Pacific Graphics 2016
E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Volume 35 (2016), Number 7

Programmable Animation Texturing using Motion Stamps

A. Milliez[1,2†] M. Guay[2] M.-P. Cani[3] M. Gross[1,2] R. W. Sumner[1,2]

[1]ETH Zurich [2]Disney Research [3]Univ. Grenoble Alpes & CNRS (LJK), Inria

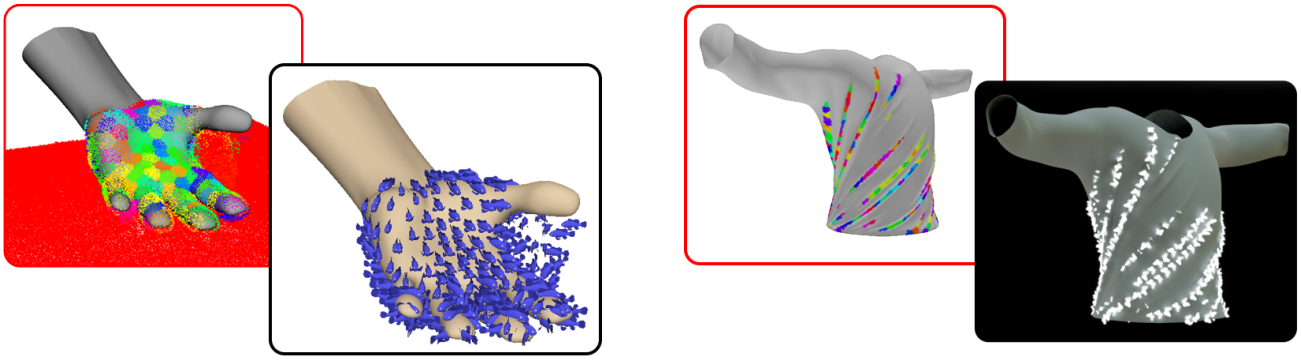

Figure 1: Our system provides a modular framework for authoring stylistic enhancements on complex animations such as fluids (left) or cloth (right). All or parts of the animation can be selected for stylization. The selected points are clustered and tracked over time. A stylization engine instantiates parameterized animations for each cluster, such as the fish and butterflies shown here that spawn when water is lifted off or when wrinkles form.

Abstract

Our work on programmable animation texturing enhances the concept of texture mapping by letting artists stylize arbitrary animations using elementary animations, instantiated at the scale of their choice. The core of our workflow resides in two components: we first impose structure and temporal coherence over the animation data using a novel radius-based animation-aware clustering. The computed clusters conform to the user-specified scale, and follow the underlying animation regardless of its topology. Extreme mesh deformations, complex particle simulations, or simulated mesh animations with ever-changing topology can therefore be handled in a temporally coherent way. Then, in analogy to fragment shaders that specify an output color based on a texture and a collection of properties defined per vertex (position, texture coordinate, etc.), we provide a programmable interface to the user, letting him or her specify an output animation based on the collection of properties we extract per cluster (position, velocity, etc.). We equip elementary animations with a collection of parameters that are exposed in our programmable system and enables users to script the animated textures depending on properties of the input cluster. We demonstrate the power of our system with complex animated textures created with minimal user input.

1. Introduction

The appearance of 3D surfaces can be controlled with 2D images through the well-known process of *texture mapping*. Local 2D parametrizations of a three dimensional object are packed in a *UV map* and exposed to artists, who can use digital painting tools to give a flat description of the appearance of the 3D model. Texture images can encode information such as color, normal perturbations, or material parameters. At render time, programmable shaders let users define the fragment colors of the rendered image as a function of the surface geometry and of its texture-mapped parameters. Texture mapping is especially powerful for locally repetitive surface appearance, where a pattern element can be defined once in texture space, and replicated over a surface through a carefully crafted UV map.

In the context of 3D animation, a wide range of tools of various

† e-mail:antoine@disneyresearch.com

complexity give artists fine control over the shape of a character, its animation and its appearance, letting them craft complex animations. However, stylizing animations with locally repetitive animated elements remains a tedious task. In particular, while modern simulation systems for fluids and cloth achieve a high degree of physical realism and deliver results that are often indistinguishable from reality, the creative vision for a computer animated film may call for a more stylized look. While some variations, such as the degree of turbulence in a fluid simulation or the apparent weight of a garment's fabric, are easily achieved through simulation parameter tuning, more dramatic art direction cannot be accommodated by changing simulation parameters. For example, the creative vision for an animated film may call for portions of a fluid simulation to be enhanced with an animated mesh that evolves over time as the fluid moves, or for the wrinkles in a cloth simulation to exhibit life of their own by spawning individual animated elements that are coherent with the evolving wrinkle patterns.

These highly art-directed effects call for animated elements that move and deform in a particular way based on the desired look, and existing tools are not designed to incorporate such animations into complex animations in a coherent and unified way. The complexity of the rich motion created by simulation systems makes this coherent stylistic adjustment a highly cumbersome task. As a result, animators must either abandon desired art-directed enhancements or, when budget allows, invest the laborious effort needed for specialized one-off effects.

Inspired by texture mapping, our work targets art-directed stylistic enhancements to complex animations using a novel algorithm for spatio-temporal clustering, and a programmable framework for animation instancing. Since complex animations such as simulations often deliver unstructured data that is independent from frame to frame, we apply our clustering algorithm to impose structure and temporal coherence on the selected points. Clusters are tracked over time and dynamically added and removed as the animation evolves. They provide information about the local behavior of the underlying animation, in analogy to UV maps that provide information about the local geometry of the underlying surface.

