Image-based Authoring of Herd Animations

Pierre Ecormier-Nocca[1], Julien Pettré[2], Pooran Memari[1], Marie-Paule Cani[1]
[1]LIX, École Polytechnique, CNRS [2]Inria

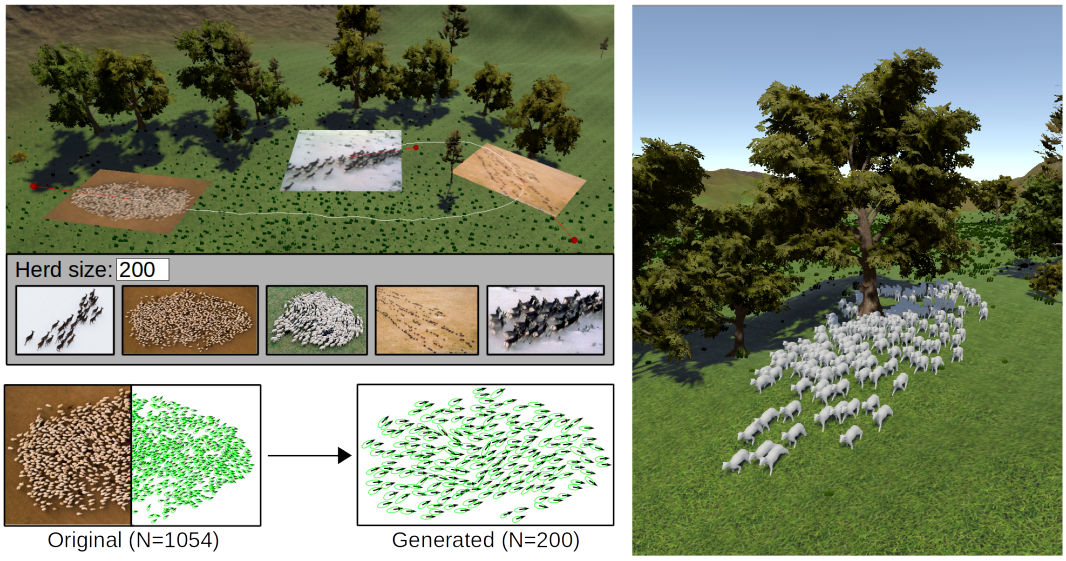


Figure 1: To create a herd animation, the user selects the desired visual aspects over time on photos, by placing them as key-frames on the terrain (top left). The analysis-synthesis method generates, for each photo, a visually similar key-herd with the target number N of animals (bottom left). The animation is then generated (right), thanks to an automatic labelling process to pair the animals of successive key-herds.

Abstract

Animating herds of animals while achieving both convincing global shapes and plausible distributions within the herd is difficult using simulation methods. In this work, we allow users to rely on photos of real herds, which are widely available, for key-framing their animation. More precisely, we learn global and local distribution features in each photo of the input set (which may depict different numbers of animals) and transfer them to the group of animals to be animated, thanks to a new statistical learning method enabling to analyze distributions of ellipses, as well as their density and orientation fields. The animated herd reconstructs the desired distribution at each key-frame while avoiding obstacles. As our results show, our method offers both high level user control and help towards realism, enabling to easily author herd animations.

Keywords: image-based animation, distribution synthesis, crowd simulation.

Introduction

While methods for easing the generation of complex virtual worlds widely spread in the last decade - leading to impressive results with detailed terrains as well as plausible distributions of rocks and vegetation - much less attention was paid so far to animal life. Yet, populating virtual worlds not only with vegetation blowing in the wind and a few birds, but also with moving groups of ground animals, is mandatory to get a lively result.

This work tackles the authoring of herd animations. The key idea is to enable intuitive authoring and enhanced realism through visual analogies with real photos. More precisely, we allow the user to key-frame the animation of a herd by copy-pasting a series of photos over the terrain using a pipette tool. In addition to the rough trajectories they define, each photo is used as a model for the local shape, distribution and density of the animals within the herd.

The input photos may indeed show quite different numbers of animals. Therefore, our solution relies on a new statistical method to analyse the photos independently from animals count and then synthesize a visually similar herd while accounting for the target number of animals, their size and local obstacles avoidance. In addition to the statistical distributions governing the relative distance between animals (modeled as oriented ellipses), our model extracts and reproduces two higher level features from each photo, namely the herd's density map (also defining its global shape) and the local orientation map for animals within the herd. If desired, these features can be edited at high level by the user using brush strokes.

Lastly, we propose a simple method to generate the individual trajectories of the animals from a herd key-frame to the next. It includes an automatic best-pairing mechanism between animals in successive key-herds and a microscopic simulation enabling animals to avoid collision while following the trajectory and having the desired relative motion in herd frame.

