DOI: 10.1111/cgf.14491
EUROGRAPHICS 2022 / R. Chaine and M. H. Kim
(Guest Editors)

Volume 41 (2022), Number 2

Interaction Fields: Intuitive Sketch-based Steering Behaviors for Crowd Simulation

A. Colas[1], W. van Toll[1,2], K. Zibrek[1], L. Hoyet[1], A.-H. Olivier[1], J. Pettré[1]

[1]Univ Rennes, Inria, CNRS, IRISA, France [2] Breda University of Applied Sciences, The Netherlands

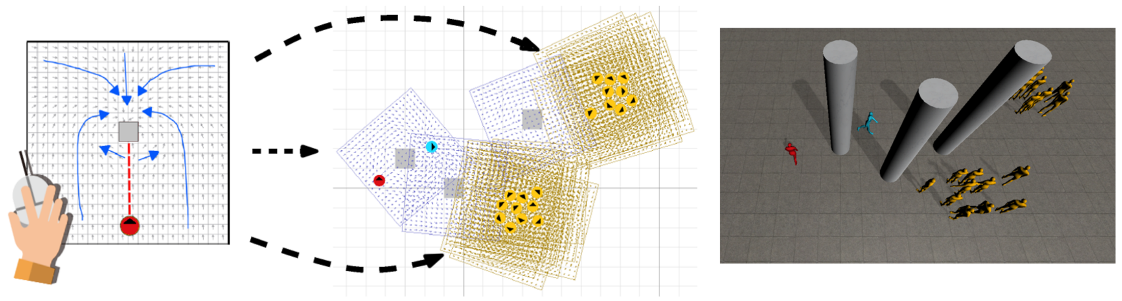


Figure 1: *We present interaction fields (IFs) for sketch-based design of local agent interactions in crowd simulation. Left: A user sketches* guide curves *(shown in blue), which are converted to an IF grid. The purpose of this specific IF is to let agents move behind one object to hide from another. Middle: 2D top view of the simulation. We let all gray obstacles and orange agents* emit *this IF. The blue agent* perceives *these IFs, causing it to hide from the red agent. Right: 3D impression of this scenario, combined with body animation per agent.*

Abstract

The real-time simulation of human crowds has many applications. In a typical crowd simulation, each person ('agent') in the crowd moves towards a goal while adhering to local constraints. Many algorithms exist for specific local 'steering' tasks such as collision avoidance or group behavior. However, these do not easily extend to completely new types of behavior, such as circling around another agent or hiding behind an obstacle. They also tend to focus purely on an agent's velocity without explicitly controlling its orientation. This paper presents a novel sketch-based method for modelling and simulating many steering behaviors for agents in a crowd. Central to this is the concept of an interaction field *(IF): a vector field that describes the velocities or orientations that agents should use around a given 'source' agent or obstacle. An IF can also change dynamically according to parameters, such as the walking speed of the source agent. IFs can be easily combined with other aspects of crowd simulation, such as collision avoidance. Using an implementation of IFs in a real-time crowd simulation framework, we demonstrate the capabilities of IFs in various scenarios. This includes game-like scenarios where the crowd responds to a user-controlled avatar. We also present an interactive tool that computes an IF based on input sketches. This IF editor lets users intuitively and quickly design new types of behavior, without the need for programming extra behavioral rules. We thoroughly evaluate the efficacy of the IF editor through a user study, which demonstrates that our method enables non-expert users to easily enrich any agent-based crowd simulation with new agent interactions.*

CCS Concepts
• ***Computing methodologies*** *→ Motion path planning; Intelligent agents; Real-time simulation;* • ***Human-centered computing*** *→* ***Graphical user interfaces;***

1. Introduction

Crowd simulation is useful for creating lively virtual scenes populated by many moving characters. This has many applications, including games, movies, and VR.

Most crowd simulation techniques are *agent-based* in that they simulate each character as an individual intelligent agent. To steer each agent through the environment in interaction with other agents, many algorithms have been developed for specific purposes such as path planning, collision avoidance, and grouping. Although the resulting models are highly successful, it is difficult to adapt them so that the agents display *new* kinds of behavior for which the algorithms were not designed. A designer can influence the overall paths that agents take, and they can tune simulation parameters to change properties of a specific algorithm, but they cannot easily let agents interact in completely different ways, such as making an agent circle around another agent or hide behind an obstacle. Furthermore, parameter tuning and scenario-specific scripting can be time-consuming.

In response to these problems, the goal of this paper is to simplify the design of local steering behaviours in crowds, by letting users sketch how agents should move in relation to other agents, obstacles, or the environment. Our central concept is an *interaction field* (IF) that defines the velocities or orientations that agents should use around a particular source, such as an obstacle or another agent. An IF can also be made *parametric* to change dynamically according to simulation parameters, such as the current speed of an agent. We also present an editor in which users can sketch IFs, allowing them to quickly and intuitively create a wide variety of new types of agent behavior. IFs can be combined with other crowd simulation techniques, so that users can focus on sketching only those behaviors for which traditional algorithms do not suffice.

Previous research has led to several other methods for artistically modifying the behavior of a crowd,cbut these methods focus on other aspects, such as controlling simulation parameters or editing global trajectories. To the best of our knowledge, we present the first method that allows users to intuitively sketch *local interactions*, i.e. to sketch how agents should move relatively to other (moving) obstacles or agents. By generating IFs from sketches, users can quickly design new behaviors that would otherwise require laborious programming and parameter tuning. Our method enables designers to easily create a wider variety of scenarios, and to drastically speed up the design process of a scenario.

In short, the main *contributions* of this paper are the following:

- We present *interaction fields* (IFs) as a simple yet effective way to model new kinds of steering behaviors in crowds. We show the capabilities of IFs in various *scenarios* that would be difficult to simulate using traditional models alone.
- We present a novel way to compute IFs based on *user sketches*. This results in an IF editor that allows designers to draw new agent behaviors in a small amount of time.
- We present the results of a thorough *user study* that confirms the efficacy of the IF editor for fast behavior sketching.

A preliminary version of this work was presented as a 2-page poster paper [CvTH*20]. We emphasize that we focus on the *steering behavior* of agents, and not on full character animation. As it is common in crowd simulation research, we will model our scenarios in 2D, with agents simplified to disks. Of course, these agents represent (humanoid) characters, and 3D applications must combine the agents' trajectories with appropriate 3D body animations. Although combining steering behaviors with realistic human animation is not the purpose of this work, we will briefly discuss several ways to handle it, and our supplementary video will show examples.

2. Related work

Crowd simulation is a large research area with many facets. Several books and surveys give a good overview [ANMS13; TM13; PAKB16; vTP21]. As this paper focuses on sketch-based design of interactions in agent-based crowd simulation, we will now discuss related work in these areas only.

2.1. Agent-based crowd simulation

Most crowds simulation models are *agent-based*, also referred to as *microscopic*. These models simulate the behavior of each person ('agent') in the crowd individually. This is in contrast to *macroscopic* approaches that simulate the crowd as a whole [TCP06]. The behavior per agent consists of multiple components, such as global path planning and local collision avoidance. Many frameworks exist that use this principle to simulate large crowds in real-time [PAB07; vTJG15; CBM16; KBB16].

The research in this domain mostly focuses on *local interactions* between agents: updating each agent's velocity at a certain frequency so that each agent proceeds to its goal while responding adequately to neighboring agents and obstacles. The most frequently studied type of local behavior is *collision avoidance*. Algorithms for this purpose let agents move according to a certain principle, such as forces [HM95; KHVO09; ZIK11; KSG14], velocity cost functions [PPD07; vdBLM08; GCC*10; KO10; MHT11; vdBGLM11], or vision [OPOD10; DMC*17; LCMP19]. The *orientation* of an agent is usually not explicitly controlled, although some exceptions to this rule exist [HOD15].

Next to collision avoidance, many other types of local behavior can be considered, such as grouping and following. Reynolds [Rey99] was among the first to propose and implement a list of such 'steering' behaviors. To add a new type of behavior to a crowd simulation, it is common to add a new algorithm, force, or cost function for that specific purpose. Each new behavior requires programming effort, knowledge of simulation details, and parameter tuning.

Some methods increase the range of possibilities by stretching the concept of an agent. For example, Yeh et al. [YCP*08] have modeled special interactions by adding invisible 'proxy agents' to the simulation. This can model (for example) an agent that makes less or more room in a crowd depending on its walking speed. Comparably, the 'situation agents' by Schuerman et al. [SSKF10] are abstract agent-like entities designed to solve specific problems, such as deadlocks at narrow passages. Although such techniques can indeed model additional behaviors, designing a new type of behavior still requires substantial effort and expert knowledge.

The *interaction fields* (IFs) of this paper are an alternative way to design local behavior. One could argue that IFs are still 'rule-based', in the sense that a single IF could be seen as a local behavioral rule. The main difference to traditional local algorithms is that the behavior of IFs follows directly from an input sketch, so new IFs for new types of behavior can easily be sketched by a novice user. We emphasize that IFs are not meant to replace existing algorithms for e.g. collision avoidance; rather, IFs introduce a new way to generate a wide variety of local behaviors based on sketches, without the need to program or tweak simulation details.

2.2. Controlling simulation setting and tuning parameters

The behavior produced by a local algorithm usually depends on a number of parameters. By tuning these parameters, designers have some control over how agents behave, but this tuning is not always intuitive and requires expert knowledge of the simulation model. To improve this, several researchers have proposed systems where a user can interactively edit parameters while the simulation is running [MR05; MMHR16], possibly even by immersing the user into the simulation itself [BBEK20]. Other work has explored ways to automatically tune simulation parameters to match particular data, such as quantitative metrics [WGO*14], controlled randomization [NLS14], textual descriptions [CWL20; LWC20], and start and goal positions of agents [ACC14].

Naturally, in all systems that operate purely on simulation parameters, the results are limited to what parameter variations can produce. Parameters alone do not enable designers to create completely new types of behavior.