Finally, each cluster is used as input to a stylization engine that instantiates parameterized animation components called *motion stamps*. Each motion stamp represents a parameterized animated scene that can have geometry, motion, and other parametric variations. Our system connects parameter values associated with the clusters, such as position, size, velocity, or other feature values, to the motion stamp parameters so that the stamp animations are directly parameterized by the animation that should be textured. This concept is analogous to fragment shaders, that let users define fragment colors depending on the input geometry parameters. Taken together, this framework provides a flexible procedure for animators to texture animations with custom, art-directed content. Examples of our method used to enhance a fluid and a cloth simulations are shown in Figure 1.

Our main contribution is a modular framework for authoring stylistic enhancements of complex animations such as fluids and cloth. On a technical level, we contribute an artistically motivated spatio-temporal clustering method to establish coherence over time, and a parameterized instancing component based on the feature values. We demonstrate our system with several examples, including a water simulation that is represented by swimming fish and erupts into a flock of birds when different components of the fluid splash together and a wrinkle pattern on cloth whose high curvature values lead to butterflies that spawn and fly away.

2. Related work

Adding realistic details to simulations. A large body of work has focused on adding *realistic* details to existing simulations. A strategy in fluid simulations is to enhance a coarse velocity field with higher resolution sub-scale turbulence. Different sub-grid models of turbulence were introduced to computer graphics [KTJG08, SB08, PTC*10, NSCL08], modeling the transfer from coarse fluid velocity to fine scale turbulence. Other methods focus on specialized procedural techniques for generating spray, droplets and foam on height-field [MY97, CM10] and 3D particle-based simulations [OCv13, IAAT12]. These methods have succeeded at increasing the realistic details of a fluid simulation, but do not accommodate expressive and artistic styles of motion. Similarly, in cloth simulation, many works seek to upscale a coarse simulation with detailed wrinkles either procedurally [HBVMT99, KWH04, RPC*10], using data-driven techniques [dASTH10, WHRO10, KGBS11, ZBO13], or through explicit user direction [BMWG07] . These methods are compatible with our work, as we can offer animated stylistic enhancements on top of the simulation output to achieve particular artistic effects.

Gross control of simulation shape. Other work enhances the art-directability of simulations by focusing on gross control over the shape of the simulation itself. Research in this area allows artists to direct simulations of smoke [TMPS03, SY05], liquids [TKPR06, RTWT12], clouds [DKNY08], cloth [WMT06], or elastic solids [MTGG11, STC*12] to take on particular shapes, such as a cow made of smoke or a horse made of water. Rather than enforcing these large modifications of the shape of the simulation, our work aims to enhance simulations with animations that target specific art-directed effects. By focusing on a modular pipeline based on feature detection, our system can support a variety of enhancements that connect fluid features to animation instances.

Stylizing fluids with 2D exemplars. One technique to enhance fluids is to use 2D texture patches along the velocity field [Ney03, SMC04, BBRF14, JFA*15]. Texture advection [Ney03] consists in transporting the UV coordinates the texture, while limiting the distortion of the UVs over time as much as possible. An alternative is to consider small individual stencils as particles and splat them to produce the final look—as is done for cartoon smoke in [SMC04]. Similar in spirit to our work, they improve the coherence over time of the stylization by modulating the stencils based on particle properties. Recent techniques reduce the distortion with a synthesis process based on self-similarity when blending between textures [BBRF14, JFA*15]. While these methods allow creating impressive looks, they are focused on 2D texture exemplars while we target 3D stylistic enhancements and provide a general framework that supports both fluids and cloth. Additionally, the internal motion of the texture example is entirely guided by the texture synthesis optimization routine and the fluid velocity. In contrast,

our programmable framework allows our motion stamps to vary according to other features such as the radius of the cluster.

Dynamic texture synthesis. One way of facilitating the process of stylizing an animation with potentially non-realistic animations is to extend texture synthesis to the dynamic case. The basic idea is to repeat a pattern over a domain following principles of coherence and causality. Traditional 2D texture synthesis methods have been extended to the 3D domain with 3D mesh texture elements [MWT11] and then to the temporal domain [MWLT13]. Given a user-provided exemplar and an animated domain to texture (a 2D surface or a set of 1D strands), their goal is to automatically cover the domain with the exemplar while ensuring coherence in both the spatial and temporal dimensions. It is unclear how to adapt their example-based structural similarity to topologically varying domains such as liquids. In contrast, our method can synthesize coherent trajectories over topologically varying domains and provides artistic control over the instantiation in time.

Motion stamps. Dynamic stylization can be time consuming when performed manually. Hence previous works have investigated the idea of linking parametric motions to larger scale movements designed by the animator [CHG14, MNB*14]. In these works, 2D or 3D animated stamps are controlled by sketching their positions or trajectories. Chevalier et al. propose an interface for designing 2D flows and controlling the parameters of overlaid 2D texture elements moving along the flow [CHG14]. Milliez et al. allow the user to sketch the position of 3D animated meshes and control additional parameters such orientation using gesturing strokes [MNB*14]. We adopt the same "motion stamp" concept proposed by Milliez et al. and extend it to work in the context of simulation enhancement. Neither method addresses the problem of feature selection or trajectory extraction in the context of complex fluid or cloth simulations.

