Interactive Inverse Spatio-Temporal Crowd Motion Design

C. D. Tharindu Mathew
Purdue University
West Lafayette, IN, USA

Bedrich Benes
Purdue University
West Lafayette, IN, USA

Daniel G. Aliaga
Purdue University
West Lafayette, IN, USA

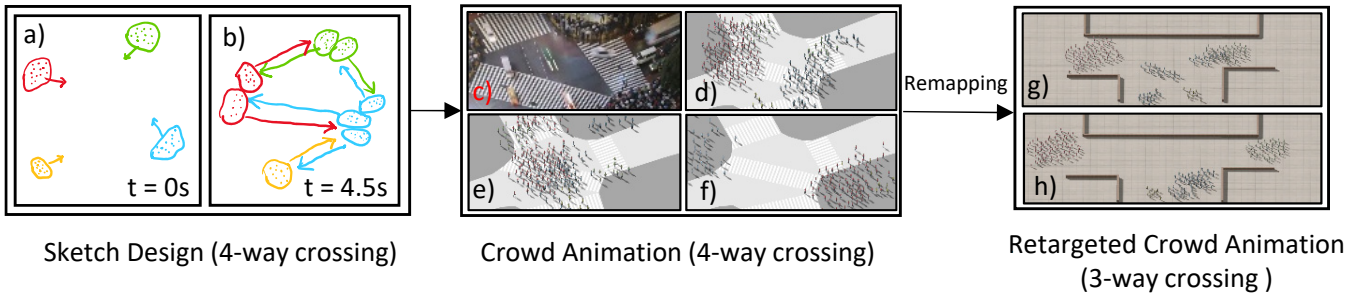

Figure 1: Crowd Motion Design. a, b) Sketching of people, source and path at keyframes t=0, t=4.5s; c) A video frame of a busy street intersection in Shibuya, Tokyo; d, e, f) Resulting crowd animation at t=6s, t=9s, t=13s for d, e, f respectively; g, h) Retargeted crowd animation from 4-way crossing to 3-way crossing (see Section 5.1) for t=7s, t=13s for g and h respectively.

ABSTRACT

We introduce a new inverse modeling method to interactively design crowd animations. Few works focus on providing succinct high-level and large-scale crowd motion modeling. Our methodology is to read in real or virtual agent trajectory data and automatically infer a set of parameterized crowd motion models. Then, components of the motion models can be mixed, matched, and altered enabling rapidly producing new crowd motions. Our results show novel animations using real-world data, using synthetic data, and imitating real-world scenarios. Moreover, by combining our method with our interactive crowd trajectory sketching tool, we can create complex spatio-temporal crowd animations in about a minute.

CCS CONCEPTS

• **Computing methodologies** → **Procedural animation**; **Motion processing**; *Motion path planning*.

KEYWORDS

Crowd Motion Control, Inverse Procedural Modeling, Sketching, Crowd Simulation

ACM Reference Format:
C. D. Tharindu Mathew, Bedrich Benes, and Daniel G. Aliaga. 2020. Interactive Inverse Spatio-Temporal Crowd Motion Design. In *I3D 2020*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3384382.3384526

I3D, 2020, May 5-7, San Francisco, CA, USA

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/3384382.3384526

1 INTRODUCTION

Interactive crowd design and animation is important in many applications, such as virtual training, urban design, gaming, and other entertainment applications. Crowd design and animation consists of a high-level crowd motion specification combined with a crowd-agent simulation governing lower-level collision avoidance, agent steering, and agent-agent and agent-environment interactions.

While many approaches exist, there is a disconnect between high and low level control. Although low-level control can be achieved manually, it is often tedious and requires a significant amount of work. Crowd simulation algorithms typically address the low-level close-range agent-interactions often with rule sets (*e.g.,* [van den Berg et al. 2011; Wolinski et al. 2014]), energy functions (*e.g.,* [Narain et al. 2009; Treuille et al. 2006]), or data-driven hints [Bera et al. 2015; Wang and O Sullivan 2016]. Crowd motion design seeks high-level virtual crowd control by, for example, combining patches of prior crowd motion (*e.g.,* [Jordao et al. 2014; Li et al. 2012; Ulicny et al. 2004; Yersin et al. 2009]), drawing simple vector fields (*e.g.,* [Patil et al. 2011]), or parameter-estimation (*e.g.,* [Bera et al. 2015]). A common design challenge is easily achieving the emergent crowd phenomena and intricate crowd motion that are often complex by-products of agent behavior and may be difficult to reproduce; for example, 1) producing complex behaviors such as vortices, dynamic-grouping, and stop-and-go motion, 2) designing complex crowd motion quickly and intuitively in space and time, and 3) supporting a widely varying number of agents.

Our methodology is based on two key inspirations. Our first key inspiration is to infer parameterized spatio-temporal crowd motion models (CMMs) of existing crowd motions (real or synthetic). Our second key inspiration is to enable a simple user interaction to design/sketch novel potentially complex crowd behavior. The design process intuitively uses and mixes the spatio-temporal CMMs and supports re-targeting them to different scenarios, all the while the underlying system automatically handles low-level behavior.

Our work belongs to inverse procedural modeling that has recently been applied to infer building models [Bokeloh et al. 2010; Vanegas et al. 2010], city models [Nishida et al. 2016; Vanegas et al. 2012], and plant models [Stava et al. 2014a], for instance. Our method takes an inverse modeling approach that hinges on crowd motion presenting itself as clusters of agents moving from source to destination locations, following recognizable spatial paths, and walking at different speed/temporal profiles (*e.g.,* slowly, quickly, changing pace). Hence, by blending and retargeting our inferred parameterized CMMs, the user can interactively sketch simple to complex and realistic virtual crowd motion. Contrary to the approaches that focus on parameter estimation or a priori example trajectories, our method enables decomposing spatio-temporal crowd motion into parameterized components, designing new motions quickly and intuitively, and supports editing the inferred models as well as re-targeting crowd motion to new environments.

Figure 1 shows an example of our work. A user observed a video of the moving crowds at a busy street intersection in the well-known Shibuya area in Tokyo, Japan. Using our interactive design tool, and CMMs inferred from virtual trajectories and real trajectories (*i.e.,* [Seyfried 2019]), the user creates a similar spatio-temporal crowd motion, in under one minute of editing time. Then, we remap this crowd motion from a four-way crossing to a three-way crossing using source and destination retargeting (Section 5.1). In addition, we can almost instantly perform edits to alter the crowd motion (see Figure 10e and paper video).