2.3. Sketch-based control of agents

Various methods exist that try to give users more intuitive control over a crowd via sketch-based techniques. Most of these provide control over *global paths*, by letting users draw curves for agents to follow [UdHT04; KSA*09; OO09; MM17; BBEK20] or directional hints that are converted to a flow field [PWJ*08; PvdBC*11; KK*14]. Some of these methods also propose sketch-based control of other simulation parameters, such as the walking speed and the smoothness of paths [UdHT04; OO09]. Other work focuses more on changing the geometry of the environment [MMHR16; MM19; SGH20], on fitting a group into a formation [GD13; HSK14], or on giving users high-level control over where agents go and which actions they perform [MR05; KSRF11; KBK16; MBA20].

Another set of solutions is more oriented towards editing existing crowd motion clips to adapt to new situations. Following up on the concept of 'crowd patches' that can be stitched together [YMPT09], there are several methods that let users intuitively deform crowd motion clips to create crowds of different shapes [JPCC14; KSKL14; KL16] or densities [JCC*15]. There is also a line of work that models a crowd as a deformable mesh, to steer it efficiently along (user-specified) paths in the presence of obstacles [KLLT08; HSK12; HSK14; ZZZY20]. A model by Ju et al. [JCP*10] converts example crowd motion to a continuous space, allowing for interpolations between types of motion. Shen et al. [SHW*18] proposed a data-driven technique that chooses the appropriate crowd motions based on multi-touch gestures.

These techniques are all successful, but they focus on other simulation aspects than the *local interactions between* agents. Thus, they operate on a different scale than the method we propose.

The idea of sketch-based control is also popular in many other computer-graphics domains [BC20], including character body animation [GCR13; GRGC15; CBL*16; CZWL20]. We remind the reader that our method focuses on generating *trajectories* for agents, and not on animating their bodies in 3D. For this reason, we will not discuss animation-related work in detail here.

2.4. Positioning

This paper proposes a new sketch-based method for designing local interactions between agents. A concept that seems close at first sight is the *navigation field* by Patil et al. [PvdBC*11]: a grid that proposes an optimal walking direction at any point in the environment, possibly based on sketches. However, whereas navigation fields specify *global* paths through the environment, interaction fields (IFs) specify how agents *locally* behave around other agents or obstacles. As such, IFs can move through the environment during the simulation, and they can change according to parameters.

We believe that the difficulty of creating new kinds of interactions has limited the variety of scenarios that can be simulated. As such, the list of local interactions proposed by Reynolds [Rey99] has never been substantially extended, despite the many developments in terms of simulation models. As discussed earlier, the agent concept can be applied in a more abstract way to model additional behaviors [YCP*08; SSKF10], but this does not necessarily make new behaviors easy to design for non-experts. By contrast, IFs are specified in a purely visual way, allowing novice users to create new behaviors with relative ease.

3. System overview

Our crowd simulation takes place in a bounded 2D environment $\mathcal{E} \subset \mathbb{R}^2$ with $m \geq 0$ *obstacles* $\{O_i\}_{i=0}^{m-1}$ and $n \geq 0$ *agents* $\{A_i\}_{i=0}^{n-1}$. It is common to implement obstacles as simple polygons and to model each agent A_i as a disk with radius r_i. However, our method does not explicitly rely on these implementation choices.

The simulation uses discrete time steps (*frames*). In each frame, every agent A_i compute a new value for its acceleration $\mathbf{a}_i$, which will induce a change in its velocity $\mathbf{v}_i$ and position $\mathbf{p}_i$. As explained in Section 2.1, the process of computing $\mathbf{a}_i$ can be based on algorithms for e.g. path following, collision avoidance, and group behavior. Each agent A_i also has an *orientation* $\mathbf{o}_i \in \mathbb{S}^1$ (a 2D unit vector) that represents the direction that A_i is facing. This paper will present *interaction fields* (IFs) as an additional way to control the velocities and orientations of agents. IFs can be used together with other navigation algorithms, as well as independently.

Figure 2 shows an overview of the proposed system that combines IF design, crowd simulation, and character animation. The details per component will be provided throughout this paper.

First, a user sketches an IF in an *editing tool*, which we will describe in Section 5. The user can draw elements onto a canvas, and this sketch is automatically converted to an IF. The user can then inspect the resulting agent behavior in the simulation (to be described below) and return to the sketching phase if they wish, until they obtain the desired agent behavior. To set up a complete simulation scenario, the user should also specify which objects *emit* the IF (as *sources*) and which agents *respond* to it (as *receivers*).

Next, the sketched IFs are applied to the *simulation* in the way presented in Section 4. (For ease of comprehension, this paper will discuss the simulation first and the IF editor second.) In each simulation frame, every agent A_i performs a sequence of tasks. First, A_i should respond to the IFs emitted by nearby sources, which results in an *IF velocity* and *IF orientation* proposed by these IFs. Next, A_i

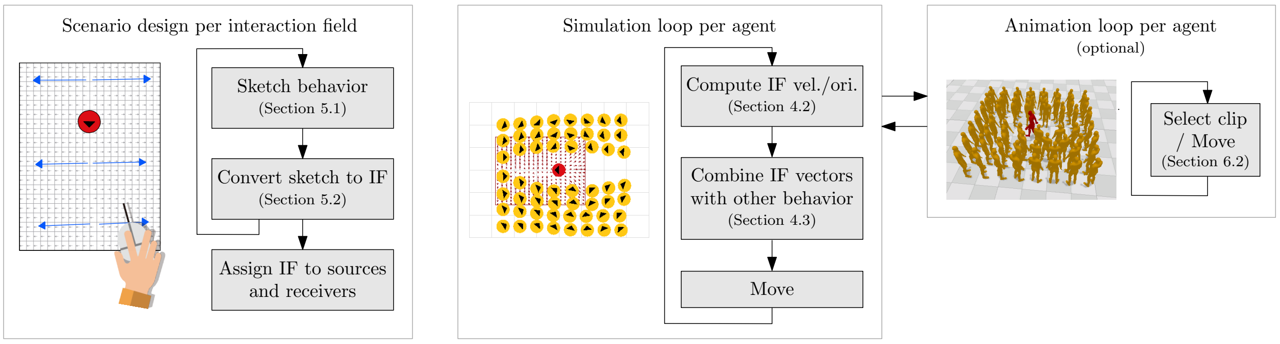

Scenario design per interaction field
Sketch behavior (Section 5.1)
Convert sketch to IF (Section 5.2)
Assign IF to sources and receivers
Simulation loop per agent
Compute IF vel./ori. (Section 4.2)
Combine IF vectors with other behavior (Section 4.3)
Move
Animation loop per agent (optional)
Select clip / Move (Section 6.2)

Figure 2: *Outline of a complete simulation system with interaction fields. See Section 3 for a textual overview.*

can combine this result with other behavior such as collision avoidance, resulting in a new velocity and orientation to use. Finally, A_i moves and rotates according to the computed vectors.

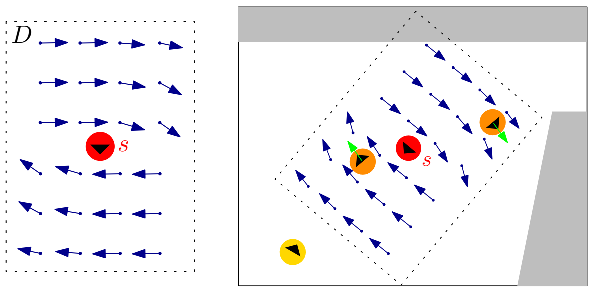
It is also possible to combine the 2D simulation output with animated 3D characters. Although this is not the focus of our work, it is an important and non-trivial component for many applications. We will discuss the options and our implementation in Section 6.2.

4. Interaction fields

This section defines the concept of an *interaction field* (IF) and explains how to integrate IFs into a crowd simulation loop.

4.1. Basic definition of an IF

Overall, a single IF describes either the *velocities* or the *orientations* that agents should use in the vicinity of a particular *source*, which we denote by s. We also say that the source *emits* the IF. A source can be an agent, an obstacle, or any other aspect of the environment that should induce a certain kind of behavior.



(a) IF definition **(b) Mapped onto the environment**

Figure 3: *(a) An interaction field is a vector field (shown here in blue) that prescribes velocity or orientation vectors in a domain D around a source object s (here: the red agent). (b) During the simulation, the IF is mapped onto the environment to match the current position and orientation of s. Other agents (in orange) use this mapped IF to compute a velocity or orientation (in green), which they can apply directly or combine with other navigation algorithms. Agents outside the domain (in yellow) are not affected.*

Because an IF prescribes behavior around a source s, we define it in a Euclidean coordinate system relative to s, with s located at the origin $(0,0)$ and oriented towards the negative y-axis. Using this, a *velocity IF* with source s and domain $D \subset \mathbb{R}^2$ is a vector field

$$VIF_{s,D} : D \to \mathbb{R}^2$$

that maps any position $\mathbf{p} \in D$ to a 2D vector $VIF_{s,D}(\mathbf{p})$, indicating the velocity that any agent should use at this position.

Likewise, an *orientation IF* is a function

$$OIF_{s,D} : D \to \mathbb{S}^1$$

that maps any $\mathbf{p} \in D$ to a 2D unit vector $OIF_{s,D}(\mathbf{p})$ that agents should use as their orientation. Figure 3(a) shows an abstract example of an IF. Whenever it does not matter whether an IF concerns velocities or orientations, we will use the notation $IF_{s,D}$. Note that an IF prescribes a vector for *all* points in the domain D. Our figures will only show sample velocities for the sake of illustration.

4.2. Applying IFs during the simulation

An IF is defined relatively to a source s. During the crowd simulation, the position $\mathbf{p}_s$ and orientation $\mathbf{o}_s$ of s may change over time, especially if s is an agent. To apply the function $IF_{s,D}$ at runtime, the IF should be translated and rotated to match the current values of $\mathbf{p}_s$ and $\mathbf{o}_s$. Informally, if we see $IF_{s,D}$ as a pre-defined 'picture' around s, we should always line up this picture with how s is currently positioned and oriented. We call the result the *mapped IF*, and we denote it by $IF'_{s,D}$. Figure 3(b) shows an example.

It is important to note that this mapping can remain implicit during the simulation. There is no need to translate and rotate complete IFs at run-time. For any position $\mathbf{q} \in \mathcal{E}$, we can easily compute the relevant IF vector $IF'_{s,D}(\mathbf{q})$ by applying the inverse mapping to $\mathbf{q}$.

