Velocity-Based Modeling of Physical Interactions in Multi-Agent Simulations

Sujeong Kim* University of North Carolina at Chapel Hill
Stephen J. Guy† University of Minnesota
Dinesh Manocha‡ University of North Carolina at Chapel Hill
http://gamma.cs.unc.edu/CrowdInteractions/

Abstract

We present an interactive algorithm to model physics-based interactions in multi-agent simulations. Our approach is capable of modeling both physical forces and interactions between agents and obstacles, while allowing the agents to anticipate and avoid collisions for local navigation. We combine velocity-based collision-avoidance algorithms with external physical forces. The overall formulation can approximately simulate various physical effects, including collisions, pushing, deceleration and resistive forces. We have integrated our approach with an open-source physics engine and use the resulting system to model plausible behaviors of and interactions among large numbers of agents in dense environments. Our algorithm can simulate a few thousand agents at interactive rates and can generate many emergent behaviors. The overall approach is useful for interactive applications that require plausible physical behavior, including games and virtual worlds.

CR Categories: I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Multiagent systems

Keywords: Multi-Agent Simulation, Physical interactions

1 Introduction

Multi-agent simulations are frequently used to model a wide variety of physical systems, including human crowds; traffic; groups of birds, bees, fish, ants; and etc. In many of these applications it is important for the agents to interact in a physical manner with each other and the environment. Agents often collide, push, and impart forces on other agents and on the obstacles in the environment, changing their trajectory or behavior. A challenging goal is to model these interactions in large multi-agent systems at interactive rates. Many algorithms based on behavior modeling, social forces, cellular automata, and velocity-based formulation have been proposed for multi-agent simulation. Most of these techniques, however, focus on only the local navigation for each agent, and do not explicitly take into account physical interactions between agents or between agents and obstacles in the environment.

At a high level, there are two different sources of physical forces which may affect an agent's trajectories: interactions with other agents and interactions with objects in the environments. For example, dynamic objects such as falling boxes or moving cars may collide with an agent, pushing it from its path. Likewise, an agent may be pushed by, or bump into, other agents in dense scenarios. This can happen because of agent's intention (e.g. aggressive agent) or because the agent was pushed by an external force.

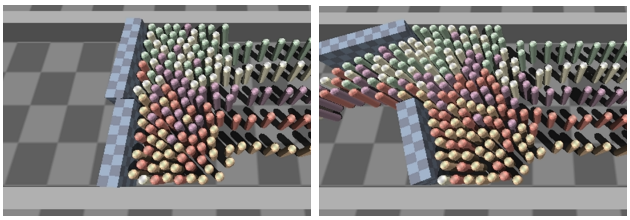


Figure 1: ***Wall Breaking****. We demonstrate the physical forces applied by cylindrical agents to breakable wall obstacles. Our algorithm can model such interactions between the agents and the obstacles in dense scenarios at interactive rates.*

While physical forces impact an agents trajectory, the agents motion will also impart forces upon the objects in his environment. This effect becomes increasingly important when the forces from many individual agents combine to produce a large effect on the environment, such as when dense, aggressive crowds bend fences or break walls. In order to simulate such scenarios, we need to develop appropriate two-way coupling techniques between autonomous agents and their physical environment.

Main Results: In this paper, we present a new method to model physical interactions between agents and objects in an interactive velocity-based multi-agent framework. Our approach incorporates both an agent's ability to anticipate the motion of other agents and avoid collisions using velocity obstacles and respond to physical forces in a single unified framework. We formulate the computation of velocity of each agent for each timestep as a linear programming problem in the velocity space. The linear constraints are computed by approximating the motion induced on an agent through Newtonian dynamics. This allows agents to account for forces from their environment and from other agents and generate a plausible trajectory. The resulting approach is efficient and can be used to simulate dense scenarios with thousands of agents at interactive rates. We have integrated our approach with the Bullet Physics Engine [AMD 2012], and reciprocal velocity obstacles [van den Berg et al. 2011], and demonstrate its performance in many complex scenarios with large number of agents and multiple moving obstacles, In practice, our approach can be used to generate physically plausible behavior for interactive multi-agent simulation.

The rest of the paper is organized as follows. Section 2 gives a brief review of related work. Section 3 gives an overview of our approach, which combines velocity-based multi-agent simulation and rigid body dynamics. We describe our approximate approach to computing velocity constraints using Newtonian dynamics for agent-agent interaction in Section 4 and for agent-obstacle interaction in Section 5. In Section 6, we highlight the performance on different scenarios and compare it with other techniques.