In summary, our three main contributions are:

- An intuitive authoring tool for key-framing herds, using the concept of pipette tool for copy-pasting the herd visual aspect from a photo and providing additional editing brushes;
- A new, two-levels method enabling to analyze both global herd features and local distributions of animals on images, and to synthesize visually similar ones with a given, target number of animals;
- A layered model using both the herd global trajectory and the desired relative motion within the herd to generate the animation between successive key-frames.

Related work

Animating animals: Microscopic simulation has long been the standard method for animating small groups of ground creatures such as herds of animals (see [1] for a survey). Reynold's pioneer work on Boids [2, 3] introduced three interaction forces of different ranges to respectively model cohesion, relative orientation and collision avoidance within a flock. Subsequent improvements, validated on groups of humans, include velocity-based [4] and vision-based [5]. Recent work also tackled very specific animal behavior such as insect swarms [6, 7]. Despite of their ability to generate nice, plausible motion, microscopic methods are user intensive since many trials and errors may be necessary to tune parameters and author an animation. While some methods can be used to animate flocks [8, 9] or pedestrian groups [10, 11] matching specified shapes, control over the distribution of elements within the shapes is not covered by the scope of these papers. In this work, we only use microscopic simulation to interpolate between herd key-frames, enabling the user to transfer the visual aspect of herds from photos.

Image-based animation: In the case of humans, trajectories extracted from videos of crowds were used to learn parameters of microscopic models [12, 13], enabling to facilitate their tuning.

These methods however involved either a large amount of manual labour or complex extraction methods, on top of requiring multiple frames of video data for each situation or crowd type. Note that such video data, showing the same creatures during a number of consecutive frames, are much more difficult to acquire for animals. In contrast, our method is based in the analysis of a few static pictures of herds which may represent different numbers of animals, making data very easy to acquire.

Analyzing real images was already used for animal animation, but in the context of key-framing the motion of a single animal, ether from a video [14] or from a static picture of a herd [15]. It was never applied, to our best knowledge, to learn the visual features of herds such as global shapes, animal distributions, den-

sity and relative orientations, the problem we are tackling here.

Learning and synthesizing distributions: Among recent authoring methodologies for virtual worlds, a strong interest was recently shown on learning statistical distributions of elements (such as rocks or vegetation) from exemplars, the latter being either user-defined, extracted from photos, or from simulation results. After learning using a pipette tool, interactive brushes were used to control synthesis over full regions of a terrain [16, 17]. In parallel, continuous Pair Correlation Functions (PCF) were introduced for more robustly handling statistical learning and synthesis [18, 19].

Although recent improvements tackled distributions of 2D shapes instead of points, the existing methods are not applicable to analyzing herd pictures: Firstly, most of them only handle non-overlapping shapes [20, 21], while animals within herds are often in close contact. The only method explicitly handling contacts and overlaps [22] is limited to disks and is not suited to our data, where animals are anisotropic and oriented. Secondly, all current methods tackled the analysis of a supposedly uniform distribution of elements. In contrast, the overall shape of a herd is important and the density of animals usually locally varies depending on distance to border. Lastly, the distribution of orientations within a herd is also an important visual feature. Therefore, we introduce our own analysis synthesis method for herds, based on the combination of PCFs with density and orientation maps.

Overview of the Authoring tool

We present a method for authoring herd animations based on data extracted from still photos.

Authoring interface

Our interactive tool enables the user to upload a virtual world which may contain a non-flat terrain with obstacles such as trees and rocks (top left of Figure 1). The user then choose a type of animal in a library, sets their maximal maximal speed V_{max} (which can be extracted if desired from zoological information) and their number N in the herd to be animated.

Our framework is inspired by traditional animation principles, namely the key-framing and interpolation workflow. In the reminder of this section, the key-frames correspond to important locations where the user requests a specific aspect for the herd, extracted from real world's pictures. In practice, the user can also add position-only key-frames to edit the trajectory without imposing a specific aspect of the herd.

Using an analogy with drawing software, we allow the user to upload reference photos of real herds in a palette window and to extract the herd aspect from them using a pipette tool to define key-frames. To provide an intuitive visual feedback, the selected photo is copy-pasted on the terrain at the position where the user would like to impose this specific aspect of the herd.

As in usual pipelines, the animation process is fully automatic once key-frames have been positioned on a timeline. Yet, the user can not only edit the timing and choice of reference images at key-frames, but also some high level parameters, namely the density and orientation maps at each key-frame, as detailed below.

Method and challenges

Key-herds from photos: From each key-frame position with an associated photo, the first step is to generate a key-position for a herd with the target number N of animals, which we call a *key-herd* (see bottom left of Figure 1). The latter should reproduce the visual aspect in terms of global shape, distribution and orientation of animals of the herd on the photo, while avoiding obstacles. The main challenges are to define and learn the right set of global and local descriptors from the photo, and then define an algorithm enabling to use them at the synthesis stage. Our solutions are detailed in the *Analysis and synthesis* section.

