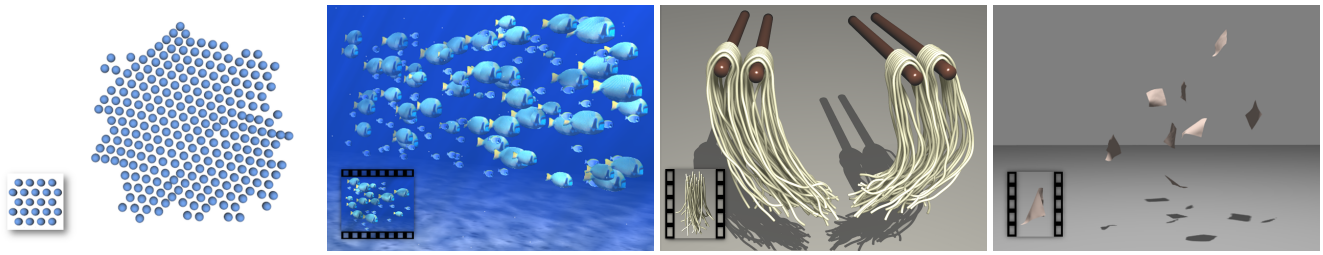
Dynamic Element Textures

Chongyang Ma[†] [*] Univ. British Columbia[†] Tsinghua Univ.[*]
Li-Yi Wei[‡] [‖] Univ. Hong Kong[‡]
Sylvain Lefebvre[§] ALICE/INRIA[§]
Xin Tong[¶] Microsoft Research[‖] Asia[¶]



(a) *particles* **(b)** *herds* **(c)** *threads* **(d)** *sheets*

Figure 1: *Dynamic element textures. Our method synthesizes a variety of repetitive spatial-temporal phenomena, such as particles, threads, and sheets, via a combination of constrained optimization and data driven computation. It offers control at both the coarse scale through spatial-temporal constraints (e.g. overall output shape and movement) and the fine scale through small input exemplars (bottom-left inlet of each image), which may be static (a) or animated (b), (c) and (d). Please refer to the accompanying video for corresponding animations of our results.*

Abstract

Many natural phenomena consist of geometric elements with dynamic motions characterized by small scale repetitions over large scale structures, such as particles, herds, threads, and sheets. Due to their ubiquity, controlling the appearance and behavior of such phenomena is important for a variety of graphics applications. However, such control is often challenging; the repetitive elements are often too numerous for manual edit, while their overall structures are often too versatile for fully automatic computation.

We propose a method that facilitates easy and intuitive controls at both scales: high-level structures through spatial-temporal output constraints (e.g. overall shape and motion of the output domain), and low-level details through small input exemplars (e.g. element arrangements and movements). These controls are suitable for manual specification, while the corresponding geometric and dynamic repetitions are suitable for automatic computation. Our system takes such user controls as inputs, and generates as outputs the corresponding repetitions satisfying the controls.

Our method, which we call *dynamic element textures*, aims to produce such controllable repetitions through a combination of constrained optimization (satisfying controls) and data driven computation (synthesizing details). We use spatial-temporal samples as the core representation for dynamic geometric elements. We propose analysis algorithms for decomposing small scale repetitions from large scale themes, as well as synthesis algorithms for generating outputs satisfying user controls. Our method is general, producing a range of artistic effects that previously required disparate and specialized techniques.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: dynamic, element, texture, control, constraints, optimization, geometry, animation, analysis, synthesis

Links: DL PDF

1 Introduction

Many natural phenomena are characterized by intricate interplays between repetitive geometric structures and dynamic motions, such as advecting particles [Zhu and Bridson 2005; Narain et al. 2010], flocking herds [Narain et al. 2009; Ju et al. 2010; Sewall et al. 2011], undulating threads [Wang et al. 2009], and waving sheets [Wang et al. 2010; Kavan et al. 2011]. Due to their ubiquity and rich variety, such phenomena have long been important for many graphics effects. A central challenge from the authoring perspective is control: manual edits can be too tedious due to the numerous repetitions, while automatic computation (e.g. procedural or physical simulation) might not offer enough creative flexibility or numerical stability for the desired effects. Another challenge is generality; most of the prior methods are dedicated to a specific phenomenon, and users have to employ a multitude of tools for different effects.

We propose a method that offers controllability and generality for authoring spatial-temporal repetitions. Based on the observation that many such repetitions can be intuitively understood as a combination of coarse scale themes and fine scale details, we devise two main categories of controls: output constraints at the coarse scales (e.g. overall shape, orientation, and motion of the output domain), and input exemplars at the fine scales (e.g. detailed element arrangements and movements). Given these, our method will then generate the desired outputs. See Figure 1 for examples.

It is challenging to provide the aforementioned controllability and generality. In particular, obeying output constraints and handling numerous details are both difficult problems *alone* for prior dynamics simulation methods. To meet these challenges, our method combines constrained optimization with data driven computation to satisfy the controls while synthesizing the details. We call this method *dynamic element textures*. Our core idea is to extend the spatial samples in Ma et al. [2011] with the temporal samples in Wei and Levoy [2000] to produce *spatial-temporal* samples as the foundation for representation, analysis, and synthesis. On the **representation** side, we use spatial-temporal sample neighborhoods

[*]Part of this work was conducted while Chongyang Ma was a visiting student in ALICE/INRIA and a research intern in Microsoft Research Asia.

analogous to the use of pixel [Efros and Leung 1999] or vertex [Turk 2001] neighborhoods in prior texture representations, so that we can apply analogous methodology to synthesize dynamic element details. On the **analysis** side, we perform spatial-temporal filtering to decompose a given input exemplar into coarse and fine scale geometry and motion. On the **synthesis** side, we provide a solver that enforces local neighborhood similarity between input and output (via data-driven computation) while observing additional constraints such as motion, geometry, topology, and physics (via constrained optimization).

Our method offers the following advantages: it is easy to use and requires only concrete output constraints and input exemplars instead of abstract procedures or parameters; it is general and can be applied to a variety of phenomena; it is robust and unconditionally stable; it offers full controllability through constraints and exemplars; and it can produce both realistic and artistic effects that are difficult to achieve via prior techniques.

2 Previous Work

A common method to synthesize detailed repetitions is by designing specialized procedures for specific phenomena, such as traffics [Sewall et al. 2011], crowds [Narain et al. 2009], and fluid turbulence [Stam and Fiume 1993; Thürey et al. 2006]. The key advantage of such methods is their controllability through fine tuning of the detailed steps and parameters of the procedures. However, they often do not apply outside their target phenomena and require significant expertise to understand the algorithms and their parameters. For this reason, we choose a data-driven approach. Data-driven synthesis tends to be more general and user friendly, as it is easier to ask users to provide examples than to understand and tweak specific algorithms. We discuss below the techniques from which we draw inspiration, as well as related data-driven techniques addressing animated content.

Neighborhood-based texture synthesis Early data-driven methods [Paget and Longstaff 1995; Efros and Leung 1999] focused on stochastic texture synthesis, as surveyed by Wei et al. [2009]. Most of these methods optimize output texture colors to reproduce the spatial neighborhoods of pixels in the input exemplar. Latest methods provide fast synthesis and user control, in particular conformity to content manually placed by the user [Lefebvre and Hoppe 2005]. We seek to bring similar benefits for synthesizing dynamic details in animated 3D content.

