DOI: 10.1111/cgf.12695
Eurographics Symposium on Geometry Processing 2015
Mirela Ben-Chen and Ligang Liu
(Guest Editors)

Volume 34 (2015), Number 5

Example Based Repetitive Structure Synthesis

Riccardo Roveri[1], A. Cengiz Öztireli[1], Sebastian Martin[2], Barbara Solenthaler[1], Markus Gross[1]

[1]ETH Zurich
[2]VirtaMed Research, Switzerland

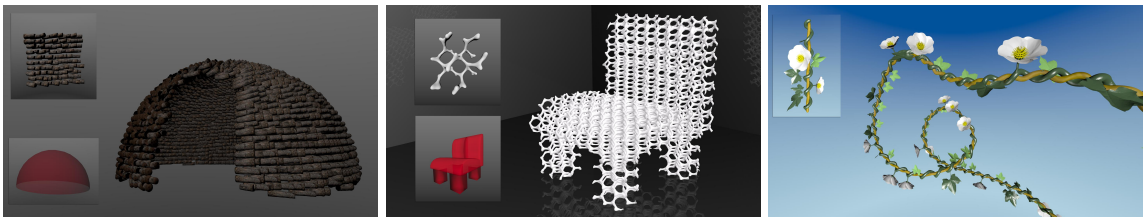

Figure 1: Our example-based structure synthesis method can be used to generate structures with discrete elements (left), continuous geometries (middle), and their mixtures (right); on different domains such as surfaces, bounding volumes, or curves.

Abstract

We present an example based geometry synthesis approach for generating general repetitive structures. Our model is based on a meshless representation, unifying and extending previous synthesis methods. Structures in the example and output are converted into a functional representation, where the functions are defined by point locations and attributes. We then formulate synthesis as a minimization problem where patches from the output function are matched to those of the example. As compared to existing repetitive structure synthesis methods, the new algorithm offers several advantages. It handles general discrete and continuous structures, and their mixtures in the same framework. The smooth formulation leads to employing robust optimization procedures in the algorithm. Equipped with an accurate patch similarity measure and dedicated sampling control, the algorithm preserves local structures accurately, regardless of the initial distribution of output points. It can also progressively synthesize output structures in given subspaces, allowing users to interactively control and guide the synthesis in real-time. We present various results for continuous/discrete structures and their mixtures, residing on curves, submanifolds, volumes, and general subspaces, some of which are generated interactively.

1. Introduction

Repetition is an integral part of nature. Modeling repetitive structures is thus essential but also challenging. A common approach is controlling the large scale structure of an object by direct modeling, and letting an algorithm automatically add the details based on an example from the repetitive structures [MWT11]. This has led to many algorithms tailored to particular applications with certain assumptions on the structures to be synthesized. Each of these algorithms thus come with application dependent constraints, which has been hindering content creation with general repetitive structures.

A classical approach for synthesizing repetitions is rasterizing them into regularly sampled images and using neighborhood matching based texture synthesis methods [WLKT09] to compute colors for each output pixel/voxel. This idea has also been extended to geometry synthesis for certain geometry representations [ZHW*06]. However, this raster based representation can only model a limited set of structures. Indeed, many repetitions in nature consist of individual elements, which should be kept intact. This has led to using geometric texture synthesis methods with discrete element textures, where individual elements and their interactions are utilized to describe the repetitive structure [MWT11,LGH13].

The discrete elements also allow resolution independent synthesis with object instancing.

However, there remain important challenges for discrete element based texture synthesis. 1) Preservation of element shapes comes at the expense of losing the ability of synthesizing continuous structures. With the current techniques, it is not possible to handle mixtures of continuous and discrete structures in general domains. 2) Representing textures with points makes the distinction between structure and sampling ambiguous, unlike raster based textures. This translates into critical dependence on initial distribution of point samples, and non-trivial neighborhood definition and matching methods, which can result in unsatisfactory synthesis especially when continuous structures are desired. 3) Content creation with repetitive structures remains to be a challenge since the current methods are designed for off-line texture synthesis and do not support interactive artistic control for general domains [MWT11, LGH13, XCW14].

We address these challenges by proposing a new method for progressive synthesis of general repetitive structures, unifying and extending previous texture synthesis techniques. The structures can consist of mixtures of discrete and continuous elements with arbitrary distributions and attributes, and reside in general domains including curved submanifolds. This allows us to extend the space of synthesizable structures, and robustly handle many structures in a unified framework. The synthesis can be intuitively and interactively guided by orientation and scaling fields defined along curves, surfaces, or volumes in 2D or 3D. The output is then synthesized on the fly automatically, allowing interactive texture brushing of general repetitive structures.

This is made possible by adopting a meshless representation and optimization framework, inspired by similar general approaches in geometry reconstruction [ABCO*01] and physically-based simulation [MKB*10]. The structures in the example and the output are first converted into a functional representation. The functions are represented by point samples with attributes extracted from the structures. The texture synthesis problem is then formulated as a smooth minimization that matches patches from the output function to those of the example function.

The generality of the meshless method allows us to seamlessly handle general structures. The functional representation results in a better neighborhood matching metric that correlates well with the visual quality of the synthesized structures. The smoothness of the optimization problem leads to robust minimization procedures with precise sampling control, which are essential to avoid bad configurations leading to incomplete or distorted structures. We show a variety of examples that range from classical texture synthesis to mixtures of continuous and discrete elements synthesized on user controlled curved domains, to illustrate the utility of our method in practice.

2. Related Work

Raster-based Texture Synthesis A classical approach to synthesizing repetitive structures based on given examples is to rasterize the example and the output, usually referred to as texture synthesis. The structures are thus represented as continuous values stored at regularly distributed sample points. For synthesis, the stored values in the output image are altered while keeping the point locations fixed. A texture image of arbitrary size can be synthesized from a given example image that contains a patch of the repetition using this technique [PL95, EL99]. Most texture synthesis methods work by matching local neighborhoods such that for each neighborhood in the output, there is a similar neighborhood in the example [WLKT09]. This idea has also been extended to colors on meshes and 3D volumes [Tur01, KFCO*07, WLKT09]. However, if the pattern consists of discrete elements, the integrity of individual elements can be lost with this method since they are rasterized [MWT11]. Furthermore, the points are assumed to stay fixed on a regular structure, and most commonly on a 2D image. These limit the application of raster-based texture synthesis to general structures.

Geometric Texture Synthesis Instead of rasterizing discrete structures, they can be directly represented with discrete elements. The goal is then to synthesize a distribution of these elements that resembles a given example [BBT*06, IMIM08, HLT*09, AdPWS10, MWT11, MWLT13, AKA13, LGH13, DHM13, HWFL14]. The methods mainly differ in the representation of the structures and the definition and matching of element neighborhoods. For simpler example patterns, it is sufficient to match aggregated neighborhood statistics for all elements [OG12, ZHWW12, HSD13]. However, for more complex examples, individual neighborhoods of elements need to be matched [MWT11, AKA13], unless more elaborate distance metrics based on element shapes are utilized [LGH13].

Most of these methods are designed for particular applications and hence operate under certain assumptions on the space, shapes, and arrangements of the elements. A notable exception is the work by Ma et al. [MWT11] that can synthesize element distributions in 2D, 3D, or on surfaces with variable element shapes arranged in arbitrary configurations, thanks to their point-based representation of element shapes. Our method also works with point samples for arbitrary domains and shapes of discrete elements. However, unlike geometric texture synthesis methods, it can synthesize continuous, as well as mixtures of discrete and continuous structures interactively. This is due to a new formulation of the neighborhood matching problem in a meshless framework that allows careful sampling control and better convergence behavior.

Most general geometric texture synthesis methods are originally designed for off-line texturing of defined regions. Recent efforts utilize and extend texture synthesis methods for generating repeated patterns in paintings [KIZD12a, LBDF13, LFB*13, KCG*14, LBW*14, XCW14]. However, these tech-

niques are designed for particular interaction scenarios and output. In contrast, our progressive synthesis method can be used to generate general structures interactively.