One special case is worth mentioning: if the source s is the entire environment $\mathcal{E}$, then $D = \mathcal{E}$ as well, and there is no mapping to apply during the simulation ($IF'_{s,D} = IF_{s,D}$). Such an IF is similar to a navigation field [PvdBC*11]: it prescribes vectors for the whole environment, and not for the neighborhood of one specific object.

The purpose of an IF is to model a single type of behavior around a source, so most simulations will feature multiple IFs at the same time. As part of the scenario's design, the user should specify for each IF which objects *emit* it and which agents *respond* to it. Consequently, it is possible for agents to respond to only *some* IFs and to ignore others, i.e. to model different behavior for different agents.

At any moment in the simulation, each agent A_i should respond to the relevant interaction fields emitted by nearby sources. To this end, let $\mathcal{I} = \{VIF_{s_j,D_j}\}_{j=0}^{k-1}$ be the set of all *velocity* IFs to which A_i can respond *and* that currently have $\mathbf{p}_i$ in their mapped domain. The *IF velocity* $\mathbf{v}_i^{\mathrm{IF}}$ for A_i is defined as a weighted average of the vectors that these IFs propose:

$$\mathbf{v}_i^{\mathrm{IF}} = \frac{\sum_j VIF'_{s_j,D_j}(\mathbf{p}_i) \cdot w_j}{\sum_j w_j} \tag{1}$$

where the w_j are weights for prioritizing between IFs, e.g. to increase the influence of an IF as an agent moves closer to the source. It will often be sufficient to use $w_j = 1$ for all j. For orientation IFs, we define the *IF orientation* $\mathbf{o}_i^{\mathrm{IF}}$ for A_i analogously, the only difference being that we explicitly normalize the result.

4.3. Combining IFs with other simulation components

There are several ways to combine the IF velocity and orientation with other simulation aspects (such as collision avoidance). In most traditional crowd simulations, the behavior of each agent A_i per simulation frame is already subdivided into multiple steps:

1. Compute a *preferred velocity* $\mathbf{v}_i^{\mathrm{pref}}$ that would send the agent towards its goal, possibly with the help of a global path.
2. Compute a *new velocity* $\mathbf{v}_i^{\mathrm{new}}$ that stays close to $\mathbf{v}_i^{\mathrm{pref}}$ while following local rules for collision avoidance, group behavior, etc. This yields an acceleration $\mathbf{a}_i := (\mathbf{v}_i^{\mathrm{new}} - \mathbf{v}_i)/\Delta t$, where Δt is the length of this simulation frame in seconds.
3. If the agent is currently colliding with other agents or obstacles, compute *contact forces* $\mathbf{f}_i^{\mathrm{c}}$ and update the acceleration: $\mathbf{a}_i := \mathbf{a}_i + \mathbf{f}_i^{\mathrm{c}}/m$, where m is the agent's mass (usually 1).
4. Update the agent's velocity and position via Euler integration:
$$\mathbf{v}_i := \mathbf{v}_i + \mathbf{a}_i \cdot \Delta t, \quad \mathbf{p}_i := \mathbf{p}_i + \mathbf{v}_i \cdot \Delta t.$$

Both $\mathbf{v}_i^{\mathrm{new}}$ and $\mathbf{v}_i$ are typically clamped to a maximum walking speed $v_i^{\max}$ to prevent unrealistically large velocities.

As mentioned earlier, most crowd simulations do not explicitly control the agent's orientation $\mathbf{o}_i$. Thus, *orientation* IFs can be trivially added to the simulation loop in a separate step:

5. Compute the IF orientation $\mathbf{o}_i^{\mathrm{IF}}$. If $\mathbf{o}_i^{\mathrm{IF}} \neq \mathbf{0}$, update the agent's orientation as $\mathbf{o}_i := \mathbf{o}_i^{\mathrm{IF}}$. Otherwise, keep $\mathbf{o}_i$ unchanged, or update it in a 'traditional' way, e.g. as an average of $\mathbf{v}_i^{\mathrm{pref}}$ and $\mathbf{v}_i^{\mathrm{new}}$.

To add *velocity* IFs to the system, we have the choice between letting the IF velocity $\mathbf{v}_i^{\mathrm{IF}}$ (Equation 1) influence an agent's *preferred* velocity (in step 1) or its *new* velocity (in step 2). We will use the first option in our implementation. This allows for an intuitive combination of IFs and collision avoidance, where IFs play an 'advising' role and collision avoidance has the final say. The navigation fields of Patil et al. [PvdBC*11] are also used in this way.

Thus, we use IFs as an alternative way to compute a preferred velocity $\mathbf{v}_i^{\mathrm{pref}}$. It is also possible to let $\mathbf{v}_i^{\mathrm{pref}}$ depend on IFs *and* on other factors (such as goal reaching) at the same time. We will use this in some of our example scenarios; Section 6.1 will describe the underlying simulation settings.

4.4. Parametric interaction fields

Next, we extend IFs so that they can change during the simulation according to parameters. These parameters may affect both the vectors and the domain of the IF. In other words, a parametric IF encapsulates different 'ordinary' IFs for different parameter values.

Formally, a *parametric velocity IF* with l scalar parameters can be described as a function

$$PVIF_s : \mathbb{R}^l \to (D^* \to \mathbb{R}^2)$$

where the resulting velocity vectors and the domain D^* now also depend on the l parameter values. A parametric *orientation* IF can be defined analogously. Theoretically, there is no limit on the number of parameters. In this work, though, we create IFs based on user sketches, and we will use at most 1 parameter to keep the design process intuitive. We will now discuss two specific types of parametric IFs that are supported by our sketching tool.

4.4.1. Keyframes and interpolation

One way to specify a parametric IF is to define IFs for a few specific values of a single parameter. These IFs then act as *keyframe IFs* at runtime, and the IF for any other parameter value is defined via linear interpolation between the two nearest keyframe IFs.

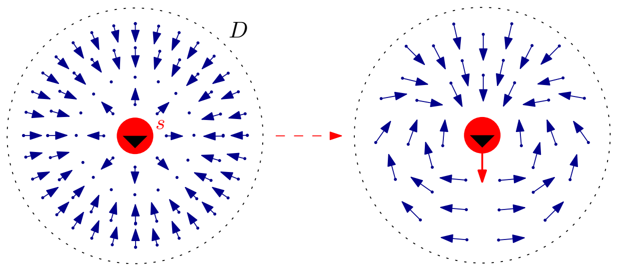
For example, Figure 4(a) shows a parametric velocity IF with two keyframes, where the parameter is the speed of the source agent s. When s is standing still, agents will gather around s in a circle. When s is moving at a certain predefined speed, agents will attempt to follow s from behind. There are infinitely many vector fields for the source speeds in-between. During the simulation, agents will use an interpolated field that matches the current speed of s.

Next to the speed of the source agent, other examples of parameters could be the width or height of a source obstacle (to apply the IF to obstacles of various sizes), the current simulation time (to model behavior that changes over time), or the local crowd density around an agent (to model density-dependent behavior). A parameter could also represent an agent's state of mind, such as its hastiness or the amount of panic it experiences.

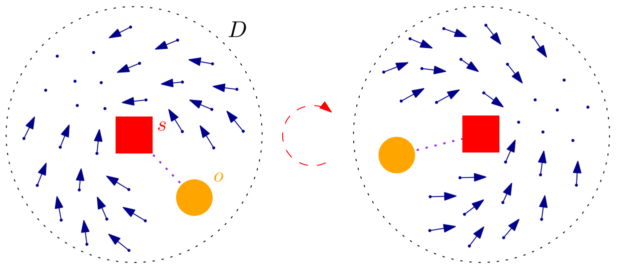
The simulation never needs to fully compute an interpolated IF. In any simulation frame, an agent only needs to compute a single output vector for each parametric IF in range.

Formally, let there be k keyframe IFs associated to k parameter values: $\{\langle q_j, KIF_j \rangle\}_{j=0}^{k-1}$, ordered by increasing q_j values. Assume for now that all keyframe IFs have the same domain D. Given a parameter value q, the parametric IF is defined as follows for any point $\mathbf{p} \in D$:

- If $q < q_0$, then $PIF_s(\mathbf{p}) = KIF_0(\mathbf{p})$.
- If $q \geq q_{k-1}$, then $PIF_s(\mathbf{p}) = KIF_{k-1}(\mathbf{p})$.
- Otherwise, $q_j \leq q < q_{j+1}$ for some j, and
$$PIF_s(\mathbf{p}) = (1-\lambda) \cdot KIF_j(\mathbf{p}) + \lambda \cdot KIF_{j+1}(\mathbf{p}),$$
where $\lambda = (q - q_j)/(q_{j+1} - q_j)$.



(a) Based on the source speed



(b) Based on an angular relation

Figure 4: *Parametric interaction fields. (a) Example of an IF that depends on the speed of its source agent s. (b) Example of an IF that depends on the angular relation between the source s and another object o. Section 4.4 explains the behavior modelled by these IFs.*

If two subsequent keyframe IFs have *different domains*, we require that any domain in-between can be obtained via linear interpolation as well. For example, this is the case if the domains are both axis-aligned rectangles or both disks. The IF vector $PIF_s(\mathbf{p})$ is then only defined if $\mathbf{p}$ lies inside the interpolated domain.

The concept of keyframe IFs can be extended to more than one parameter. In that case, each keyframe will be associated to a point in a higher-dimensional parameter space. As mentioned earlier, though, we will focus on single-parameter examples because these are still relatively intuitive for non-expert users to design.

4.4.2. Relation between objects

A parameter of an IF could also be a relation between two objects a and b. Possible examples are the distance between $\mathbf{p}_a$ and $\mathbf{p}_b$, or the angle between the vector $\mathbf{p}_b - \mathbf{p}_a$ and the x-axis.

As a concrete example, Figure 4(b) shows a velocity IF that lets agents move behind a source s (typically an obstacle) to hide from another object o (typically a specific agent A_k). In this specific example, the parameter of the IF is the angle α between $\mathbf{p}_o - \mathbf{p}_s$ and the x-axis. The effect of α is that it simply rotates the IF: it does not affect the IF vectors themselves, but it only changes how the IF is mapped onto the environment. In contrast to regular IFs, this mapping now no longer depends on the orientation of the source s.

Note that this example can theoretically be combined with keyframe IFs, where the keyframes determine the IF vectors and the angular relation determines the mapping onto $\mathcal{E}$. The result would be a parametric IF with two parameters.