Our approach consists of two main steps (see Figure 2 for an overview of our approach). During an *inference phase*, our system is provided with motion trajectory data either as "brush strokes" provided by our interactive trajectory design tool or as agent traces from a real-world observation. The trajectory data is then used to infer parameterized CMMs consisting of i) sources and destinations of agents, ii) spatio-temporal clusters, iii) spatio-temporal motion paths, and iv) corresponding displacement and speed profiles. During a *design phase*, our interactive design tool enables retargeting and blending crowd motions from one or more inferred CMM models to the same or different environments. Then, the design motion model is used together with agent-based crowd simulation engine and renderer to produce a crowd animation. In particular, we demonstrate results using our crowd simulation engine based on space colonization [de Lima Bicho et al. 2012], but others could be used and compared.

Our results include examples to create parameterized CMMs from real-world trajectory data (*e.g.,* [Seyfried 2019]) and from interactive sketches. We also use the CMMs to interactively design crowd motions similar to crowds observed in video sequences and in steering model-based virtual crowd scenarios. The inference phases in our examples each takes under four seconds. Our interactive design tool, combined with our CMMs, enables creating realistic crowd animations based on observations in a video in about one minute of iterative and interactive design. Furthermore, our crowd simulation implementation is able to simulate 25,000 agents at interactive update rates.

Our main contributions include: 1) inverse crowd motion modeling - inferring parameterized crowd motion models from virtual/real trajectory data; 2) crowd motion retarget - a method to integrate and blend various crowd motion models to same/new environment; and 3) crowd design tool - an interactive brush-based sketching tool to easily set up virtual trajectories and sketch new spatio-temporal crowd motions.

2 RELATED WORK

Related work can be divided into *crowd simulation* methods, *crowd-motion design* methods, and *inverse modeling* methods.

Crowd simulation methods focus on the agent-agent and agent-environment interactions and collision avoidance. In general, crowd simulation methods include macro (*e.g.,* [Treuille et al. 2006]) and micro approaches (*e.g.,* [Dutra et al. 2017; Golas et al. 2013; Karamouzas et al. 2014, 2018, 2017; Kim et al. 2012; Narain et al. 2009; Normoyle et al. 2014; Singh et al. 2011; van den Berg et al. 2011; Wang et al. 2016; Wolinski et al. 2014, 2016; Wong et al. 2018]). Also, related are techniques that focus on improving the behavior of individuals or characters (*e.g.,* [Aberman et al. 2019; Juarez-Perez and Kallmann 2018; Kapadia et al. 2016; Park et al. 2016; Shoulson et al. 2013]). In our system, we do not produce a new crowd simulator but instead make use of a crowd simulation algorithm and add a prior step to design the overall crowd scenario.

Crowd-motion design addresses the higher-level goal of prescribing the crowd motion as a whole and it has been tackled by various options. A first option is to optimize the parameters of the underlying crowd simulation to produce a desired motion as best as possible. For instance, Wolinski et al. [2014] described a comparative framework that uses parameter optimization to tune several a priori provided simulation algorithms to best match a reference motion. Sung et al. [2004] defines low-level individual character situation-based behavior but the overall crowd scenario is prescribed. Kapadia et al. [2009] defines time varying metrics as indicators of crowd behavior but does not enable specification of the overall crowd scenario. Jordao et al. [2015] only supports specifying crowd motion by giving desired density and flow values over the crowd environment. These so called group formation technologies [Ju et al. 2010; Takahashi et al. 2009] do not support the flexibility and expressiveness for creating complex scenarios as is possible by our approach (see Section 7). Some compelling commercial crowd simulation and authoring software systems exist as well, such as Massive. As per the feature list of Massive, it provides high-level crowd control only through keyframes (i.e., manually specifying agent variables at different instants), flow fields (*e.g.,* [Patil et al. 2011]), or handles to directly control individual characters. Overall, our method infers space-time motion models from virtual or real data and provides an interactive tool to easily mix and match the motion components yielding novel space-time crowd motions in the same or in re-targetted new environments.

A second option is to patch together existing trajectory data. Lerner et al. [2007] combines example real-world trajectories. Li et al. [2012] seeks for almost periodic patterns and almost symmetric arrangements to enable connecting together motion patches, extending the work [Yersin et al. 2009]. Jordao et al. [2014] also uses example patches and deforms them to stitch them together. A variation is to enable editing the trajectories using a deformable cage that can change spatial and temporal arrangements [Kim et al. 2014]. Another variation is to draw directed lines, called navigational fields, to define a crowd motion [Patil et al. 2011]. The lines can be

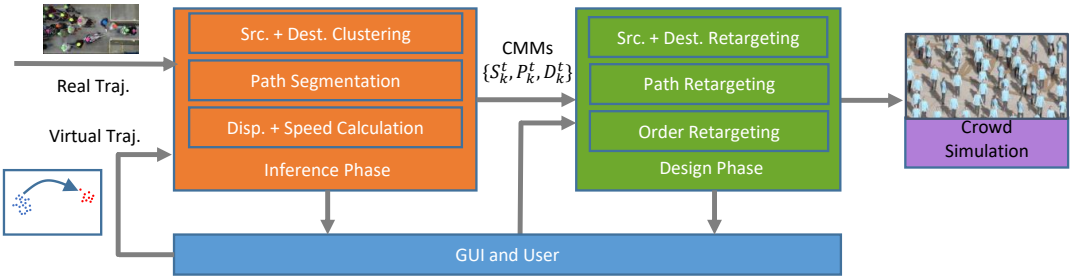

Figure 2: System pipeline providing a summary of our method.

blended and/or transformed to be placed together. This last work uses simple local collision avoidance and does not support complex motions like motion patterns, queueing, etc. Unlike our approach, these methods *do not address* parameterizing and/or changing the motion within each patch or navigational field nor combining portions of one motion with another. Instead, our technique defines source and destination areas, connected by an arbitrary space-time path, and supports retargeting the source, destination, and path to new environment as well as changing the segments and temporal order of the path segments (*e.g.,* adding the stop-and-go behavior of one motion to the spatial pattern of another).

A third option is a data-driven method to learn a crowd trajectory. Bera et al. [2015] incrementally learns pedestrian motion and behaviors from crowd videos. Wang et al. [2016] discovers and models recurring activity patterns from video. Lee et al. [2007] learns crowd simulation from video. However, these appproaches do not focus on combining motion models to yield new spatio-temporal animations and do not re-target to new environments.

Close to our solution are the methods generating compelling group formation, such as Ju et al. [2010] and Takahashi et al. [2009], that are compared in Section 7.