Clustering. A core component of our system addresses temporally coherent clustering of 3D points respecting a user-specified size. Data clustering is a research field in itself, and Xu and Wunsch provide a survey of clustering algorithms [XW05]. We are in particular inspired by spatial hierarchical clustering techniques, of which a detailed review is available in [HKT01]. In order to support an artist-inspired workflow, we favor an application-oriented clustering and tracking algorithm. Related techniques for spatio-temporal clustering target data such as moving pedestrians or vehicles [LHY04, HP04, NP06, AAR*09, JLO07, JYZ*08, KMNR10]. One of the core differences with our work is that these techniques are designed to track grouped points over time, while we support more general feature-based space-time clustering where the sets of points constituting the features being clustered are allowed to change over time.

3. Overview

Our method enables artists to texture complex animations by extracting temporally coherent structures that locally describe these animations. The artists can then specify a correspondence between the characteristics of the extracted structures and parameters of user-defined animation elements called *motion stamps*. Our method allows one to specify the size of the structures that locally describe the underlying animation, and is applicable even when the set of animated points change over time.

When analyzing artistic stylization in hand-drawn animations, we observe that animators often stylize visually salient areas of an animation in order to emphasize or exaggerate their qualities. Based on this observation, we let artists choose to texture a whole animation or only parts of it. We provide a UI letting artists specify relevant areas based on features such as vertex curvature, position, or velocity. Texturing salient parts of an animation such as waves rolling over an ocean or cloth wrinkles sliding along the surface of a garment is made possible thanks to our structure extraction that is independent of the topology of the underlying animation.

Our solution includes two main components:

- **Space-time clustering.** Since the points that define a simulation are often unstructured without temporal consistency, we propose a novel spatio-temporal clustering method to establish coherence in the simulation data. Clusters may appear or disappear, split or merge over time, triggering the appearance or disappearance of motion stamps when and where needed, based on the evolution of the animation.
- **Motion stamp control.** For each cluster, a stylization engine instances a motion stamp whose parameters are connected to the detected feature values. Our method allows one to control time cycles of motion stamps, temporal warping, changes in scale and orientation, as well as deformation parameters such as the amount of stretch or twist of an animated shape.

The parts of our work are respectively described in the next two sections.

4. Dynamically Coherent Clustering

Given a set of 3D points $\mathbf{P}_s$ defining all or parts of an animation, our goal is to create a uniform distribution of clusters that meet the desired radius of the animated stamps. While many methods targeting spatial clustering exist (see Section 2), they do not allow directly specifying the radius of the computed clusters. Since the clusters will be used as support to instantiate motion stamps, it is crucial that our method provides control over the size of those clusters. We let users specify a radius and a tolerance threshold, defining the range of acceptable cluster sizes. This enables the detection of clusters whose size varies over time. The second challenge is to create a dynamically coherent clustering, i.e. the ability to have clusters that move over time without temporal discontinuities.

To allow the user to specify a range of acceptable cluster radii, we employ a hierarchical clustering strategy (described below), and track the clusters over time by mixing advection and dynamic re-clustering. In other words, we initialize a clustering at the beginning of the animation, and then advect the clusters over time by sampling the velocities of the clustered points, and re-clustering areas that hold new features that appeared over time.

4.1. Spatial Clustering

Our spatial clustering routine is a hierarchical clustering which results in a tree graph with bottom leafs holding a cluster with a single 3D point, and the root node a single cluster containing all the

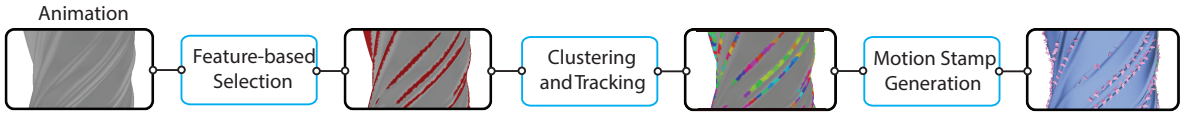

Figure 2: Overview of our pipeline.

points. Each node describes a set of points and its radius, a node's parent always has a larger radius, while a node's child always has a smaller one. Given a range of acceptable cluster radii, we can parse the tree and select clusters at the appropriate hierarchical level.

To create the hierarchical clustering structure, we start by creating a cluster for each 3D point, setting its radius to 0. Then we iteratively merge the two nearest clusters into a larger one until all clusters have been merged into a root node containing all 3D points (see Figure 3). In practice, building the whole hierarchy is not necessary since the root cluster will often be larger than the maximum desired radius. Instead, we keep track of the radius of each cluster at each iteration. If two clusters get merged and the new cluster's radius is above the maximum allowed value, we stop this iteration and look at the two clusters getting merged. If a cluster has its radius larger than the minimum authorized radius, then it is tagged as a valid cluster and will be propagated further down our pipeline.

Note that this algorithm computes clusters only if their radii are within the acceptable range and therefore does not necessarily provide a partition of the selected 3D points. For instance, water droplets smaller than the prescribed radius and too far apart will merge into a cluster with an undesirable large clusters, and they will be discarded since they are too small, which complies to the user-specified cluster size.

4.2. Space-time clustering

Given two clusterings of the animation at two different moments in time, the correspondence between the two sets of clusters can be ambiguous; it can be hard to know which cluster at a time step corresponds to which cluster at the next step. One possible strategy to overcome this ambiguity is to perform a hierarchical space-time clustering of the animated points. Such an algorithm would be similar to the spatial clustering described in Section 4.1. Each particle at each time frame would be assigned to one cluster, and the two closest clusters in both time and space would be iteratively merged. While our experiments have shown that this method provides satisfying results, the cluster hierarchy computed over a complex animation is very memory and time consuming, which is not desirable when interactively manipulating animations.

Our strategy is instead to create a first spatial clustering at time t_k, and to then advect the clusters, predicting their positions at time t_{k+1}. We can use the predicted cluster positions to efficiently compute a clustering at t_{k+1} as shown in Figure 4, and iterating this step results in a fast algorithm producing temporally coherent cluster data.