Time-varying textures Several approaches modify texture synthesizers to generate animated sequences [Szummer and Picard 1996; Wei and Levoy 2000; Kwatra et al. 2003]. These algorithms extend spatial neighborhoods in the temporal domain, augmenting them with colors from previous frames. A similar methodology allows synthesis of textures flowing along a vector field, correcting distortions from motion through texture synthesis [Kwatra et al. 2005; Lefebvre and Hoppe 2006; Kwatra et al. 2007]. Schödl et al. [2000] note that some video sequences exhibit a stochastic behavior only in time. They detect low-cost transitions between frames to generate infinitely looping sequences. All these schemes consider *regular* neighborhoods of pixels across time and space.

Element-based textures Many natural appearances result from the composition of similar discrete elements. These basic components can be extracted manually or automatically [Ahuja and Todorovic 2007; Cheng et al. 2010]. Dischler et al. [2002] produce textures by distributing texture patches with irregular boundaries, considering co-occurrence statistics. Recent schemes synthesize distributions of elements by matching *irregular* neighborhoods of elements [Barla et al. 2006; Ijiri et al. 2008; Ma et al. 2011]. This makes an interesting connection to neighborhood based synthesis, which we exploit in this work.

Symbol	Meaning
s_i	Input sample
s_o	Output sample
$\mathbf{p}(s)$	Position of sample s
$\hat{\mathbf{p}}(s, s')$	Relative position between s and s'
$t(s)$	Time stamp of sample s
$\mathbf{u}(s)$	Concatenated spatial-temporal location of sample s
$\mathbf{n}(s)$	Spatial-temporal neighborhood of sample s
$\mathcal{I}$	Sample collections of input exemplar
$\mathcal{O}$	Sample collections of synthesis output

Table 1: *Notations.*

Our technique is at the crossroads of time-varying and element-based techniques: we extend the irregular neighborhoods of Ma et al. [2011] in the temporal domain and add topological constraints to address different geometries. This enables a more general framework: animation is no longer limited to video sequences, and our method is applicable to particles, threads, and sheets (Figure 1).

Data-driven animation synthesis Several data driven synthesis methods target specific animated content, such as motion capture data [Kovar et al. 2002; Pullen and Bregler 2002], crowds [Ju et al. 2010; Li et al. 2012], sequences of meshes [James et al. 2007], and cloth wrinkles [Wang et al. 2010; Kavan et al. 2011]. While these techniques offer state-of-the-art results for the specific applications they address, our technique applies to a wider range of phenomena and provides more control to the user.

Controllable animation synthesis Control over the result is especially important for animators: it is often necessary to direct the animation, for instance, specifying the poses at certain keyframes. Several techniques for simulation of dynamic effects offer such controls, including rigid body [Popović et al. 2000], crowd formation [Kwon et al. 2008], and deformable objects [Barbič et al. 2009]. However, none of the existing methods can handle large scale controls and small scale details simultaneously. In addition, non-physics-based effects are preferred under a variety of usage scenarios [Cho et al. 2007]. This is a key advantage of our data-driven technique compared to physically based animation.

3 Core Ideas

We use spatial-temporal samples to represent both the geometric and dynamic aspects as well as to define a sample-based local neighborhood to characterize textures. These form the basis of our analysis and synthesis algorithms. Notations are summarized in Table 1 for easy reference.

3.1 Samples

To handle different element geometries and motions, we use a sample based representation as inspired by Ma et al. [2011] for static geometry as well as Müller and Chentanez [2011] for dynamic motion. The algorithm of Ma et al. [2011] stores with each sample s its spatial location $\mathbf{p}(s)$ as the main information, plus optional application dependent properties $\mathbf{q}(s)$ such as id, color, or texture coordinates. In our case, since we need to handle dynamic motions, we also store with each sample a time-stamp t:

$$\mathbf{u}(s) = (\mathbf{p}(s),\, t(s)) \tag{1}$$

where $\mathbf{u}(s)$ indicates the spatial-temporal location of s, a concatenation of its spatial position $\mathbf{p}(s)$ and time stamp $t(s)$. This simple representation allows us to handle a variety of objects with different

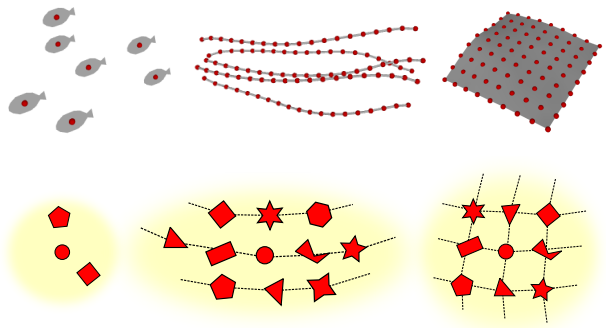


Figure 2: *Elements, samples, and neighborhoods. Top row: elements (gray shapes) and samples (red points). Bottom row: the corresponding* spatial *neighborhoods with samples illustrated in different shapes and dashed lines indicating topological constraints.*

shapes and motions. See Figure 2 for examples.

3.2 Neighborhood

Local neighborhood has shown to be a very simple and effective representation for textures [Paget and Longstaff 1995; Efros and Leung 1999]. Here, we define neighborhoods based on the spatial-temporal samples introduced above.

Representation Our basic idea is for each neighborhood $\mathbf{n}$ to cover nearby spatial-temporal samples considering both static and dynamic aspects. Specifically, for each element s' in the spatial-temporal neighborhood of s, we calculate their relative spatial-temporal difference as:

$$\hat{\mathbf{u}}(s', s) = \mathbf{u}(s') - \mathbf{u}(s) \tag{2}$$

This allows us to define the neighborhood $\mathbf{n}$ around s as a collection of spatial-temporal differentials $\hat{\mathbf{u}}(s', s)$:

$$\mathbf{n}(s) = \{\hat{\mathbf{u}}(s', s)\} \tag{3}$$

where s' is within a user-defined spatial-temporal region around s. (See Figure 3.) Essentially, we express each neighborhood $\mathbf{n}(s)$ in terms of relative differences of each s' with respect to the center sample s, therefore discarding the global information and allowing for neighborhoods in different spaces and times to match.

Distance measure The dissimilarity of two samples' neighborhoods $\mathbf{n}(s_o)$ and $\mathbf{n}(s_i)$ is measured by our notion of neighborhood "distance" defined as:

$$|\mathbf{n}(s_o) - \mathbf{n}(s_i)|^2 = \sum_{s'_o \in \mathbf{n}(s_o)} \left|\hat{\mathbf{u}}(s'_o, s_o) - \hat{\mathbf{u}}(s'_i, s_i)\right|^2 \tag{4}$$

where s'_o runs through all samples in $\mathbf{n}(s_o)$, and $s'_i \in \mathbf{n}(s_i)$ is the "matching" sample of s'_o. We match only s'_i and s'_o with identical time differences to s_i and s_o. We determine the exact matches via the Hungarian algorithm [Kuhn 1955] to minimize Equation 4.