Continuous Geometry Synthesis Since the previously mentioned methods cannot be used to synthesize continuous structures when the domain is not a 2D or 3D volume, several other methods have been developed to synthesize continuous geometry for terrains [ZSTR07], mesh-based geometry [ZHW*06, LHGM05] or 3D models [MM08]. In addition, structured 2D pattern synthesis along curves has been recently proposed in [LBW*14] and [ZLL13], and later made suitable for fabrication in [ZJL14]. The continuity assumption comes at the expense of the possibility of losing shapes of individual elements, similar to raster-based texture synthesis methods. Furthermore, the structures should have application specific representations for these methods to work. Our method can handle continuous synthesis in general domains in combination with discrete elements, and for all structure representations once they are converted into point samples.

3. Overview

Our method is based on geometries represented by point samples with associated attributes (Section 4.1). In order to robustly compare the input exemplar and the synthesized structure, point data is converted into a functional representation encoding both the spatial configuration and attributes in the form of a sum of Gaussians (Section 4.3). A similarity measure between two smooth functions is then constructed (Sections 4.2 and 4.3). As in previous neighborhood-based texture synthesis approaches, our optimization alternates a matching step for local neighborhoods, and a merging step where point locations and attributes are updated according to the matching (Section 5.1). Our smooth formulation allows the computation of analytic gradients, thus robust methods such as the gradient descent can be applied. A multi-scale approach is used to optimize first large scale structures and then fine details, and a dynamic sampling control strategy based on the presented similarity measure guarantees the generation of a proper number of output samples (Section 5.1). Several large-scale control possibilities are presented in Section 6.1.

Our geometry representation based on points and attributes allows to model different kinds of structures in the same framework. Continuous structures (Figure 1 center, Figure 13) can be generated by storing a scalar or vector as attribute at each point location (for example, the surface normal vector). Discrete elements can be synthesized by either representing them with multiple points or with single samples (Figure 1 left). With the same optimization procedure, point data representing mixtures of discrete elements and continuous structures can be processed (Figure 1 right, Figure 18).

4. Measuring Structure Similarity

In this section we first present a flexible and powerful geometric representation that allows us to define a robust similarity measure between the generated and the example geometries that serves as the basis for synthesis. We strive to design a measure that accurately describes general structures and their similarities, accepts an adaptive and efficient discrete representation, and is smooth for utilizing robust and efficient optimization procedures.

4.1. Geometry Representation

In order to derive a general method for synthesizing new geometry from a set of examples, we need a general way of representing arbitrary geometries. We allow that they can have different dimensionality (1D/2D/3D) with possibly non-manifold connections. One obvious option would be to choose simplicial complexes (line segments, triangles, tetrahedra) as the underlying discrete representation [LGH13, ZQC*14]. However as we do not want to deal with explicit connectivity between vertices, we choose to follow the general point-based representation of Ma et al. [MWT11] and treat material connectivity implicitly. That is, we represent all geometries by a set of tuples

$$\{(\mathbf{x}_i, \mathbf{a}_i), i = 1..n\} \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ are the point locations and $\mathbf{a}_i \in \mathbb{R}^{d'}$ is a vector of associated continuous attributes and encode additional geometric or appearance information. Choosing such a general representation allows us to cover a large variety of different applications as we will see later.

However, unlike Ma et al. [MWT11], we do not use the point samples themselves as the representation for the neighborhoods. Instead, we construct auxiliary smooth functions defined in terms of these samples, and compute a matching measure based on these functions, as we explain next.

4.2. Continuous Similarity Measure

In order to construct a robust similarity measure for our point-based geometry representation, we first study the problem of measuring similarity between two continuous functions and will then show how to perform a meshless discretization in order to derive the actual numerical scheme.

Similarity Error Density Let us first consider the problem of measuring *local* similarity between an output function $\mathbf{f}(\mathbf{x})$ and the example function $\mathbf{e}(\mathbf{x})$. For this purpose, we define a window function $w(.)$ to delimit a local neighborhood (Figure 2, a) and a discontinuous mapping function $\mathbf{m}(\mathbf{x})$ that matches the output domain point $\mathbf{x}$ to a matching point within the example function. Using these definitions, we can define the *similarity error density* for the location $\mathbf{x}$ and a current

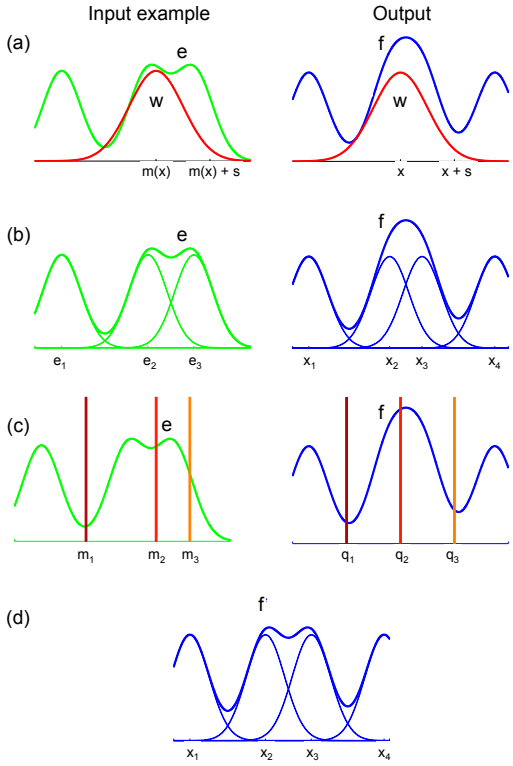

Figure 2: (a) The example and output functions, (b) their discrete representation as a sum of Gaussians, (c) the quadrature points $\mathbf{q}_k$ and their matching points $\mathbf{m}_k$, (d) the final matched output function, which now has the same shape as the example function $\mathbf{e}$ in the region defined by the window function $\mathbf{w}$.

matching $\mathbf{m}(\mathbf{x})$ as

$$S(\mathbf{f}(\mathbf{x}),\mathbf{e}(\mathbf{m}(\mathbf{x}))) = \int_{\mathbb{R}^n} |\mathbf{f}(\mathbf{x}+\mathbf{s}) - \mathbf{e}(\mathbf{m}(\mathbf{x})+\mathbf{s})|^2 \, w(\mathbf{s}) \mathbf{ds}. \tag{2}$$

The size and shape of the window function w characterize the actual matching. Large support sizes demand for large scale structures to match well, while smaller sizes only require small scale details to match.

Similarity Error The *total similarity error* is then given as

$$T = \int_{\Omega} S(\mathbf{f}(\mathbf{x}),\mathbf{e}(\mathbf{m}(\mathbf{x}))) \mathbf{dx}, \tag{3}$$

where Ω is a subspace in the output. The idea is now to minimize this total error alternatively for $\mathbf{m}(\mathbf{x})$ by finding the best matching neighborhoods for the current solution $\mathbf{f}(\mathbf{x})$, and the output function $\mathbf{f}(\mathbf{x})$ by finding the most similar output to the example, given the matching. Hence, we follow the common idea of neighborhood matching based texture synthesis methods, but reformulate it in a general way to handle different types of structures in the same framework.

Once the structures in the example and output are converted into their smooth functional representations, the measure T gives a robust two-way matching.