*e-mail:sujeong@cs.unc.edu
†e-mail:sjguy@cs.umn.edu
‡e-mail:dm@cs.unc.edu

2 Related Work

In this section, we give a brief overview of some related work in multi-agent and physically-based character simulation.

2.1 Multi-Agent and Crowd Simulation Models

Many approaches have been proposed to simulate the motion of large number of agents and crowds. Often these models are based on rules, which are used to guide the movement of each agent. An early example of such an approach is the seminal work of Reynolds [1999], which uses simple rules to model flocking behavior.

Force-based methods, such as the social force model [Helbing and Molnár 1995], use various forces to model attraction and repulsion between agents. These forces are not physically based; rather, they provide a mechanism to model the psychological factors that govern how agents approach each other. Helbing et al. [2000] model panic behavior with two additional physical forces (body force and sliding friction) in addition to the social forces. Yu and Johansson [2007] model the turbulence-like motion of a dense crowd by increasing the repulsive force. Other approaches model collision-avoidance behavior with velocity-based techniques [van den Berg et al. 2011; Pettré et al. 2009; Karamouzas and Overmars 2012] or vision-based steering approaches [Ondřej et al. 2010].

Other techniques have been proposed to model complex social interaction. HiDAC [Pelechano et al. 2007] uses various rules and social forces to model interactions between agents and obstacles; collision avoidance and physical interactions between agents and objects are handled using repulsive forces. The composite agent formulation [Yeh et al. 2008] uses geometric proxies to model social priority, authority, guidance, and aggression. Many other multi-agent simulation algorithms use techniques from sociology [Musse and Thalmann 1997], biomechanics [Guy et al. 2012], and psychology [Sakuma et al. 2005; Durupinar et al. 2011; Guy et al. 2011; Kim et al. 2012] to model different aspects of agent behaviors and decision models. These approaches are able to generate realistically heterogeneous behaviors for agents. Our approach to model physical interactions can also be combined with many of these approaches.

Many researchers have proposed cognitive and decision-making models to generate human-like behaviors [Shao and Terzopoulos 2005; Yu and Terzopoulos 2007; Ulicny and Thalmann 2002], or use data-driven approaches to the problem [Lee et al. 2007; Lerner et al. 2009].

Other approaches for modeling crowds are based on continuum or macroscopic models [Hughes 2003; Treuille et al. 2006; Narain et al. 2009]. In particular, Narain et al. [2009] present a hybrid technique using continuum and discrete method for aggregate behaviors in large and dense crowds. They are mainly used to simulate the macroscopic flow and do not model the interaction between the crowd and obstacles. In contrast, our approach simulates agent-agent and agent-obstacles physical interaction.

2.2 Force-Based Techniques for Character Animation

There has been extensive work on using physics-based models to improve character animation. Sok et al. [2010] use a force-based approach to ensure that the resulting motions are physically plausible. Other approaches consider geometric and kinematic constraints [Shum et al. 2012] or use interactive methods for character editing [Kim et al. 2009]. These techniques, which are primarily based on enhancing motion-captured data, can be used to simulate behaviors of and interactions between the characters and obstacles in their environment.

Many hybrid techniques have been proposed that bridge the gap between physics-based simulation of character motion and pre-recorded animation of characters to model responsive behavior of character [Shapiro et al. 2003; Zordan et al. 2005]. Muico et al. [2011] propose a composite method to improve the responsiveness of physically simulated characters to external disturbances by blending or transitioning multiple locomotion skills.

Our approach is quite different from these methods. Unlike character animation techniques that mainly focus on generating the full-body motion of a relatively small number of characters, we focus on generating physically plausible interactions between a large number of agents in dense scenarios.

2.3 Crowd Simulation in Game Engines

Some commercial game engines or middleware products can simulate character motion or crowd behavior. This includes Natural Motion's Euphoria, which simulates realistic character behavior based on biomechanics and physics simulation. There are also commercial AI middlewares for game engines that combine crowd and physics simulation: Kynapse, Havok AI, and Unreal Engine are examples of these. These systems primarily focus on the local and global navigation of each agent using navigation meshes and local rules. Our approach to generating physical interactions can be combined with these systems to improve local interactions between the agents and the obstacles in the scene.