Topology constraint Samples belonging to the same element, such as a thread or sheet as illustrated in Figure 2, have to obey additional topological constraints during the matching process described above. Our sheets have a regular 2D topology along their surface akin to pixels in images (Figure 2). Therefore we forgo our element-based neighborhood collection and instead we gather a 7×7 neighborhood guided by this topology [Lefebvre and Hoppe 2005]. Threads are the 1D equivalent of this. The directions, such as

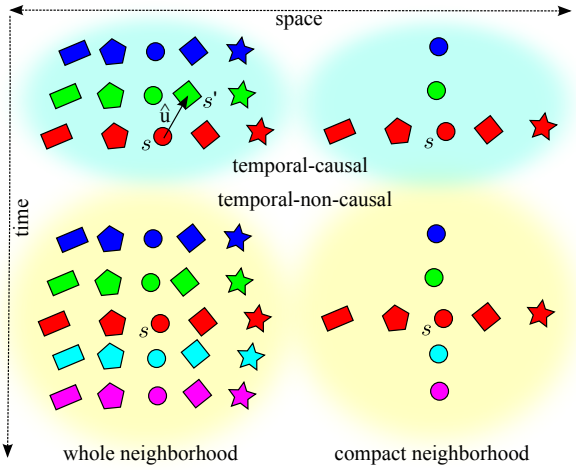


Figure 3: *Spatial-temporal neighborhood shape. Here we visualize 2D spatial (horizontal) temporal (vertical) neighborhoods, with colors indicating samples at the same time frame and shapes indicating samples across different time frames. Two orthogonal concepts are illustrated here: temporal-causal versus temporal-non-causal (top and bottom) and whole versus compact shape (left and right). We use temporal-causal neighborhoods (top) for frame-by-frame synthesis and temporal-non-causal neighborhoods (bottom) for all-frame synthesis. A full neighborhood* $\mathbf{n}(s)$ *considers all neighboring samples near the central sample* s *(left), while our compact neighborhood considers only spatial neighbors and nearby temporal doppelgangers (right).*

left and up, are determined by user-specified orientation fields. These topological constraints precede the Hungarian algorithm above, which applies only to sample pairs (s'_o, s_o) without any topological constraints.

Temporal causality Depending on which portions of the output already contain known texture values, different neighborhood shapes may need to be deployed. A key concept is causality. In Wei and Levoy [2000], specifically, a causal neighborhood is used for scanline-order synthesis (to cover only those already synthesized pixels), whereas a non-causal neighborhood is used for full synthesis (as refinement of an already synthesized output). For dynamic element textures, we adopt an analogous concept in the temporal dimension. See Figure 3 for illustrations. A temporal-non-causal neighborhood is a neighborhood that contains samples in later frames. We use temporal-causal neighborhoods for synthesizing output frame-by-frame, and use temporal-non-causal neighborhoods for optimizing all output frames together when the synthesis is performed under user constraints.

Compact shape The neighborhood $\mathbf{n}(s)$ introduced so far incorporates all spatial-temporal samples within a certain distance from a central sample s as shown in Figure 3 (left). For better efficiency, we have adopted a more compact neighborhood shape with only spatial neighbors and 1D temporal trails, as shown in Figure 3 (right), which reduces computation significantly (from $O(r^4)$ to $O(r^3)$ in a 3D spatial + 1D temporal space with circular neighborhood of radius r). This idea is analogous to the use of 2D slices instead of full 3D neighborhoods for accelerating solid texture synthesis [Dong et al. 2008]. While results are found adequate with the compact neighborhood, we leave a more thorough study of where a full neighborhood is necessary as future work.

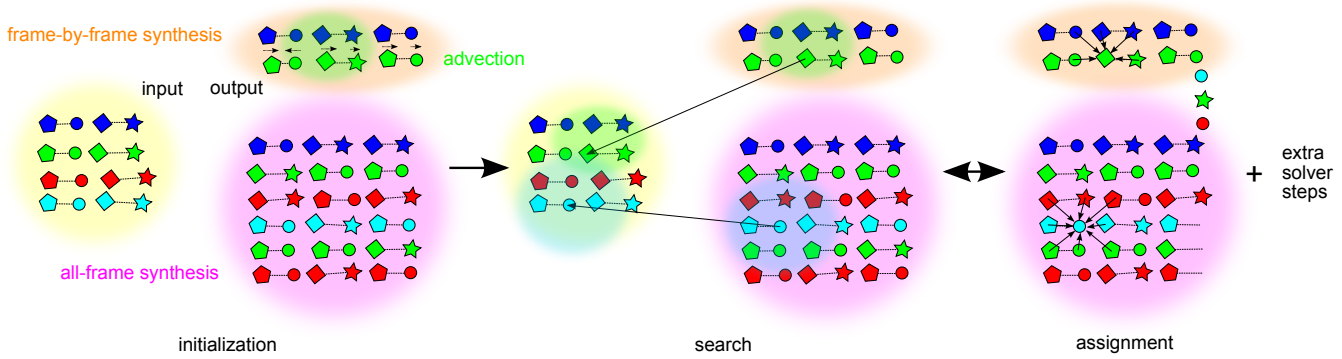


Figure 4: *Illustration of our synthesis pipeline. Following the convention in Figure 3, colors indicate samples at the same time frames and dashed lines connect samples of the same elements. The input is on the left while the output is on the right. Depending on the sequencing, the output is initialized with either the first frame or the whole spatial-temporal volume. Our method then iterates among the search, assignment, and potential extra solver steps.*

4 Synthesis

Our basic methodology is inspired by texture optimization [Kwatra et al. 2005]. Let $\mathcal{I}$ be the input exemplar and $\mathcal{O}$ the corresponding synthesis output. We can measure their distance based on local neighborhoods via the following general energy formulation:

$$E(\mathcal{O};\mathcal{I}) = \sum_{s_o \in \mathcal{O}} \min_{s_i \in \mathcal{I}} |\mathbf{n}(s_o) - \mathbf{n}(s_i)|^2 + \Theta(\mathcal{O};\mathcal{I}) \qquad (5)$$

The goal is to compute an output $\mathcal{O}$ that minimizes $E(\mathcal{O};\mathcal{I})$ for a given input $\mathcal{I}$. In the above equation, the first term measures the difference between the local neighborhoods $\mathbf{n}$ as defined in Equation 4. Specifically, for each output sample $s_o \in \mathcal{O}$, we find the corresponding input sample $s_i \in \mathcal{I}$ with the most similar neighborhood, and sum their squared neighborhood differences. The second term Θ contains application specific constraints.

4.1 Basic Synthesis

Our basic algorithm first initializes the output according to the input exemplar(s) and output constraints. It then optimizes the output through iterative search and assignment steps as inspired by Kwatra et al. [2005], plus potential extra solver steps. The whole process is visualized in Figure 4. More detailed descriptions are as follows.

Sequencing We can synthesize the output either frame-by-frame or all the frames together. The former is easier to implement and has smaller memory footprint, while the latter is necessary to handle spatial-temporal constraints. All the subsequent steps depend on this choice of sequencing, as well as the choice of neighborhood shape as depicted in Figure 3.

Initialization For frame-by-frame synthesis, we initialize the first frame via the patch-based approach in Ma et al. [2011]. Each subsequent frame is initialized from the previous frame via advection following the sample velocities.

For all-frame synthesis, we extend the patch-based initialization in Ma et al. [2011] to include the time dimension as well. Specifically, we divide the input exemplar into spatial-temporal patches and then randomly copy these patches into the output domain. We set the patch size to be identical to the user selected neighborhood size (Section 3.2). To avoid partial/broken objects, we always copy complete elements.

When constraints are present, we perform this patch copy starting from the most constrained to the least constrained regions. Specifically, we first identify the most constrained regions in the output, find the input patches that best satisfy these constraints, and move the constrained elements to the use specified positions for hard constraints. We repeat the same process for gradually less constrained regions while ensuring patch boundary compatibility as in traditional patch based synthesis. Since we do not add or remove elements during the subsequent optimization, we need to ensure that all frames are populated with the same number of elements. In addition, when copying patches, we take into account user controls, such as aligning patches with local orientations as well as preferring input patches with similar boundary conditions to the output region.