4.3. Discrete Similarity Measure

Similarity Error Density In order to turn this continuous minimization problem into a numerically treatable form, we need to discretize the functions as well as the integrals involved. We achieve this by a general geometry representation, and a combination of analytic and numerical integration with meshless methods. Both output and example geometries, $\{(\mathbf{x}_i,\mathbf{a}_i), i = 1..n\}$ and $\{(\mathbf{e}_i,\mathbf{b}_i), i = 1..m\}$ respectively, are transformed into corresponding continuous functions as

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= \sum_i \mathbf{a}_i g(\mathbf{x}-\mathbf{x}_i,\sigma) \\ \mathbf{e}(\mathbf{x}) &= \sum_i \mathbf{b}_i g(\mathbf{x}-\mathbf{e}_i,\sigma), \end{aligned} \tag{4}$$

that is, we place Gaussians $g(\mathbf{x},\sigma) = e^{-|\mathbf{x}|^2/\sigma^2}$ at all point locations that 'smear' the point attributes into their neighborhood (Figure 2, b). Switching to a sum-of-Gaussians representation for the point data allows us to encode both their spatial configuration and attributes into the shape of continuous functions and to use the presented similarity measures for continuous functions. As shown in the supplementary material, the similarity error density (2) can be analytically evaluated by representing the window function $\mathbf{w}$ with a Gaussian or a box function. If the latter is chosen, the evaluation results in the *discrete error density measure*

$$\begin{aligned} &S(\mathbf{f}(\mathbf{x}),\mathbf{e}(\mathbf{m}(\mathbf{x}))) = \\ &\textstyle\sum_{ij}(\mathbf{a}_i\cdot\mathbf{a}_j) g(\mathbf{x}_i-\mathbf{x}_j,\sqrt{2}\sigma) \\ &-2\textstyle\sum_{ij}(\mathbf{a}_i\cdot\mathbf{b}_j) g((\mathbf{x}_i-\mathbf{x})-(\mathbf{e}_j-\mathbf{m}(\mathbf{x})),\sqrt{2}\sigma) \\ &+\textstyle\sum_{ij}(\mathbf{b}_i\cdot\mathbf{b}_j) g(\mathbf{e}_i-\mathbf{e}_j,\sqrt{2}\sigma), \end{aligned} \tag{5}$$

where the points $\mathbf{e}_i$ and $\mathbf{x}_i$ are in the neighborhood defined by the window function, for the example and output domains, respectively, and the equality is up to a constant. Hence, the functions $\mathbf{f}$ and $\mathbf{e}$ are replaced by their representations with the sets of point locations and attributes. The resulting discrete similarity measure for the case when a Gaussian of width δ is chosen for representing the window function, i.e., $w(\mathbf{s}) = g(\mathbf{s},\delta)$, is shown in the supplementary material. For numerical efficiency, we usually truncate Gaussians below a given threshold value, making similarity density measure local. Thus, we only consider the points that are 3δ distance apart from $\mathbf{x}$ in the output, and $\mathbf{m}(\mathbf{x})$ in the input example.

Similarity Error This discrete similarity density can now be used to measure the total similarity T between a synthesized point set and the example point set as defined in Equation 3. This requires computing another integral over all points in the output domain. Note that the first and last

terms in Equation 5 do not depend on $\mathbf{x}$. However, the second term involves the mapping function $\mathbf{m}(\mathbf{x})$ that assigns each point $\mathbf{x} \in \Omega$ to the point in the input example domain with the matching neighborhood. This makes it impossible to take the final integral analytically. Hence, we resort to a numerical scheme where the output domain Ω is sampled with a regular grid. Since the integrand is a sum of shifted Gaussians of standard deviation σ, it has a fixed effective bandwidth, allowing us to use an optimal spacing between the grid points.

Let $\mathbf{q}_k$ denote the background integration points, all having a constant integration domain associated (Figure 2, c, right). Furthermore, let $\mathbf{m}_k = \mathbf{m}(\mathbf{q}_k)$ the associated best matching location for the quadrature points in the example domain (Figure 2, c, left). This then leads to the *discrete similarity error*

$$T = \sum_k S(\mathbf{f}(\mathbf{q}_k), \mathbf{e}(\mathbf{m}_k)), \tag{6}$$

which compares the geometric neighborhood structure in the output around each quadrature point $\mathbf{q}_k$ to the best matching corresponding neighborhood in the example set.

4.4. Discussion

The proposed similarity error T measures how well each neighborhood in the output matches to its neighborhood in the input example. Thus, it follows the patch based texture synthesis approaches that rely on a Markov Random Field model [KEBK05]. Similar to these raster based approaches, it conceptually compares functions defined in the whole domain. However, the free parameters are no longer only the functional values, but also the point locations themselves. Similar to meshless surface reconstruction methods, the functions are adaptively represented such that only relevant structures are sampled. Representing structures with point samples gives us full flexibility in handling structures of different kinds and their combinations. This is contrast with methods that rely on certain assumptions on what the point samples represent and design the matching metrics accordingly [LGH13, LBW*14, XCW14].

Neighborhood matching metric We thus share the generality of point sampling based discrete element texture synthesis methods [MWT11, MWLT13]. In contrast to these methods, however, we do not directly match difference vectors in the neighborhoods. Defining matching metrics directly on points is challenging, as the matching scores critically depend on the sampling in that case. Indeed, previous works [MWT11, MWLT13] have a one-way matching score, where each vector $\mathbf{x}_j - \mathbf{x}_i$ for $\mathbf{x}_j$ in the neighborhood of $\mathbf{x}_i$ in the output is matched to the most similar vector in the input example. This implies that for output point sets where the points do not completely represent the desired structures, the matching energy will still be low, as illustrated in Figure 3. In contrast, our energy utilizes a two-way matching, leading to lower energy values only when the neighborhoods in the output and input are structurally similar (Figure 3).

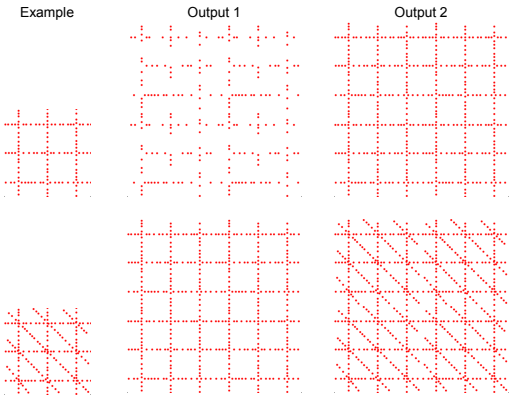

Figure 3: We show two different outputs for each input example. For both outputs 1 and 2, the neighborhood matching energy defined by Ma et al. [MWT11, MWLT13] stays the same due to the one-way matching the pairs difference vectors from the output to those in the input example. In contrast, our method has 2 times higher energy for the wrong output 1 for each example.

The sampling dependency of previous methods [MWT11, MWLT13] also means that the neighborhoods of only sample points are matched. Thus, depending on the sampling, some neighborhoods might be left unmatched, if they are covered with less samples. In contrast, we separate the structure representation (sample points $\mathbf{x}_i$ and $\mathbf{e}_i$), and neighborhood matching (quadrature points $\mathbf{q}_k$), enforcing that all neighborhoods are matched equally well, regardless of the sampling. With the previous methods, the $\mathbf{q}_k$ and $\mathbf{m}_k$ are constrained to be among $\mathbf{x}_i$ and $\mathbf{e}_i$, respectively. We will see in Section 7 that this results in better preservation of local structures, which is very important especially when continuous structures in general domains are to be synthesized.

Attributes The representation of output and example functions in Equation 4 is different from previous attribute representations, where they are treated as additional point locations. Treating attributes as scaling factors for the Gaussians significantly reduces the dimensionality of the matching problem, especially when the attributed $\mathbf{a}_i$ live in high dimensional spaces. Since we regularly sample the output domain, this is an important consideration. Furthermore, we do not need to consider relative scaling between the spatial domain and the attributes, as in the previous works [MWT11, MWLT13]. If the attributes are discrete, we snap to the closest discrete value after each optimization step, as described in Section 5. Equation 4 implies that putting two points at the sample location, or doubling an attribute (assuming it is scalar) at that location results in the same representation. As we elaborate in the next section, we avoid such cases by carefully controlling the density of the points, and optimizing for the attributes and the point location separately.

Smooth approximation The smooth representation with

Algorithm 1: Multi-scale Local-Global Solver

initialize $\mathbf{x}_i$
initialize window size δ
loop *multiScaleIterations* **times**
  loop *samplingControlIterations* **times**
    loop *localGlobalIterations* **times**
      matching: $\{\mathbf{m}_i\} \leftarrow \arg\min_{\{\mathbf{m}_i\}} T$
      merging: $\{\mathbf{x}_i\}, \{\mathbf{a}_i\} \leftarrow \arg\min_{\{\mathbf{x}_i\},\{\mathbf{a}_i\}} T$
    end
    sampling control: add/remove $\mathbf{x}_i$ based on $P(\cdot)$
  end
  $\delta \leftarrow \alpha \cdot \delta$ where $\alpha \in [0,1]$
end

sums of Gaussians (Equation 4) allows us to compute analytic integrals and derivatives, which are essential for efficient and accurate synthesis via well-established optimization methods, as we will illustrate in the next sections.