Advecting clusters requires the definition of cluster velocity. Two velocities can be defined for a specific cluster: the average of its element (vertex or particle) velocities, or the velocity obtained by finite differences over the cluster's successive positions. We found in our experiments that both definitions can be appropriate depending on the application. When clustering wrinkles moving over a shirt, the vertices constituting the wrinkles follow a trajectory mostly along the normal direction to the shirt, while the wrinkles slide tangentially to the cloth. When clustering particles in a flowing river simulation however, a cluster should move along the water and follow the particle velocities. Ignoring the particle velocities could result in static clusters, that would be surrounded by particles at each time step while having a null velocity.

This observation lead us to offer the user the choice to compute clustering based on point velocities or on cluster velocity. When choosing to use cluster velocities, we initialize the cluster velocities after the initial spatial clustering to be zero, they are then updated as the spatio-temporal clustering progresses.

During spatio-temporal clustering, after any time step t_k, we estimate each cluster's position at time t_{k+1} by advecting its previous position using the velocity type chosen by the user. We use a priority queue to store all the pairs (estimated cluster position, 3D point) at time t_{k+1} and use it to successively assign 3D points to their closest estimated cluster center. Each assignment is discarded if it conflicts with the user specifications: if the assignment would bring the cluster radius over the maximum authorized one, or if the displacement of the cluster would increase the cluster velocity over the maximum authorized value.

Once the assignment step done, the remaining points (features appearing in the animation, or points that would have enlarged existing clusters beyond user specifications) are clustered using our spatial clustering method and added to the set of computed clusters. Note that since this spatial hierarchical clustering is only computed when and where needed, the computation time of our method does not suffer much from rebuilding a cluster hierarchy.

Finally, we detect splits and merges between clusters. Using the initial velocity of a cluster, we backtrack its first position one frame in time, and look at the backtracked position's neighborhood. If the existing cluster closest to that position is within $\delta t * max_{vel}$, with max_{vel} the maximum authorized cluster velocity, we mark this as a split event: the already existing cluster has its own trajectory, but splits to spawn the new one. We apply a similar process to detect merges at the end of a cluster's life. Later in the pipeline, the user can chose to use that split/merge knowledge for stylization.

5. Programmable motion stamps parameterization

Our goal is to texture animations with animated motion. In the previous section, we described how to obtain dynamic clusters of preselected size moving along the animation. Each cluster provides local spatio-temporal information about local parts of the animation.

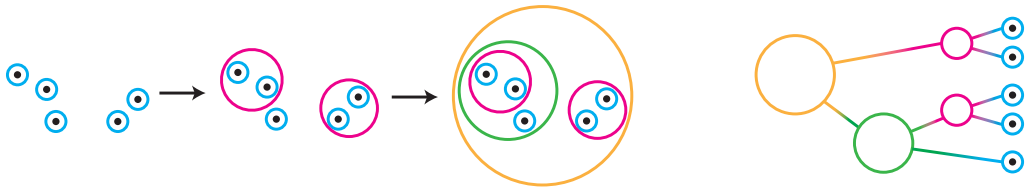

Figure 3: Each 3D point is assigned to an individual cluster of radius 0 (blue). We iteratively merge the two closest clusters until we reach one cluster covering all the data (orange). The cluster hierarchy (right) can then be used to get clusters with a desired radius.

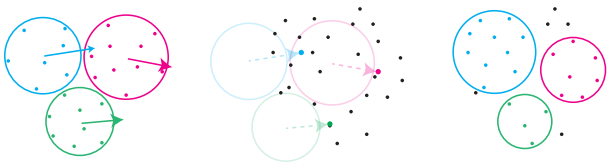

Figure 4: The cluster velocities at time t_k (left) are used to predict the cluster positions at t_{k+1} (center). Assigning 3D points to the closest predicted centers (right) yields a partial clustering at t_{k+1}. The non-clustered points (black) are then clustered in an independent step.

We build an analogy with traditional texture mapping. To render a textured surface, it is discretized into faces, bounded by vertices and edges. Each vertex can carry information such as its position, color, normal, or texture coordinate, the latter helping to define local 2D parameterizations of the 3D surface. By writing a vertex shader, the user defines how 3D vertices are projected onto the screen. Then by writing a fragment shader, the user can specify an output color, defined by four parameters (red, green, blue, alpha), depending on the information carried by the vertices that are projected onto the screen.

In our framework, the base animation is discretized into clusters, that carry information sampled from their components: their cluster velocity and component velocity, their position, radius, normal, etc. The user can write scripts that we call *motion shaders* to parameterize animation instances. We use the formalism introduced by Milliez et al. [MNB*14] and define elementary animations as *motion brushes*. A motion brush can be as flexible as a procedurally defined shape and as rigid as a baked 3D animation. A motion brush is analogous to a 2D texture. Instances of motion brushes in a scene are called *motion stamps*, and carry a set of parameters such as position, scale, orientation, opacity multiplier, animation speed, etc. When writing motion shaders, users define the values of a single motion stamp's parameters based on a single input cluster, the same way fragment shader programs define the color of a single fragment at a time (Figure 5).

As described in 4.2, our space-time clustering detects if a cluster has splitted from or is merging into another cluster. That information is available in our motion shader framework and the user can read the parameters of the clusters splitted from or merged into. The motion shader can ignore it, or can adapt the instantiated motion stamp's parameters accordingly. For example in the provided supplemental videos, the butterly motion shader ignores those events and every butterfly appears with a rising animation, and disappears by flying away. The fish motion shader however takes note of the split information. When a cluster appears splitting from another one, the motion shader aligns the motion stamp parameters with the older cluster for a few frames. The result animation does not result in a *popping* fish animation, but shows fish splitting in two.

