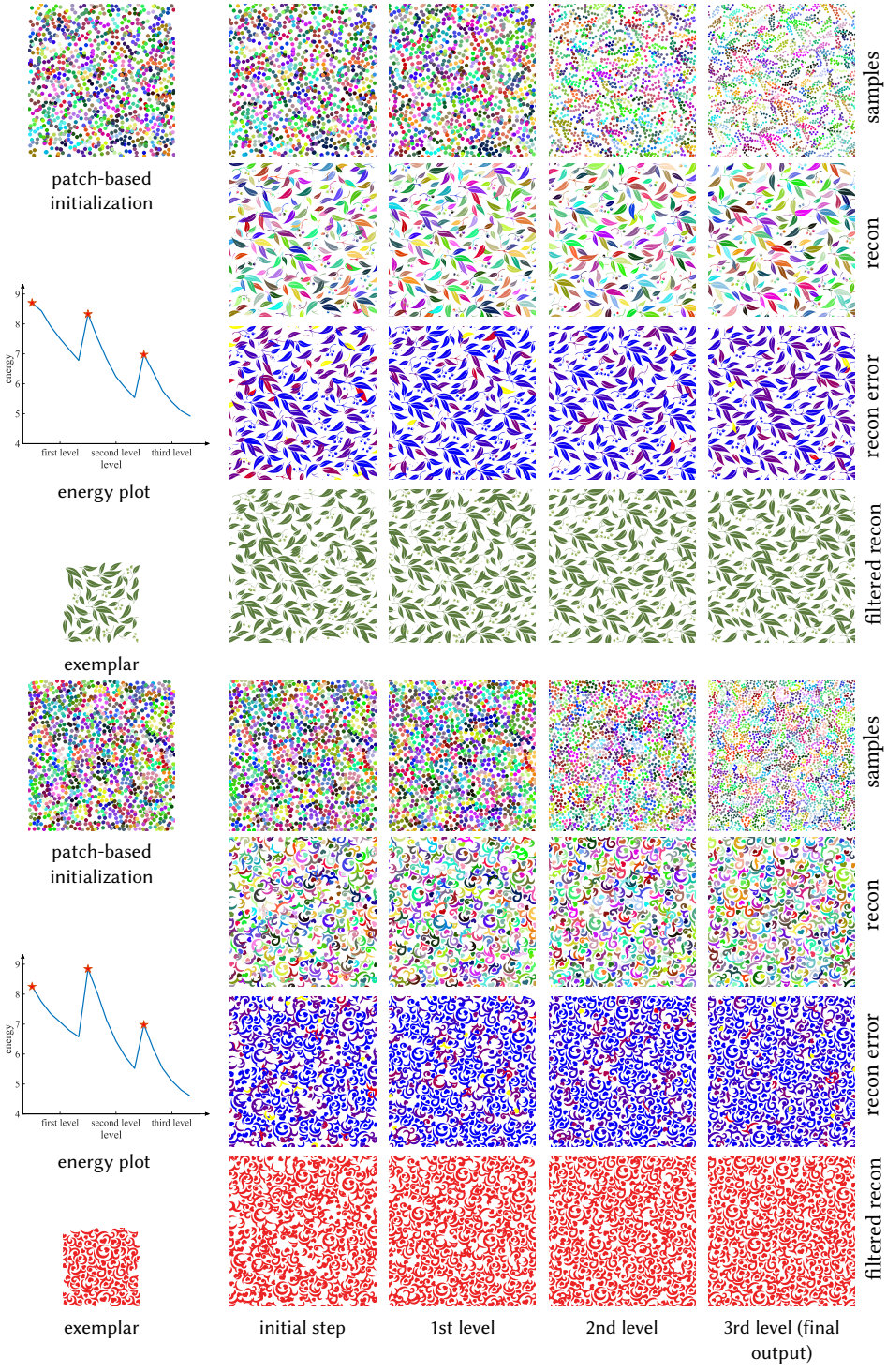


Fig. 18. *Continuation from Figure 17.*

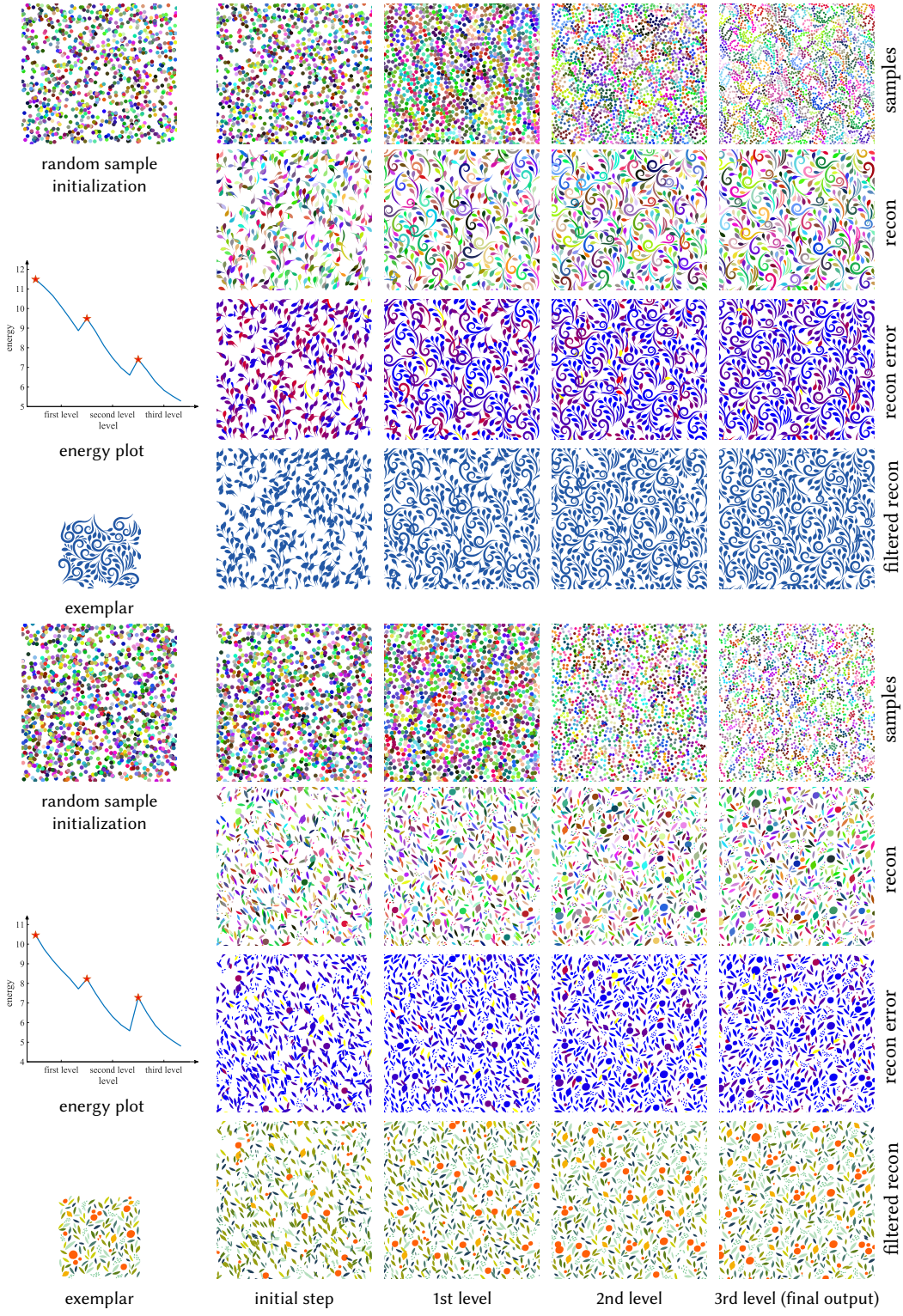


Fig. 19. *Optimization process with random sample initialization.* We show the intermediate samples (first/fifth rows, where the first column shows initialization), reconstructed patterns (second/sixth rows), reconstruction errors (third/seventh rows) and filtered reconstructions (forth/eighth rows) after the very first optimization step (initial step) and the last steps in each hierarchical levels. The energy plots are shown in the first column where red stars annotate the start of each optimization hierarchy. Corresponding clusters and elements are visualized in the same colors. In third/seventh rows, red and blue indicate high and low reconstruction errors; yellow indicates filtered-out elements with errors surpassing a threshold.