3 Overview

Our framework simulates *agents* and *objects* differently, based on two fundamental assumptions about the nature of their motion. Agents are assumed to be autonomous and self-actuated. In the absence of external forces, we use velocity-based collision avoidance techniques to control the paths of the agents, who avoid collisions using anticipatory techniques. In contrast, objects in the environment move only when physical forces act on them. The positions of objects are updated by solving Newton's equations of motion; contacts are handled with a constraint-based method. This section gives an overview of our proposed approach to simulating agents and objects together in a shared space.

Local navigation and anticipatory collision avoidance of agents can be efficiently modeled using reciprocal velocity obstacles, which imposes linear constraints on an agent's velocity to help it navigate its environment. We extend this framework by representing the effect of physical forces on agents also as linearized velocity constraints. This allows us to use linear programming to compute a new velocity for each agent – one which takes into account both the navigation and force constraints imposed upon that agent.

Agent simulation is typically performed over discrete timesteps. Agents are assumed to have a *preferred velocity*. This is the velocity at which the agent would travel if there were no anticipatory collision-avoidance or physical constraints. This velocity is used to define the cost function for linear programming or constrained optimization. At each timestep, an agent computes a new velocity that satisfies the velocity constraints, then updates its position based on this velocity. A new set of velocity constraints are then computed based on the new positions and velocities.

There are two types of constraints which we impose on an agent's velocity:

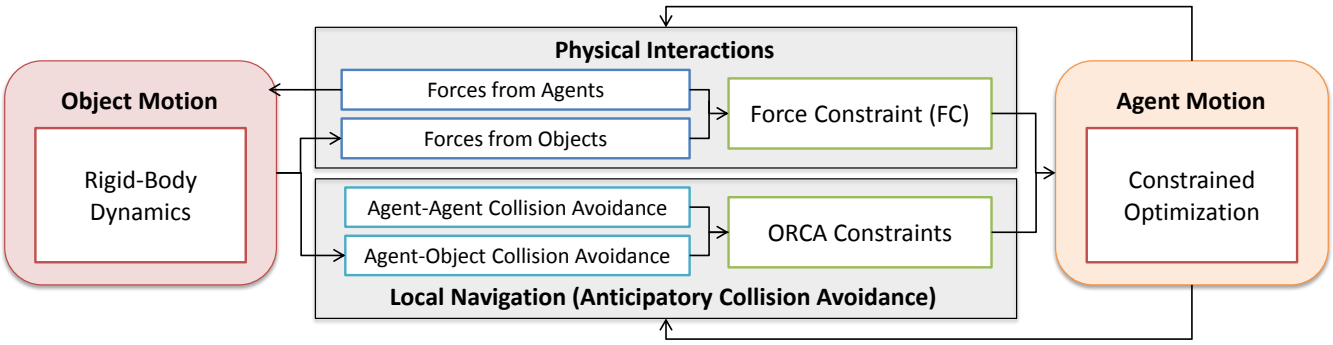


Figure 2: ***System Overview.*** *The motions for objects and agents are computed by a rigid-body dynamics solver and a constrained optimizer, respectively. Physical interactions between agents and obstacles determine forces. For obstacles, the forces serve as inputs to the rigid-body system; for agents, they become force constraints. These force constraints are combined with the original ORCA planning constraints and serve as inputs to optimization algorithm.*

- **ORCA Constraints** define the space of velocities which are expected to remain collision-free for a given period of time. The derivation of agent-agent ORCA constraints is given in Section 3.1, and that of agent-object ORCA constraints in Section 5.1.1.
- **Force Constraints** are constraints which arise out of forces initiated by physical interactions with other agents and objects. The details are given in Sections 3.2, 4, and 5.

Fig. 2 gives an overview of the full simulation system for computing these constraints and for updating an agent's position and velocity.