Within computer graphics, **inverse (procedural) modeling** has found success in determining the procedure (*e.g.,* the rule set and/or parameter values) yielding a desired output. For example, Stava et al. [2010] found procedural descriptions of 2D vector geometry, trees [Stava et al. 2014b], Bokeloh et al. [2010] and Demir et al. [Demir et al. 2016] discovered symmetries and repetition to find a procedural description of a 3D urban model, Talton et al. [2012; 2011] stochastically drives a procedural model so as to obtain an output following a desired global shape, Garcia-Dorado et al. [2014] changed a road network so as to obtain a desired vehicular traffic pattern over time. Procedural brushes were used to learn and reapply distribution of stochastic elements in a virtual scene [Emilien et al. 2015] and Nishida et al. [2018] inversely finds a procedural grammar based on a photograph. In our work, we seek to infer the parameterized CMMs that capture the behavior in virtual and/or real trajectory data enabling their subsequent combination, and re-targeting to new environments.

3 OVERVIEW

Our approach (see Figure 2) enables designing a spatio-temporal crowd motion scenario by interactive sketching and on-the-fly retargeting of one or more parameterized CMMs. In this section, we describe the crowd environment data structure, the trajectory input used to infer CMMs, and our design tool.

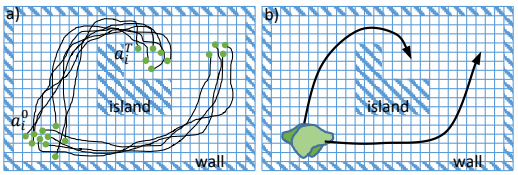

Figure 3: Input: Environment and Trajectories. We show an example of a grid-cell-based environment. Further, (left) we show a depiction of real-world trajectory data and (right) using our interactive sketching tool to specify trajectory data.

3.1 Environment

We assume crowd motion occurs in an environment defined by a rectangular perimeter inside which there may be solid static obstacles (*e.g.,* walls, islands, etc). The environment is represented by a 2D grid, typically of 10×10 cm grid cells. Grid cells representing obstacles are flagged as such (Figure 3). The inferred CMMs and crowd simulation variables are stored in the grid (see Section 6). A sparse representation could be used to accomodate to very large environments.

3.2 Trajectory Input

The input to inference phase is one or more sets of trajectory data in the form

$$\{\mathbf{a_i^t}\} = \{[\mathbf{a}_1^1, \mathbf{a}_2^1, \ldots, \mathbf{a}_N^1], \ldots, [\mathbf{a}_1^T, \mathbf{a}_2^T, \ldots, \mathbf{a}_N^T]\}, \quad (1)$$

where $a_i^t = [x_i^t, y_i^t]$ is the i-th agent's position at time t. Such data might be obtained

- by an interactive sketching tool for defining virtual trajectory data (*e.g.,* , a user may observe a video and sketch inspired motion patterns);
- by providing a file with the desired motion pattern (*e.g.,* a multi-agent trace from Pedestrian Dynamics Archive [Seyfried 2019]); or
- by a prior method that tracks agents during a video sequence [Rodriguez et al. 2011].

In this paper, and inspired by Ulicny et al. [2004] and by Emilien et al. [Emilien et al. 2015], we provide an interactive sketching tool

consisting of three brush types that can be applied at user-defined time steps: people, source-path, and geometry brushes. The people brush facilitates creating of an initial grouping of people. The brush has metaphors for specifying the size and density of the cluster. The source-path brush enables drawing a source polygon to select a group of agents, and a desired path with a desired speed. The geometry brush supports defining the obstacles in an environment (*e.g.,* walls). Please see video for examples.

The sketched curves are converted into a sequence as in Equation 1 for later use. In particular, the sketched lines, the speed and acceleration with which they are drawn are used to create a set of agent positions over time (*i.e.,* a virtual trajectory). In case the user sketches lines over static obstacles, our method uses a shortest path algorithm to determine the actual trajectory (*i.e.,* the Jump Point Search shortest path algorithm [Lawler 1972] which works efficiently with grid cells). For example, if a sketched line enters an obstacle on one side and then exits at the other side, our approach computes the shortest path through the open-space grid cells that connects the entry and exit points.

3.3 Crowd Motion Design

Our graphical user interface (GUI) provides controls for the inference and design phases. The user first selects a trajectory dataset: it can be loaded from file or sketched interactively. The inference phase then converts the dataset into a parameterized CMM. This phase can be performed for several datasets altogether. Next, during the (iterative) design phase, a crowd environment is designed and CMMs can be inserted, edited or combined. Further, the CMMs are automatically re-targeted to the new environment despite consisting of a new layout, size, or number of agents. Finally, the crowd simulation engine is invoked to produce a crowd animation.

4 INFERENCE PHASE

The first part of our method uses trajectory data to infer crowd motion models (Figure 2). The crowd motion model and the three steps of the inference approach are defined in this section.

4.1 Crowd Motion Model (CMM)

We use the following set to define the crowd motion model

$$C = \{S_k^t, P_k^t, D_k^t\}, \tag{2}$$

which represents agent group $k = \{1, \ldots, G\}$ moving from source area S_k to destination area D_k using path P_k. Each CMM is defined to start at certain discrete time step $t = 1, \ldots, T$. The path P_k (note: for notational simplicity we omit the time superscript from the following path definition) for agent group k consists of a set of linear segments l_{kj} for $j = 1, \ldots, M_k$ with corresponding speeds s_{kj} and displacement d_{kj} (*i.e.,* the displacement, or distance, of the agent from the path centerline and thus a representation of how tightly the path is actually followed); thus

$$P_k = \left\{ \langle l_{k1}, s_{k1}, d_{k1} \rangle, \ldots, \langle l_{kM_k}, s_{kM_k}, d_{kM_k} \rangle \right\}. \tag{3}$$

This CMM is flexible as it is able to capture a wide range of crowd motions. For instance, the path component is able to capture motion patterns of vortex and spiral motion, both of which appear as relatively intuitive patterns in crowd motion space (x, y, t). Since the path segments also have a speed component, the CMM can also represent stop-and-go motion that are commonly observed in large crowds [Seyfried 2019]. In addition, the sources and destinations of multiple models can overlap and also be sequenced over time. This supports agents diverging from one area and heading to multiple locations as well as intertwined, seemingly coordinated spatio-temporal motions (*e.g.,* such as a crowd crossing at street intersections) or queueing and dequeing in an open space for an event).

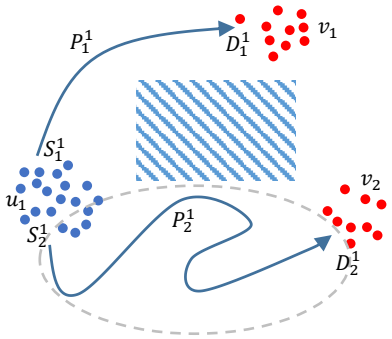

Figure 4: Source and Destination Clustering. Spatial clustering determines clusters u_1, v_1, and v_2 which are then used to form the two CMMs $\{S_1^1 = u_1', P_1^1, D_1^1 = v_1\}$ and $\{S_2^1 = u_1'', P_2^1, D_2^1 = v_2\}$ where u_1' is the subset of u_1 that navigates to v_1 and $u_1'' = u_1 - u_1'$.