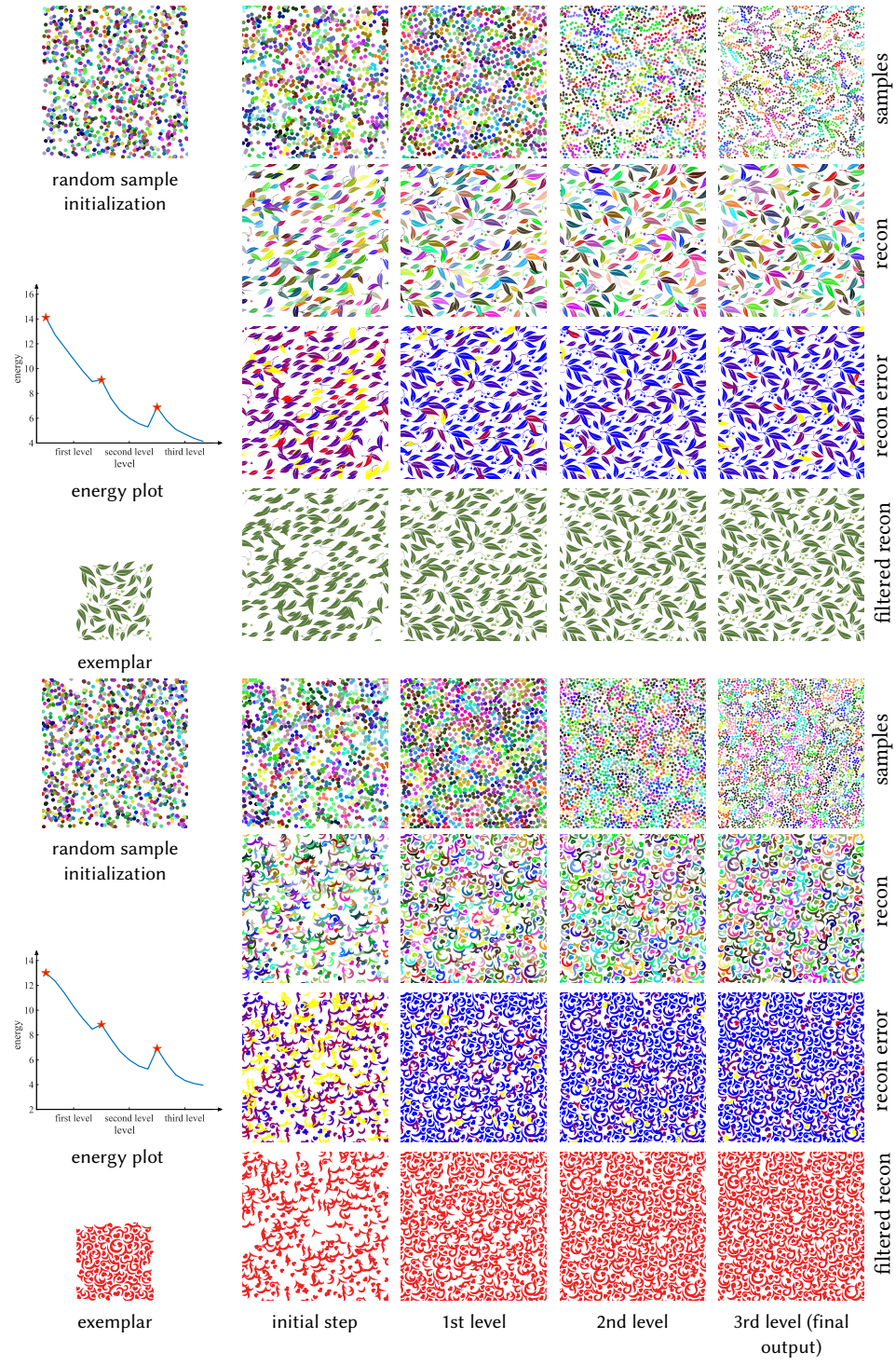


Fig. 20. *Continuation from Figure 19.*

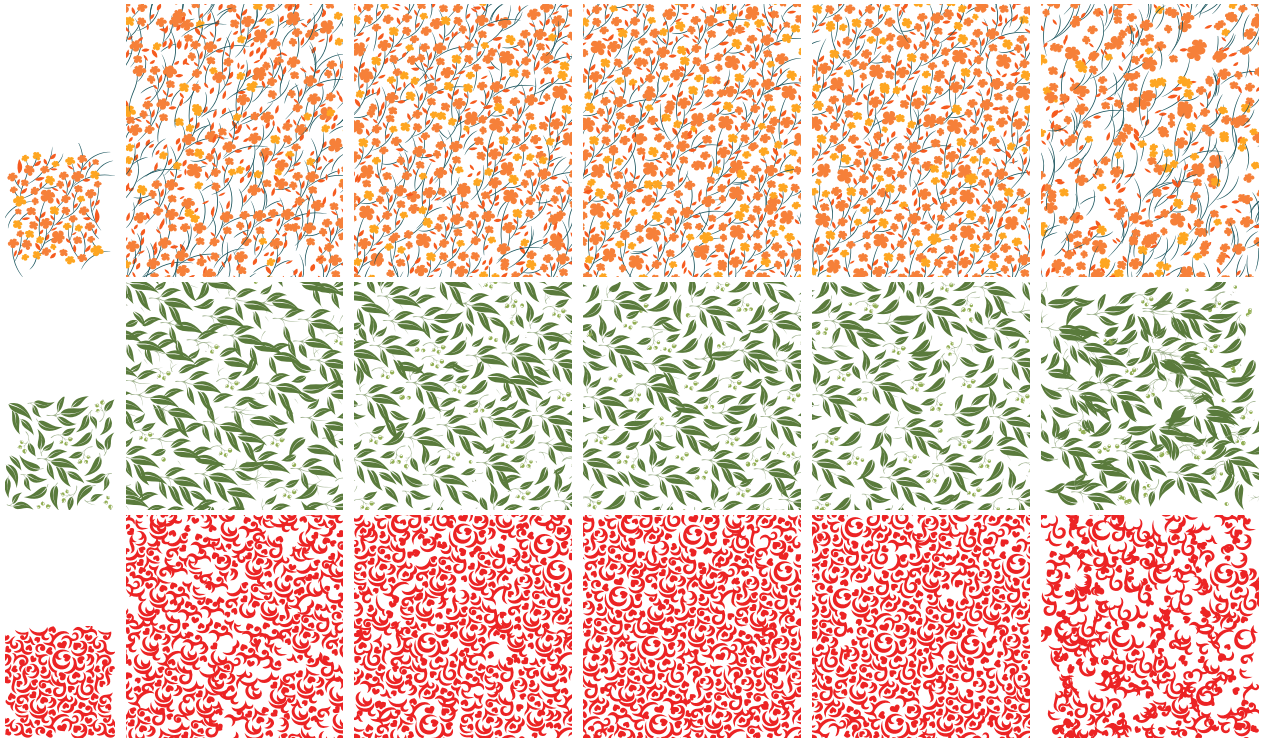


Fig. 21. *Synthesis results using different hyperparameters* w_f. $w_f = 10$ is the chosen parameter.