Search step During the search step, we find, for each output sample s_o, the best matching input sample s_i with the most similar neighborhood (according to Equation 4). This search is conducted by exhaustively examining every input sample if the exemplar is sufficiently small, or further accelerated by adopting a k-coherence search [Tong et al. 2002] for constant time computation.

Assignment step Here we determine the output sample positions $\{\mathbf{p}(s_o)\}_{s_o \in \mathcal{O}}$ to minimize Equation 5. At the beginning of the assignment step, we have multiple input neighborhoods $\{\mathbf{n}(s_i')\}$ overlapping every output sample s_o, where $\mathbf{n}(s_i')$ is the matching input neighborhood for output sample s_o' as determined in the search step, and s_o' is sufficiently close to s_o so that the spatial extent of $\mathbf{n}(s_i')$ covers s_o. Each such $\mathbf{n}(s_i')$ provides a prediction $\hat{\mathbf{p}}(s_o, s_o')$ for the relative position between s_o and s_o':

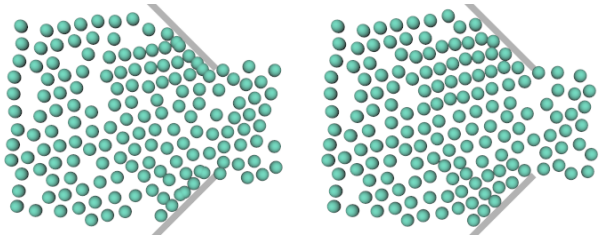
$$\hat{\mathbf{p}}(s_o, s_o') = \mathbf{p}(s_i) - \mathbf{p}(s_i') \qquad (6)$$

where s_i,s_i' indicates the matching input sample for s_o,s_o' respectively as described in the neighborhood metric (Equation 4). (Each such prediction is visualized as an arrow in Figure 4.) Note that since $\mathbf{n}$ covers neighbors both spatially and temporally, our method above can assign positions considering spatial arrangements and temporal motions. We extract from Equation 5 the $\mathbf{p}(s_o)$ variables for all output samples $s_o \in \mathcal{O}$ into the following energy function:

$$E_p(\{\mathbf{p}(s_o)\}) = \sum_{s_o \in \mathcal{O}} \sum_{s_o' \in \mathbf{n}(s_o)} \left|(\mathbf{p}(s_o) - \mathbf{p}(s_o')) - \hat{\mathbf{p}}(s_o, s_o')\right|^2 \qquad (7)$$

Equation 7 is a quadratic function of $\{\mathbf{p}(s_o)\}$ and can be minimized via least squares, i.e. solving a positive definite sparse linear system.

Parallelization The search step above can be trivially parallelized. To parallelize the assignment step, we fix $\mathbf{p}(s_o')$ (considering as constants rather than variables as inspired by Lefebvre and Hoppe [2005]) in Equation 7, so that $\mathbf{p}(s_o)$ can be solved by examining only a small set of local neighbors.



(a) *pure synthesis* **(b)** *hybrid solver*

Figure 5: *Comparison between pure synthesis and hybrid solver with a physical sub-step. Notice the higher amount of collisions near the boundary obstacle for (a) versus (b). By detecting and resolving particle collisions in the advection step, our hybrid solver can reduce physical artifacts compared to pure data driven synthesis with global flow. Both of the results are obtained using the input exemplar shown in Figure 8a.*

Extra energy terms We can incorporate additional controls for various application specific effects such as control maps [Lu et al. 2007; Wei et al. 2008], orientation fields, boundary conditions, domain shapes, and constrained selections. These can be achieved through additional energy terms Θ in Equation 5 analogous to how Ma et al. [2011] handled static elements.

Extra solver steps Additional solver steps can be incorporated depending on particular application needs. One possibility is to add a physical simulation step to help resolve collisions and interpenetrations, making our whole pipeline a *hybrid* between data driven and physical simulation. We have found this hybrid approach helpful for more constrained situations such as output boundary conditions not present in the input exemplars, as demonstrated in Figure 5. However, we wish to emphasize that all these extra steps are optional and our basic steps are already quite adequate. We have *not* applied a physical sub-step to our other results.

4.2 Smooth Synthesis

Since the input exemplar provides only a finite discrete sampling of the entire neighborhood state space, the best matching input neighborhoods during two successive search steps may change significantly. Furthermore, even when the best matching input neighborhoods remain the same, the number of matched neighbor pairs may change during the assignment step. These may cause our basic algorithm in Section 4.1 to produce temporal jitters if the input exemplar is too small to adequately sample the state space. Note that such jittering also exists in color textures, but human perception is more sensitive to motion jittering than color jittering.

We alleviate these issues by introducing smoothness into every major step of our basic algorithm, including both the search and assignment steps as well as the neighborhood distance measure.

Distance measure Inspired by [Efros and Leung 1999], we use a Gaussian falloff to smooth the boundary of each neighborhood:

$$|\mathbf{n}(s_o) - \mathbf{n}(s_i)|^2 \tag{8}$$

$$= \sum_{s'_o \in \mathbf{n}(s_o)} \frac{\kappa(s'_o, s_o) + \kappa(s'_i, s_i)}{2} \left|\hat{\mathbf{u}}(s'_o, s_o) - \hat{\mathbf{u}}(s'_i, s_i)\right|^2 \tag{9}$$

where $\kappa(s', s)$ is a Gaussian kernel with user specified parameters σ_p and σ_t for spatial and temporal domain extents:

$$\kappa(s', s) = \exp\Big(- \frac{(\mathbf{p}(s') - \mathbf{p}(s))^2}{\sigma_p} - \frac{(t(s') - t(s))^2}{\sigma_t} \Big) \tag{10}$$

Intuitively, this smooth falloff reduces the impacts of samples far from the neighborhood centers and alleviates the temporal jittering when the number of neighbors changes drastically between two consecutive frames.

Search step For each output neighborhood $\mathbf{n}(s_o)$ we find the nearest N input neighborhoods $\{s_{ij}\}_{j=0:N-1}$ instead of just one as in our basic algorithm.

Assignment step We generalize Equation 6 to consider these $N > 1$ best matching input neighborhoods as follows:

$$\hat{\mathbf{p}}(s_o, s'_o) = \frac{1}{\sum_j \omega_j} \sum_j \omega_j \cdot (\mathbf{p}(s_{ij}) - \mathbf{p}(s'_{ij})) \tag{11}$$

where the interpolation weight ω_j is determined based on the neighborhood distances:

$$\omega_j = 1.0 - \frac{|\mathbf{n}(s_o) - \mathbf{n}(s_{ij})|^2}{\sum_k |\mathbf{n}(s_o) - \mathbf{n}(s_{ik})|^2} \tag{12}$$

In our current implementation we use $N = 2$, as our experiments indicate that this already smoothes out most noticeable jittering effects.

4.3 Graph Synthesis

Our algorithm described so far relies on inputs with sufficient spatial and temporal extents in order to produce good results. However, this might not be always feasible or desirable; captured data may have inherent limitations on the spatial-temporal extents, and manual preparations can quickly become tedious for large inputs.

To address this issue, we propose to enrich the input exemplars through *neighborhood graph*, as inspired by those animation-from-still methods [Wang et al. 2006; Xu et al. 2008]. The graph consists of neighborhoods as graph nodes and edges connecting nodes with weights proportional to the nodal neighborhood distance, as defined in Equation 4. Thus, walking along a low-weight path can provide a plausible animation, even for inputs with limited temporal extents (including static inputs with a single frame).