In our IF editor, for simplicity, an angular relation between s and another object o is currently the only object relation that users can draw. A *distance-based* relation between two objects could be implemented with the help of keyframes again: the user specifies which two objects determine the distance parameter, and then they draw keyframes with different distances between these objects.

5. Sketch-based construction of interaction fields

We have developed a graphical interface in which users can intuitively sketch IFs. This section describes the components of this 'IF editor' and their mathematical meaning for the IF being drawn.

In the IF editor, the user starts by defining a bounding shape D^b, which will serve as the IF domain D. The IF editor then creates a rectangular canvas on which the user can draw. Next, the user can draw elements onto the IF canvas to specify parts of the IF. Section 5.1 will describe these elements in more detail.

Finally, the program can convert a drawing to a discretized IF: a rectangular grid of vectors, with a user-specified level of precision. This conversion process, which we will describe in Section 5.2, uses an interpolation scheme to fill in any regions where the user has not drawn. Of course, the user can adapt this result if desired, by drawing additional elements and then rebuilding the grid.

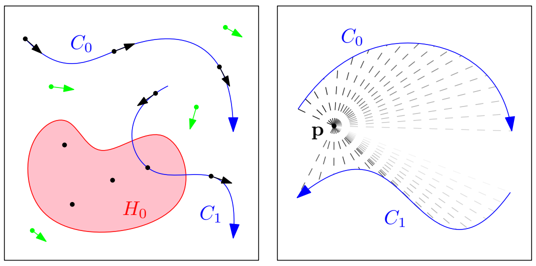
5.1. Main elements of the IF editor

The user can draw three main types of elements in the IF editor, and a sketch can contain multiple elements of each type.

An **object** is anything that can serve as the *source* of an IF. In our IF editor, it can be an agent (visualized as a disk) or a polygon (which can represent an obstacle or something more abstract). One of the objects on the canvas can be marked as the source object s. Other (non-source) objects can be drawn as a visual aid, or to help define a *parametric* IF. We will explain this further in Section 5.3.

A **guide curve** is a curve $C_i : [0,1] \to \mathbb{R}^2$, with an associated magnitude v_i, that exactly specifies the IF vectors along that curve. For any point $\mathbf{p}$ that lies on C_i (i.e. if $\mathbf{p} = C_i(t)$ for some value t), the curve prescribes a vector $\mathbf{c}_i$ with magnitude v_i and direction $\frac{d}{dt}C_i(t)$. Figure 5(a) contains two examples of a guide curve. In the final IF, the vector $IF_{s,D}(\mathbf{p})$ at any point $\mathbf{p}$ will be an interpolation of the vectors proposed by all guide curves. Section 5.2.1 will describe this interpolation. In the IF editor, users can draw a guide curve as a piecewise-linear curve or as a freehand curve. For velocity IFs, the default value for v_i is the maximum walking speed of our agents (1.8 m/s), but the user can change this value per curve. For orientation IFs, v_i is fixed to 1 so that C_i proposes unit vectors.

Finally, a **zero area** is a region $H_j \subset \mathbb{R}^2$ where the IF is 'empty'. For velocity IFs, H_j prescribes the zero vector, meaning that an agent will stand still when it is located inside H_j. For orientation IFs, H_j acts as a hole in the domain D, i.e. as a region where the IF does not propose any specific orientation. Figure 5(a) contains one example of a zero area. Note that zero areas always have priority over guide curves, as will be further clarified in Section 5.2. In the IF editor, users can draw zero areas with a paintbrush tool, or they can erase IF vectors after converting their sketch to a grid.



(a) IF editor elements **(b) Interpolation**

Figure 5: *Concepts of the IF editor. (a) The user can draw guide curves (blue) and zero areas (red) to specify IF vectors; example vectors are shown in black. IF vectors for points in-between will be interpolated (green). (b) For any point p outside all zero areas, the IF vector is a weighted average of all vectors along all guide curves, where weights depend on the distance to p.*

5.2. Converting a sketch to an IF

The user draws the elements of Section 5.1 onto the canvas. We now describe how to convert this sketch to an IF.

5.2.1. Interpolating between guide curves

An important aspect of the conversion is to 'fill in' the IF for areas where nothing has been drawn. To infer a meaningful IF vector for any point $\mathbf{p}$ in the domain D, we interpolate between all vectors proposed along all guide curves. This interpolation is based on *inverse distance weighting* [She68], a commonly used method for estimating values among scattered data points.

Given a set of c guide curves $\mathcal{C} = \{C_i\}_{i=0}^{c-1}$, the estimated IF vector for a point $\mathbf{p} \in D$ is the following:

$$\mathbf{u}(\mathbf{p}, \mathcal{C}) = \frac{\sum_{i=0}^{c-1} \left(\int_0^1 w(\mathbf{p}, C_i(t)) \cdot \mathbf{v}_i(t)\, dt \right)}{\sum_{i=0}^{c-1} \left(\int_0^1 w(\mathbf{p}, C_i(t))\, dt \right)} \quad (2)$$

Here, $w(\mathbf{p}, \mathbf{q}) = \frac{1}{\|\mathbf{p}-\mathbf{q}\|^{\kappa}}$, and $\kappa \in \mathbb{R}^+$ is a power parameter that determines how strongly the influence of a curve point decays along with the distance to $\mathbf{p}$. Preliminary experiments have led to a use of $\kappa = 1.9$ in our implementation. This yields IFs where all vectors are meaningful even with a small number of guide curves. We remind the reader that users can still edit their drawing after the conversion, in case the resulting IF does not match their expectations.

In practice, the integrals in Equation (2) can be approximated by sums, using regularly spaced sample points on each curve. Figure 5(b) gives a visual impression of this interpolation scheme. Note that the number of samples does not affect the curve's importance; it only determines the precision by which C is approximated.

This type of interpolation has several useful properties. First, if a point $\mathbf{p}$ lies exactly on a curve point $C_i(t)$, then $\mathbf{u}(\mathbf{p}) = \mathbf{v}_i(t)$, and other curves do not matter (unless $\mathbf{p}$ is visited multiple times due to curve intersections). Second, if there *are* intersections between or within curves, they do not need to be handled explicitly: the interpolation scheme will simply produce an average vector at an intersection point. Third, the distance-based decay of a curve's influence is only *relative* and not *absolute*. Moving away from a curve point $C_i(t)$ does not 'shrink' the vector that it proposes; it only reduces the relative weight by which it is taken into account.

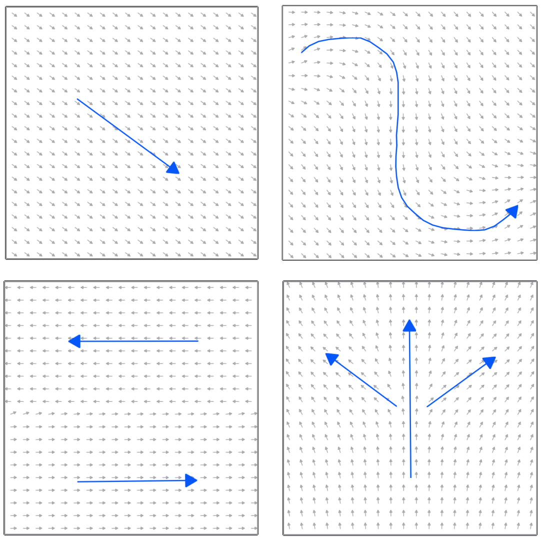


Figure 6: *Examples of guide curves (shown in blue) and their resulting IFs. The gray arrows are the IF vectors (following from the interpolation scheme of Section 5.2.1) on a* 20×20 *sample grid.*

Figure 6 shows a number of examples of IFs for different guide curves. Note that the simplest example contains only one straight guide curve, and its IF contains a uniform vector everywhere.

5.2.2. Computing the final IF

We now define the overall interaction field that can be obtained from a source s, a bounding shape D^b, a set of guide curves $\mathcal{C} = \{C_i\}_{i=0}^{c-1}$, and a set of zero areas $\mathcal{H} = \{H_j\}_{j=0}^{h-1}$.

For a *velocity* IF, the domain D is equal to D^b, and the velocity function $VIF_{s,D}$ works as follows for any point $\mathbf{p} \in D$:

- If $\mathbf{p}$ is inside any zero area $H_j \in \mathcal{H}$, then $VIF_{s,D}(\mathbf{p}) = \mathbf{0}$.
- Otherwise, $VIF_{s,D}(\mathbf{p}) = \mathbf{u}(\mathbf{p}, \mathcal{C})$ (see Equation (2)).

For an *orientation IF*, recall that zero areas are treated as holes in the domain. In other words, the domain D is equal to $D^b - \bigcup_{i=0}^{h-1} H_i$, i.e. the set of points that is not covered by any zero area. For any point $\mathbf{p}$ in the remaining domain D, the final orientation function normalizes the interpolated vector from Equation (2) to unit length:

$$OIF_{s,D}(\mathbf{p}) = \frac{\mathbf{u}(\mathbf{p}, \mathcal{C})}{\|\mathbf{u}(\mathbf{p}, \mathcal{C})\|}.$$

The IF editor finally converts a drawing to a grid by computing $IF_{s,D}(\mathbf{p}_i)$ for a set of regularly sampled grid points $\mathbf{p}_i$. The resulting grid of vectors can be used in the crowd simulation.

5.3. Sketching parametric IFs

Recall from Section 4.4 that a *parametric IF* is an IF that depends on additional scalar parameters. To draw a parametric IF based on *keyframes*, the user can simply draw separate IFs and specify the corresponding parameter values. To draw a parametric IF based on an *object relation*, the user can draw a line-segment connection (a *link*) between the two relevant objects. As mentioned in Section 4.4, the IF editor currently only supports a link between the source s and another object o, and this link implies an angle-based relation between s and o. We leave other types of links for future work.

6. Implementation details

We have implemented the IF editor and an IF-enriched crowd simulation in platform-independent C++. To convert a drawing to an IF, guide curves are sampled at curve-length intervals of 0.1 meters.

6.1. Crowd simulation framework and settings

We have extended UMANS, an existing real-time agent-based crowd simulation framework [vTGG*20], to support interaction fields. The simulation represents each IF by a grid. We compute an IF vector using bilinear interpolation between the nearest grid cells. For parametric IFs based on keyframes, recall from Section 5.3 that any interpolated IFs are not explicitly computed. However, we sometimes *visualize* an interpolated IF for the sake of illustration.