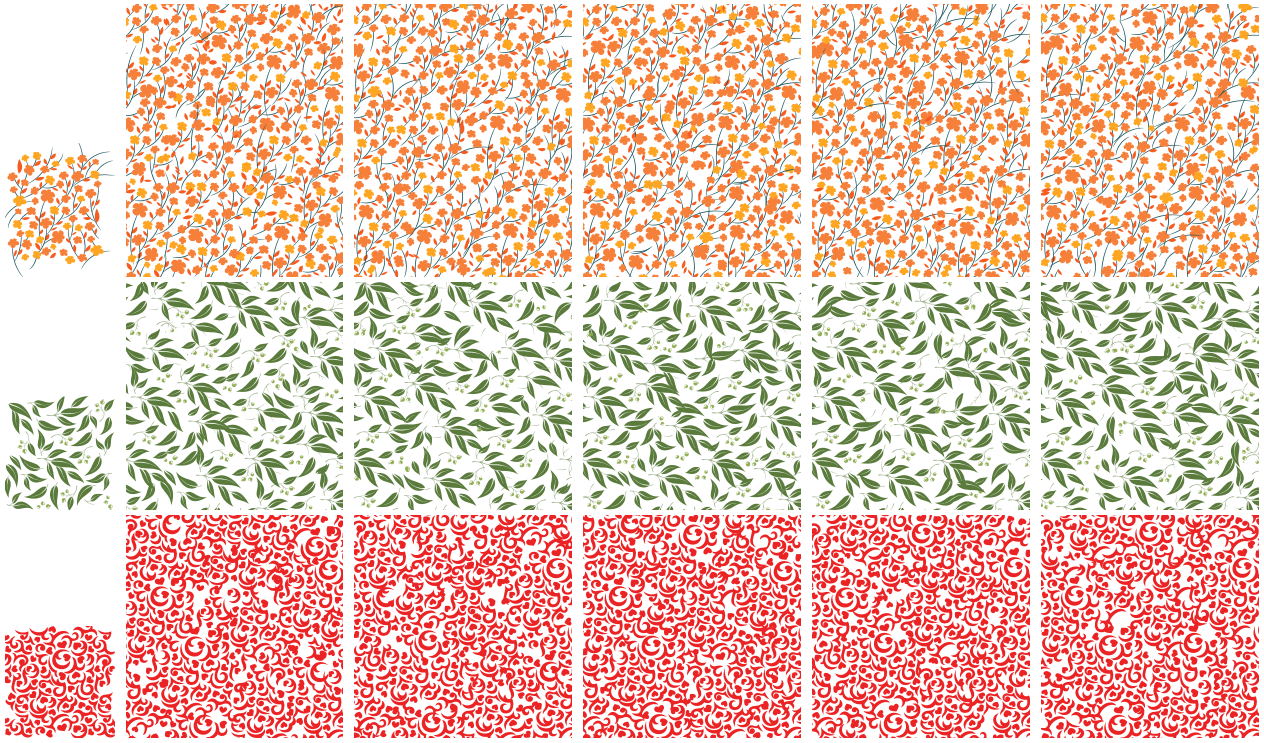


Fig. 22. *Synthesis results using different hyperparameters* w_s. $w_s = 4$ is the chosen parameter.