5. Structure Synthesis

The smooth representation of the structures allows us to formulate a robust optimization procedure with standard optimization methods. Our optimization follows the same basic iterative steps of previous neighborhood-based texture synthesis methods: a matching step for the neighborhoods, and a merging step that computes the positions of the sample points based on the matched neighborhoods. However, unlike previous methods, the matching measure is decoupled from the sampling points that represent the structures, i.e., we have the quadrature points $\mathbf{q}_k$ to measure how well the neighborhoods match. This distinction allows us to robustly handle the matching step, regardless of how the structures are sampled or represented.

5.1. Multi-scale Local-Global Solver

To synthesize the output point set, we minimize the total error T defined in Equation 6 with respect to the point locations $\mathbf{x}_i$ and attributes $\mathbf{a}_i$. We perform this optimization with an alternating approach as outlined in Algorithm 1, where each step is guaranteed to decrease the similarity error T.

Local Step: Matching We first compute the best match for each quadrature point neighborhood by finding the matching $\mathbf{m}_k$ for each quadrature point $\mathbf{q}_k$. A notable property of this approach is that the matching point $\mathbf{m}_k$ for a given $\mathbf{q}_k$ can be optimized independently of the other quadrature points since $\partial T/\partial \mathbf{m}_k = \partial S/\partial \mathbf{m}_k$. This makes this local step of the problem highly parallelizable. We employ a simple gradient descent procedure to find the best matching locations. The necessary gradients $\partial S/\partial \mathbf{m}_k$ are given in the supplementary material. This optimization is prone to get stuck in a local minimum and not finding the best match. Therefore we run this optimization with five different random initial positions for $\mathbf{m}_k$ and choose the best match. The repetitive pattern of the input exemplar guarantees the existence of many good local minima, thus we found five seeds to be sufficient for the optimization to succeed.

Global Step: Merging After having found best matches for all quadrature points, we adapt the output point set structure to be as similar as possible to these local neighborhoods as demanded by the similarity metric, i.e., we minimize T for all output points $\mathbf{x}_i$ and attributes $\mathbf{a}_i$. This is a globally coupled nonlinear optimization problem. We again employ a gradient descent procedure where we sequentially optimize the points and attributes, one after the other, in a Gauss-Seidel manner. Optimizing first for points $\mathbf{x}_i$, combined with a sampling control stage as explained below, prevents points from clustering to compensate for the difference in attributes. The required gradients $\partial T/\partial \mathbf{x}_i$ and $\partial T/\partial \mathbf{a}_i$ are provided in the supplementary material.

Choosing such a simple optimization scheme allows us to parallelize this step due to the local support of the window function w, and to have a consistent decrease in the energy at each step. The non-convexity of our similarity error function leads to the presence of multiple local minima. For the goal of geometry synthesis this is expected, as there are multiple configurations exhibiting the desired structures. It is in fact an advantage: having different regions ending up in different local minima increases the diversity among generated configurations.

Multi-scale Optimization Once the optimization converges sufficiently for a given window size, the best concense has been found for this feature size. However, this compromise can result in geometries where local features are not close to example features. In order to fix the smaller scale structures, we apply a multi-scale optimization where we decrease the window size starting from the initially provided window size, such that the algorithm can continue the descent to improve the local small scale details. By going

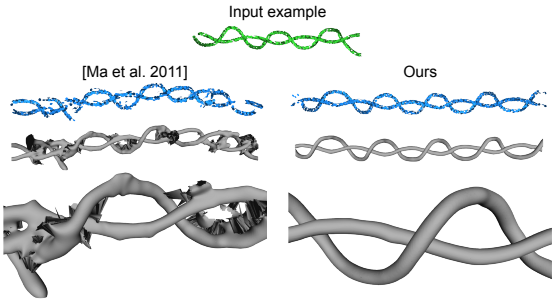

Figure 5: Given the input example with points and normals (top), outputs point sets are synthesized (second row) and reconstructed (third and fourth row) using moving least squares surfaces [OGG09]. Our representation and synthesis algorithm accurately handles such continuous structures.

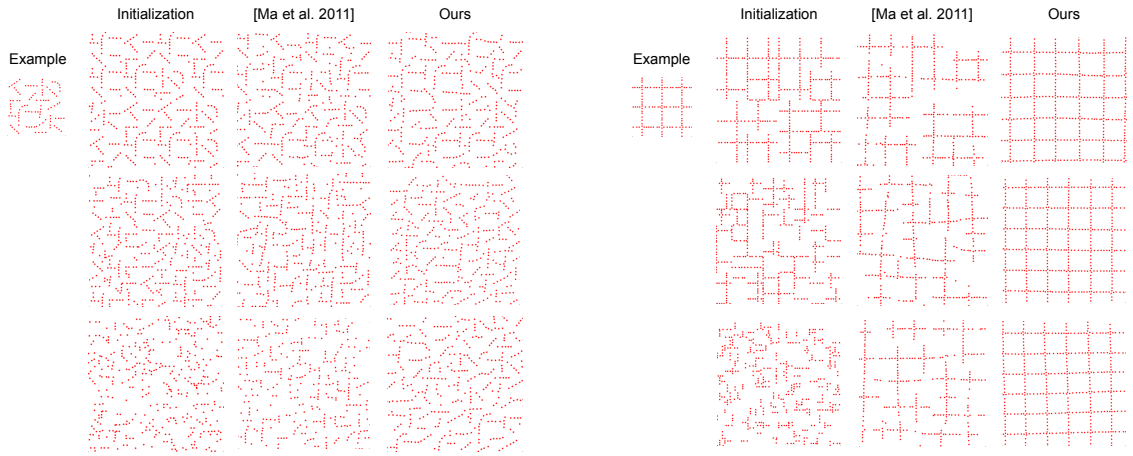

Figure 4: Different initializations computed by copying patches of different sizes from the example to the output are used for each synthesis result. For all initializations, our method can generate accurate discrete (left) as well as continuous (right) structures. The previous methods require copying of considerably larger patches to initialize the synthesis, resulting in less randomness and a patchy look as illustrated in the left-topmost synthesis result. When continuous structures are present (right), even such large patches are not sufficient to generate accurate structures.

from larger to smaller scales, we make sure that the larger structures, which are harder to reproduce, are matched first. The optimization then continues with refinements in smaller scales.

Sampling Control So far we assumed that the regions in the output domain always contain the optimal amount of points regarding the similarity measure, i.e., that there is no mismatch in the number of points and that errors only occur from non-optimal point locations and attributes. However, too many or too few points in a region impair our similarity measure, as for all point sample based texture synthesis methods. If there are not enough points, the structures in the example will be partially reproduced. Conversely, too many points will result in excess points that force the optimization to destroy structures to accommodate the extra points. In order to prevent this, we employ a sampling control strategy during the optimization, based on the same principle of our similarity error density measure.

In order to detect the deviation in the sampling density around a quadrature point we define a *sampling error density* function

$$P(\mathbf{x},\mathbf{m}(\mathbf{x})) = \int (f(\mathbf{x}+\mathbf{s}) - e(\mathbf{m}(\mathbf{x})+\mathbf{s}))\, w(\mathbf{s})\mathbf{ds} \qquad (7)$$

that measures the *signed* difference between the output and example functions. However, here $f(\cdot)$ and $e(\cdot)$ are special instances of the two functions where $\mathbf{a}_i = \mathbf{b}_i = 1$ such that solely point locations are taken into account. If $P(\mathbf{x},\mathbf{m}(\mathbf{x})) < 0$ for a given point $\mathbf{x}$, the function e in the example domain is on average larger than the output f, implying removal of points. Similarly, $P(\mathbf{x},\mathbf{m}(\mathbf{x})) > 0$ calls for adding a point to match the functions f and e. For robustness, we introduce a threshold $\varepsilon = 10^{-6}$ and add a new point in vicinity of the quadrature point if $P < -\varepsilon$, or remove an unnecessary point if $P > \varepsilon$. Once a new random point is added, we optimize for its location and attribute, and decide to keep the optimized point or not by checking whether the energy increases or decreases. The same check is performed for the case of removing a point.