4.2 Source and Destination Clustering

First, our method needs to determine the sets of agents starting from relatively nearby source (or start) locations, and going to relatively nearby destinations. For virtual trajectory data, the source and destination clustering might be explicitly provided but that is not typical for real-world trajectory data (unless the real trajectories are of exactly one group of people going from one area to another).

To determine the source and destination clusters, we analyze the agents at each key frame. Using a density-based spatial clustering method of [Ester et al. 1996], we determine the set of spatial clusters $U = \{u_i\}$ at time step $t = 1$ and the set of spatial clusters $V = \{v_i\}$ at the time step T. Empirically, we found a minimum clustering distance of 2m to work well for examples used in this paper. Afterwards, each agent is labeled with a cluster from U and a cluster from V. We then group all agents having the same U and V labels into the same CMM, thus defining the source area S_k^t and destination area D_k^t for that CMM. For example, in Figure 4 clustering determines u_1, v_1, and v_2 which are then paired as (u_1', v_1) and (u_1'', v_2) to form two CMMs as shown. This does not enforce the agents in the same CMM to follow the same path nor does it guarantee they all arrive at nearly the same time to the destination. Nevertheless, in practice we found it yields a good subdivision into source and destination clusters.

4.3 Path Segmentation

As a second step, our approach decomposes the spatio-temporal path P_k^t between each source and destination pair into a set of linear segments in (x, y, t) space as well as displacement and speed

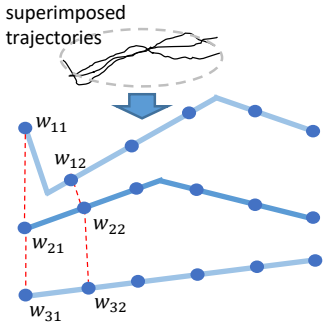

Figure 5: Linear Path Segments. Our method computes for each agent a solution to SLLS and then chooses the linear segments solution that best fits to all solutions. For this equidistant points $b = \{1, \ldots, B\}$ points are sampled along each trajectory $a = \{1, \ldots, A\}$ (*e.g.,* w_{ab} represents equidistant point sampled along the trajectory b), and used to find the solution most similar to all other solutions.

variables. To prevent choosing a priori how many line segments are needed for an arbitrary path, our system finds a sequence that minimizes a weighted combination of the number of lines and the distance error of the agents from the closest line segment.

Our solution is based on an extension to the dynamic programming solution of segmented linear least squares (SLLS) [Bellman 2013] which also accounts for our desired temporal sequencing during the path. The dynamic programming solution to SLLS is:

$$SLLS(j) = min_{1 \leq i \leq j} \left(e(i, j) + C + SLLS(i - 1) \right), \tag{4}$$

where $SLLS(j)$ implies the minimum SLLS cost for a sequence of points $\{p_1, \ldots, p_j\}$ and $e(i, j)$ is the minimum squared distance error for the sequence of points $\{p_i, \ldots, p_j\}$. In our case, the sequence of points refers to agent positions over time (*i.e.,* (x, y, t)). The constant C refers to the penalty cost for each line segment ($C = 5$ works well for all examples in this paper).

In our problem, we have multiple agents approximately following the path P_k. Since SLLS is defined for a sequence of points, we cannot simply merge all agent positions over time into a single (x, y, t) point cloud. Instead, we compute the SLLS solution for each agent and then choose the SLLS solution that is most similar to (*i.e.,* best approximates) all other solutions. Since the multiple SLLS solutions might vary in the number of segments, we use a canonical representation and re-sampling to determine the similarities between SLLS solutions.

In particular, we calculate an arc-length $t = [0, 1]$ parameterization for each SLLS solution and sample W points equidistantly within $[0, 1]$. To determine the similarity between two SLLS solutions, we compute the sum of the distances between all points with the same arc-length parameter value. To compute the best overall SLLS solution, we find the solution whose similarity to all other solutions is highest (*i.e.,* the aforementioned distance sums are the smallest). The resultant SLLS solution defines the segments l_{kj}. In Figure 5, we observe a setup to compute the SLLS solution that best fits a subset of P_2^1 from Figure 4 – note that the three shown multi-linear segments are actually superimposed (they are separated in the figure only for explanatory purposes).

Path segmentation is performed considering both space and time; thus, i) agents following a similar spatial path at a similar velocity are grouped and ii) agents remaining approximately spatially static for some time are also grouped – agents moving at the same speed but not spatially near are *not* grouped.

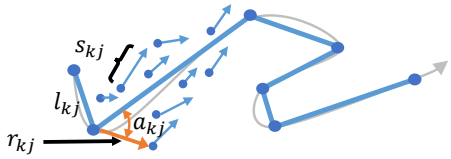

Figure 6: Displacements and Speeds. We associate with each agent a $d_{kj} = (r_{kj}, a_{kj})$ relative to the closest linear segment start point. We also store with each agent its speed s_{kj}. Blue dots represent agents, line segments are starting points, blue arrows are agents' velocity, and orange arrows represent the angle and the displacement vector respectively.

4.4 Displacement and Speed Calculation

In the third and final inference step, we also characterize the displacements d_{kj} and speeds s_{kj} of the agents relative to the extracted line segments. For each displacement $d_{kj} = (r_{kj}, a_{kj})$, we store a sampling of two paired values: the vector r_{kj} from the agent to the closest line segment start point earlier in the path and the angle a_{kj} of said distance vector to the closest line segment (see Figure 6). Since we typically expect agents continue their (forward) motion along the path, the linear path segments act essentially as the averaged crowd path (though globally it may still exhibit spirals and other intricate path shapes) and the displacements represent how the agents usually deviate from the averaged crowd path.

When subsequently re-targeting the displacements and speeds, our approach makes use of probability distribution functions (PDF) computed based on the aforementioned samples. The PDFs enable the novel motion to be longer than the original trajectories, for example (as will be shown in Figure 9).

5 INTERACTIVE DESIGN PHASE

During the interactive design, we map one or more CMMs to a potentially newly sketched environment, alter CMM variables, and/or mix and match CMM components. A simple retarget example would be to alter the speed of a linear path segment while a more complex retarget example would be to map the displacements and speeds of one CMM to the path segments of another CMM, and position all within a new environment.