3.1 Velocity Constraints for Local Navigation

ORCA constraints are defined by a set of velocities that are guaranteed to avoid upcoming collisions with other nearby agents [van den Berg et al. 2011]. The constraints are represented as the boundary of a half plane containing the space of feasible, collision-free velocities. Given two agents, A and B, which we represent as 2D discs, we compute the minimum vector $\mathbf{u}$ of the change in relative velocity needed to avoid collision. ORCA enforces this constraint by requiring each agent to change their current velocity by at least 1/2 $\mathbf{u}$. The ORCA constraint on A's velocity induced by B would be:

$$ORCA_{A|B} = \{\mathbf{v} | (\mathbf{v} - (\mathbf{v}_A + \frac{1}{2}\mathbf{u})) \cdot \hat{\mathbf{u}} \geq 0\}, \qquad (1)$$

where $\mathbf{v}_A$ is A's current velocity and $\hat{\mathbf{u}}$ is the normalized vector $\mathbf{u}$.

If A has multiple neighboring agents, each will impose its own ORCA constraint on A's velocity. Local navigation is computed by finding the new velocity for A ($\mathbf{v}^{new}$) which is closest to its preferred velocity ($\mathbf{v}^{pref}$) while respecting all the ORCA constraints.

3.2 Velocity Constraints from Physical Forces

The set of neighbors involved in physical interactions with an agent include both nearby agents and obstacles. We define a radius and an angle that are then used to define a range of physical interactions for each agent. When an agent is pushed, either by another agent or by an obstacle, the agent experiences an external force. By Newton's second law, the net force acting on an agent implies a net acceleration. Given a known timestep, we can compute the change in velocity exactly. We represent this change in velocity induced by a force as an additional constraint on the agent velocity.

One benefits of applying forces as a form of constraint is the ability of an agent to adapt to the forces. While the constraint guarantees an acceleration at least as large as that implied by the dynamics, the actual acceleration from the forces may be greater than that. When pushed, rather than simply falling sideways, an agent could accelerate faster to reach a stable, controlled state.

We classify these forces into two types, depending on the origin of the force:

Forces from agents are generated when an agent pushes (or is pushed into) another agent, or when there is a collision between two agents. This effect of a pushing force can persist across multiple time-steps depending on the agent's response. The effect of a pushing force on an agent can also propagate to other agents as a result of the first agent's being pushed into others. The force imparted by the agent onto an object is given as an input to a rigid body dynamic simulation, which we use to simulate the behavior of the objects in the environment. This simulation accounts for the impact of agent's force on the motion of the object.

Forces from objects are the forces an agent receives from objects. Note that forces acting upon agents from objects are only those caused by the collision, i.e. the reaction force. These forces are then summed up and represented as a Force Constraint, an additional constraint to the velocity computation.

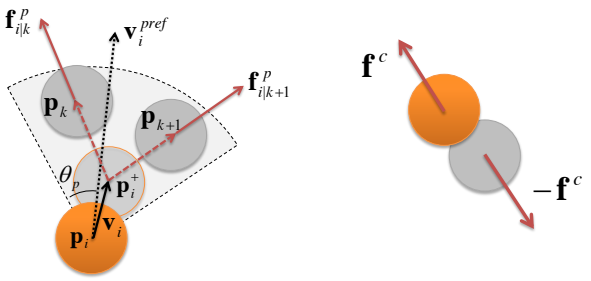
3.3 Velocity Computation Algorithm

We can summarize our new velocity computation algorithm as follows: Given an agent A with neighbors B, we define the *permitted velocities* for A, PV_A as the intersection of all velocity constraints. We can state our agent update algorithm as an optimization problem. Formally:

$$PV_A = FC_A \cap \bigcap_{B \neq A} ORCA_{A|B} \qquad (2)$$

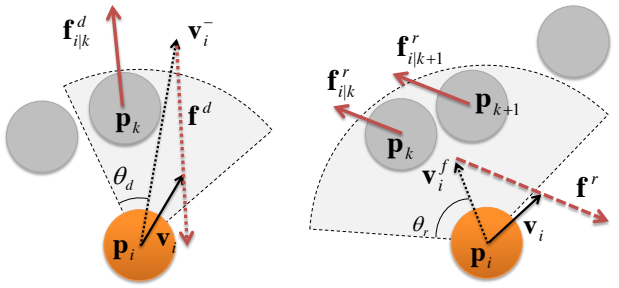
$$\mathbf{v}^{new} = \underset{\mathbf{v} \in PV_A}{\operatorname{argmin}} \|\mathbf{v} - \mathbf{v}^{pref}\|. \qquad (3)$$

In conditions where the preferred velocity of an agent is only determined by physical forces (i.e., in the absence of navigation constraints) the formulation will reproduce the motion based on Newton's second Law. This is because the closest velocity to the agent's current velocity will be the perpendicular distance to the velocity constraint FC_A.