Animation: Once key-herds have been generated and projected onto the terrain, a global trajectory is computed between their centroids. Individual trajectories are then automatically generated for each animal. Achieving a fluid motion is a challenge, since it requires consistently

labelling the animals within the successive key-herd. Moreover, the individual trajectories need to insure collision avoidance between animals and with obstacles on the terrain, while following the herd global path and matching the required change of position in herd frame. Our solutions to these problems are detailed in the *Herd animation* section.

Analysis and synthesis of herds

Our processing pipeline for generating a key-herd from a photo is detailed in Figure 2. The photo is first pre-processed to extract oriented ellipses giving the position and orientation of the animals. We analyze this resulting distribution to compute correlation curves modeling the local interactions between animals, as well as editable features in the form of density and orientation maps. We take the global orientation of the herd trajectory, the possible obstacles on the terrains and the size of the target herd to modify these maps, which may also be interactively edited. Finally, we use both curves and features to synthesize a perceptually similar herd of the right size. We details these steps below.

Data extraction from a single image

We use a semi-automatic method to extract data from an image. It only requires manual tuning in ambiguous regions and greatly eases the extraction work for large images.

We ask the user to manually annotate a few spots on the image (see Figure 3) as *foreground* (the animals) or *background* (for ground or other unwanted features). A specific Support Vector Machine (SVM) is then created and trained to discriminate foreground from background pixels based on their colour in the user-defined annotations, and subsequently used to create a black and white mask of these features. Different regions, usually corresponding to different animals, are then extracted using a Watershed Transform. Finally, we fit ellipses on each of these regions to retrieve their position, orientation and dimension. The global direction given to all ellipses is chosen arbitrarily for an image and can be flipped if necessary.

Some animals can be wrongly captured as a single, large region if they were too close to each other. We detect these cases with an analysis on the distribution of extracted region sizes. In such situations, the created ellipses are too large, too small or of unusual proportions. They are automatically detected and manually corrected with the right shapes.

A PCF-based method for interactions

We model the local interactions between animals by extending the concept of the Pair Correlation Function (PCF) to distributions of ellipses within arbitrary shapes. PCFs are smooth functions that represent the average density of objects in a distribution with respect to an increasing distance between those objects. They are known to have many interesting properties such as being density invariant and characterising well the visual aspect of distributions (see [22] for details).

The fact that PCF are smooth curves allows robustness to noise. They are computed by weighting the contributions of neighbors at distance r with the Gaussian Kernel $k_\sigma(x) = \frac{1}{\sqrt{\pi}\sigma} e^{-x^2/\sigma^2}$. For the case of a 2D point distribution within the unit square domain, the PCF can be expressed by:

$$\mathrm{PCF}(r) = \frac{1}{A_r n^2} \sum_{i \neq j} k_\sigma(r - d_{ij}) \qquad (1)$$

where A_r is a normalisation factor defined as the area of the ring of radius r and thickness 1, and d_{ij} is the distance between points i and j.

From point to ellipse distributions: To extend the framework to distributions of ellipses, we need an appropriate distance encoding both the size and the orientation of each ellipse. We simplify the problem of computing an ellipse-to-ellipse distance by using two ellipse-to-point distance computations, one for each ellipse of every pair. This choice is consistent with the fact that PCFs compute and average the distances of each element w.r.t. the others in the distribution.

In order to take into account the size and the orientation of the elements while computing ellipse-to-point distances, each point is expressed in local coordinates relative to the ellipse. More formally, for the point (x, y) in local coordinates, we compute its distance to the