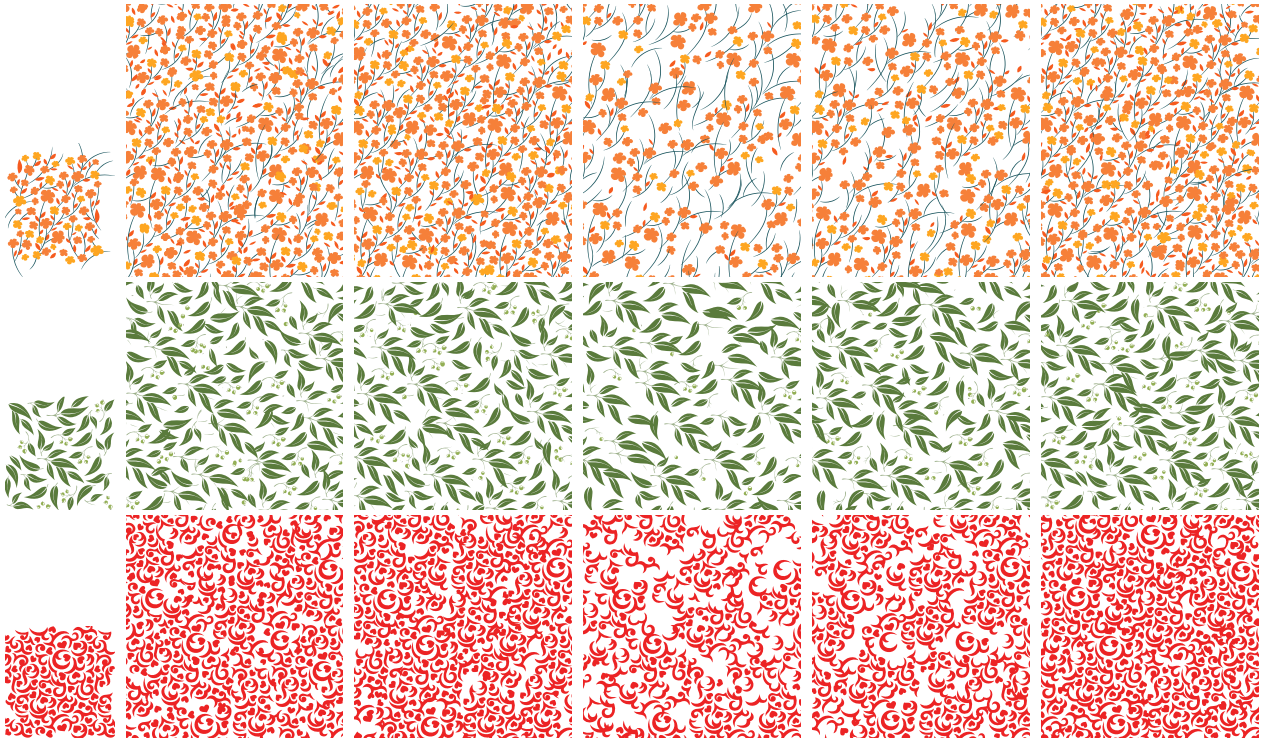


Fig. 23. *Synthesis results using different combinations of clustering operators.* Our method tend to produce empty or broken regions without these clustering operators.

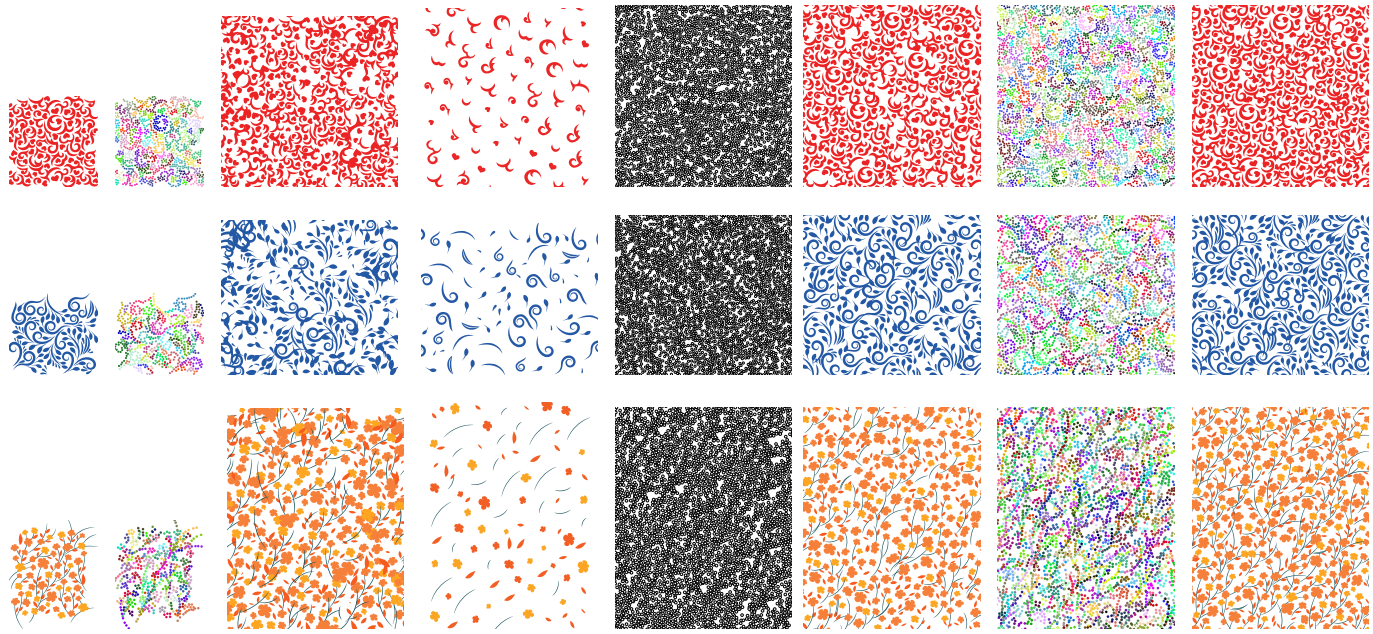


Fig. 24. *More comparisons with previous methods.* Please see Figures 3 and 13 for detailed descriptions.