In line with other research, our simulations use Euler integration and a fixed frame length $\Delta t = 0.1$ s. Each agent has a disk radius of 0.3 m, unit mass, a preferred speed of 1.3 m/s, and a maximum speed of 1.8 m/s. For contact forces in case of collisions, we use the model by Helbing et al. [HFV00] with coefficients $K_{\mathrm{ag}} = \frac{5000}{80}$ for agent forces and $K_{\mathrm{obs}} = \frac{2500}{80}$ for obstacle forces. These values are commonly used in literature when the agents have unit mass.

Next to these overall simulation settings, each agent A_i will use one of the following *behavior profiles*:

- *IFs-Only*: A_i uses the IF velocity $\mathbf{v}_i^{\mathrm{IF}}$ directly as the preferred velocity $\mathbf{v}_i^{\mathrm{pref}}$ and as the new velocity $\mathbf{v}_i^{\mathrm{new}}$. There is no additional goal reaching or collision avoidance.
- *IFs+GoalReaching*: A_i computes $\mathbf{v}_i^{\mathrm{pref}}$ as the average of $\mathbf{v}_i^{\mathrm{IF}}$ and a velocity that sends A_i to a pre-defined *goal* at the preferred speed. There is no collision avoidance, so $\mathbf{v}_i^{\mathrm{new}} := \mathbf{v}_i^{\mathrm{pref}}$.
- *IFs+RVO*: A_i computes $\mathbf{v}_i^{\mathrm{pref}}$ using IFs. It then computes $\mathbf{v}_i^{\mathrm{new}}$ using the RVO algorithm for collision avoidance [vdBLM08], using the default settings suggested by its authors. Overall, RVO looks for a velocity close to $\mathbf{v}_i^{\mathrm{pref}}$ that has a low collision risk.
- *UserControl*: A_i receives $\mathbf{v}_i^{\mathrm{pref}}$ and $\mathbf{v}_i^{\mathrm{new}}$ directly from a user (e.g. via keyboard or controller input). The agent still receives contact forces in case of a collision. In our figures and videos, user-controlled agents will always be visualized in red.

Of course, and most importantly, each scenario will use its own specific *interaction fields* to model specific types of behavior, and different agents can emit and receive different IFs.

6.2. Coupling with character animation

To visualize our results using animated 3D characters for our supplementary video, we have connected the crowd simulation to the Unity game engine. Synchronizing a 2D simulation (of 10 FPS) with an animated 3D scene (of a higher framerate) is not a trivial task. There are at least two options to choose between:

Simulation priority: Let the 3D characters move exactly to the positions produced by the crowd simulation, and use interpolation to fill in the additional animation frames. For body animation, apply a suitable motion clip to each character, accepting possible artifacts such as footsliding.

Animation priority: Use the *output* of the simulation as *input* for an animation system that chooses an appropriate motion clip per character. The chosen animation determines where a character actually moves, and this overrides the simulation results.

The first option is often used in crowd simulation papers, whenever a perfect correspondence to the simulation is more important than animation accuracy. The second option is popular for e.g. controllable characters in games, where the animation should be smooth and natural. It can also help filter out motion for which no animation clips exist, such as fast backward motion or sudden rotations.

For crowd simulations with IFs, we see use cases for both options. In our supplementary video, we consistently use the second option, based on a Unity plugin for *motion matching* [Ani20].

7. Demonstration of results

This section shows the capabilities of interaction fields in a number of example scenarios. Our main purpose is to demonstrate specific features of IFs (such as the use of parameters), and to show that these can easily be combined into more complex scenarios.

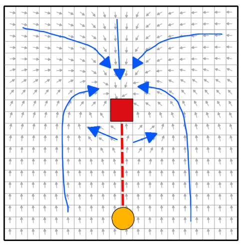
For each scenario, we will show the input IFs created in our editor, as well as screenshots of the resulting simulation. All simulation screenshots include a grid with cells of 1×1 m, to illustrate the scale of the environment. For visualization purposes, we also show several IFs mapped onto the environment. Recall from Section 4.2 that the simulation itself does not need to compute any mapped IFs.

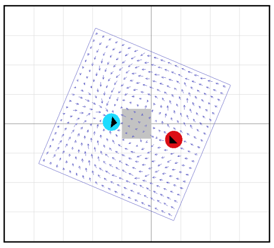
We also invite the reader to watch the supplementary video of this paper, which shows several results in motion, including the IF design process and combinations with 3D character animation.

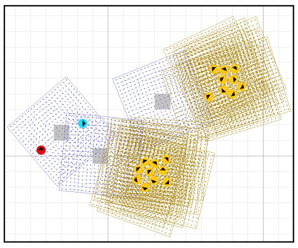
7.1. Scenario 1: Hide and seek

Our first scenario uses an angle-dependent parametric velocity IF to let an agent hide behind an object. This IF, shown in Figure 7(a), was drawn using 7 guide curves and a rotation link. In the simplest version of the scenario, one obstacle O emits this IF, with a user-controlled agent A_0 as the linked object. An agent A_1 with the *IFs-Only* profile responds to the IF. As the user moves A_0 around, A_1 automatically hides behind O depending on where A_0 is located. Figure 7(b) shows a screenshot of the simulation.

In the extended scenario shown in Figure 7(c), we have added several obstacles and agents (with the *IFs+RVO* profile) that all emit the same IF. Consequently, the agent A_1 hides behind whichever object is nearby, treating obstacles and agents in the same way. The extra agents do not respond to any IFs, but they use collision avoidance to make way for the user if necessary. Figure 7(d) and the supplementary video visualize the scenario in 3D.


(a) Velocity IF (5×5 **m)**


(b) Simulation (simple)


(c) Simulation (extended)

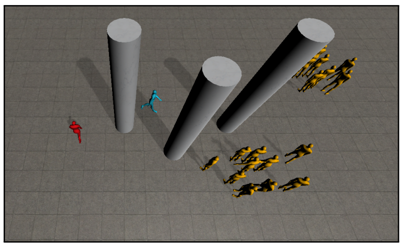
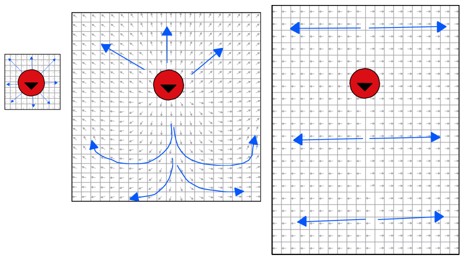
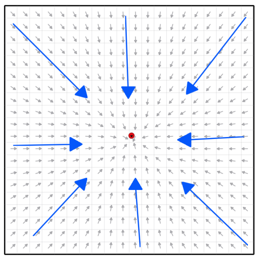
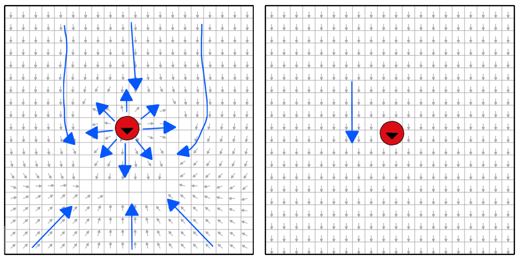
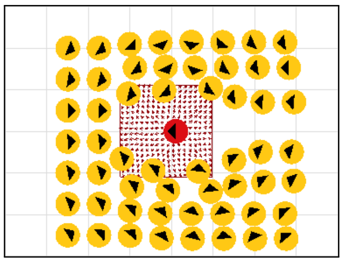

(d) 3D visualization

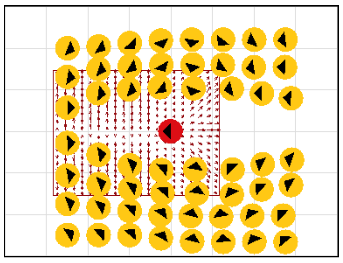
Figure 7: *Results for the* Hide and Seek *scenario (Section 7.1). (a) A velocity IF with a rotation link (red dashed segment) between the source (red) and a second object (orange). Guide curves are shown in blue. (b) A simulation where the blue agent uses this IF to hide from the user-controlled red agent. (c) A simulation where the blue agent can hide behind all obstacles and orange agents, each emitting the same IF. (d) A 3D impression with the two main agents on the left.*


(a) Velocity IF perceived by the crowd, for $v = 0$ **m/s** (1×1 **m),** 1 **m/s** (3×3 **m), and** 1.8 **m/s** (3×4 **m)**


(b) Orientation IF perceived by the crowd (20×20 **m)**


(c) Velocity IF perceived by the bodyguards, for $v = 0$ **m/s and** $v = 1$ **m/s** (5×5 **m)**


(d) Simulation (VIP speed: 0.6 **m/s)**


(e) Simulation (VIP speed: 1.8 **m/s)**

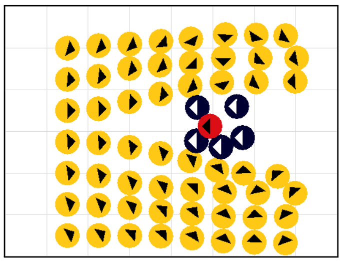
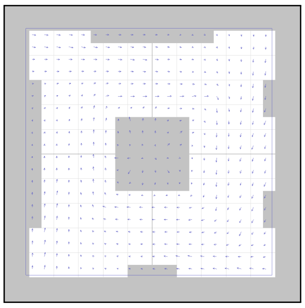
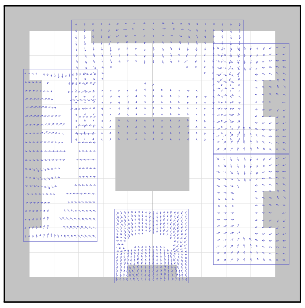

(f) Simulation with bodyguards

Figure 8: *Results for the* VIP in a Crowd *scenario (Section 7.2). (a) Keyframes of the velocity IF used by the crowd. (b) The orientation IF used by the crowd. (c) Keyframes of the velocity IF used by the bodyguards. (d–e) Simulation examples with different speeds for the VIP (in red). The interpolated IF is shown as well. (f) Simulation example with bodyguards (in dark blue). Here, all IFs are omitted for clarity.*


(a) Velocity IF (corridor)


(b) Velocity IFs (paintings)

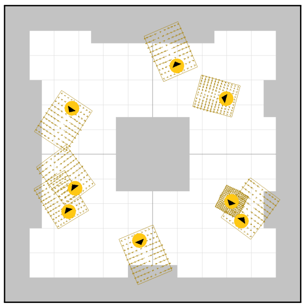

(c) Simulation

Figure 9: *Results for the* Museum *scenario (Section 7.3). (a) One of the velocity IF for walking around the central pillar. (b) The velocity IFs for all five paintings. (c) Screenshot of the simulation, also showing the parametric IFs around standing and moving agents.*

7.2. Scenario 2: VIP in a crowd

Next, we show an example where a crowd makes room for a user-controlled ‘VIP’ agent. To model this, we make the VIP agent emit two IFs: a parametric velocity IF that depends on the source speed (Figure 8(a)) and an orientation IF that makes agents look at the source (Figure 8(b)). For the velocity IF, the domain grows and the pushing effect becomes stronger as the speed increases.