5.2. Discussion

Robustness to initialization Our two-way neighborhood matching, as discussed in Section 4.4, combined with the optimization algorithm presented in the last section makes our method robust to initializations, as compared to the previous methods that critically depend on the initial distribution of points in the output [MWT11, MWLT13]. We illustrate the robustness to initialization in Figure 4. It is especially important for synthesizing continuous structures (Figure 4, right), where even a single neighborhood mismatch can lead to visually disturbing reconstructions (Figure 5).

Control sampling Dynamically controlling the sampling by adding and removing points in the optimization is a key component for interactive synthesis, where the user continuously extends the region he wants to texture by brushing (Section 6.2). This is in contrast with the previous offline discrete element texture synthesis methods. Adding/removing points for interactive synthesis has recently been explored for 2D drawing applications [XCW14], and proposed as an optimization method for shape processes via MCMC [LGH13]. In contrast to these approaches, our technique offers a general unified adding/removing strategy that is interleaved with the optimization steps that move the points for accurate place-

ment. Such accurate placement takes considerably more time to obtain by merely adding entities [LGH13].

6. Controlling Structures

We control the synthesized structures in a two-scale approach, where the small scale geometry is given by the example, and the large scale behavior is controlled by the user, similar to the existing methods [MWT11, MWLT13]. The main strength of our structure definition and synthesis method is that we can accurately handle arbitrary structures represented in various forms. This turns our method into a powerful tool for users to create complex general output structures interactively, in contrast to the previous methods offline [MWT11, MWLT13], or interactive [KIZD12a, XCW14] methods. In this section, we first show how the generic definition can be leveraged to generate and control various types of output representations by simply changing what the point locations and attributes represent. We then show how orientation and scaling fields can be used to steer the structure synthesis by specifying geometries of different dimensions.

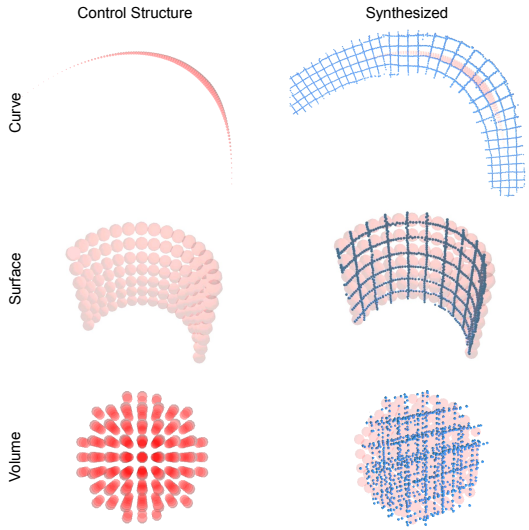

Figure 6: The guiding fields can be defined along curves, surfaces, or volumes to control the large-scale geometry.

6.1. Structure Representations

Our point samples can represent a variety of structures sparsely, ranging from discrete to continuous. This allows us to handle different structures, and their mixtures, in the same framework.

Continuous Structures Continuous geometry can be achieved by storing scalar and/or vectorial point attributes. A scalar value, such as a point color or radius, can be assigned and used for texturing and rendering, i.e., to extract an isosurface. Equipping each particle with a vector-valued surface normal attribute further allows to employ recent meshless surface reconstruction techniques [OGG09] within our framework to synthesize high-quality surfaces.

Discrete Elements A discrete element can be represented by one or more samples in our framework. Representation with a single sample works well for cases where collisions are not a problem, such as the leaves in Figure 1. However, for more complex structures, we should have a sufficient sampling [MWT11, LGH13]. We represent such discrete elements with multiple points, often sampled on the surface of the element, and sometimes equipped with normals. Points belonging to a single element are added, removed, and moved together in the optimization, by treating them as a single point, i.e. by setting $\mathbf{x}_i = \mathbf{x}_0$ for all points $\mathbf{x}_i$ in the discrete structure, and optimizing with respect to $\mathbf{x}_0$. Note that the optimization procedure stays exactly the same as for the continuous structures.

Mixtures of Structures The common definition for continuous and discrete structures allows us to handle their mixtures seamlessly, since both are represented and optimized in the same way. An example synthesis results with structure mixtures containing discrete elements such as the organic structure and the discrete gems is illustrated in Figure 16. We can also use mixtures of discrete and continuous attributes if the structures require such a representation. As an example, we used a grouping attribute for the points to synthesize the structure in Figure 12, such that each bean is assigned a different grouping number. Note that this does not prevent the beans to exhibit random variations, as they are still represented as continuous structures with surface points and normals.

6.2. Large-scale Control

The large scale geometry is intuitively controlled by the user as orientation and scaling fields are defined along curves, triangular surfaces, or bounded volumes. We first discuss how our system can be extended to handle rotation and scale, and then explain how these guiding geometries are defined.

Rotation and Scale We extend the optimization to account for orientation and scale of the output geometry by adding a rotation $\mathbf{R}(\mathbf{x})$ and scaling factor $s(\mathbf{x})$ to the similarity error density shown in Equation 2, resulting in

$$S = \int |\mathbf{f}(\mathbf{x}+\mathbf{s}) - \mathbf{e}(s(\mathbf{x})\mathbf{R}(\mathbf{x})(\mathbf{m}(\mathbf{x})+\mathbf{s}))|^2 \, w(\mathbf{s})\mathbf{ds}. \quad (8)$$

Note that since $\mathbf{R}$ and s define fixed fields independent of the synthesized function $\mathbf{f}$, no modification to the derivatives and hence the optimization procedure is required. As implied by the expression, in practice, we implement this optimization by rotating and scaling the example $\mathbf{e}$ for each point $\mathbf{x}$. Note that in the discretized energy, $\mathbf{x}$ is represented by the quadrature points, and hence these fields are also stored at the quadrature points. Once they are stored, we can then run the

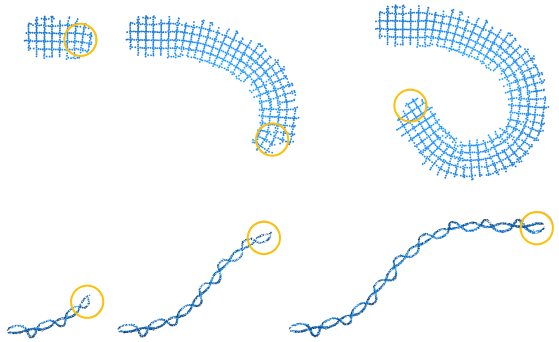

Figure 7: The structures grow in real-time as the user interactively brushes (orange circle indicate brushing region).

same optimization. The fields $\mathbf{R}$ and s depend on the design metaphor utilized as discussed next.

Brush strokes With our system we can generate new geometries by drawing lines in 3D space. Quadrature points are automatically generated along and near the drawn lines. The drawing direction can be exploited to automatically set the rotation $\mathbf{R}$ of each quadrature point, such that the direction of the input example is aligned with the stroke tangent. The scaling factor s is manually set by the user as she/he draws the strokes. The user can also control the brush size, which determines the region to be textured. Examples of this type of control is given in Figures 6, top, 7, and the accompanying video. The brush direction defines the orientation, and the varying scaling factors smoothly change the pitch.

Triangular surfaces The guiding fields can also be defined on surfaces. As an example, we used the principle curvature directions as an orientation field in Figure 6, middle. Hence, we can add fine-scale details to existing surface geometries with our technique, as we further illustrate in the next section.

Volumetric synthesis This naturally extends to 2D and 3D volumes, where the fields are defined throughout the ambient space. The user can also utilize a 3D surface to shape the output structure as shown in Figure 6, bottom, and the chair example in Figure 1. The quadrature points are sampled inside the provided volume to avoid unnecessary computations.