Our motion stamp engine then runs the user-specified motion shader on each computed cluster and automatically instantiates motion stamps within the scene. The attached video shows various examples of textured animations including changes between different motion stamps during the life of a cluster.

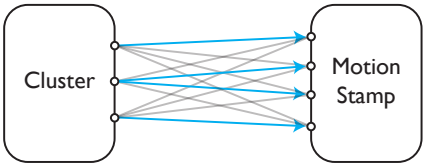

Figure 5: Users can choose how the cluster data is used to parameterize the motion stamps.

6. Results

We implemented our pipeline as a plugin for Autodesk Maya ([May16]), in order to take advantage of the production pipeline professional artists are acquainted with. Both cloth and liquid animations seen in this paper were simulated using technology natively available in Maya.

The user can open a scene, select a simulation object, and load our plugin. When desired, we provide a visualisation of the selected parts of an animation by coloring the selected particles or the mesh vertices. Our plugin provides UI controls to adjust thresholds for a variety of vertex and particle parameters, letting users select parts of the animation depending on that information. In the case of complex feature selection, the user can even script the value of the score for each 3D point.

The user starts the clustering procedure by first using UI elements to specify the desirable cluster radius and the tolerance for admissible cluster radii, as well as the maximum authorized cluster velocity. Our clustering algorithm is fast and if the user is not satisfied with the result, he or she can change the parameters and run the clustering algorithm again. We present some performance results in Table 1.

Finally, the user can edit motion shaders that parametrize motion

	nPoints	nFrames	nClust.	time
Water Fish	134,651	127	238	6s
Water Bird Drops	132,120	130	435	8s
Butterfly Shirt	457,748	215	2578	38s

Table 1: For each result, the number *nPoints* of 3D points clustered in the animation, over *nFrames* frames, into *nClust* clusters. Computation time reported in the last column.

stamps depending on the cluster data. The collection of already implemented shaders is exposed to the user, who can decide to reuse previous shaders, or to write a new one.

Figure 6 shows a simple toy example: a simulated cloth rectangle is held by two handles that exerce a vertical motion. The cloth simulation produces a wave motion. The goal here was to cover the cloth with butterflies, that take off as the wave motion propagates along the cloth.

The user could define the selection based on the y position of vertices, as well as the animation time. The desired cluster radius was defined as the desired butterfly radius, and the maximum authorized velocity for each cluster was set relatively high (3 times the cluster radius) to ensure proper tracking during the fast cloth movement. Once the clustering was performed, the user could use two motion brushes describing butterfly animations: *idle* and *take off*, and script the motion stamp instantiation through the following script (presented here as pseudo code).

```
while (cluster is alive)
{
  stamp.position ← cluster.center
  stamp.scale ← cluster.radius / stamp.radius
  stamp.upDirection ← (0,1,0)
  loop play idle animation
}
for (frame = cluster.end +1
     to cluster.end + take off duration)
{
  keep the same position, scale, orientation
  play take off animation once
}
```

For the wrinkle stylization seen in Figure 1, the wrinkles on the T-shirt were selected based on the vertex curvatures (wrinkles exhibit a high curvature) and positions (so as to ignore the wrinkles on the arms). Butterfly motion stamps were then scripted to follow the cluster centers, first playing a spawning animation once, followed by a looped idle animation. When clusters disappear, the stamps were scripted to keep the same position and play a take off animation once.

We also created different liquid simulation scenarios. First, as shown in Figure 7, water is thrown with a high initial horizontal velocity, and is projected in the air by a bump on the ground. The user could select various parts of the animation to cluster them, and use the clusters to texture the liquid with animated fish. In a similar fashion to previous results, the fish radii are dictated by the cluster sizes, while the fish positions and orientations are parametrized by the cluster positions and velocities respectively. Replacing the fish by another type of motion stamps like the birds seen in Figure 7 is simply done by changing the type of motion stamp used in the motion shader.

Motion stamp parameters are not limited to the ones used in our results so far. We demonstrate the flexibility of our method by parametrizing the deformation of spring-like shapes. Each spring's stretch factor is parametrized by the underlying cluster's velocity, its bending by the curvature of the cluster's trajectory, and its revolution angle is scripted to increase over time. Considering the spring shape as a 1-frame animation, this example fits in the formal definition of motion stamps described in [MNB*14]. However, we are essentially producing the shape animation on the fly depending on the cluster data, instead of reusing animated examples.

Finally, we show more complex examples of animation texturing. In the first, two liquids collide and create a splash, as shown in Figure 8. Selecting particles based on their position and velocity, the user could successively stylize the two sets of fast moving particles aiming at the collision area. By setting the same desired cluster properties on the two sets, and texturing them using the fish parametrization described above, the user could easily instantiate two crowds of fish with similar radii aiming at a frontal collision. Then, selecting particles based on their position and time of existence, the user could isolate the particles that constitute the splash. After clustering them, a bird parameterization similar to the previously described one was used, thus texturing the splashing drops with flying birds. We show a composition of the fish and birds, rendered along with the meshed particles.

The last example seen in Figure 1 shows a hand lifting water from a pond. After selecting particles based on their vertical position, the user textured the water lifted off by the hand, turning the water into fish as it is being manipulated.