(a) *Pushing Force* **(b)** *Collision Force*

Figure 3: ***Contact Forces*** *An agent (orange disk) can affect nearby neighbors (grey disks) through physical contact expressed as impulse forces. These physical forces can be used to push other agents as in (a) or resolve physical collisions as in (b). The red arrows display the direction of the resulting forces.*



(a) *Deceleration Force* **(b)** *Resistive Force*

Figure 4: ***Inferred Forces****: Forces between agents can be inferred based on local navigation. If an agent has a large change in velocity in the absence of force applied to the agent, as in (a), then a deceleration force, $\mathbf{f}^d$, is inferred to have caused the change, and is applied to nearby agents. (b) Likewise, when a force is applied to an agent which produces no change in agent's velocity, we model in terms of resistive force $\mathbf{f}^r$, which implicitly opposes this motion. This inferred resistive force is also applied to nearby agents.*

4 Force Computation

In this section, we present our approach for computing velocity constraints from physical forces. We propose an approximate approach because we need to handle a large number of agents in dense environments. As a result, we approximate the physical interactions based on appropriate velocity constraints. We assume that these forces are initiated from a collision or by pushing. When an agent experiences these forces, the impact on its motion lasts more than one timestep because of its effort to recover momentum. We approximate the effects of momentum by using two *inferred forces*: a resistive force and a deceleration force. These two additional forces have the net effect of propagating the momentum through the crowd.

4.1 Contact Forces

Pushing Forces: Pushing is one of the ways for agents to *physically* interact with each other [Pelechano et al. 2007]. Agents can impart a pushing force on nearby agents. In our formulation, agents can impart a pushing force to nearby agents; this pushing force follows the approximate direction of the pushing agent's preferred velocity and pushes the blocking agent out of the pushing agent's path (see Fig. 3a). The pushing force imparts an impulse to the nearby agents in the direction of the normal vector from the pushing agent towards the pushed agent. Formally, the pushing force $\mathbf{f}^p_{i|k}$ exerted by an agent i pushing another agent k can be given as:

$$\mathbf{f}^p_{i|k} = \rho_k f_p \frac{\mathbf{p}_k - \mathbf{p}_i^+}{\|\mathbf{p}_k - \mathbf{p}_i^+\|}, \tag{4}$$

where $\mathbf{p}_i$ and $\mathbf{p}_k$ indicate the positions of agent i and k, respectively, and $\mathbf{p}_i^+ = \mathbf{p}_i + \mathbf{v}_i \Delta t$ is the pushing agent's future position at the next time step. f_p is used to define the weight of pushing force, and we formulate it as an inverse of number of agents that are pushed.

Collisions: In case of collisions between agents, a collision resolution force is applied (Fig. 3b). This force is computed based on the physically-based simulation approach proposed by [Baraff 1997], and depends on the mass and velocity of colliding agents. We consider only linear momentum and simulate agents as radially symmetric disks. As a result, we do not take into account the orientation of the agents. For an agent i colliding with agent k, the collision force $\mathbf{f}^c$ is computed as follows:

$$\mathbf{f}^c = \left(\frac{-(1+\epsilon)\mathbf{v}^{rel}}{1/m_i + 1/m_k} \cdot \mathbf{n}\right)/\Delta t, \tag{5}$$

where $\mathbf{n}$ is the collision normal, pointing towards agent i from agent j; v^{rel} is relative velocity; and m_i and m_k are the mass of agent i and agent k, respectively. ϵ is the coefficient of restitution.

4.2 Inferred Forces

We also define two forces, deceleration force and resistive force, to model agent's ability to adjust their motion when external force is applied. The contact forces that result from agents colliding or pushing each other are computed as impulses. After being bumped or pushed, an agent will naturally exert forces in order to quickly recover its preferred velocity. Forces will therefore propagate through a dense crowd, since one agent is likely to push others in order to recover from the external pushing force.

This kind of behaviors are inspired from biomechanics, an observation about how humans react to recover their balance in various conditions including when the external forces are applied to the body. More details are given in [Kim et al. 2013].