5.1 Source and Destination Retargeting

A first retarget step is to map the source S_k^t and destination D_k^t of one CMM from its original environment to a new environment. To perform this re-mapping we first apply an initial linear transformation to the spatial locations of the sources and destinations. Then,

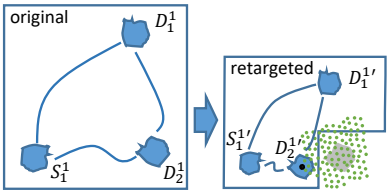

Figure 7: Source and Destination Retarget. An original environment and sources and destinations networks are remapped to a new environment and the network is altered so as to ensure the same reachability as in the original environment. $D_2^{1'}$ is moved from its transformed location to a suitable location by searching for incrementally better positions near the original location.

we perform a refinement to reduce the spatial distortion of the new sources and destinations network as compared to the original.

Given a quadrilaterial bounding box of the original environment, we compute the linear homography matrix that transforms it to the quadrilateral bounding box of the new environment. While this linear transformation does not capture all possible retargetings, it does offer a significant ability. Nonetheless, it distorts the geometrical relationships within the original sources and destinations network and might position sources or destinations in invalid locations (*e.g.,* inside walls of the new environment or with no source-to-destination reachability).

To refine the locations of the sources and destinations (*e.g.,* move a destination accidentally transformed to be in a region that cannot be reached from its corresponding source), we then use a Monte Carlo approach. For each source or destination, we choose a location from a set of nearby randomly-generated positions that improves a similarity metric between the original sources and destinations network and the new one. The similarity metric computes the Euclidean distance between corresponding sources and destinations in the two networks, normalized to a canonical space (*e.g.,* centered at the origin and rescaled to $[-1, +1]$ in x and y). If the new environment contains walls or islands, we perform an additional step that computes the reachability from each source to its destination – we use the fast Jump Point Search algorithm (Section 3.2) to verify reachability. In this way, we can detect if the new network results in valid source-destination connectivity (see Figure 7).

We perform the random-based optimization sequentially for each source and destination pair and repeat until a valid source-destination network is found with no further significant improvement in similarity or a maximum number of iterations is reached. Since we limit the iterations, the runtime is linear in the total number of sources and destinations which implies a fast runtime.

5.2 Path Retargeting

A second retarget step is to map the displacement and speed components of a P_k^t to a new path Q_k^t. Since displacements and speeds are defined relative to the line segments, the aforementioned mapping can be performed as well as changes to the displacements and speeds themselves (note: changes can be per-agent or overall). For example, the re-mapping of the displacements naturally re-scales the distance values to the length of new line segments. If desired, the user can alter the angle component of the displacement to produce a tighter or looser following of the path segments. Similarly, the user can alter the speed component to produce slower or faster agent motion. The displacement and speed retargeting can also be done by adding a random variation. For example, if PDFs of displacement and/or speed are defined, the retargeted value can be randomly chosen from the corresponding probability distribution.

5.3 Order Retargeting

A third re-target step is to alter the number or the order of line segments in the new path Q_k^t as compared to the original path P_k^t. If necessary, we can clamp or repeat the line segments in the original path. For example, if the new path has less segments, we map less segments from P_k^t; however, if a new path has more segments, we map segments in a wrap-around fashion from P_k^t.

Another variant is to map the line segments from P_k^t to the new path in a random order. This random variation is particularly applicable in scenarios with various styles of haphazard motion, such as stop-and-go and zig-zag patterns. The new path will obtain a similar style motion as the original yet not be identical and can be of a different length (*e.g.,* similar to the seemingly infinite repeatability of video textures [Schödl et al. 2000]).

6 CROWD SIMULATION

Previous sections describe how to learn the parameters and how to define motion of the agents. In order to model the individual behavior of our agents, we can use one of several crowd simulation engines. In our work, we extend a crowd simulation model based on space colonization [de Lima Bicho et al. 2012] to use our CMM and combine it with the grid-cell data structure. We place an inferred CMM within the grid cells spanning its source area. A CMM's path segments (Figure 5) define intermediate way points which are fed to the crowd simulator. Thus, when an agent enters the source area, and at the appropriate time step, it will "read-in" the CMM and commence following the way points until the destination area. Once an agent reaches its destination it remains dormant until a new CMM commences in that grid cell. The motion of the new CMM will gradually, over time, adjust an agent's speed and direction.

The grid cell also stores data for the crowd simulator. In particular, our simulator is based on the work of de Lima Bicho et al. [2012] which places *markers* within each open-space grid cell. Agents in the environment can only move to areas where they can perceive and grab available markers within their observational radius. This approach scales well and implicitly provides guaranteed collision-free agent movement. In such a space colonization method, obstacles are very easy to represent as zones without any markers. In addition, we add several performance enhancements that yield approximately one order of magnitude increase in simulation performance over [de Lima Bicho et al. 2012]. In particular, we use space discretization (*i.e.,* a grid) to accelerate spatial operations and we calculate and cache approximate shortest path, instead of the exactly best path.

7 RESULTS AND EVALUATION

Our system is implemented in C++ using Qt and OpenGL, and runs on a computer equipped with Intel i7 and NVIDIA GTX 1080

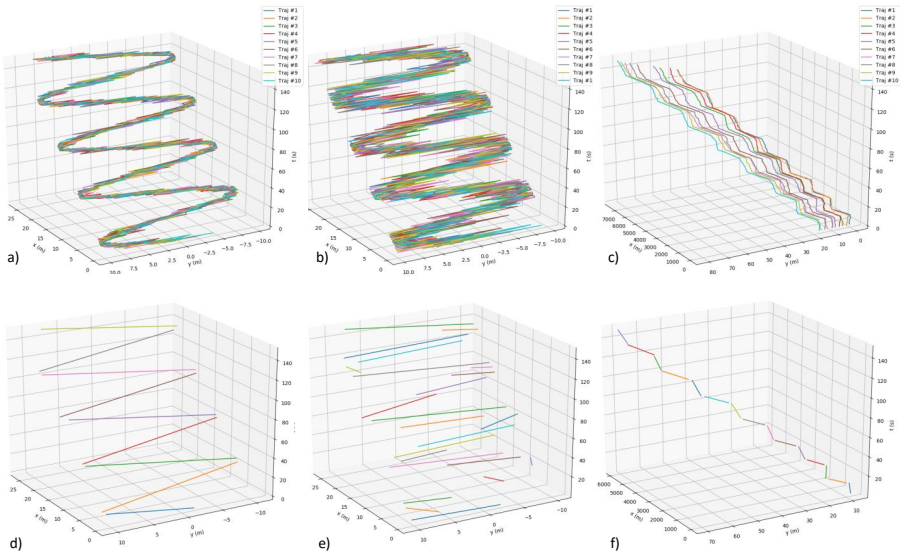

Figure 8: Inference of the underlying motion despite noise and random deviations. a,b) Show 10 agents with mild to strong random deviations added to an underlying sinusoidal path. d,e) show the recovered sinusoidal path when large random deviations are present. c,f) Show how a stop-n-go motion can be recovered.

graphics card. All the operations during inference and design phases of our examples occurs at interactive or nearly interactive rates (from subsecond to a few seconds at most). The typical time to interactively design a new crowd scenario is about one minute. Our current crowd simulation and rendering engine is able to simulate up to 25,000 agents and render the resulting animation at 10 frames per second. The graphics card is used only for rendering purposes.