The simulation features a small crowd of agents with the *IFs+GoalReaching* profile. The goal of each agent is set to its starting position, so that the agents move back to their old position after the VIP has passed. Figures 8(d) and 8(e) show how the crowd responds differently depending on the speed of the VIP agent.

Finally, we extend the scenario to include five ‘bodyguard’ agents with the *IFs-Only* profile. We make the VIP agent emit another velocity IF to which only the bodyguards respond. This IF (shown in Figure 8(c)) is parametric again: it lets the bodyguards align with the VIP when it is moving, and (re-)group around the VIP when it stands still. The latter keyframe IF uses zero areas to let the bodyguards stop in a circle around the VIP. Furthermore, the bodyguards themselves also emit the same pushing IF as the VIP. Figure 8(f) shows an example of the simulation with bodyguards.

7.3. Scenario 3: Museum

Our final example is a museum scenario where 8 *IFs+RVO* agents move through a corridor and look at paintings. The central pillar emits two velocity IFs for walking around it in a clockwise or counterclockwise way; each agent uses one of these two IFs. Figure 9(a) shows the clockwise IF. Next, each painting emits a velocity IF with a zero area that lets agents stand still at a certain distance from that painting; these IFs are shown in Figure 9(b). Each painting also emits an orientation IF that lets agents face the painting; we have omitted these IFs from our figures for clarity reasons.

Also, each agent A_i emits a parametric velocity IF that prevents others from entering A_i’s line of sight when it is standing still. This way, others will avoid A_i politely when it is looking at a painting. Figure 9(c) shows a screenshot of the simulation and the agents’ IFs. Again, more results can be found in our supplementary video.

To let agents switch between walking around and studying a painting, we have added the ability to (de)activate IFs using timers. Whenever an agent enters the domain of a painting velocity IF for the first time, the agent will ignore the corridor IF for a number of seconds. When this timer has passed, the agent ignores the painting IF and uses the corridor IF again, so it continues exploring the museum. However, the *orientation* IFs stay active all the time, so that agents always face paintings that are in range. The timer system is not part of the IF technique itself, and it required some additional modelling/programming effort specifically for this scenario. Note that this is the only example with such an extra system.

8. User study

To evaluate the efficacy of IFs and the IF editor for non-expert users, we conducted a user study with 22 users who were familiar with computer animation but not with IFs. Our goal was to evaluate how easily they could learn to independently sketch IFs to design specific agent interactions. Please note that the scenarios in this study are different from those in Section 7: thus, the user study shows even more examples of IF use cases. We will describe the study only briefly in this section. For in-depth experimental details and results, we refer the reader to the supplementary material.

All participants completed the study at the institution with the experimenter present, using two 24-inch screens with 1 GUI window to draw the fields and 1 simulation window to see the resulting behavior. They could always update their IF sketch interactively until they were satisfied with the simulation results. All participants started with a short video-guided training session, where they could freely explore our IF tool and interact with the experimenter.

After the introduction phase, participants were asked to sketch IFs for scenarios of increasing complexity.

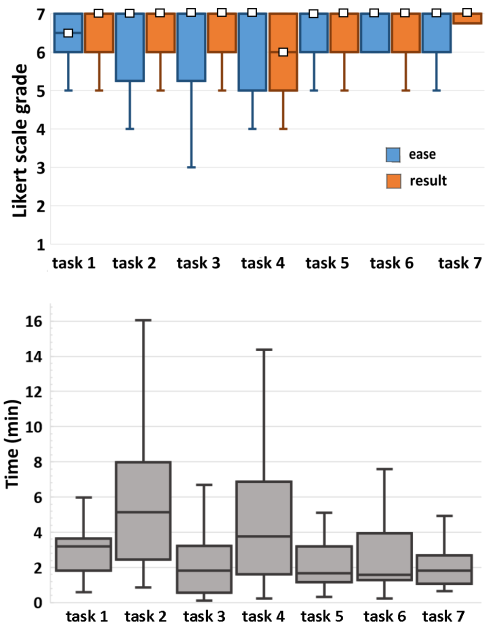


Figure 10: *Results of the user study, per task. Box plots show medians, interquartile ranges and maximum/minimum values (excluding outliers). Top: user ratings for the ease of design (in blue) and satisfaction with the result (in orange). Bottom: completion time for each task (in minutes).*

Each scenario started with a video example and training tasks covering a specific concept, such as controlling the velocity or creating parametric IFs. Each training was followed by one or more evaluation tasks, where participants were asked to create a given scenario based on a number of high-level instructions. There were seven of these evaluation tasks in total. The tasks were designed to require a small number of IFs each (e.g. one orientation IF and one velocity IF), and ordered so as to gradually introduce the users to all IF features (e.g. by saving parametric IFs for last). After each evaluation task, participants reported their satisfaction with their result on a 7-point Likert scale using an online form. They also filled out a usability questionnaire based on SUS [Bro96] at the end. The time to complete the study varied between participants but never exceeded 2 hours.

Figure 10 shows that the participants found the tool easy to use and were very satisfied with the behaviors they designed. The average completion time per task was between 2 minutes 24 (for the fastest task) and 5 minutes 43 (for the slowest task). The final usability questionnaire showed a high average score of 80.6 percentile, which gives our IF editor a A- rating on the Sauro-Lewis Grading scale [LS18]. Knowing that the IF editor is a simple interface not yet designed for commercial use, this grade shows a very high usability performance. Overall, our study suggests that novice users can easily use the IF editor to sketch agent interactions.

9. Discussion

Advantages and relation to other methods Interaction fields allow users to intuitively sketch local agent interactions for real-time crowd simulations. To our knowledge, this is the first method with this specific focus. Our scenarios show that interesting simulations can be obtained based on a few simple IF sketches. Thus, sketching IFs is an effective way to create particular behaviors in a crowd.

Compared to traditional models where the behavior of agents can only be influenced via parameters, IFs can be drawn directly and have a more intuitive visual effect on agent behavior. This makes IFs ideal for scenarios where a designer has in mind how agents should move, without knowing how to define this motion mathematically. We acknowledge that there are also simulation tasks (such as multi-agent collision avoidance) for which highly successful traditional algorithms already exist. This is why we have shown that IFs can easily be combined with such algorithms; such a combination is more sensible than attempting to use IFs for everything.

The concept of letting agents move according to vector fields is not revolutionary. As mentioned before, our method is inspired by the navigation fields (NFs) of Patil et al. [PvdBC*11]. NFs and IFs can both control the *global paths* of agents: an IF with the entire environment as its source is essentially an NF. However, IFs can also model the *local interactions between* agents because of two distinctive properties: they are defined *relative to sources* that can move during the simulation (such as other agents), and they can change according to *parameters* (such as the source's speed or a relation to another object). Our scenarios focused on these distinctive aspects, as well as on the ease of designing IFs visually.

Applicability It is important to understand which types of interactions an IF can(not) encode. Overall, a non-parametric IF describes an agent's velocity or orientation at different positions relative to another object. A parametric IF can change this behavior according to scalar parameters or to a relation between two objects. In theory, IFs support any behavior that can be described in such terms. However, behavior depending on *more than two objects* is difficult to encode in an IF. Examples include moving towards the center of a group of agents, or avoiding multiple agents at the same time without considering them individually. Other techniques are more suitable for this. As mentioned, IFs are not meant to fully replace traditional algorithms, and techniques can easily be combined.

When combining many IFs for different purposes, it is possible that the results average out and agents become indecisive. However, this is also the case for traditional methods, and modeling many behaviors simultaneously is a difficult problem in general.

Usability and authoring capabilities Our user study indicates that the IF editor is a powerful tool for non-expert users to design agent interactions. Recall that the design process itself is interactive as well: in our study, users could edit their IFs on the fly and immediately see the effect in the simulation. Overall, the IF editor allows users to design new types of behavior that typically take much more time and effort to model using traditional techniques.

Setting up a complete scenario still required some additional manual work, e.g. for assigning IFs to sources and receivers, and for specifying the behavior profile of each agent. Likewise, there is not yet a reusable system for (de)activating IFs over time; we have scripted this manually for the Museum scenario in Section 7.3. Of course, our software can be improved to a fully integrated design pipeline with such additional features, but this is engineering labor that does not yield new research insights.

Parametric fields The concept of parametric IFs can still be explored further. As mentioned, other examples of scalar parameters could be the local crowd density (for creating density-dependent behavior) and the elapsed simulation time (for creating behavior that evolves temporally). We also have not yet considered IFs with *multiple* parameters, mostly because drawing keyframes in a higher-dimensional space is less intuitive. Also, we are interested in implementing other types of *object relations*, e.g. to create an IF that depends on the distance between two objects.

Computational performance In terms of performance, our scenarios are too small for meaningful time measurements. However, the software that we use as a basis can simulate tens of thousands of agents in real-time. It is well-known in our community that *nearest-neighbor queries* between agents is the least scalable simulation task, which will dominate the overall running time when the crowd is large. *Collision avoidance* can also be an expensive task, depending on the algorithms used. Relatively, IFs have very little impact on the simulation complexity. In a simulation frame, each agent A_i performs simple arithmetic operations for each perceived IF. This is similar in complexity to e.g. force-based collision avoidance. Thus, nothing prevents IFs from being usable for large crowds.