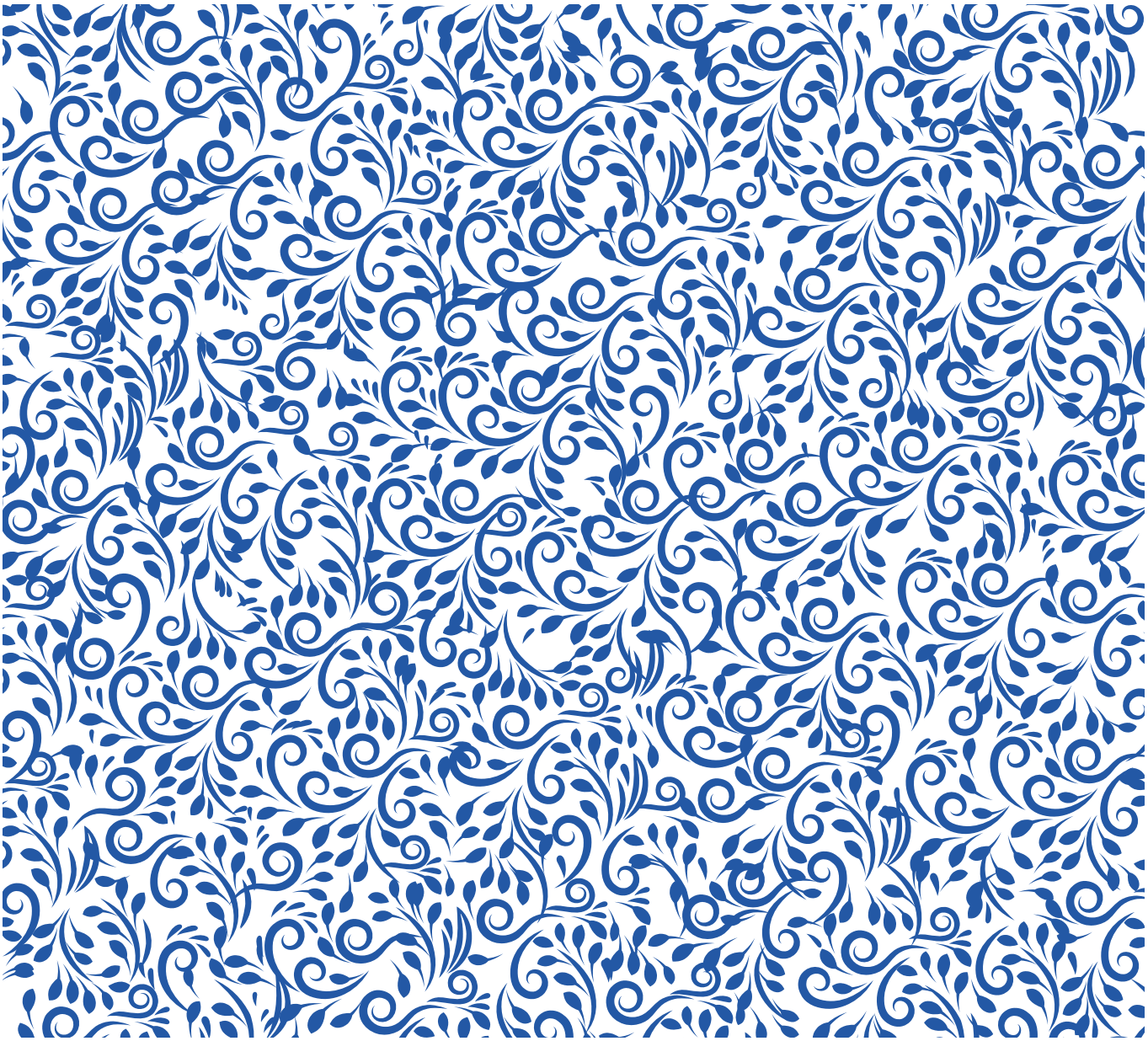


Fig. 25. *A large synthesis result.* This result is generated by enlarging the exemplar (Figure 11j) 4×4 times.