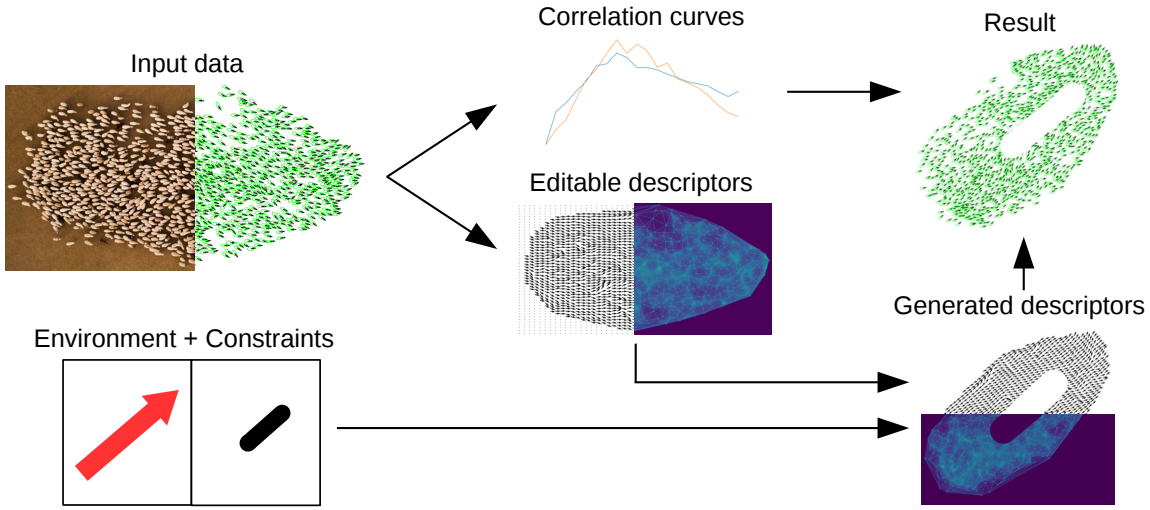


Figure 2: Pipeline for computing a key-herd: From oriented ellipses extracted from the photo (top left), correlation curves and editable descriptors are computed. The descriptors are adapted to the trajectory orientation and obstacles (bottom), and then used in conjunction with the correlation curves to produce a key-herd with N animals (right).

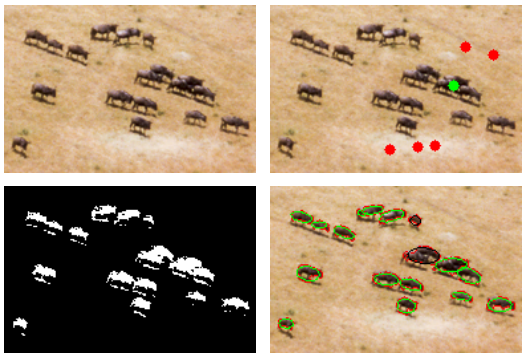


Figure 3: Data extraction. From left to right, top to bottom: input, annotated image, binary segmentation, extracted ellipses with detected outliers in red.

axis-aligned ellipse using:

$$d(x,y) = \sqrt{\frac{x^2}{a^2} + \frac{y^2}{b^2}} \qquad (2)$$

where x and y are the local coordinates of the point, and a, b re respectively the semi-major and semi-minor axes of the ellipse. More intuitively, this distance represents how much larger the ellipse would have to be for the point to be lying on its contour. This scaling value is equal to 1 for all points on the ellipse.

Generalising the domain: Another challenge is to adapt the formulation of PCFs to arbitrary domains. Indeed, the domain shape needs to be taken into account in the PCF equation in order to correct the bias induced by its borders. A usual correction is to take weigh the number of neighbors by the ratio between the area or perimeter of a disk and its intersection with the domain. While this is relatively straightforward for a square domain, it can become quite complex for arbitrary shapes, and we would lose too much information by reducing herds to squares.

Instead, we choose to define the domain as the convex hull of the data points, which is an arbitrary convex polygon. Although accurately computing the area of an elliptic ring inside a convex polygon is complex, we can efficiently provide a good approximation of the solution: We consider an approximation of the ellipse as a collection of as many triangles as there are edges in the convex hull of the domain. Each of these triangles is composed of the origin of the ellipse, and two consecutive points on the hull polygon. We translate the points of the hull to the intersection between the ellipse and the convex hull. The area of the part of the ellipse in the domain is computed as the sum of areas of the triangles. The area of the inner ellipse is subtracted to get the area of the elliptic ring within the domain.

In practice these computations are only done for a few rings at regular distances around each point, the weighing coefficients given by the ratio of the ring in the domain vs. the area of the

full ring being interpolated in-between.

Editable descriptors

On top of the correlation curves used to keep track of the local relationship between objects, at least one other descriptor, the orientation field, is required to produce a result faithful to the original data. One last optional descriptor, the density map, can also be used in conjunction with the rest of framework to replicate the density of regions in the same location in our output.

Orientation field: The orientation field is defined everywhere in the domain, and allows us to assign a plausible orientation at any hypothetical point within the original convex hull.

We compute this field by extracting the Delaunay triangulation of the centers of the extracted ellipses representing animals, and assigning the orientation of the ellipse to each vertex of the resulting mesh, as can be seen in Figure 4. When querying the field for the orientation of a new arbitrary point, we find the triangle that contains this location and interpolate the three angles of the vertices using barycentric coordinates.

Density map: Extracting density maps as well helps creating distributions as close to the original as possible by reproducing the position of the denser areas and empty regions.

Our approach to do so is similar to the one used for the orientation field. From the Delaunay triangulation of the ellipse centers, we use the area of the 1-ring, i.e. the collection of neighboring vertices, as an indicator of how much free space is available around at this location. We take the inverse of this area as an approximation of the local density and assign it to each point of the triangulation. We use barycentric coordinates interpolation withing each triangle to define density everywhere in the domain.