Figure 8 shows the robustness of the inference method in determining a representative set of linear path segments for a given set of agent trajectories. Thus, during retargeting the general style of the motion can be re-mapped. Figure 8a contains the (x, y, t) trajectories of 10 synthetic agents moving roughly along a sinusoidal path at constant speed (each agent is shown in a different color). Figure 8d shows how the expected clean sinusoidal-like path is recovered even with a certain amount of random perturbations of the agents. However, if we inject a significant amount of random displacement to the agents (as in Figure 8b), then the inferred path segments no longer resemble the expected path (Figure 8e).

Figure 8c shows a stop-n-go motion of 10 agents – observe the staircase-like pattern in (x, y, t) space where the vertical axis is t. When an agent stops, it corresponds to a vertical segment (*i.e.,* only t changes). When in motion, it corresponds to a diagonal segment (*i.e.,* x, y, t change). Collectively these form the staircase-like appearance. From the multiple noisy agent trajectories, our method is able to infer the underlying path (Figure 8f). For the top-row of Figure 8, each agent is colored differently. In the bottom-row of the figure, each linear path segment is given a different color.

Figure 9 demonstrates several retargeting tasks. Figure 9a shows a spiral trajectory retargetted to a path having more segments resulting in the segments repeated in the same order along the retargeted path (Figure 9b). The spiral like trajectory is maintained and extended in the retargeted scenario. Figure 9c is a stop-n-go

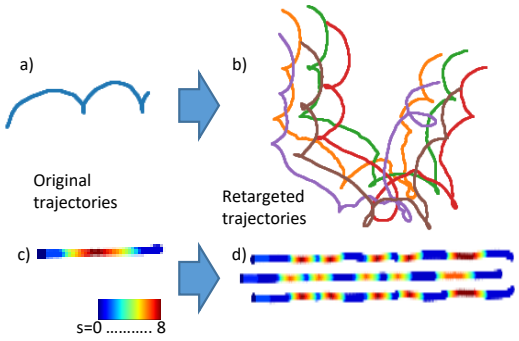

Figure 9: Retarget Examples. a-b) An original trajectory retargeted to multiple agents along a longer U-shaped path. c-d) One stop-n-go agent retargeted to multiple stop-n-go motions by random order retargeting. The color represents speed as per the included JET color map from 0 to 8 m/s.

type motion drawing using a JET colormap to represent agent speed. The inference phase automatically decomposes this real-world spatio-temporal trajectory (from [Seyfried 2019]) into several segments. Figure 9d is a view of the segments retargeted after specifying the option to randomly perturb the order of the original segments and to extend to three agents, thus effectively yielding a longer sequence of stop-n-go motion for more agents.

Figure 10 shows another example that used our interactive design tool to create crowd motions immitating complex real-world scenarios (Figure 10a-b, see also the video). Figure 10c contains an image of a dense crowd motion during a Comic Fest in Japan.

Figure 10: Real-World Scenarios. a, b) Sketch segments (cropped) at keyframes t=0s, t=12s of busy market; c) a video frame of dense, organized motion in a city park (i.e., Comic Market in Tokyo); d) resulting crowd animation of the recreation of the busy market at t=5s; e) A retargeted crowd animation of the same using an artistic squiggly motion.

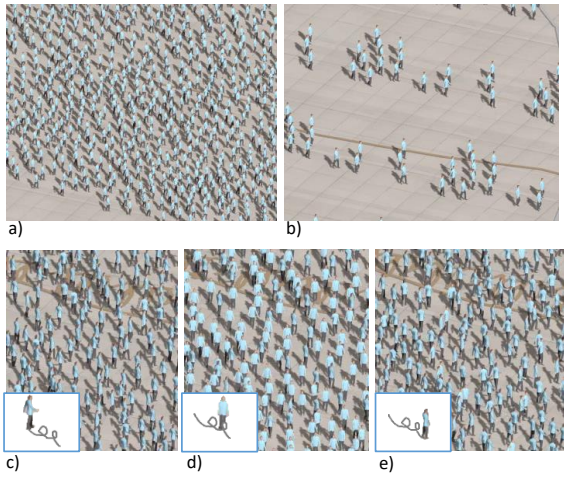

Figure 11: Crowd Motion Mixing and Matching. a) The original animation; b) a grouping style was pasted to the same animation; c-e) three frames of the crowd animation to which a spiral-like pattern was pasted.

Figure 10d shows a frame from our corresponding crowd animation. Figure 10e shows the result of a retargeting using an artistic squiggly motion applied to one crowd segment.

Finally, Figure 11 contains several crowd motion changes to the original crowd motion in Figure 11a. The original crowd motion is about 1,000 agents moving across the scene. The user can retarget the original motion scenario to a new scenario where agents move in groups, as now observed in Figure 11b. Alternatively, the user can sketch a spiral-like trajectory and effectively "paste" this trajectory onto the original trajectory. The effect is the spiral-like trajectory being retargeted to the crowd motion of 1,000 agents and agents appearing to perform spiral like motions asynchronously (see Figure 11c-e). In an analogous way, the stop-n-go motion is inferred from real-world trajectories (*i.e.,* from [Seyfried 2019]) and is pasted into the original crowd motion to produce stop-n-go motion for many agents (see video).

Comparison. We highlight that the most similar prior systems to our method are the compelling group formation approaches of Takahashi et al. [2009] and Ju et al. [2010]. The former enables interpolating between spatial arrangements (and not spatio-temporal components). The latter does not enable specifying the overall crowd motion (in their single example it is user specified). Their trajectories are limited to combinations of short (one second) straight or curved motion. Thus, for example the scenario from Figure 1 is not possible, a stop-n-go motion is not possible (Figure 11), and a spiral motion is cumbersome for it requires sequencing many formation and trajectory models (Figure 11).

Limitations. Our method is not without limitations. In particular, our current CMM inference assumes that the motions within the trajectories provided to it are similar in the spatio-temporal sense. Thus, while trajectories between CMMs can vary, we assume each CMM has self-similar trajectories. In addition, we have not included optimizations to handle a large number of CMMs. Finally, our displacement and speed parameterization for path segments is not able to represent all types of trajectories (*e.g.,* structured/regular trajectories or symmetrical arrangements).