Graph construction Given a set of sample neighborhoods $\{\mathbf{n}(s)\}$, we build a graph out of these neighborhoods as nodes. The weight of the edge connecting each pair of nodes is computed via our neighborhood distance measure. In particular, the lower the weight is, the more similar the two neighborhoods are. We denote the largest weight of the edge in the graph as δ, which is used as a normalization factor below.

Path computation For each node $\mathbf{n}(s_a)$, we first identify its immediate neighbor node $\mathbf{n}(s_b)$ with the largest edge weight from $\mathbf{n}(s_a)$. We then determine a *motion path* between these two nodes $\{\mathbf{n}(s_0), \mathbf{n}(s_1), \ldots, \mathbf{n}(s_n)\}$, which satisfies $s_0 = s_a$, $s_n = s_b$ and minimizes the following energy function:

$$\sum_{i=0}^{n-1} \frac{|\mathbf{n}(s_{i+1}) - \mathbf{n}(s_i)|^2}{\delta} + \alpha\Big(1 - \frac{\sum_{0 \le i,j \le n} |\mathbf{n}(s_i) - \mathbf{n}(s_j)|^2}{\delta \cdot n \cdot (n+1)}\Big) \tag{13}$$

where α is a user specified constant weight ($\alpha = 0.5$ worked well in all our results). Intuitively, the first term in Equation 13 ensures a smooth transition between $\mathbf{n}(s_a)$ and $\mathbf{n}(s_b)$, while the second term favors larger node diversity in the path. We solve the discrete optimization problem in Equation 13 using simulated annealing.

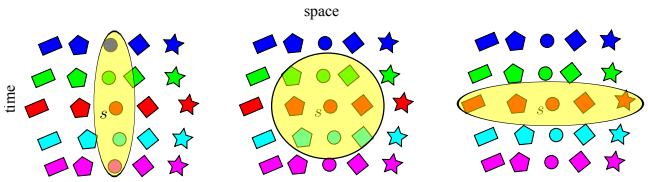


Figure 6: *Spatial-temporal filtering. Shown here are different filter kernel sizes and shapes. The filter kernels are colored in yellow, with the sample visualization following the convention of Figure 3. Left: temporally elongated. Middle: spatial-temporal isotropic. Right: spatially elongated.*

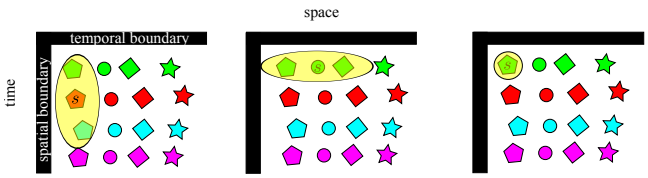


Figure 7: *Adaptive filter sizes near boundaries. Starting with the isotropic filter in Figure 6 middle, the filter needs to shrink for samples near boundaries, in the spatial (left), temporal (middle), or both (right) directions.*

For each motion path, we use the morphable model in [Ju et al. 2010] to generate smooth temporal transitions between nodes, with blending weights determined by considering the path nodes as fourth-order NURBS data points.

5 Analysis

Our synthesis method relies on inputs that contain only details without global structures. However, creating such pure details in isolation can be difficult, as certain phenomena can be easier to obtain alongside non-trivial global structures (e.g. interesting thread vibrations with certain hair styles). Therefore we provide an analysis tool to decompose general inputs into local and global motions and geometries.

Our basic idea is to perform a low pass filtering of spatial-temporal sample positions of a given input $\mathcal{I}$ to obtain the coarse spatial-temporal sample positions $\mathcal{C}$, with the details $\mathcal{D}$ computed as their difference $\mathcal{D} = \mathcal{I} - \mathcal{C}$. $\mathcal{C}$ directly provides a coarse sample geometry in a per time frame basis, and the coarse motion is computed as the difference of corresponding samples at adjacent time frames of $\mathcal{C}$. Since both filtering and differencing are linear operations, filtering positions followed by differencing for velocity is the same as differencing for velocity first followed by filtering.

The size and shape of the filter kernel can be customized according to particular application needs and semantics (see Figure 6 for visualization). Let us take group motions as an example. If we assume the global flow changes slowly over time, we can set the kernels to have large temporal duration. However, if the flow has higher temporal variation, we can set a shorter kernel size in the temporal direction.

To avoid bias, the filter kernel should maintain symmetry both spatially and temporally; otherwise, the filtered spatial-temporal location may be biased/shifted towards another spatial or temporal direction. This implies that for samples near spatial or temporal domain boundaries, the filter size will need to shrink accordingly. See Figure 7 for an illustration.

6 Applications

We apply our approach to a variety of phenomena. Here, for the purpose of presentation we organize the applications according to the topological degree of constraints, including particles (Section 6.1), threads (Section 6.2), and sheets (Section 6.3). Orthogonal to all these categories, our approach handles different input exemplars that are realistic or artistic (e.g. Figure 10), dynamic or static (e.g. Figure 11 & Figure 12), as well as output constraints such as shapes, boundaries (e.g. Figure 8), motions (e.g. Figure 9), and physics (e.g. Figure 5). We also present our analysis method for decomposition and recombination (e.g. Figure 14). To our knowledge, even though techniques exist for specific phenomena, none of them are capable of handling all such general and diverse effects.

Category	topology constraint	Figure	neighborhood size		σ_p
			spatial	temporal	
particles	none	8e	2.0	1	1.0
		8f	2.25	1	1.0
		8g	1.75	1	1.0
		8h	1.75	1	3.0
fish	none	9b	2.0	10	∞
		9d	2.5	10	∞
		9f	2.0	10	∞
		11b	2.0	10	3.0
spaghetti	1D	10	3.0	7	∞
papers	2D	13	3.0	10	∞

Table 2: *Parameter settings for our results. The spatial neighborhood size is measured with respect to the nearest distance between two samples in the exemplar. The temporal neighborhood size counts the number of successive frames in each spatial-temporal neighborhood. The spatial smooth falloff range σ_p is measured with respect to the spatial neighborhood size, while no temporal smooth falloff has been introduced for all the results (i.e. $\sigma_t = \infty$).*

Parameters We have found that our algorithm behaves well with a wide range of parameters. For example, the spatial neighborhood size is set to approximately cover the 1-ring neighborhood of each sample, while the temporal content of the neighborhood just needs to be comparable to the length of a single textural behavior (such as a fish transiting from one group to another in Figure 9e). We have listed the parameters for each demo in Table 2.

6.1 Particles

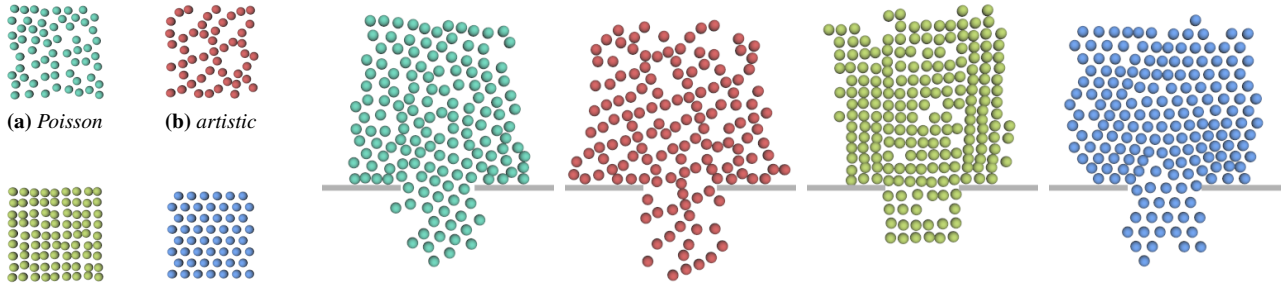
This category includes individual elements or agents freely moving around each other without any topological constraints, such as particles or herd motions.