Key-herd synthesis algorithm

The editable descriptors computed from the input image are first transformed and modified to account for the general orientation of the herd trajectory at the key-frame, the obstacles in the environment and the size of the herd (each animal being of constant size, a herd of 100 will not take the same surface area on the terrain as a herd of 1000): after rotation and projection of the density map on the terrain, the part covered by obstacles is masked out. the map in then scaled in or out until it fits the requested number of animals. The same transformation is then applied to the orientation map.

Our synthesis algorithm, taking as input the PCFs of the input distribution, the density and orientation maps, and the number of ellipses required in the target distribution, is summarized in Algorithm 1, and further detailed in this section. The method used is a modified version of the dart throwing algorithm.

Input: PCF, orientation field O, density map D, number of elements N
Output: Ellipse distribution

$fails \leftarrow 0$;
while $< N$ *elements accepted* **do**
  Sample ellipse from D;
  Sample orientation from O;
  Update PCF with new ellipse;
  if *error decreased or* $fails > max\_fails$ **then**
    Accept best ellipse;
    $fails \leftarrow 0$;
    continue;
  end
  $fails \leftarrow fails + 1$;
end

Algorithm 1: Synthesis algorithm

We first pick a random ellipse respecting the probability distribution of the density map. This sampling is done by computing running sums of densities per column of a discretized version of the density map, and then from one column to the next. This gives us the probability of having a point in each column, and for each column the probability of having a point at each pixel. We use these cumulative density functions to sample random ellipses according the density map. A rotation is then assigned to the ellipse using an orientation field query at its location. We update the current PCF with the impact of the new ellipse and compute the error E compared to the target PCF, after normalization to account for the difference in element count. We use:

$$E = \sum_r \frac{(PCF(r) - PCF_0(r))^2}{PCF_0(r)} \qquad (3)$$

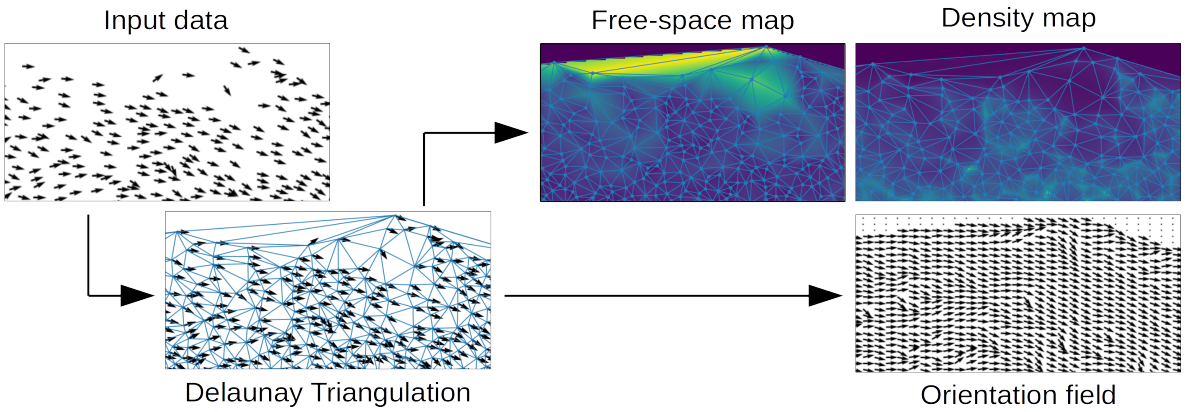


Figure 4: The descriptors are computed by interpolation inside triangles from the Delaunay triangulation of the input data.

where r spans distances to the ellipse.

The ellipse is accepted if adding it to the distribution reduces the error. Otherwise, to make sure that the algorithm does not get stuck in an infinite loop, we keep track of the best candidates while searching for ellipses. If the algorithm cannot find a good enough solution before reaching a threshold max_fails, the best candidate from the previous tries is accepted.

Descriptors as control tools

While it is possible to extract every component of our model from images and use them to create key-herds, the user can also edit the orientation map and density field to obtain a finer control over the result.

The density map is a completely optional parameter and removing it will yield an image that is similar but has local density extrema in different places. Manually painting a density map can be used for artistic purposes to alter the look of the resulting distribution.

Similarly, editing the orientation field leads to another form of control over the result. Indeed, replacing every orientation in the field by the same orientation produces a uniform flow, and smoothing or adding noise to the field can be used to control the level of detail in the emerging distribution.

The extent of this control is shown in Figure 5, with the top row showing the effects of orientation field changes while the bottom row showcases the impacts of different density maps. Sub-figure 5b shows the result after merging the orientation field with orientations pointing straight right, and figure 5c shows the effect of a pure rotational orientation field on this example. While changes in the density map are more discreet, its effects can still clearly be seen. Indeed, the most important empty spots are mainly located to the left of the herd in figure 5e while they are in the center in sub-figure 5f.