8 CONCLUSION AND FUTURE WORK

We have introduced an inverse modeling method to determine a set of crowd motion models from real or virtual trajectory data and showed how to interactively generate novel scenarios. Our approach has two main phases: during the inference phase source and destination clusters are identified, followed by decomposing the motion into piecewise linear path segments and additional displacement and speed profiles. This decomposition then enables mixing altering components from one or more crowd motion models. During the design phase, the sources and destinations, path segments, and path segment order can be altered and mapped to a new environment. Our system has been applied to real data, sketched data immitating real-world scenarios, and to fully virtual environments.

Going forward, there are several avenues of future work. While we demonstrated results using a space colonization crowd simulation model [de Lima Bicho et al. 2012], we would like to showcase our approach using others crowd simulators including ORCA [van den Berg et al. 2011] and Power-Law [Karamouzas et al. 2014]. Moreover, since we have potentially many agents approximately following the path segments, we could train an artificial neural network to capture the additional subtle details from real-world data, rather than only via the displacement and speed profiles. During retargeting, the trained network could then reproduce the subtle details not well captured by our method. Finally, we look to a GPU implementation to simulate and design even larger crowd motions.

ACKNOWLEDGMENTS

This research was funded in part by National Science Foundation grants #10001387 and #1835739.

REFERENCES

Kfir Aberman, Rundi Wu, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. 2019. Learning Character-agnostic Motion for Motion Retargeting in 2D. *ACM Trans. Graph.* 38, 4, Article 75 (2019), 75:1–75:14 pages.

Richard Bellman. 2013. *Dynamic programming*. Courier Corporation.

Aniket Bera, Sujeong Kim, and Dinesh Manocha. 2015. Efficient Trajectory Extraction and Parameter Learning for Data-driven Crowd Simulation. In *Proc. of the 41st Graphics Interface Conference (GI '15)*. 65–72.

Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. 2010. A connection between partial symmetry and inverse procedural modeling. In *ACM Trans. Graph*, Vol. 29. ACM, 104.

Alessandro de Lima Bicho, Rafael Araújo Rodrigues, Soraia Raupp Musse, Cláudio Rosito Jung, Marcelo Paravisi, and Léo Pini Magalhães. 2012. Simulating crowds based on a space colonization algorithm. *Comp. & Graph.* 36, 2 (2012), 70–79.

Ilke Demir, Daniel G Aliaga, and Bedrich Benes. 2016. Proceduralization for editing 3d architectural models. In *3D Vision (3DV)*. IEEE, 194–202.

T. B. Dutra, R. Marques, J.B. Cavalcante-Neto, C. A. Vidal, and J. Pettré. 2017. Gradient-based Steering for Vision-based Crowd Simulation Algorithms. *Comput. Graph. Forum* 36, 2 (May 2017), 337–348. http://dl.acm.org/citation.cfm?id=3128975.3129006

Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. 2015. WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds. *ACM Trans. Graph.* 34, 4, Article 106 (July 2015), 11 pages.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.

Ignacio Garcia-Dorado, Daniel G. Aliaga, and Satish V. Ukkusuri. 2014. Designing large-scale interactive traffic animations for urban modeling. *Comp. Graph. Forum* 33, 2 (2014), 411–420.

Abhinav Golas, Rahul Narain, and Ming Lin. 2013. Hybrid Long-range Collision Avoidance for Crowd Simulation. In *Proc. of I3D (I3D '13)*. ACM, 29–36.

Kevin Jordao, Panayiotis Charalambous, Marc Christie, Julien Pettré, and Marie-Paule Cani. 2015. Crowd art: density and flow based crowd motion design. In *MIG*.

Kevin Jordao, Julien Pettré, Marc Christie, and Marie-Paule Cani. 2014. Crowd sculpting: A space-time sculpting method for populating virtual environments. *Comput. Graph. Forum* 33 (2014), 351–360.

Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. 2010. Morphable Crowds. In *ACM SIGGRAPH Asia*. ACM, Article 140, 10 pages. https://doi.org/10.1145/1866158.1866162

Alain Juarez-Perez and Marcelo Kallmann. 2018. Fast Behavioral Locomotion with Layered Navigation Meshes. In *Proc. of I3D (I3D '18)*. ACM, Article 8, 6 pages. https://doi.org/10.1145/3190834.3190841

Mubbasir Kapadia, Shawn Singh, Brian Allen, Glenn Reinman, and Petros Faloutsos. 2009. SteerBug: An Interactive Framework for Specifying and Detecting Steering Behaviors. In *Proc. of SCA (SCA '09)*. ACM, 209–216.

Mubbasir Kapadia, Xu Xianghao, Maurizio Nitti, Marcelo Kallmann, Stelian Coros, Robert W. Sumner, and Markus Gross. 2016. Precision: Precomputing Environment Semantics for Contact-rich Character Animation. In *Proc. of I3D (I3D '16)*. 29–37.

Ioannis Karamouzas, Brian Skinner, and Stephen J Guy. 2014. Universal power law governing pedestrian interactions. *Physical Rev. letters* 113, 23 (2014), 238701.

Ioannis Karamouzas, Nick Sohre, Ran Hu, and Stephen J. Guy. 2018. Crowd Space: A Predictive Crowd Analysis Technique. *ACM Trans. Graph.* 37, 6, Article 186 (Dec. 2018), 14 pages. https://doi.org/10.1145/3272127.3275079

Ioannis Karamouzas, Nick Sohre, Rahul Narain, and Stephen J. Guy. 2017. Implicit Crowds: Optimization Integrator for Robust Crowd Simulation. *ACM Trans. Graph.* 36, 4, Article 136 (2017), 13 pages. https://doi.org/10.1145/3072959.3073705

Jongmin Kim, Yeongho Seol, Taesoo Kwon, and Jehee Lee. 2014. Interactive Manipulation of Large-scale Crowd Animation. *ACM Trans. Graph.* 33, 4, Article 83 (July 2014), 10 pages.

Sujeong Kim, Stephen J. Guy, Dinesh Manocha, and Ming C. Lin. 2012. Interactive Simulation of Dynamic Crowd Behaviors Using General Adaptation Syndrome Theory. In *Proc. of I3D (I3D '12)*. ACM, 55–62.

Eugene L Lawler. 1972. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management science* 18, 7 (1972), 401–405.