These propagation forces can be inferred when the motion computed using constrained optimization does not match the motion expected from external physical forces. For example, when an agent decelerates at a faster rate than that implied by the external forces, we infer that the agent must be pushing against other agents or obstacles in order to be able to slow down so quickly. Likewise, when an agent accelerates at a rate less than that implied by external forces, we infer the agent must be pushing against other agents or obstacles, which resist the effect of the forces. These inferred propagation forces are applied to the appropriate neighboring agents during the subsequent timestep. We describe the formulation for each of these forces below.

Deceleration Forces: When an agent reduces speed while preserving direction to within a certain threshold (θ_d), we introduce a force into the system based on this velocity change. The deceleration force generated by agent i's deceleration is defined as:

$$\mathbf{f}^d_i = \begin{cases} k_{thresh} m_i \Delta \mathbf{v}_i / \Delta t & \text{if } (\Delta \hat{\mathbf{v}}_i \cdot \hat{\mathbf{v}}_i) < -cos(\theta_d), \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $\Delta \mathbf{v}_i = \mathbf{v}_i - \mathbf{v}_i^-$ is the change in velocity from the previous time step to the current time step. Because agents are not rigid bodies, they can absorb or transform forces. We approximate this behavior by introducing a parameter k_{thresh}.

We assume that the speed reduction arises from one of two sources: self-will (e.g. sudden change of preferred velocity) or agent interaction (e.g. impending collision avoidance). When there are no interacting agents, we assume it is the former case, and the deceleration force is applied back to the agent itself. In the latter case, where the deceleration is caused by interaction with the agents neighbors, the behavior of those neighbors should also change as a result of the interactions; we thus distribute the deceleration force among them in the case of collision avoidance. Furthermore, a neighboring agent k causes such behavior if it lies within a cone centered on $\mathbf{v}_i^-$ and is within an angular space of $2\theta_d$ degrees (as shown in Fig. 4a). For each interacting neighbor k of agent i, the portion of the deceleration force acting on agent k is defined as:

$$\mathbf{f}_{i|k}^d = -\delta_k \mathbf{f}_i^d, \tag{7}$$

where δ_k is a parameter that indicates how the deceleration force is transferred to agent k. We set this parameter to $1/n$, where n is the number of interacting agents.

Resistive Forces: Resistive forces occur when an agent's computed velocity does not account for the entire change in velocity expected from the external force. This difference is propagated to neighboring agents via the resistive forces. This force is computed by the difference between the velocity $\mathbf{v}$ computed by (3) and the velocity $\mathbf{v}^f$ computed only from the net force applied to the agent. The resistive force of an agent i experiencing the discrepancy between $\mathbf{v}^f$ and $\mathbf{v}$ is:

$$\mathbf{f}_i^r = \begin{cases} k_{thresh} m_i (\mathbf{v}_i - \mathbf{v}_i^f)/\Delta t & \text{if } \mathbf{v}_i^f \neq \mathbf{0} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

As in the case of deceleration force, the resistive force is applied to the agent i when there is no interacting agent. Otherwise, the resistive force is distributed equally among the interacting agents, whose position is inside a cone centered on $\mathbf{v}_i^f$ and with an angular span of $2\theta_r$ degrees (as shown in Fig. 4b). The resistive force $\mathbf{f}_{i|k}^r$ applied to agent k is given as:

$$\mathbf{f}_{i|k}^r = -\gamma_k \mathbf{f}_i^r, \tag{9}$$

where γ_k is a weighting parameter for agent k (we use $1/n$).

The resistive force and deceleration force can be viewed as complementary to one another. The resistive force is non-zero only in the presence of external physical forces on an agent, and the deceleration force is non-zero only in the absence of such forces.

4.3 Force Constraints

The net force $\mathbf{f}$ is the sum of all the forces applied to the agent. Mathematically, force $\mathbf{f}$ used to compute force constraint FC (described in (12)) is computed as follows:

$$\mathbf{f} = \sum \mathbf{f}^c + \sum \mathbf{f}^d + \sum \mathbf{f}^r + \sum \mathbf{f}^p. \tag{10}$$

The force constraint FC induced by the net force $\mathbf{f}$ is computed as follows:

$$\mathbf{v}^f = \mathbf{v} + \frac{\mathbf{f}}{m}\Delta t \tag{11}$$

$$FC = \{\mathbf{v} | (\mathbf{v} - \mathbf{v}^f) \cdot \hat{\mathbf{f}} \geq 0\}. \tag{12}$$

FC is a half plane whose boundary, a line through $\mathbf{v}^f$, is perpendicular to the normalized force $\hat{\mathbf{f}}$. This half plane contains a set of velocities that is equal to or greater than the minimum velocity change required by the force $\mathbf{f}$. This term is used for velocity computation in Equation (2).