Herd animation

Global herd trajectory

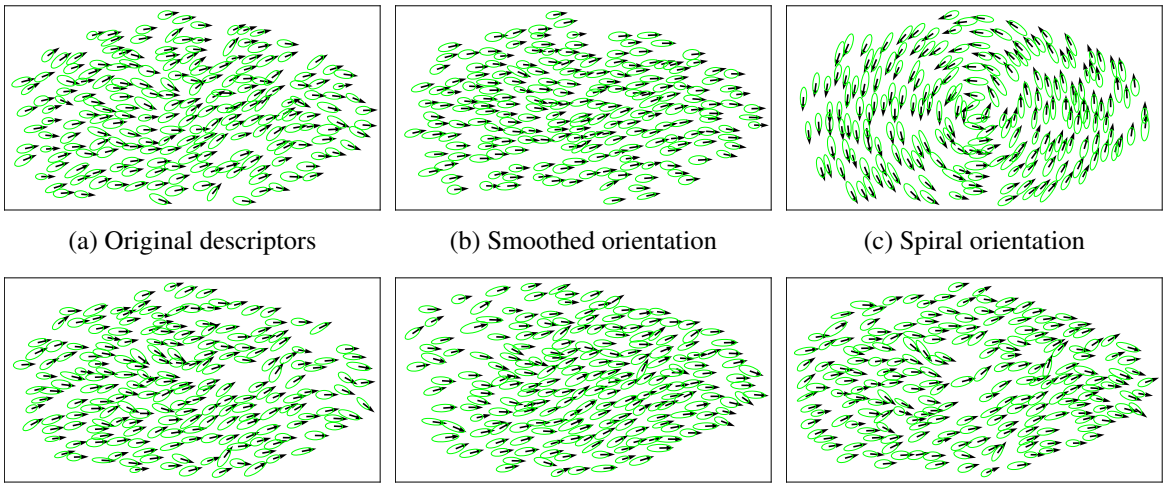
In this work, we extend the idea of key-framing an animation to a group of animals. While traditional key-frames can be used to represent important poses of an individual character, our key-herds encode a full group of animals at a specific point in time.

From this input, a global trajectory for the herd, modeled using an interpolation spline between the centroids of key-herds and the extra position-only key-frames, is generated and projected onto the terrain. The herd can be seen as a local frame that moves along this global trajectory, and within which the animals will have some adequate relative motion.

Generating individual trajectories

We first compute a consistent labeling to pair animals within successive key-herds, and then use microscopic simulation to define individual trajectories, as described next.

Pairing based on optimal transport: Establishing a good correspondence between the ani-



(a) Original descriptors (b) Smoothed orientation (c) Spiral orientation

(d) Uniform density (e) Left-to-right density (f) Radial density

Figure 5: Effect of the editable descriptors on the synthesised result, with changes to the orientation field (top) and density map (bottom).

mals of a key-herd and the following one is essential to generate fluid motion. In our work, we use optimal transport [23] for this computation, after virtually superposing the two successive key-herds to best align their centroids and orientations (by convention, the X axis in a local frame). More precisely, the mapping is computed as the optimal transport of one unit of mass for each animal located in local coordinates relative to the key-herd, to the local positions of the animals in the next key-herd. Depending on the type of animation required, the metric used for this computation can be the Euclidean distance, the Euclidean distance to the nth power to increase the penalty on distant mappings, or a metric that penalises matching along one axis more than the other.

Herd-frame-guided simulation: The key idea to generate individual animal trajectories that both interpolate their assigned position in two successive key-herds, but also follow the herd trajectory in-between, is to use microscopic simulation following moving guides defined in the herd frame. This way, while the guides model the necessary relative motion within the herd, the steering behavior in world frame enables to avoid collisions with other animals and with obstacles.

We use a simple steering model based on five forces to generate individual motion. The first three forces are separation, alignment and cohesion [2]. They are responsible for the general behaviour of the individuals. They are completed by an extra force to handle collision with the environment: it steers the animals away from obstacles if their predicted future position given their current position and velocity gets too close.

The last force controls the animal movement by giving them guides to follow. The latter straightly move, in the local herd frame, from their previous position in a key-herd to the next. Guide positions are transformed to world frame at each time step, while the herd frame follows its global trajectory. In addition, to account for the actual speed of the associated animals, which can be slowed down by other interactions, the position of the herd frame along its trajectory is set to move forwards only when all the associated animals are about to reach their guide.

Note that in our current implementation, the orientation of animals can be computed either from their movement or by interpolating the orientations of successive key-herds.

Results and discussion

Our framework implemented in Python for herd analysis and synthesis and in C# on the Unity Engine for user interaction and rendering. Timings reported in Table 1 were run on an Intel Core i5 processor at 3.3GHz equipped with 8GB of memory.