We start with the simplest case of isotropic elements, serving as a didactic demo and a base-line test for our method, in line with prior texturing algorithms [Efros and Leung 1999]. As shown in Figure 8, we can use different input exemplars to fully control and produce different output effects, some of which contain artistic patterns and are beyond traditional procedural or simulation-based methods. We can also use different output constraints including initial domain shape, boundary obstacles, and global flow field.

Phenomena with more complex spatial-temporal structures can also be easily handled by our method. In Figure 9, we show herd motion consisting of individual fish with relatively complex geometries and dynamics. With our approach, users can produce a variety of effects by simply providing proper exemplars, such as both spatially and temporally homogeneous animation (case 9a), heterogeneous motion with some agents behaving differently from others (case 9c), or spatially heterogeneous arrangements with agents of different sizes (case 9e). The only output constraint for all these versatile cases is a simple cubic output domain shape.

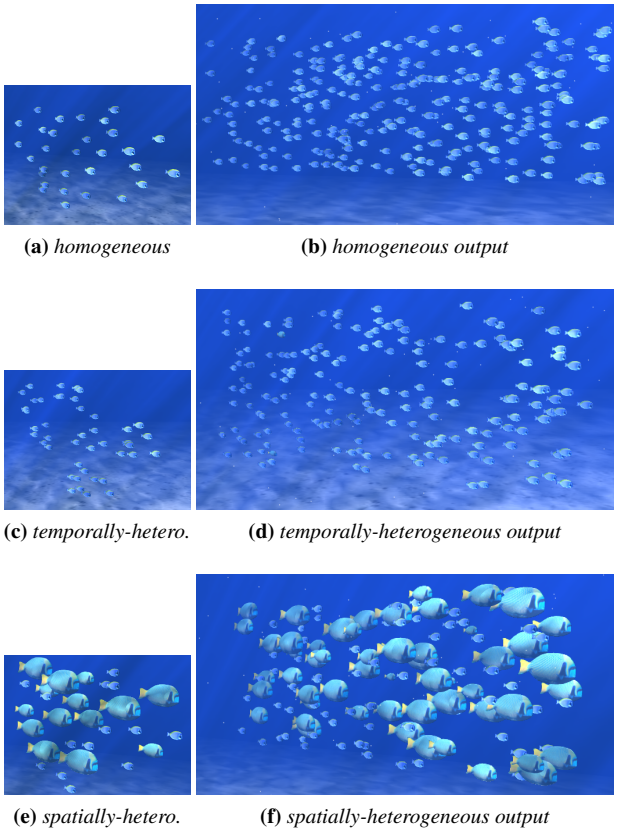
6.2 Threads

This category includes elongated elements such as noodles, hairs, and branches. Such elongated elements need multiple samples per element for accurate characterization, and samples corresponding to



(a) *Poisson* **(b)** *artistic* **(c)** *jittered* **(d)** *regular* **(e)** *stochastic output* **(f)** *artistic output* **(g)** *jittered output* **(h)** *regular output*

Figure 8: *Synthesis of particles. Out method can synthesize animation of particles with various local distributions, including stochastic (a), artistic (b), as well as regular (d) arrangements. Input (b) is manual art while the rest procedurally generated: (a) Poisson disk, (c) jittered grid, (d) hexagonal grid.*



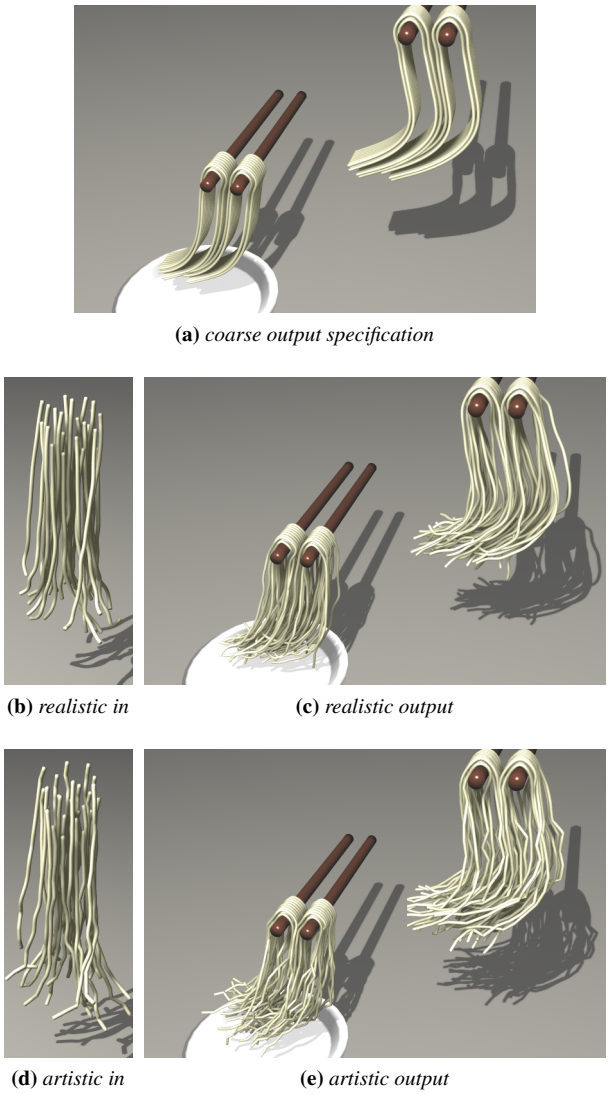
(a) *homogeneous* **(b)** *homogeneous output* **(c)** *temporally-hetero.* **(d)** *temporally-heterogeneous output* **(e)** *spatially-hetero.* **(f)** *spatially-heterogeneous output*

Figure 9: *Synthesis of fish. Here we manually create three different input exemplars: a homogeneous distribution (a), a temporally heterogeneous behavior with some fishes rapidly transiting between groups (c), and a spatially heterogeneous distribution of fishes with different sizes (e).*

the same element have to obey additional 1D constraints.

In Figure 10, we apply our method for dynamic spaghetti/noodle threads. It is very difficult to control the exact appearance and behavior of such a complex phenomenon through physical simulation [Soares et al. 2012] at either the global or the local scale, such as the overall shape of the noodle bundle or the detailed motion of each individual thread.

Our system allows direct control of both scales through output constraints and input exemplars. In Figure 10, users first specify the motion for a single output noodle via simulation or manual manipulation and simply duplicate the motion to all the other output noodles to get the coarse control. Then they can use our system to add details from the input exemplar, introduce natural variations,



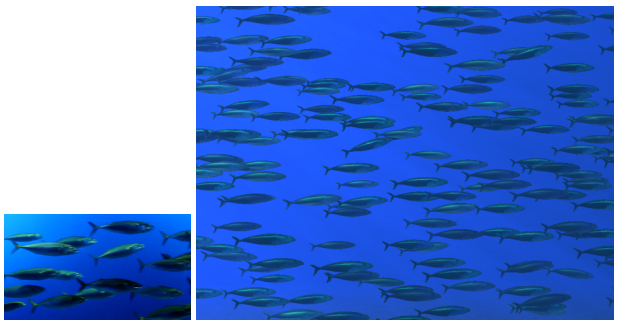
(a) *coarse output specification* **(b)** *realistic in* **(c)** *realistic output* **(d)** *artistic in* **(e)** *artistic output*

Figure 10: *Synthesis of spaghetti. With a coarse output motion (a) and different detailed input exemplars, (b) for realistic effects and (d) for artistic effects ("jagged" noodles for the 40th SIGGRAPH), our method can produce corresponding results with different effects (c) (e).*

and/or achieve artistic effects.