7. Conclusion

In this work, we have devised an animation texturing framework that allows animators to author stylizations on top of complex animations. Our programmable approach allows artists to select salient areas of basic animations, as well as to link the stylization behavior and motion of the instantiated motion stamps to the attributes of the underlying animation. Our clustering algorithm can be efficiently implemented without copying any point data, and we have shown results of its performance. However, its sequential design makes a performance improvement through parallelization challenging. We have implemented our work within Autodesk Maya for artistic convenience: it provides all the necessary tools for animation authoring, and our method integrates well within that workflow. However, while motion shaders are simple programs and our motion shader processor can be straightforwardly parallelized, the time spent processing the shaders is negligible compared to the processing time required by the Maya API to read motion brush files from disk and to set motion stamp parameters and keyframes.

While our method lets artists semi-automatically create complex arrangements of animations that would otherwise be very challenging to produce by hand, we could already identify interesting areas for future work.

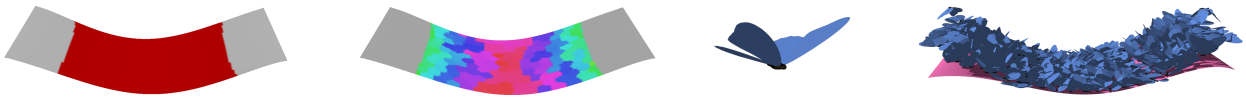

Figure 6: From left to right: original animation with colored selection of vertices, clustering over the selected vertices, motion stamp, textured animation.

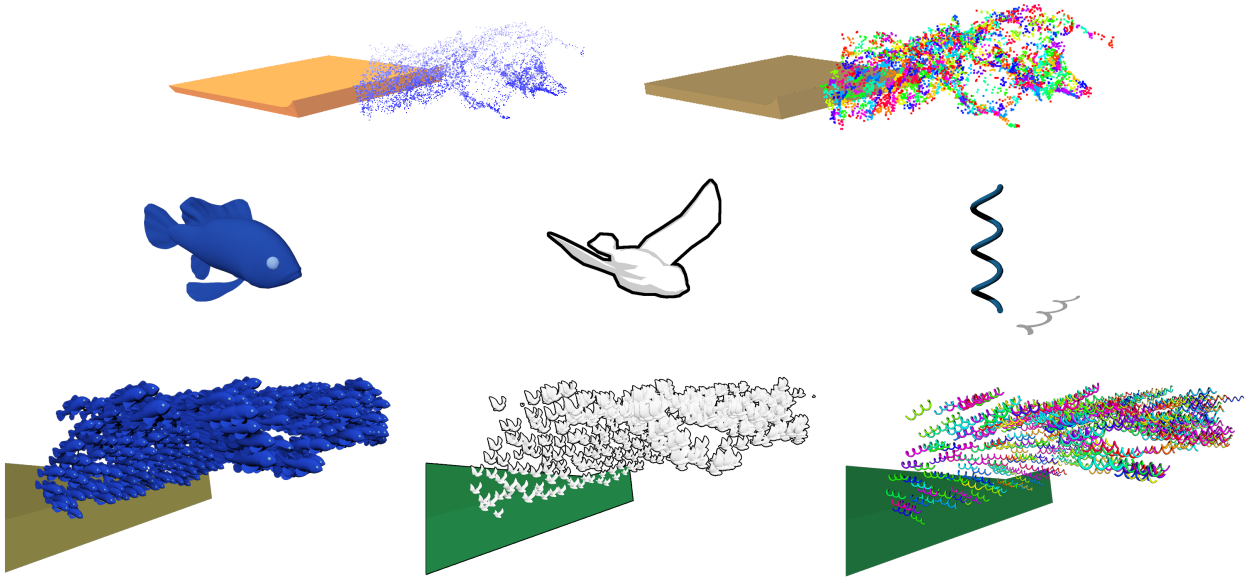

Figure 7: From top to bottom: the original animation and a clustering obtained using our algorithm. The motion stamps used to create the present results. Textured results of our algorithm.

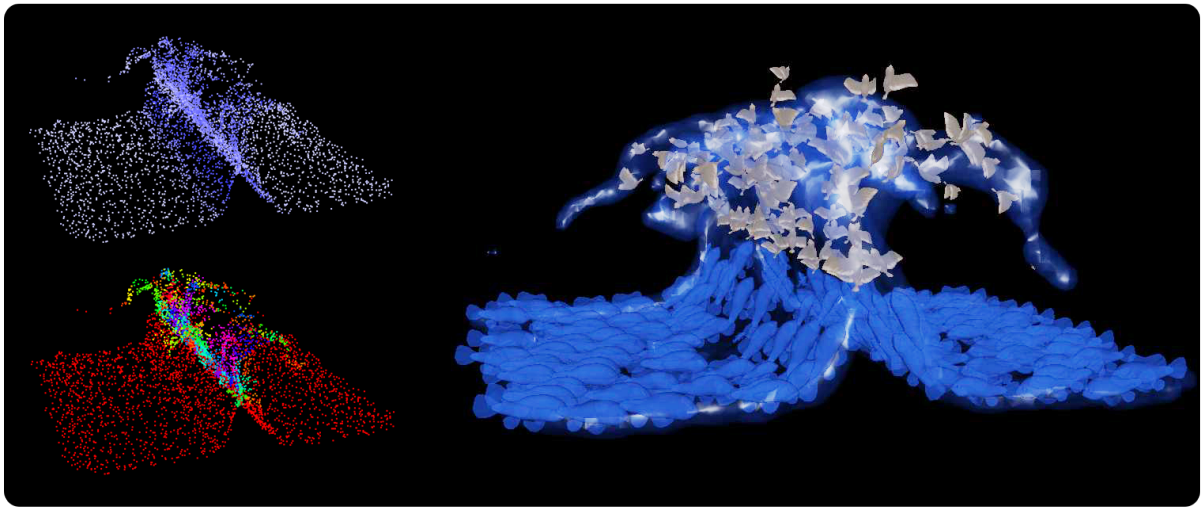

Figure 8: Original, clustered data and textured animation.