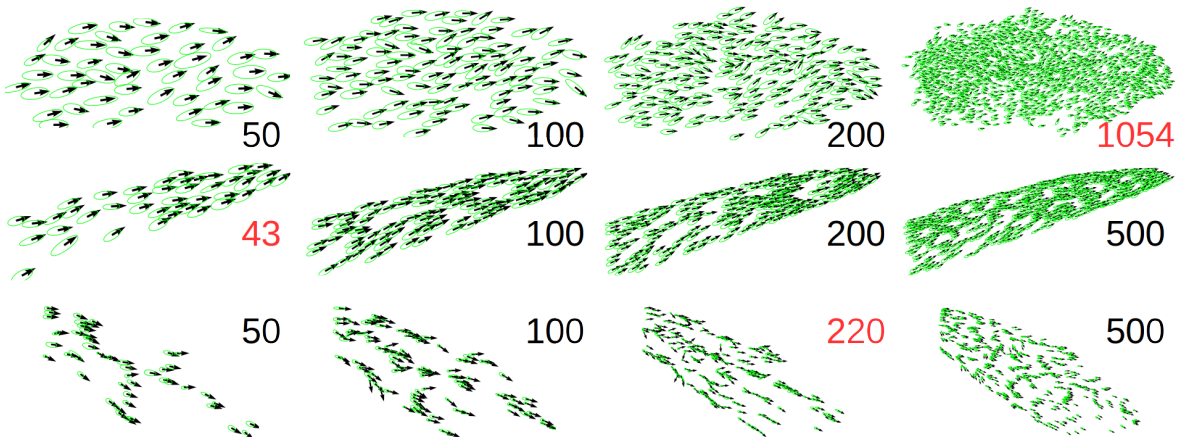


Figure 6: Generated herds with different target herd sizes. The size in the input image is in red.

Figure 7: Frames from an animation showing interaction with an obstacle.

Herd size	Ex.1 $N = 1054$	Ex.2 $N = 43$	Ex.3 $N = 220$
5	1.8s	0.5s	0.7s
25	1.9s	0.5s	0.8s
100	2.2s	5.1s	6.2s
200	3.5s	16.2s	15.4s
500	15.9s	73.4s	59.3s

Table 1: Benchmark of synthesis time

Figure 6 shows synthesis results for three different input images depicting herds of widely varying sizes and visual patterns. For each input, generation is performed with different herd sizes to illustrate the robustness of our method.

Animation results are depicted in Figures 1 and 7. As these results show, our method can handle challenging animation scenarios with significant changes in the herd shapes due to user input or environmental constraints, while maintaining the global aspect of the herd distribution learned from the input herd photos. A demo is provided in the supplementary video.

Limitations: Firstly, although orientation fields are properly reconstructed at herd key-frames, we failed to achieve an animation of individual animals that matches both this orientation field and results in natural motion, as can be seen in the supplementary material and Figure 8. This comes from the fact that linearly rotating guides between their target position and adding angular forces within the steering behavior produces strange behavior such as animals moving side-way or even back-way. Achieving animation that match orientation constraints is there-

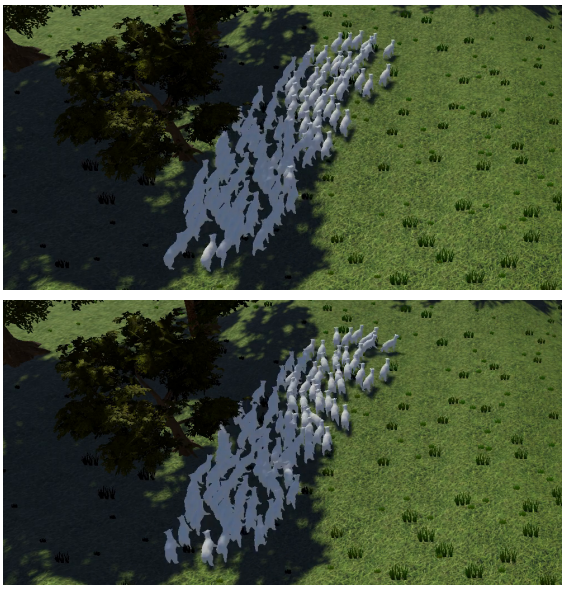


Figure 8: Compared to computing orientations based on animal movement (top), interpolating orientations between key-herds (bottom) helps to reproduce variety in a herd, but can result in sideway movement.

fore left for future work. Secondly, although we offer a precise control of the statistical distributions of animals at key-frames, our method does not offer any guarantee on the plausibility of the distribution in-between. In particular, even when the same reference image is used for two successive key-herds, the fact we use microscopic simulation to process local interactions in-between, may alter the intermediate distribution. A last limitation is the difficulty to reproduce constrained formations as the queue of animals in Figure 6, bottom. This is due to the interpolation of the density map, which diffuses density where there was none in the input image, and to the distance used that does not discriminate front from side, apart for the inherent shape of the ellipses.