7. Results and Discussion

7.1. Implementation and Parameters

For our representation, σ gives the standard deviation of the Gaussians used in defining the functions $\mathbf{f}$. The parameter σ determines the smoothness of the matched functions, which in turn depends on the sampling of the structures. If the structures are sampled densely in the input example, we can set it to lower values to capture smaller scale details. Thus, in our implementation, it is set to the average spacing between the points $\mathbf{e}_i$ in the example. Once σ is set, the optimum spacing of the quadrature points $\mathbf{q}_k$ in the output can be analytically determined as the Gaussians are band-limited. The parameter δ determines the spatial extent of the window function w. The neighborhood size and hence the δ is provided by the user based on the expected scale of repetitions.

In the synthesis algorithm (Algorithm 1), the optimization by gradient descent is run till the average movement of the points $\mathbf{x}_i$ is below a threshold $\varepsilon = 10^{-6}$ or the maximum number of iterations (we used 30) is reached. The step size for the gradient descent is set to 0.03. After this optimization, each quadrature point is checked for the condition on P as defined in Section 5.1, and a point is removed or added accordingly. These steps are repeated for a new shrunken neighborhood size by setting $\alpha = 0.9$. Once the new neighborhood size is half of the original size, the algorithm is stopped. We use a kd-tree to speed up neighborhood queries.

7.2. Analysis and Comparisons

Our new matching metric combined with the synthesis algorithm with sampling control leads to accurate reproduction of repeated patterns, independent of the initializations and complexity of the structures (please see Sections 4.4 and 5.2 for discussions on these properties). We illustrate robustness to initial conditions in Figures 4 and 5. As illustrated in Figure 4, top row, utilizing the initialization strategy proposed by Ma et al. [MWT11] leads to preservation of structures when continuity is not essential, although the resulting synthesized point sets look quite similar to the initializations with visible patches from the input example. However, continuous structures cannot be accurately synthesized with this technique [MWT11], as shown in Figure 4, top row, right. Furthermore, even small changes in the initial point sets can lead to convergence to bad local minima, resulting in distorted structures with the method of Ma et al. [MWT11] (Figure 4, second and third rows). In contrast, our algorithm is not affected by the initialization. This property is especially important if the resulting point sets are used for reconstructing smooth surfaces, as shown in Figure 5.

Contrary to Ma et al. [MWT11], our smooth formulation does not require solving a linear system at each iteration. We can thus achieve better performance which also allows for interactive results, and use input examples with a large number of points (usually more than 1000), which is essential for representing continuous structures. Figure 8 shows the convergence behavior for a point set of 1000 points for the input examples shown in Figure 7, top (2D), and Figure 18, bottom (3D). Table 1 shows the run time for the results in Figures 7 (bottom), 18 (bottom) and 1 (center), achieved with our method and with the technique of Ma et al. [MWT11]. An interactive brush was used for the helix example in Figure 7, thus the presented run time refers to the points within the brush (around 100). We run our method until convergence, and the one by Ma et al. [MWT11] for 10 iterations, as in the

Example	#input points	#output points	Run time Us	Run time Ma et al.
helix	600	100	4 s	12 s
sand	1600	7200	2 min	9 min
chair	1100	23000	6 min	40 min

Table 1: Run times for three different results.

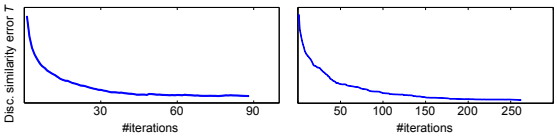

Figure 8: Discrete similarity error T as a function of the number of iterations for the optimization of 1000 points in the output. On the left, the 2D grid input in Figure 7, on the right, the 3D beach input in Figure 18, bottom is analyzed.

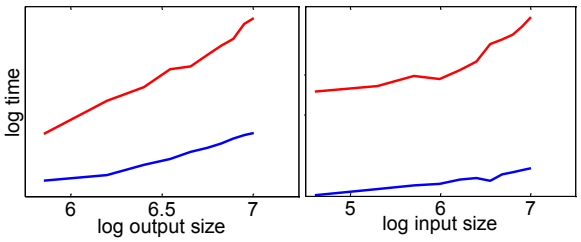

Figure 9: Run time for one iteration for our method (blue) and Ma et al. [MWT11] (red). On the left image, the input exemplar contains 100 points, and the output structure varies from 300 to 3000 points. On the right, the output structure contains 1000 points, and the input exemplar varies from 100 to 1000.

mentioned work. The neighborhood size in these three examples was about one fourth of the size of the input. Moreover, in Figure 9 we present the time required to run one iteration for the two methods, with different input and output sizes. One can notice that our method scales better with respect to the input and output size, due to our simpler gradient descent based optimization. The results were tested on a PC with an Intel i7-3770K CPU.

The neighborhood size, provided by the user, offers control over the scale of expected repetitions in the input and output. We illustrate its effect on the synthesis results in Figure 10. A bigger size can be used to preserve large scale structures, while a smaller size leaves room for more randomness.

Our technique allows for interactive brushing in general domains, and synthesizing mixtures of continuous and discrete structures. We provide examples of such results in the next sections.

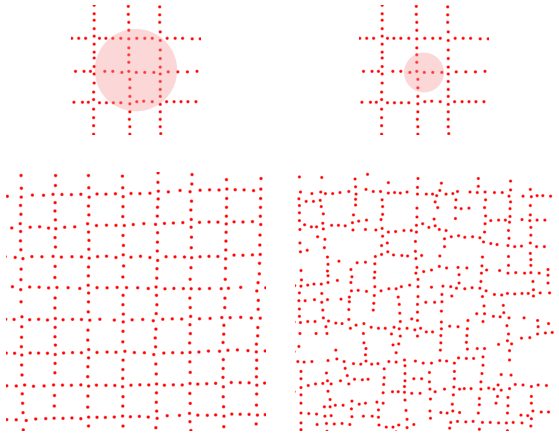

Figure 10: Influence of the neighborhood size in the input example (top, red circle) on the synthesized result (bottom).

7.3. Synthesis Examples

We applied the technique to synthesize various structures ranging from discrete to continuous in different domains including curves, surfaces, and volumes. Some of the controlling domains are sketched interactively (please also see the accompanying video). For all the continuous structures in 3D, except for the one in Figure 13, we used points with normals sampled on the surface in the examples, synthesized the output point cloud with normals, and reconstructed the results with a moving least squares based surface reconstruction method [OGG09]. For the example shown in Figure 13, we reconstructed the isosurface by placing a metaball (blob) at each point location. Locations and sizes of the metaball are optimized according to the input. For discrete structures in 3D, we simply kept the original geometry representation, e.g. if the points are vertices of a mesh in the input example, the same mesh also appears in the output. For some of the results, we also included colors as attributes, as depicted in the corresponding figures.

In Figure 1, models of wooden sticks (discrete elements) have been placed on the hut surface shown in red.

The output structures representing the intertwined helix in Figure 11 and the chain in Figure 12 have been interactively generated along curves. The user draws a curve and specifies orientation and scaling fields along it, and the points are then synthesized accordingly. For smaller input examples (less than about 500 points), the output can be synthesized in real-time with a brushing interface, as in Figure 7 for the 2D grid and the 3D helix. The continuous surfaces have been reconstructed offline using moving least squares. Synthesiz-

ing continuous surfaces onto a triangular surface is shown in Figure 14. The input surface, shown on the left in red, defines the global shape of the structure, i.e., the deep-water waves. With our method we can synthesize small-scale capillary waves onto the surface, improving the realism of the water surface. In the same way, bumps have been generated on the stone structure in Figure 17. An example of using bounded volumes is presented in Figure 1. A 3D model of a chair is included, as shown on the right in red, defining the global shape. The chair is sampled by quadrature points during initialization, and then the volume is synthesized by an organic structure. In a post-processing step, the borders of the reconstructed output structure are cut using the same model of the chair. The structure in Figure 13 has been generated using metaballs of different sizes. Branches at the extremities are thinner, thus generated from smaller metaballs than the ones close to the main branch. Like in the case for moving least squares, our control sampling strategy is essential to reconstruct continuous, connected structures from the point clouds.