While we have demonstrated our implementation on cloth and particle simulations, we believe that our method can easily extend to other types of animated data. Eulerian simulations for example, are typically used to represent smoke or fire. Each cell in a Eulerian simulation grid provides a velocity vector that we could rely on in our spatio-temporal clustering algorithm, and scalar data such as density or temperature could be used to parameterize motion stamps.

While our method instantiates and tracks clusters based on the underlying animation, there is so far no interaction between the clusters. Using the detected splits and merge informations, the user

can decide to temporarily align rigidly new appearing clusters to existing ones, giving the impression that instantiated stamps are splitting, as is seen in the accompanying video. However, one could imagine designing, in addition to a motion stamp animation, a motion stamp merging animation, that would be played when two clusters merge. While our current method rigidly transforms a motion stamp to instantiate it, instantiating merging motions would require a non-rigid deformation of the motion stamps' trajectories, which we do not currently support.

Our clustering is based solely on point positions, which is an acceptable limitation for the results targeted in this paper. However, one could envision use-cases where more metrics would be relevant. For example in the complex case of particles moving in different directions while being in the same vicinity, one might want to cluster them based on velocity. It is unclear whether exposing several metrics to the user would result in an intuitive and predictable interface, which would be an interesting question to explore when extending this work.

Another feature that could contribute to interesting results would be a remeshing step to merge the animation mesh and the instantiated motion stamps. While our method is already satisfying for instantiating fish in water for example, it is not well suited to changing the shape of the water. If a user wants to select every wave in an ocean to change their shape, the current approach would be limited to instantiating new wave shapes on top of the existing ones. Computing a smooth mesh over the animation and the instantiated geometry would provide more appealing results.

Finally, we feel like our clustering algorithm could be used for level of detail stylization of 3D scenes. Indeed, by incorporating a camera model into our clustering, one could imagine seeing more clusters appear on a scene as the camera zooms in. The radius occupied by each cluster on the screen could also be used to parameterize the motion stamp animations.

References

[AAR*09] ANDRIENKO G., ANDRIENKO N., RINZIVILLO S., NANNI M., PEDRESCHI D., GIANNOTTI F.: Interactive visual clustering of large collections of trajectories. In *Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on* (Oct 2009), pp. 3–10. 3

[BBRF14] BROWNING M., BARNES C., RITTER S., FINKELSTEIN A.: Stylized keyframe animation of fluid simulations. *NPAR 2014, Proceedings of the 12th International Symposium on Non-photorealistic Animation and Rendering* (June 2014). 2

[BMWG07] BERGOU M., MATHUR S., WARDETZKY M., GRINSPUN E.: Tracks: Toward directable thin shells. *ACM Trans. Graph. 26*, 3 (2007). 2

[CHG14] CHEVALIER F., HABIB R., GROSSMAN T.: DRACO: Bringing Life to Illustrations with Kinetic Textures. *ACM CHI Conference ...* (2014), 351–360. 3

[CM10] CHENTANEZ N., MÜLLER M.: Real-time simulation of large bodies of water with small scale details. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2010), SCA '10, Eurographics Association, pp. 197–206. 2

[dASTH10] DE AGUIAR E., SIGAL L., TREUILLE A., HODGINS J. K.: Stable spaces for real-time clothing. *ACM Trans. Graph. 29*, 4 (July 2010), 106:1–106:9. 2

[DKNY08] DOBASHI Y., KUSUMOTO K., NISHITA T., YAMAMOTO T.: Feedback control of cumuliform cloud formation based on computational fluid dynamics. *ACM Trans. Graph. 27*, 3 (2008). 2

[HBVMT99] HADAP S., BANGERTER E., VOLINO P., MAGNENAT-THALMANN N.: Animating wrinkles on clothes. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years* (Los Alamitos, CA, USA, 1999), VIS '99, IEEE Computer Society Press, pp. 175–182. 2

[HKT01] HAN J., KAMBER M., TUNG A. K. H.: Spatial clustering methods in data mining: A survey. In *Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS* (2001), Miller H. J., Han J., (Eds.), Taylor and Francis. 3

[HP04] HAR-PELED S.: Clustering motion. *Discrete & Computational Geometry 31*, 4 (2004), 545–565. 3

[IAAT12] IHMSEN M., AKINCI N., AKINCI G., TESCHNER M.: Unified spray, foam and air bubbles for particle-based fluids. *Vis. Comput. 28*, 6-8 (2012), 669–677. 2

[JFA*15] JAMRIŠKA O., FIŠER J., ASENTE P., LU J., SHECHTMAN E., SÝKORA D.: Lazyfluids: Appearance transfer for fluid animations. *ACM Trans. Graph. 34*, 4 (July 2015), 92:1–92:10. 2

[JLO07] JENSEN C. S., LIN D., OOI B. C.: Continuous clustering of moving objects. *Knowledge and Data Engineering, IEEE Transactions on 19*, 9 (Sept 2007), 1161–1174. 3

[JYZ*08] JEUNG H., YIU M. L., ZHOU X., JENSEN C. S., SHEN H. T.: Discovery of convoys in trajectory databases. *Proc. VLDB Endow. 1*, 1 (Aug. 2008), 1068–1080. 3