Conclusion and future work

We presented an easy to use authoring tool for herd animation. It enables the user to key-frame a herd over a terrain by extracting the successive visual appearances from photos, while automatically taking the 3D environment constraints as well as the target number of animals into account. The animals in successive key-frames are then automatically paired and animated.

In addition to matching orientation fields, the animation could also be enriched using some known animal features such as their type of movement (going from Brownian motion for flying insects to smooth displacements or jumping for other animals).

Among the original features of our method, learning herd animation from photos rather than videos was a design choice. Indeed, images allow more efficient learning are also much more accessible. In the context of herd animation, finding segments of videos where each animal remains visible for a long enough time period would have been difficult.

Still, the velocity of animal motion cannot be learned from a photo. Being able to extract it from videos, even short and with only a few fully visible animals, would be an excellent complement to our work, which we plan to investigate in the future.

References

[1] L. Skrba, L. Reveret, F. Hétroy, M-P. Cani, and Carol O'Sullivan. Animating Quadrupeds: Methods and Applications. *Computer Graphics Forum*, 28(6), 2009.

[2] Craig W. Reynolds. Flocks, herds and schools: a distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4), 1987.

[3] Craig W Reynolds. Steering behaviors for autonomous characters. 1999.

[4] Sébastien Paris, Julien Pettré, and Stéphane Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Computer Graphics Forum*, 26(3), 2007.

[5] Jan Ondřej, Julien Pettré, Anne-Hélène Olivier, and Stéphane Donikian. A synthetic-vision based steering approach for crowd simulation. *ACM Transactions on Graphics*, 29(4), 2010.

[6] Xinjie Wang, Xiaogang Jin, Zhigang Deng, and Linling Zhou. Inherent noise-

aware insect swarm simulation. *Computer Graphics Forum*, 33(6), 2014.

[7] Weizi Li, David Wolinski, Julien Pettré, and Ming C Lin. Biologically-inspired visual simulation of insect swarms. In *Computer Graphics Forum*, volume 34, 2015.

[8] Qin Gu and Zhigang Deng. Generating freestyle group formations in agent-based crowd simulations. *IEEE Computer Graphics and Applications*, 33(1), 2013.

[9] Mingliang Xu, Yunpeng Wu, Yangdong Ye, Illes Farkas, Hao Jiang, and Zhigang Deng. Collective crowd formation transform with mutual information-based runtime feedback. *Computer Graphics Forum*, 34(1):60–73, February 2015.

[10] Jiayi Xu, Xiaogang Jin, Yizhou Yu, Tian Shen, and Mingdong Zhou. Shape-constrained flock animation. *Computer Animation and Virtual Worlds*, 19(3-4), 2008.

[11] Xinjie Wang, Linling Zhou, Zhigang Deng, and Xiaogang Jin. Flock morphing animation. *Computer Animation and Virtual Worlds*, 25(3-4):351–360, May 2014.

[12] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. Group behavior from video: a data-driven approach to crowd simulation. In *SCA'2007*, 2007.

[13] Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. Morphable crowds. ACM Press, 2010.

[14] Laurent Favreau, Lionel Reveret, Christine Depraz, and Marie-Paule Cani. Animal gaits from video. *Graphical Models*, 68(2), 2006.

[15] Xuemiao Xu, Liang Wan, Xiaopei Liu, Tien-Tsin Wong, Liansheng Wang, and Chi-Sing Leung. Animating animal motion from still. *ACM Trans. Graph. (Siggraph Asia issue)*, 27(5), 2008.

[16] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. WorldBrush: interactive example-based synthesis of procedural virtual worlds. *ACM Transactions on Graphics*, 34(4), 2015.

[17] James Gain, Harry Long, Guillaume Cordonnier, and Marie-Paule Cani. EcoBrush: interactive control of visually consistent large-scale ecosystems. *Eurographics*, 36(2), 2017.

[18] Cengiz Öztireli and Markus Gross. Analysis and synthesis of point distributions based on pair correlation. *ACM Transactions on Graphics (TOG)*, 31(6), 2012.

[19] Riccardo Roveri, A. Cengiz Öztireli, and Markus Gross. General point sampling with adaptive density and correlations. *Computer Graphics Forum*, 36(2), 2017.

[20] Chongyang Ma, Li-Yi Wei, and Xin Tong. Discrete element textures. 2011.

[21] Pierre-Edouard Landes, Bruno Galerne, and Thomas Hurtut. A shape-aware model for discrete texture synthesis. In *Computer Graphics Forum*, volume 32, 2013.

[22] Pierre Ecormier-Nocca, Pooran Memari, James Gain, and Marie-Paule Cani. Accurate synthesis of multi-class disk distributions. *Computer Graphics Forum*, 38(2), 2019.

[23] Nicolas Bonneel, Michiel van de Panne, Sylvain Paris, and Wolfgang Heidrich. Displacement interpolation using lagrangian mass transport. *ACM Trans. Graph.*, 30(6):158:1–158:12, December 2011.