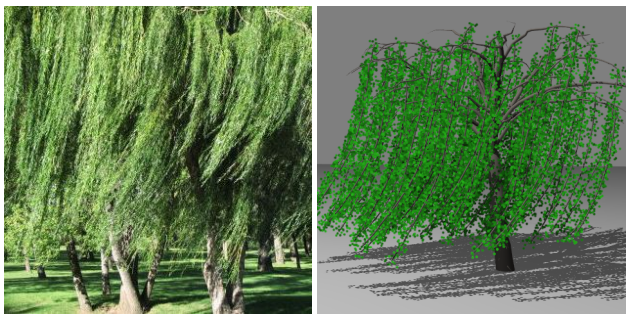
6.3 Sheets

This category includes flat elements such as papers and clothes. These elements need multiple samples per element to accurately



(a) *captured input* **(b)** *synthesis output*

Figure 11: *Synthesis from captured video input.*



(a) *captured input* **(b)** *synthesis output*

Figure 12: *Synthesis from captured image input.*

characterize geometries and motions both within and across elements, including the 2D topological constraints. Adding dynamic details for sheets via data driven synthesis has recently attracted a lot of attention [Wang et al. 2011; Kavan et al. 2011]. However, most of these methods aim to reproduce physical realism with more efficient computations. With different input exemplars, our approach produces not only physically plausible motions but also artistic effects as shown in Figure 13. Here we use the coarse motion in Figure 13c as the only output constraint.

6.4 Captured Input

Our method can be applied to not only manually or computationally prepared inputs as demonstrated in our earlier demos, but also captured inputs, which can significantly save user preparation time, analogous to how prior image texture synthesis methods can produce outputs directly from images snapped from the web [Efros and Leung 1999].

In Figure 11, we demonstrate a case where the input is a captured video of swimming fish and the output contains synthetic models with spatial distributions and temporal motions computed by our method. This case is inspired by prior motion-capture like scenarios (e.g. [Ju et al. 2010]), where synthetic outputs are computed by data captured inputs. Here, the users only need to prepare polygonal model for the fish, with the spatial distributions and temporal motions extracted from the input.

In Figure 12, we demonstrated a case where the input can be simply captured images. This can be a quite handy usage scenario, as users only need to shoot a few static images instead of a whole video of a real scene. They can then use our algorithm as described in Section 4.3 to produce dynamic outputs from static inputs. In this particular example, the input consists of a single photo of a willow tree (with branch structures manually traced by a user) and the output contains a dynamic motion with polygonal models.

6.5 Analysis

We can apply our analysis algorithm (Section 5) for decomposing an input into coarse and fine spatial-temporal structures as well as re-combining different coarse and fine structures.

Decomposition We apply our analysis method to decompose an input dynamic geometry into the corresponding coarse and fine scales. Since the notion of coarse and fine can be subjective (similar to image textons with hierarchical structures), we let users decide the proper neighborhood size for the decomposition. Figure 14 demonstrates our decomposition results for dynamic hairs.

Recombination We can also recombine the coarse motion of one sequence with the details of another. See Figure 14 for examples. Here the extracted coarse motion is used to guide the synthesis of fine-scale features and acts like the control map of the origin input, similar to the filtered source image in image analogies [Hertzmann et al. 2001].

7 Limitations and Future Work

Our method handles only textural motions and geometries with sufficiently small and local scales. Take the fish demo in Figure 9 as a concrete example. If the relative motions between different fish are sufficiently large relative to their aggregate speed, our method may produce output with some fish swimming backwards. A potential remedy is to add additional constraints beyond those described in Section 4.1.

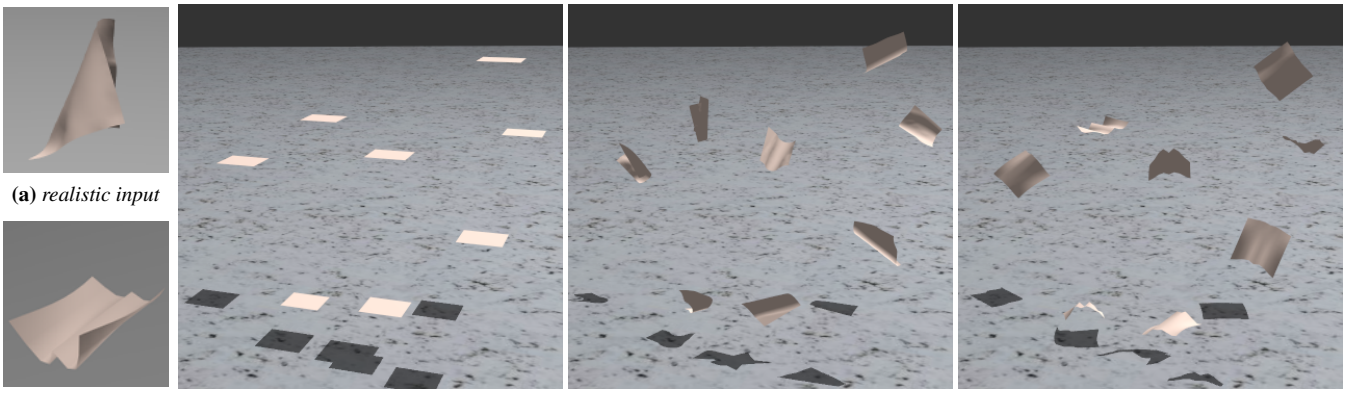
Our current core metric uses simple spatial-temporal samples. As demonstrated in other domains such as crowds [Guy et al. 2012], more domain-specific metrics may be needed for accurate evaluation. A potential direction is to incorporate such analysis models into our synthesis framework to produce outputs that more faithfully reproduce input behaviors.

Our current implementation for constrained synthesis (Section 4.1) requires the storage of the entire output spatial-temporal volumes. Potential computation and storage savings can be achieved through order-independent synthesis [Lefebvre and Hoppe 2005].

We have applied our method to particles, threads, and sheets. A potential interesting direction is to explore other applications and representations such as volumetric solids.

A crucial issue for any data driven synthesis is the preparation of input exemplars. This is usually quite easy for image textures but can be more difficult for dynamic elements. We have demonstrated the possibility of using various input methods, including simulation, manual art, and captured data to achieve different desired output effects. We believe more can be done in this direction, especially in combining vision/machine-learning methods with a good UI for human intervention. Similarly, we believe our algorithm would become much more accessible by improving user interaction for contact creation in the spirit of Kazi et al. [2012].

Acknowledgements We would like to thank Florence Bertails-Descoubes for hair animation data, Shuitian Yan and Weiwei Xu for input preparation, Lvdi Wang, Xin Sun and Weikai Chen for final rendering, Steve Lin and David Brown for video dubbing, Will Chang for proofreading, as well as anonymous reviewers for their valuable suggestions. This work was partially supported by ERC grant *ShapeForge* (StG-2012-307877) and general research fund *Dynamic Element Textures* (HKU 717112E).