[KGBS11] KAVAN L., GERSZEWSKI D., BARGTEIL A. W., SLOAN P.-P.: Physics-inspired upsampling for cloth simulation in games. *ACM Trans. Graph. 30*, 4 (July 2011), 93:1–93:10. 2

[KMNR10] KISILEVICH S., MANSMANN F., NANNI M., RINZIVILLO S.: Spatio-temporal clustering: a survey. In *Data Mining and Knowledge Discovery Handbook*, Maimon O., Rokach L., (Eds.). Springer US, 2010, pp. 855–874. 3

[KTJG08] KIM T., THÜREY N., JAMES D., GROSS M.: Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008 27*, 3 (2008), 50. 2

[KWH04] KIMMERLE S., WACKER M., HOLZER C.: Multilayered wrinkle textures from strain. In *VMV* (2004), Girod B., Magnor M. A., Seidel H. P., (Eds.), Aka GmbH, p. 225Ű232. 2

[LHY04] LI Y., HAN J., YANG J.: Clustering moving objects. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2004), KDD '04, ACM, pp. 617–622. 3

[May16] MAYA: Autodesk Maya, 2016. http://www.autodesk.com/maya. 5

[MNB*14] MILLIEZ A., NORIS G., BARAN I., COROS S., CANI M.-P., NITTI M., MARRA A., GROSS M., SUMNER R. W.: Hierarchical motion brushes for animation instancing. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2014), NPAR '14, ACM, pp. 71–79. 3, 5, 6

[MTGG11] MARTIN S., THOMASZEWSKI B., GRINSPUN E., GROSS M.: Example-based elastic materials. *ACM Trans. on Graphics (Proc. SIGGRAPH) 30*, 4 (2011), 72:1–72:8. 2

[MWLT13] MA C., WEI L.-Y., LEFEBVRE S., TONG X.: Dynamic element textures. *ACM Transactions on Graphics 32*, 4 (2013), 1. 3

[MWT11] MA C., WEI L.-Y., TONG X.: Discrete element textures. *ACM Transactions on Graphics 30*, 4 (2011), 1. 3

[MY97] MOULD D., YANG Y.-H.: Modeling water for computer graphics. *Computers & Graphics 21*, 6 (1997), 801–814. 2

[Ney03] NEYRET F.: Advected textures. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 147–153. 2

[NP06] NANNI M., PEDRESCHI D.: Time-focused clustering of trajectories of moving objects. *Journal of Intelligent Information Systems 27*, 3 (2006), 267–289. 3

[NSCL08] NARAIN R., SEWALL J., CARLSON M., LIN M. C.: Fast animation of turbulence using energy transport and procedural synthesis. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2008 27*, 5 (2008). 2

[OCv13] ONDERIK J., CHLÁDEK M., ĎURIKOVIČ R.: Sph with small scale details and improved surface reconstruction. In *Proceedings of the 27th Spring Conference on Computer Graphics* (New York, NY, USA, 2013), SCCG '11, ACM, pp. 29–36. 2

[PTC*10] PFAFF T., THÜREY N., COHEN J., TARIQ S., GROSS M.: Scalable fluid simulation using anisotropic turbulence particles. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2010 29*, 6 (2010), 174. 2

[RPC*10] ROHMER D., POPA T., CANI M.-P., HAHMANN S., SHEFFER A.: Animation wrinkling: Augmenting coarse cloth simulations with realistic-looking wrinkles. In *ACM SIGGRAPH Asia 2010 Papers* (New York, NY, USA, 2010), ACM, pp. 157:1–157:8. 2

[RTWT12] RAVEENDRAN K., THUEREY N., WOJTAN C., TURK G.: Controlling liquids using meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2012), SCA '12, Eurographics Association, pp. 255–264. 2

[SB08] SCHECHTER H., BRIDSON R.: Evolving sub-grid turbulence for smoke animation. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation* (2008). 2

[SMC04] SELLE A., MOHR A., CHENNEY S.: Cartoon rendering of smoke animations. In *Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering* (New York, NY, USA, 2004), NPAR '04, ACM, pp. 57–60. 2

[STC*12] SCHUMACHER C., THOMASZEWSKI B., COROS S., MARTIN S., SUMNER R., GROSS M.: Efficient simulation of example-based materials. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2012), SCA '12, Eurographics Association, pp. 1–8. 2

[SY05] SHI L., YU Y.: Controllable smoke animation with guiding objects. *ACM Trans. Graph. 24*, 1 (Jan. 2005), 140–164. 2

[TKPR06] THÜREY N., KEISER R., PAULY M., RÜDE U.: Detail-preserving fluid control. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), SCA '06, Eurographics Association, pp. 7–12. 2

[TMPS03] TREUILLE A., MCNAMARA A., POPOVIĆ Z., STAM J.: Keyframe control of smoke simulations. *ACM Trans. Graph. 22*, 3 (July 2003), 716–723. 2

[WHRO10] WANG H., HECHT F., RAMAMOORTHI R., O'BRIEN J. F.: Example-based wrinkle synthesis for clothing animation. *ACM Trans. Graph. 29*, 4 (July 2010), 107:1–107:8. 2

[WMT06] WOJTAN C., MUCHA P. J., TURK G.: Keyframe control of complex particle systems using the adjoint method. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2006), The Eurographics Association. 2

[XW05] XU R., WUNSCH II D.: Survey of clustering algorithms. *Trans. Neur. Netw. 16*, 3 (May 2005), 645–678. 3

[ZBO13] ZURDO J. S., BRITO J. P., OTADUY M. A.: Animating wrinkles by example on non-skinned cloth. *IEEE Trans. Vis. Comput. Graph. 19*, 1 (2013), 149–158. 2