Animation and motion quality To give users freedom in designing agent behavior, our technique deliberately controls velocities and orientations separately, and it allows these vectors to change quickly during the simulation. Also, users are free to draw velocity IFs with 'local minima' where agents end up standing still (e.g. using zero areas or by drawing curves that cancel each other out). This freedom of design may cause agent behavior that is not realistic or 'human-like', which may be seen as a disadvantage for some applications, especially when animated 3D characters are involved.

We have chosen not to place any 'realism'-related constraints inside the IF method itself, to keep the method generic and suitable for many applications. Instead, we believe that the best place for filtering out unrealistic motion (if desired) is the *animation* loop. As described in Section 6.2, we have created an implementation where the 3D character animation system has the final say over a 2D agent's trajectory, thus overriding the simulation result if necessary. This way, the (transitions between) motions of an agent are limited to what the provided animation clips support. Note that this also allows for the personalization of behavior by providing different animations per character.

Current limitations The IF editor converts sketches to interaction fields using inverse-distance weighting of guide curves. While this type of interpolation has several advantages (as explained in Section 5.2), there may be situations where the result is not yet ideal. For example, a point in the IF is currently always influenced by *all* guide curves to some extent, even by (parts of) curves that are far away. To prevent this, we could let users control the influence distance of a guide curve, so that far-away points are ignored.

Our simulation currently approximates IFs by grids. This has limitations in terms of precision (e.g. for describing behavior around detailed obstacles) and scalability to large environments. To alleviate this, the simulation could also directly use the guide curves and zero areas from the IF editor. This would come at a computational cost, though, as computing a velocity or orientation from one IF would no longer take constant time.

There are a few other limitations to how we currently define and use IFs. First, we do not enforce any *smoothness*, and trajectories may be non-smooth especially when an agent enters or exits the domain of an IF. As mentioned, a coupling with character animation improves this greatly. On a simulation level, we could also consider letting the influence of an IF decay smoothly around the border of its domain. Second, IFs cannot yet specify *variability*: the same situation will always result in the same agent trajectories. This makes it difficult to, for example, design how agents can move around an obstacle in different ways. Such variability could be added by letting the simulation choose (or interpolate) between multiple IFs. Third, we currently combine the results of multiple IFs as a weighted average. We can imagine cases where more advanced combination mechanisms are useful, e.g. to prioritize certain IFs only if certain conditions are satisfied. Such additional rules could be part of an advanced authoring tool. In short, none of these issues are fundamental, and they can be resolved in future work.

Finally, we have deliberately decided that velocity IFs prescribe absolute velocities, and not relative velocities (or even acceleration vectors). Such alternative representations could make certain scenarios easier to model, but they would strongly reduce the method's intuitiveness and user-friendliness. Similarly, applying concepts such as incompressibility may lead to smoother velocity fields, but not necessarily to a more intuitive user experience. In future work, we plan to explore if/how such extensions can be integrated into a user-friendly tool.

10. Conclusions and future work

In agent-based crowd simulations, the behavior of each agent is usually described via rules and equations. In this paper, we have presented *interaction fields (IF)* as an alternative way to model agent behavior in crowds. An IF specifies the velocities or orientations that agents should use around a given object, possibly depending on parameters such as that object's speed. Combined with an editor that computes IFs based on user sketches, we obtain a system for efficiently and intuitively sketching new agent behaviors. Our example scenarios show that complex simulation results can be obtained using a few simple sketches. IFs can easily be combined with other simulation components (such as collision avoidance) without affecting real-time performance. Our user study indicates that non-expert users can easily use the IF editor to draw agent behaviors that match overall instructions.

Next to the discussion points mentioned in Section 9, there are several other directions for future work. First, all IFs in this paper have been drawn by hand in the IF editor. However, for some types of behaviors with a clear geometric meaning (e.g. moving to a given point or following an agent), it should be possible to generate an IF *automatically* without guide curves. An IF could also be generated based on recordings of a real crowd, by using the recorded trajectories as guide curves. By leveraging machine learning, it may even be possible to detect patterns in such data and create a separate IF for each 'sub-behavior'. This would yield a 'palette' of IFs from which many different types of crowds can be composed. Another interesting direction for future work would be to enable IF authoring based on text input, e.g., associating words with IFs and building a 'grammar' to describe agent interactions.

Second, while is common to visualize a crowd simulation from a 2D top view, IFs are also useful for modelling subtle personal interactions that are better judged from an agent's perspective. We therefore intend to integrate our simulation into a *virtual-reality* environment where the user itself is embedded as an agent. This is an important step towards evaluating the realism of our results.

Overall, we are confident that interaction fields give an unprecedented level of creative control over the steering behaviors in crowds. This is an important step towards fully immersive crowd simulations where all agents display human-like behavior, and where the crowd responds realistically to the user's actions.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 856879 PRESENT.

References

[ACC14] Allain, Pierre, Courty, Nicolas, and Corpetti, Thomas. "Optimal crowd editing". *Graphical Models* 76.1 (2014), 1–16 3.

[Ani20] Animation Uprising. *Motion Matching for Unity*. 2020. url: https://assetstore.unity.com/packages/tools/animation/motion-matching-for-unity-145624 8.

[ANMS13] Ali, S., Nishino, K., Manocha, Dinesh, and Shah, M. *Modeling, Simulation and Visual Analysis of Crowds: A Multidisciplinary Perspective*. Springer, 2013 2.

[BBEK20] Bönsch, Andrea, Barton, Sebastian J, Ehret, Jonathan, and Kuhlen, Torsten W. "Immersive sketching to author crowd movements in real-time". *Proc. ACM International Conference on Intelligent Virtual Agents*. 2020, 1–3 3.

[BC20] Bhattacharjee, Sukanya and Chaudhuri, Parag. "A survey on sketch based content creation: from the desktop to virtual and augmented reality". *Computer Graphics Forum* 39.2 (2020), 757–780 3.

[Bro96] Brooke, John. "SUS: a quick and dirty usability scale". *Usability Evaluation in Industry* 189 (1996) 10.

[CBL*16] CHOI, BYUNGKUK, BLANCO I RIBERA, ROGER, LEWIS, J. P., et al. "SketchiMo: Sketch-based motion editing for articulated characters". *ACM Transactions on Graphics* 35.4 (2016) 3.

[CBM16] CURTIS, SEAN, BEST, ANDREW, and MANOCHA, DINESH. "Menge: A modular framework for simulating crowd movement". *Collective Dynamics* 1.A1 (2016), 1–40 2.

[CvTH*20] COLAS, AD'ELE, van TOLL, WOUTER, HOYET, LUDOVIC, et al. "Interaction Fields: Sketching Collective Behaviours". *Proc. ACM SIGGRAPH Conference on Motion in Games (Poster)*. 2020 2.

[CWL20] CHEN, CHIEN-YUAN, WONG, SAI-KEUNG, and LIU, WEN-YUN. "Generation of small groups with rich behaviors from natural language interface". *Computer Animation and Virtual Worlds* 31.4-5 (2020), e1960 3.

[CZWL20] CANNAVÒ, ALBERTO, ZHANG, CONGYI, WANG, WENPING, and LAMBERTI, FABRIZIO. "Posing 3D characters in virtual reality through in-the-air sketches". *Proc. International Conference on Computer Animation and Social Agents*. 2020, 51–61 3.

[DMC*17] DUTRA, TEOFILO B., MARQUES, RICARDO, CAVALCANTE-NETO, JOAQUIM B., et al. "Gradient-based steering for vision-based crowd simulation algorithms". *Computer Graphics Forum* 36.2 (2017), 337–348 2.

[GCC*10] GUY, STEPHEN J., CHHUGANI, JATIN, CURTIS, SEAN, et al. "PLEdestrians: A least-effort approach to crowd simulation". *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association. 2010, 119–128 2.

[GCR13] GUAY, MARTIN, CANI, MARIE-PAULE, and RONFARD, RÉMI. "The line of action: An intuitive interface for expressive character posing". *ACM Transactions on Graphics* 32.6 (2013) 3.

[GD13] GU and DENG. "Generating freestyle group formations in agent-based crowd simulations". *IEEE Computer Graphics and Applications* 33.1 (2013), 20–31 3.

[GRGC15] GUAY, MARTIN, RONFARD, RÉMI, GLEICHER, MICHAEL, and CANI, MARIE-PAULE. "Space-time sketching of character animation". *ACM Transactions on Graphics* 34.4 (2015) 3.

[HFV00] HELBING, DIRK, FARKAS, ILLÉS, and VICSEK, TAMÁS. "Simulating dynamical features of escape panic". *Nature* 407 (2000), 487–490 8.

[HM95] HELBING, DIRK and MOLNÁR, PÉTER. "Social force model for pedestrian dynamics". *Physical Review E* 51.5 (1995), 4282–4286 2.

[HOD15] HUGHES, ROWAN, ONDŘEJ, JAN, and DINGLIANA, JOHN. "Holonomic Collision Avoidance for Virtual Crowds". *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2015, 103–111 2.

[HSK12] HENRY, JOSEPH, SHUM, HUBERT, and KOMURA, TAKU. "Environment-Aware Real-Time Crowd Control". *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2012, 193–200 3.

[HSK14] HENRY, JOSEPH, SHUM, HUBERT, and KOMURA, TAKU. "Interactive formation control in complex environments". *IEEE Transactions on Visualization and Computer Graphics* 20 (2014), 211–222 3.

[JCC*15] JORDAO, KEVIN, CHARALAMBOUS, PANAYIOTIS, CHRISTIE, MARC, et al. "Crowd art: Density and flow based crowd motion design". *Proc. ACM SIGGRAPH Conference on Motion in Games*. 2015, 167–176 3.

[JCP*10] JU, EUNJUNG, CHOI, MYUNG GEOL, PARK, MINJI, et al. "Morphable crowds". *ACM Transactions on Graphics* 29.6 (2010), 1–10 3.

[JPCC14] JORDAO, KEVIN, PETTRÉ, JULIEN, CHRISTIE, MARC, and CANI, M-P. "Crowd sculpting: A space-time sculpting method for populating virtual environments". *Computer Graphics Forum* 33.2 (2014), 351–360 3.