(a) *realistic input* **(b)** *artistic input* **(c)** *coarse motion* **(d)** *realistic output* **(e)** *artistic output*

Figure 13: *Synthesis of deformable paper scraps. Given a coarse motion (c), we produce outputs with different flavors through different inputs, including physical simulation (a) and manual art (b).*

(a) *original* **(b)** *detail* **(c)** *coarse 1* **(d)** *detail 1* **(e)** *coarse 2* **(f)** *detail 2*

(g) *in1* **(h)** *input 1 + coarse 2* **(i)** *in2* **(j)** *input 2 + coarse 2* **(k)** *input 3* **(l)** *input 3 + detail 2*

Figure 14: *Decomposition and recombination of hair strands. (Top row, analysis for decomposition:) Given an input (a), our method extracts both the coarse and fine scale spatial-temporal textures depending on the user specified neighborhood sizes, as in (c)+(d)\(e)+(f) for smaller\larger neighborhood sizes. All details are visualized via adding on to an initial straight bundle in (b). Note that (e) is smoother than (c) while (f) is more detailed than (d). (Bottom row, re-editing from our analysis results:) We replace the original details in (a) by different input details in (g) and (i) to produce novel outputs in (h) and (j). Conversely, we can also apply analyzed detail motion (f) to a novel coarse input (k) to produce (l).*

References

AHUJA, N., AND TODOROVIC, S. 2007. Extracting texels in 2.1D natural textures. *ICCV 0*, 1–8.

BARBIČ, J., DA SILVA, M., AND POPOVIĆ, J. 2009. Deformable object animation using reduced optimal control. In *SIGGRAPH '09*, 53:1–9.

BARLA, P., BRESLAV, S., THOLLOT, J., SILLION, F., AND MARKOSIAN, L. 2006. Stroke pattern analysis and synthesis. In *Computer Graphics Forum (Proc. of Eurographics 2006)*, vol. 25.

CHENG, M.-M., ZHANG, F.-L., MITRA, N. J., HUANG, X., AND HU, S.-M. 2010. Repfinder: finding approximately repeated scene elements for image editing. In *SIGGRAPH '10*, 83:1–8.

CHO, J. H., XENAKIS, A., GRONSKY, S., AND SHAH, A. 2007. Course 6: Anyone can cook: inside ratatouille's kitchen. In *SIGGRAPH 2007 Courses*.

DISCHLER, J., MARITAUD, K., LÉVY, B., AND GHAZANFARPOUR, D. 2002. Texture particles. In *EUROGRAPH '02*, vol. 21, 401–410.

DONG, Y., LEFEBVRE, S., TONG, X., AND DRETTAKIS, G. 2008.

Lazy solid texture synthesis. In *Computer Graphics Forum (EGSR)*.

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

GUY, S. J., VAN DEN BERG, J., LIU, W., RYNSON, L., LIN, M. C., AND MANOCHA, D. 2012. A statistical similarity measure for aggregate crowd dynamics. In *SIGGRAPH Asia '12*, 190:1–190:11.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *SIGGRAPH '01*, 327–340.

IJIRI, T., MECH, R., IGARASHI, T., AND MILLER, G. 2008. An example-based procedural system for element arrangement. *EUROGRAPH '08 27*, 2, 429–436.

JAMES, D. L., TWIGG, C. D., COVE, A., AND WANG, R. Y. 2007. Mesh ensemble motion graphs: Data-driven mesh animation with constraints. *ACM Trans. Graph. 26*.

JU, E., CHOI, M. G., PARK, M., LEE, J., LEE, K. H., AND TAKAHASHI, S. 2010. Morphable crowds. In *SIGGRAPH Asia '10*, 140:1–140:10.

KAVAN, L., GERSZEWSKI, D., BARGTEIL, A., AND SLOAN, P.-P. 2011. Physics-inspired upsampling for cloth simulation in games. In *SIGGRAPH '11*, 93:1–10.

KAZI, R. H., IGARASHI, T., ZHAO, S., AND DAVIS, R. 2012. Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustraion. In *CHI'12*.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH '02*, 473–482.

KUHN, H. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly 2*, 1-2, 83–97.

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

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. In *SIGGRAPH '05*, 795–802.

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

KWON, T., LEE, K. H., LEE, J., AND TAKAHASHI, S. 2008. Group motion editing. In *SIGGRAPH '08*, 80:1–8.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. In *SIGGRAPH '05*, 777–786.

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

LI, Y., CHRISTIE, M., SIRET, O., KULPA, R., AND PETTRÉ, J. 2012. Cloning crowd motion. In *SCA '12*.

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

MA, C., WEI, L.-Y., AND TONG, X. 2011. Discrete element textures. In *SIGGRAPH '11*, 62:1–10.

MÜLLER, M., AND CHENTANEZ, N. 2011. Solid simulation with oriented particles. In *SIGGRAPH '11*, 92:1–92:10.

NARAIN, R., GOLAS, A., CURTIS, S., AND LIN, M. 2009. Aggregate dynamics for dense crowd simulation. In *SIGGRAPH Asia '09*, 122:1–8.

NARAIN, R., GOLAS, A., AND LIN, M. C. 2010. Free-flowing granular materials with two-way solid coupling. In *SIGGRAPH Asia '10*, 173:1–173:10.

PAGET, R., AND LONGSTAFF, I. D. 1995. Texture synthesis via a nonparametric markov random field. In *Proceedings of DICTA-95, Digital Image Computing: Techniques and Applications*, vol. 1, 547–552.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. 2000. Interactive manipulation of rigid body simulations. In *SIGGRAPH '00*, 209–217.

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: texturing and synthesis. In *SIGGRAPH '02*, 501–508.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *SIGGRAPH '00*, 489–498.

SEWALL, J., WILKIE, D., AND LING, M. C. 2011. Interactive hybrid simulation of large-scale traffic. In *SIGGRAPH Asia '11*, 135:1–12.

SOARES, O., RAJA, S., HURREY, R., AND IBEN, H. 2012. Curls gone wild: hair simulation in brave. In *SIGGRAPH '12 Talks*, 22:1–22:1.

STAM, J., AND FIUME, E. 1993. Turbulent wind fields for gaseous phenomena. In *SIGGRAPH '93*, 369–376.

SZUMMER, M., AND PICARD, R. W. 1996. Temporal texture modeling. In *IEEE Intl. Conf. Image Processing*, vol. 3, 823–826.

THÜREY, N., KEISER, R., PAULY, M., AND RÜDE, U. 2006. Detail-preserving fluid control. SCA '06, 7–12.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH '02*, 665–672.

TURK, G. 2001. Texture synthesis on surfaces. In *SIGGRAPH '01*, 347–354.

WANG, J., TONG, X., LIN, S., PAN, M., WANG, C., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Appearance manifolds for modeling time-variant appearance of materials. In *SIGGRAPH '06*, 754–761.

WANG, L., YU, Y., ZHOU, K., AND GUO, B. 2009. Example-based hair geometry synthesis. In *SIGGRAPH '09*, 56:1–9.

WANG, H., HECHT, F., RAMAMOORTHI, R., AND O'BRIEN, J. 2010. Example-based wrinkle synthesis for clothing animation. In *SIGGRAPH '10*, 107:1–8.

WANG, H., , RAMAMOORTHI, R., AND O'BRIEN, J. F. 2011. Data-driven elastic models for cloth: Modeling and measurement. In *SIGGRAPH '11*, 71:1–11.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00*, 479–488.

WEI, L.-Y., HAN, J., ZHOU, K., BAO, H., GUO, B., AND SHUM, H.-Y. 2008. Inverse texture synthesis. In *SIGGRAPH '08*, 52:1–9.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. In *Eurographics '09 State of the Art Report*, 93–117.

XU, X., WAN, L., LIU, X., WONG, T.-T., WANG, L., AND LEUNG, C.-S. 2008. Animating animal motion from still. In *SIGGRAPH Asia '08*, 117:1–117:8.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. In *SIGGRAPH '05*, 965–972.