Kang Hoon Lee, Myung Choi, Qyoun Hong, and J. Lee. 2007. Group behavior from video: A data-driven approach to crowd simulation. *Proc. of SCA* 2007, 109–118. https://doi.org/10.1145/1272690.1272706

Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. 2007. Crowds by Example. *Comput. Graph. Forum* 26 (2007), 655–664.

Yi Li, Marc Christie, Orianne Siret, Richard Kulpa, and Julien Pettré. 2012. Cloning Crowd Motions. In *Proc. of SCA (SCA '12)*. Eurographics Association, 201–210.

Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C. Lin. 2009. Aggregate Dynamics for Dense Crowd Simulation. *ACM Trans. Graph.* 28, 5, Article 122 (2009), 122:1–122:8 pages.

Gen Nishida, Adrien Bousseau, and Daniel G Aliaga. 2018. Procedural Modeling of a Building from a Single Image. In *Comp. Graph. Forum*, Vol. 37. 415–429.

Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. 2016. Interactive Sketching of Urban Procedural Models. *ACM Trans. Graph.* 35, 4, Article 130 (July 2016), 11 pages.

Aline Normoyle, Maxim Likhachev, and Alla Safonova. 2014. Stochastic Activity Authoring with Direct User Control. In *Proc. of I3D (I3D '14)*. ACM, 31–38.

Chonhyon Park, Jae Sung Park, Steve Tonneau, Nicolas Mansard, Franck Multon, Julien Pettré, and Dinesh Manocha. 2016. Dynamically Balanced and Plausible Trajectory Planning for Human-like Characters. In *Proc. of I3D (I3D '16)*. 39–48.

Sachin Patil, Jur van den Berg, Sean Curtis, Ming C. Lin, and Dinesh Manocha. 2011. Directing Crowd Simulations Using Navigation Fields. *IEEE Transactions on Visualization and Computer Graphics* 17, 2 (Feb. 2011), 244–254.

Mikel Rodriguez, Ivan Laptev, Josef Sivic, and Jean-Yves Audibert. 2011. Density-aware person detection and tracking in crowds. *2011 International Conference on Computer Vision* (2011), 2423–2430.

Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. 2000. Video Textures. In *Proc. of Siggraph*. 489–498.

Boltes Maik Seyfried, Armin. 2019. *Pedestrian Dynamics Data Archive*. website: http://ped.fz-juelich.de/da/doku.php?id=start; accessed January 10, 2019.

Alexander Shoulson, Nathan Marshak, Mubbasir Kapadia, and Norman I. Badler. 2013. ADAPT: The Agent Development and Prototyping Testbed. In *Proc. of I3D (I3D '13)*. 9–18.

Shawn Singh, Mubbasir Kapadia, Billy Hewlett, Glenn Reinman, and Petros Faloutsos. 2011. A Modular Framework for Adaptive Agent-based Steering. In *I3D (I3D '11)*. ACM, 141–150 PAGE@9.

Ondrej Stava, B. Benes, Radomír Měch, D. G. Aliaga, and P. Kristof. 2010. Inverse procedural modeling by automatic generation of L-systems. 29, 2 (2010), 665–674.

Ondrej Stava, Sören Pirk, Julian Kratt, Baoquan Chen, Radomir Mech, Oliver Deussen, and Bedrich Benes. 2014a. Inverse Procedural Modelling of Trees. *Comp. Graph. Forum* 33, 6 (2014), 118–131.

Ondrej. Stava, Sören. Pirk, Julain Kratt, Baoquan Chen, Radomir Měch, Oliver Deussen, and Benes Benes. 2014b. Inverse Procedural Modelling of Trees. *Comp. Graph. Forum* 33, 6 (2014), 118–131. https://doi.org/10.1111/cgf.12282 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12282

Mankyu Sung, Michael Gleicher, and Stephen Chenney. 2004. Scalable behaviors for crowd simulation. *Comp. Graph. Forum* (2004).

Shigeo Takahashi, Kenichi Yoshida, Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Sung Yong Shin. 2009. Spectral-Based Group Formation Control. *Comput. Graph. Forum* 28 (2009), 639–648.

Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. 2012. Learning design patterns with bayesian grammar induction. In *UIST*. ACM, 63–74.

Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. 2011. Metropolis proced. modeling. *ACM Trans. Graph.* 30, 2 (2011), 11.

Adrien Treuille, Seth Cooper, and Zoran Popović. 2006. Continuum crowds. *ACM Trans. Graph.* 25, 3 (2006), 1160–1168.

Branislav Ulicny, Pablo de Heras Ciechomski, and Daniel Thalmann. 2004. Crowdbrush: Interactive Authoring of Real-time Crowd Scenes. In *Proc. of SCA (SCA '04)*. Eurographics Association, 243–252.

Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. 2011. Reciprocal n-Body Collision Avoidance. In *Robotics Research*, Cédric Pradalier, Roland Siegwart, and Gerhard Hirzinger (Eds.). Springer Berlin Heidelberg, 3–19.

Carlos A. Vanegas, Daniel G. Aliaga, and Bedrich Benes. 2010. Building reconstruction using manhattan-world grammars. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 0 (2010), 358–365.

Carlos A. Vanegas, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Paul Waddell. 2012. Inverse Design of Urban Procedural Models. *ACM Trans. Graph.* 31, 6, Article 168 (Nov. 2012), 11 pages.

He Wang and Carol O Sullivan. 2016. Globally Continuous and Non-Markovian Crowd Activity Analysis from Videos. In *Computer Vision – ECCV 2016*, Vol. 9909.

He Wang, Jan Ondřej, and Carol O Sullivan. 2016. Path Patterns: Analyzing and Comparing Real and Simulated Crowds. In *Proc. of I3D (I3D '16)*. ACM, 49–57.

D. Wolinski, S. J. Guy, A.-H. Olivier, M. Lin, D. Manocha, and J. Pettré. 2014. Parameter Estimation and Comparative Evaluation of Crowd Simulations. *Comput. Graph. Forum* 33, 2 (May 2014), 303–312.

David Wolinski, Ming C. Lin, and Julien Pettré. 2016. WarpDriver: Context-aware Probabilistic Motion Prediction for Crowd Simulation. *ACM Trans. Graph.* 35, 6, Article 164 (Nov. 2016), 11 pages.

Sai-Keung Wong, Yi-Hung Chou, and Hsiang-Yu Yang. 2018. A Framework for Simulating Agent-based Cooperative Tasks in Crowd Simulation. In *Proc. of I3D (I3D '18)*. ACM, Article 11, 10 pages. https://doi.org/10.1145/3190834.3190839

Barbara Yersin, Jonathan Maïm, Julien Pettré, and Daniel Thalmann. 2009. Crowd Patches: Populating Large-scale Virtual Environments for Real-time Applications. In *Proc. of I3D (I3D '09)*. ACM, 207–214.