[KBB16] KIELAR, PETER M., BIEDERMANN, DANIEL H., and BORRMANN, ANDRÉ. *MomenTUMv2: A modular, extensible, and generic agent-based pedestrian behavior simulation framework*. Tech. rep. TUM-I1643. Technische Universität München, Institut für Informatik, 2016 2.

[KBK16] KRONTIRIS, ATHANASIOS, BEKRIS, KOSTAS E, and KAPADIA, MUBBASIR. "Acumen: Activity-centric crowd authoring using influence maps". *Proc. International Conference on Computer Animation and Social Agents*. 2016, 61–69 3.

[KHVO09] KARAMOUZAS, IOANNIS, HEIL, PETER, VAN BEEK, PASCAL, and OVERMARS, MARK H. "A predictive collision avoidance model for pedestrian simulation". *Proc. International Workshop on Motion in Games*. 2009, 41–52 2.

[KK*14] KANG, SHIN JIN, KIM, SOO-KYUN, et al. "Crowd control with vector painting". *Journal of Research and Practice in Information Technology* 46.2-3 (2014), 119 3.

[KL16] KIM, JONGMIN and LEE, JEHEE. "Interactive editing of crowd animation". *Simulating Heterogeneous Crowds with Interactive Behaviors*. 2016, 115–130 3.

[KLLT08] KWON, TAESOO, LEE, KANG HOON, LEE, JEHEE, and TAKAHASHI, SHIGEO. "Group motion editing". *ACM Transactions on Graphics* 27.3 (2008), 1–8 3.

[KO10] KARAMOUZAS, IOANNIS and OVERMARS, MARK H. "A velocity-based approach for simulating human collision avoidance". *Proc. International Conference on Intelligent Virtual Agents*. 2010, 180–186 2.

[KSA*09] KAPADIA, MUBBASIR, SINGH, SHAWN, ALLEN, BRIAN, et al. "SteerBug: an interactive framework for specifying and detecting steering behaviors". *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2009, 209–216 3.

[KSG14] KARAMOUZAS, IOANNIS, SKINNER, BRIAN, and GUY, STEPHEN J. "Universal power law governing pedestrian interactions". *Physical Review Letters* 113 (23 2014), 238701:1–5 2.

[KSKL14] KIM, JONGMIN, SEOL, YEONGHO, KWON, TAESOO, and LEE, JEHEE. "Interactive manipulation of large-scale crowd animation". *ACM Transactions on Graphics* 33.4 (2014), 1–10 3.

[KSRF11] KAPADIA, MUBBASIR, SINGH, SHAWN, REINMAN, GLENN, and FALOUTSOS, PETROS. "A behavior authoring framework for multi-actor simulations". *IEEE Computer Graphics and Applications* 31 (2011) 3.

[LCMP19] LÓPEZ, AXEL, CHAUMETTE, FRANÇOIS, MARCHAND, ERIC, and PETTRÉ, JULIEN. "Character navigation in dynamic environments based on optical flow". *Computer Graphics Forum* 38.2 (2019), 181–192 2.

[LS18] LEWIS, JAMES R and SAURO, JEFF. "Item benchmarks for the system usability scale". *Journal of Usability Studies* 13.3 (2018) 11.

[LWC20] LIU, WEN-YUN, WONG, SAI-KEUNG, and CHEN, CHIEN-YUAN. "A natural language interface with casual users for crowd animation". *Computer Animation and Virtual Worlds* 31.4-5 (2020), e1965 3.

[MBA20] MATHEW, THARINDU, BENES, BEDRICH, and ALIAGA, DANIEL. "Interactive inverse spatio-temporal crowd motion design". *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 2020, 1–9 3.

[MHT11] MOUSSAÏD, MEHDI, HELBING, DIRK, and THERAULAZ, GUY. "How simple rules determine pedestrian behavior and crowd disasters". *Proc. National Academy of Sciences* 108 (17 2011), 6884–6888 2.

[MM17] MONTANA, LUIS RENE and MADDOCK, STEVE. "Sketching for real-time control of crowd simulations". *Proc. Conference on Computer Graphics & Visual Computing*. 2017, 81–88 3.

[MM19] MONTANA GONZALEZ, LR and MADDOCK, SC. "A sketch-based interface for real-time control of crowd simulations that use navigation meshes". *Proc. International Conference on Computer Graphics Theory and Applications*. Vol. 1. 2019, 41–52 3.

[MMHR16] McIlveen, James, Maddock, Steve C, Heywood, Peter, and Richmond, Paul. "PED: Pedestrian Environment Designer". *Proc. Conference on Computer Graphics & Visual Computing*. 2016, 105–112 3.

[MR05] Millán, Erik and Rudomin, Isaac. "Agent paint: Intuitive specification and control of multiagent animations". *Proc. International Conference on Computer Animation and Social Agents*. Vol. 2. 3. 2005 3.

[NLS14] Normoyle, Aline, Likhachev, Maxim, and Safonova, Alla. "Stochastic activity authoring with direct user control". 2014, 31–38 3.

[OO09] Oshita, Masaki and Ogiwara, Yusuke. "Sketch-based interface for crowd animation". *International Symposium on Smart Graphics*. 2009, 253–262 3.

[OPOD10] Ondřej, Jan, Pettré, Julien, Olivier, Anne-Hélène, and Donikian, Stéphane. "A synthetic-vision based steering approach for crowd simulation". *ACM Transactions on Graphics* 29.4 (2010), 123 2.

[PAB07] Pelechano, Nuria, Allbeck, Jan M., and Badler, Norman I. "Controlling individual agents in high-density crowd simulation". *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2007, 99–108 2.

[PAKB16] Pelechano, Nuria, Allbeck, Jan M., Kapadia, Mubbasir, and Badler, Norman I. *Simulating Heterogeneous Crowds with Interactive Behaviors*. CRC Press, 2016 2.

[PPD07] Paris, Sébastien, Pettré, Julien, and Donikian, Stéphane. "Pedestrian reactive navigation for crowd simulation: A predictive approach". *Computer Graphics Forum* 26.3 (2007), 665–674 2.

[PvdBC*11] Patil, Sachin, van den Berg, Jur, Curtis, Sean, et al. "Directing crowd simulations using navigation fields". 17.2 (2011), 244–254 3–5, 11.

[PWJ*08] Paravisi, M., Werhli, A., Junior, J. J., et al. "Continuum crowds with local control". *Proc. Computer Graphics International*. 2008, 108–115 3.

[Rey99] Reynolds, Craig W. "Steering behaviors for autonomous characters". *Game Developers Conference*. Vol. 1999. Citeseer. 1999, 763–782 2, 3.

[SGH20] Savenije, Noud, Geraerts, Roland, and Hürst, Wolfgang. "CrowdAR table : An AR system for real-time interactive crowd simulation". *Proc. IEEE International Conference on Artificial Intelligence and Virtual Reality*. 2020, 57–59 3.

[She68] Shepard, Donald. "A two-dimensional interpolation function for irregularly-spaced data". *Proc. ACM National Conference*. 1968, 517–524 7.

[SHW*18] Shen, Yijun, Henry, Joseph, Wang, He, et al. "Data-driven crowd motion control with multi-touch gestures". *Computer Graphics Forum* 37.6 (2018), 382–394 3.

[SSKF10] Schuerman, Matthew, Singh, Shawn, Kapadia, Mubbasir, and Faloutsos, Petros. "Situation agents: Agent-based externalized steering logic". *Computer Animation and Virtual Worlds* 21.3-4 (2010), 267–276 2, 3.

[TCP06] Treuille, Adrien, Cooper, Seth, and Popović, Zoran. "Continuum crowds". *ACM Transactions on Graphics* 25.3 (2006), 1160–1168 2.

[TM13] Thalmann, Daniel and Musse, Soraia R. *Crowd Simulation*. 2nd ed. Springer, 2013 2.

[UdHT04] Ulicny, Branislav, de Heras Ciechomski, Pablo, and Thalmann, Daniel. "CrowdBrush: Interactive authoring of real-time crowd scenes". *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2004, 243–252 3.

[vdBGLM11] Van den Berg, Jur P., Guy, Stephen J., Lin, Ming C., and Manocha, Dinesh. "Reciprocal n-body collision avoidance". *Proc. International Symposium of Robotics Research*. 2011, 3–19 2.

[vdBLM08] Van den Berg, Jur, Lin, Ming, and Manocha, Dinesh. "Reciprocal velocity obstacles for real-time multi-agent navigation". *Proc. IEEE International Conference on Robotics and Automation*. IEEE. 2008, 1928–1935 2, 8.

[vTGG*20] Van Toll, Wouter, Grzeskowiak, Fabien, Gandía, Axel López, et al. "Generalized microscropic crowd simulation using costs in velocity space". *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. New York, NY, USA: Association for Computing Machinery, 2020. isbn: 9781450375894 8.

[vTJG15] Van Toll, Wouter, Jaklin, Norman, and Geraerts, Roland. "Towards believable crowds: A generic multi-level framework for agent navigation". *ASCI.OPEN*. 2015 2.

[vTP21] Van Toll, Wouter and Pettré, Julien. "Algorithms for microscopic crowd simulation: Advancements in the 2010s". *Computer Graphics Forum* 40.2 (2021) 2.

[WGO*14] Wolinski, David, Guy, Stephen J., Olivier, Anne-Hélène, et al. "Parameter estimation and comparative evaluation of crowd simulations". *Computer Graphics Forum* 33.2 (2014), 303–312 3.

[YCP*08] Yeh, Hengchin, Curtis, Sean, Patil, Sachin, et al. "Composite agents". *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2008 2, 3.

[YMPT09] Yersin, Barbara, Maïm, Jonathan, Pettré, Julien, and Thalmann, Daniel. "Crowd patches: Populating large-scale virtual environments for real-time applications". *Proc. Symposium on Interactive 3D Graphics and Games*. 2009, 207–214 3.

[ZIK11] Zanlungo, Francesco, Ikeda, Tetsushi, and Kanda, Takayuki. "Social force model with explicit collision prediction". *EPL (Europhysics Letters)* 93.6 (2011), 68005 2.

[ZZZY20] Zhang, Yong, Zhang, Xinyu, Zhang, Tao, and Yin, Baocai. "Crowd motion editing based on mesh deformation". *International Journal of Digital Multimedia Broadcasting* 2020 (2020) 3.