As our method generalizes previous approaches, we can simply combine discrete and continuous attributes. We present several examples to illustrate such mixed structures. The ivy example presented in Figure 1 shows a continuous ivy structure that is synthesized along a curve, similarly to the intertwined helix (Figure 11). The leaves and flowers are discrete elements represented as proxy samples in the input point set. Other examples using mixed structures are shown for a stone wall, a decorative object with gems, and a beach with shells in Figures 15, 16 and 18, respectively. In these cases, around 20 points have been sampled on the surface of each discrete element. In addition, the beach example shows how using different sand patterns as input affects the generated continuous surface.

7.4. Limitations

The structure representation and thus the synthesis algorithm depends on a smooth approximation with Gaussians. As described in Section 7.1, we determine a global standard deviation given by σ for the Gaussians based on point spacing, to accurately capture the structures. This was sufficient for reproducing a variety of structures as presented. However, an adaptive σ can further improve the results, especially when there are far apart tight clusters of points in the input example.

Similar to the previous methods, we utilize a neighborhood size to capture the repetitions in the input examples. As shown in Figure 10, this size is an example-dependent parameter and thus should be provided by the user for artistic control, reflecting his/her view of the scale of repetitions in the example. However, if there are repetitions of different scales in the same example, setting a single size can be detrimental for the preservation of the repetitive structures at the other scales. In particular, we encountered this problem only in some of the discrete-continuous mixtures we experimented

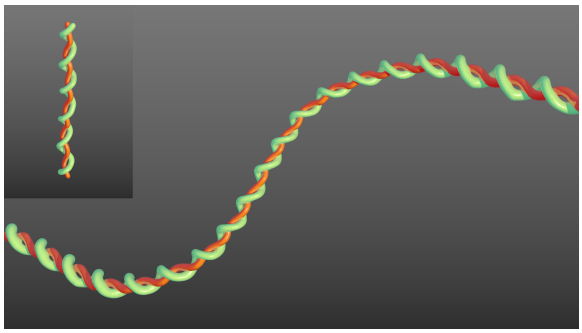

Figure 11: The spiral is synthesized by specifying orientation and scaling fields along a curve.

Figure 12: The chain is synthesized with surface normals as attributes, and reconstructed with a smooth surface definition.

with, where samples representing discrete elements have a considerably larger scale of repetitions than the ones composing continuous structures. As illustrated in Figure 18, top, the arrangement of the sea shells (each of which is regarded as a discrete element) is not fully preserved, since the scale of the repetition of that pattern is much larger than that of the continuous background. On the other hand, purely continuous or discrete exemplars are not affected by this limitation, as these structures usually exhibit repetitions of similar scales, allowing us to use our approach for further applications, such as painting [KIZD12b].

Moreover, like in the other neighborhood-based texture synthesis methods, our exemplars are required to have a repetitive enough pattern. This allows our matching optimization with multiple initial positions to avoid bad local minima, which would distort the synthesized structures.

8. Conclusion

We presented a new method for synthesizing general repeated structures given by an input example. In contrast to previous geometric texture synthesis methods, our technique offers treating continuous and discrete structures residing on general domains in the same framework, exhibits better and initial-

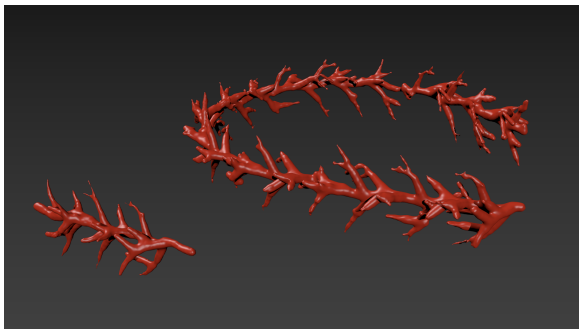

Figure 13: Metaballs of different sizes are used to reconstruct a continuous surface along a curve.

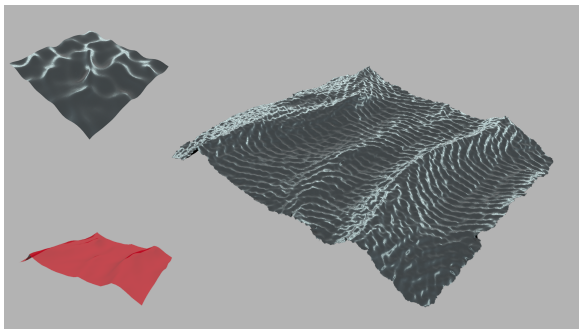

Figure 14: Capillary waves synthesized onto coarse mesh defining the deep water waves.

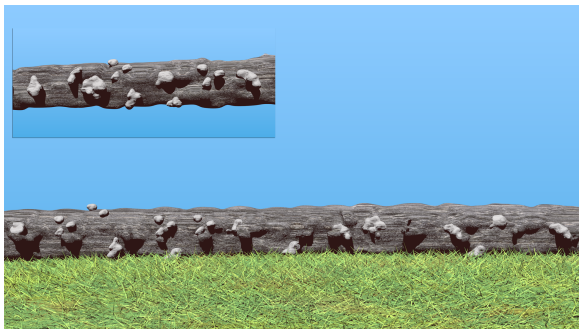

Figure 15: The mixture of the discrete elements of stones of lighter color and continuous background wall are synthesized.

ization independent preservation of structures, and allows for interactive repetitive synthesis of general structures.

Future work

We presented only a few applications of our method in the scope of this paper. Our method can be utilized in all applications where repetitive structures are utilized for synthesizing missing or augmenting existing structures. Some immediate applications are surface and image reconstruction, completion, consolidation, inpainting, or superresolution. The technique can be combined with physical or fabrication constraints for simulations or fabrication oriented design. By adding a time dimension, similar to previous methods [MWLT13], it can also be used for animation generation, and in combination with physical simulations.

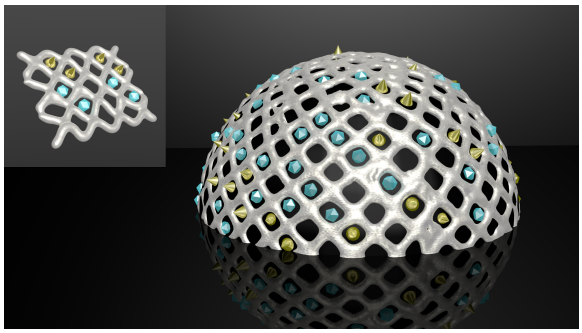

Figure 16: Another example of synthesizing mixtures of discrete elements (gems) with continuous structures.

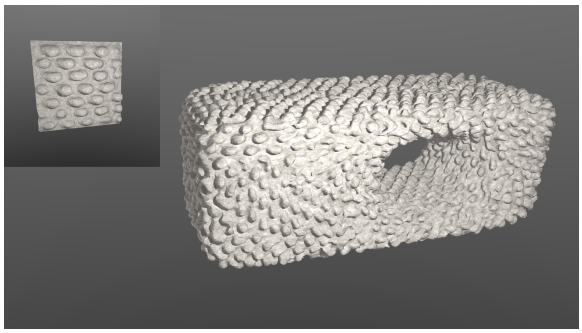

Figure 17: Our method can also be used to synthesize continuous structures on surfaces.

The technique is well-suited for interactive authoring of such difficult structures. We utilized a simple stroke drawing interface for real-time results. It can be extended with more sophisticated and possibly application dependent metaphors for interactive synthesis.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *Proceedings of Visualization* (2001), VIS '01, pp. 21–28. 2

[AdPWS10] ALVES DOS PASSOS V., WALTER M., SOUSA M.: Sample-based synthesis of illustrative patterns. In *Computer Graphics and Applications* (2010), PG '10, pp. 109–116. 2

[AKA13] ALMERAJ Z., KAPLAN C. S., ASENTE P.: Patch-based geometric texture synthesis. In *Proceedings of the Symposium on Computational Aesthetics* (2013), CAE '13, pp. 15–19. 2

[BBT*06] BARLA P., BRESLAV S., THOLLOT J., SILLION F., MARKOSIAN L.: Stroke pattern analysis and synthesis. In *Computer Graphics Forum (Proc. Eurographics)* (2006), vol. 25, pp. 663–671. 2

[DHM13] DU S.-P., HU S.-M., MARTIN R. R.: Semiregular solid texturing from 2d image exemplars. *IEEE Trans. Vis. Comput. Graph. 19*, 3 (2013), 460–469. 2

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision* (1999), vol. 2 of *ICCV '99*, pp. 1033–1038. 2

[HLT*09] HURTUT T., LANDES P.-E., THOLLOT J., GOUSSEAU Y., DROUILLHET R., COEURJOLLY J.-F.: Appearance-guided synthesis of element arrangements by example. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (2009), NPAR '09, pp. 51–60. 2

[HSD13] HECK D., SCHLÖMER T., DEUSSEN O.: Blue noise sampling with controlled aliasing. *ACM Transactions on Graphics 32*, 3 (2013), 25:1–25:12. 2

[HWFL14] HUANG Z., WANG J., FU H., LAU R. W. H.: Structured mechanical collage. *IEEE Trans. Vis. Comput. Graph. 20*, 7 (2014), 1076–1082. 2

[IMIM08] IJIRI T., MECH R., IGARASHI T., MILLER G. S. P.: An example-based procedural system for element arrangement. *Computer Graphics Forum 27*, 2 (2008), 429–436. 2

[KCG*14] KAZI R. H., CHEVALIER F., GROSSMAN T., ZHAO S., FITZMAURICE G.: Draco: Bringing life to illustrations with kinetic textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2014), CHI '14, ACM, pp. 351–360. 2

[KEBK05] KWATRA V., ESSA I., BOBICK A., KWATRA N.: Texture optimization for example-based synthesis. *ACM Trans. Graph. 24*, 3 (July 2005), 795–802. 5

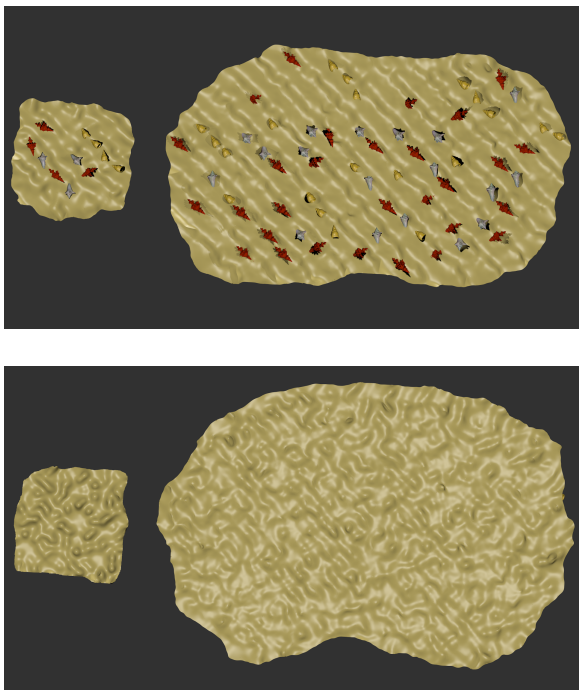

Figure 18: Discrete elements (shells) are placed on a continuous structure (sand) (top). Another sand pattern is used to show how the synthesis is affected (bottom).

[KFCO*07] KOPF J., FU C.-W., COHEN-OR D., DEUSSEN O., LISCHINSKI D., WONG T.-T.: Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3 (2007), 2:1–2:9. 2

[KIZD12a] KAZI R. H., IGARASHI T., ZHAO S., DAVIS R.: Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2012), CHI '12, ACM, pp. 1727–1736. 2, 8

[KIZD12b] KAZI R. H., IGARASHI T., ZHAO S., DAVIS R.: Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2012), CHI '12, ACM, pp. 1727–1736. 11

[LBDF13] LU J., BARNES C., DIVERDI S., FINKELSTEIN A.: Realbrush: Painting with examples of physical media. *ACM Trans. Graph. 32*, 4 (July 2013), 117:1–117:12. 2

[LBW*14] LU J., BARNES C., WAN C., ASENTE P., MECH R., FINKELSTEIN A.: DecoBrush: Drawing structured decorative patterns by example. In *ACM Transactions on Graphics (Proc. SIGGRAPH)* (Aug. 2014). 2, 3, 5

[LFB*13] LUKÁČ M., FIŠER J., BAZIN J.-C., JAMRIŠKA O., SORKINE-HORNUNG A., SÝKORA D.: Painting by feature: Texture boundaries for example-based image creation. *ACM Trans. Graph. 32*, 4 (July 2013), 116:1–116:8. 2

[LGH13] LANDES P.-E., GALERNE B., HURTUT T.: A shape-aware model for discrete texture synthesis. *Computer Graphics Forum 32*, 4 (2013), 67–76. 1, 2, 3, 5, 7, 8

[LHGM05] LAI Y.-K., HU S.-M., GU D. X., MARTIN R. R.: Geometric texture synthesis and transfer via geometry images. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling* (New York, NY, USA, 2005), SPM '05, ACM, pp. 15–26. 3

[MKB*10] MARTIN S., KAUFMANN P., BOTSCH M., GRINSPUN E., GROSS M.: Unified simulation of elastic rods, shells, and solids. *ACM Transaction on Graphics (Proc. SIGGRAPH) 29*, 3 (2010), 39:1–39:10. 2

[MM08] MERRELL P., MANOCHA D.: Continuous model synthesis. *ACM Transaction on Graphics 27*, 5 (2008). 3

[MWLT13] MA C., WEI L.-Y., LEFEBVRE S., TONG X.: Dynamic element textures. *ACM Transactions on Graphics 32*, 4 (2013), 90:1–90:10. 2, 5, 7, 8, 12

[MWT11] MA C., WEI L.-Y., TONG X.: Discrete element textures. In *ACM Transactions on Graphics (Proc. SIGGRAPH)* (2011), pp. 62:1–62:10. 1, 2, 3, 5, 7, 8, 9, 10

[OG12] ÖZTIRELI A. C., GROSS M.: Analysis and synthesis of point distributions based on pair correlation. *ACM Transaction on Graphics 31*, 6 (2012). 2

[OGG09] ÖZTIRELI A. C., GUENNEBAUD G., GROSS M.: Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum 28*, 2 (2009), 493–501. 6, 8, 10

[PL95] PAGET R., LONGSTA D.: Texture synthesis via a non-parametric markov random field. In *Proceedings of Digital Image Computing: Techniques and Applications* (1995), vol. 1, pp. pp. 547–552. 2

[Tur01] TURK G.: Texture synthesis on surfaces. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques* (2001), SIGGRAPH '01, pp. 347–354. 2

[WLKT09] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.:

State of the art in example-based texture synthesis. In *Eurographics '09 State of the Art Reports (STARs)* (March 2009). 1, 2

[XCW14] XING J., CHEN H.-T., WEI L.-Y.: Autocomplete painting repetitions. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 172:1–172:11. 2, 5, 7, 8

[ZHW*06] ZHOU K., HUANG X., WANG X., TONG Y., DESBRUN M., GUO B., SHUM H.-Y.: Mesh quilting for geometric texture synthesis. *ACM Trans. Graph. 25*, 3 (2006), 690–697. 1, 3

[ZHWW12] ZHOU Y., HUANG H., WEI L.-Y., WANG R.: Point sampling with general noise spectrum. *ACM Transaction on Graphics 31*, 4 (2012), 76:1–76:11. 2

[ZJL14] ZHOU S., JIANG C., LEFEBVRE S.: Topology-constrained synthesis of vector patterns. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 215:1–215:11. 3

[ZLL13] ZHOU S., LASRAM A., LEFEBVRE S.: By-example synthesis of curvilinear structured patterns. *Comput. Graph. Forum 32*, 2 (2013), 355–360. 3

[ZQC*14] ZHU B., QUIGLEY E., CONG M., SOLOMON J., FEDKIW R.: Codimensional surface tension flow on simplicial complexes. *ACM Transaction on Graphics (Proc. SIGGRAPH) 33*, 4 (2014), 111:1–111:11. 3

[ZSTR07] ZHOU H., SUN J., TURK G., REHG J. M.: Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (2007), 834–848. 3