5 Interaction with Obstacles

A key part of our approach is to model interactions between the agents and static and dynamic objects, i.e. two-way coupling between agents and obstacles. The behavior of agents towards the objects around them includes anticipatory collision avoidance, pushing, and unintended collisions. An agent might also impose forces from its motion (e.g., resistive force and deceleration force) on obstacles, as it does to other agents. If there is a collision, then objects also exert forces on the agent. In this section, we present an efficient algorithm to model these interactions for interactive applications.

5.1 Dynamic Objects

There are some significant differences between agent-agent and agent-obstacle interactions, both in terms of the motion computation and in how an agent responds to those obstacles. Importantly, the motion of obstacles (e.g. rigid bodies) is governed by Newtonian physics, since these objects have no will and are unable to initiate movement on their own. As a result, the agents cannot assume that the obstacles will anticipate collisions and change trajectory to avoid them. Moreover, the rigid body simulation is performed on the obstacle motion in 3D space, while the agents are constrained to move on a 2D plane.

5.1.1 Anticipatory Collision Avoidance

In our approach, agents attempt to anticipate and avoid collisions with the obstacles. Since the agent's navigation is performed in 2D space, we project the boundary of the dynamic obstacle onto the 2D plane (see Fig. 5).

The dynamic object O is represented, like the agents, as an open disc centered at $\mathbf{p}$ with the radius r of the bounding sphere of the object. While we use this bounding shape for collision avoidance with the agents, the underlying rigid body simulation uses an exact 3D object representation for collision detection and for response to other objects in the scene.

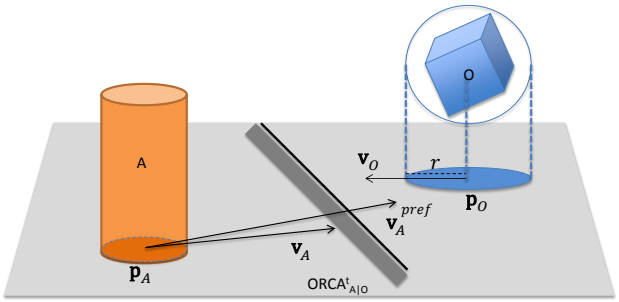


Figure 5: ***Collision Avoidance and Anticipation with a 3D object projected onto 2D plane*** *We take into account the object location in computing appropriate collision avoidance constraints for agent A, shown in the shaded region.*

Agents try to avoid collisions with dynamic obstacles, just as they try to avoid collisions with other agents, whenever the dynamic obstacles are within agent's visual range. However, agents do not

assume objects will reciprocate in avoiding collisions. Therefore, assuming that a change in velocity of $\mathbf{u}$ (Section 3.1) is required to avoid an anticipated collision with an obstacle, the agent will modify its velocity by at least $\mathbf{u}$; this is twice as large as the velocity bound using ORCA algorithm.

Therefore, the collision avoidance constraint for agent A induced by object O is:

$$ORCA^{\tau}_{A|O} = \{\mathbf{v}|(\mathbf{v} - (\mathbf{v}_A + \mathbf{u})) \cdot n \geq 0\}. \quad (13)$$

5.1.2 Agent-Object Collisions

When there is a collision between an agent and an object, the impulse force $\mathbf{f}^c$ is computed by the method used in [Baraff 1997]. We only consider rotational factors in the computation of object motion, not for the agents. We can compute the impulse force $\mathbf{f}^c$ from the collision between an agent a and object o is as follows:

$$\mathbf{f}^c = (\frac{-(1+\epsilon)\mathbf{v}^{rel}}{1/m_a + 1/m_o} \cdot \mathbf{n})/\Delta t, \quad (14)$$

where m_o is the mass of object o, $\mathbf{v}^{rel}$ and $\mathbf{n}$ are the relative velocity and the contact normal between the contact points, respectively. A force with the same magnitude but with the opposite direction is applied to the object, which also results in change of angular motion generated by the torque τ^c:

$$\tau^c = \mathbf{f}^c \times \mathbf{r}_o, \quad (15)$$

where $\mathbf{r}_o$ is the displacement vector for the contact point of the object.

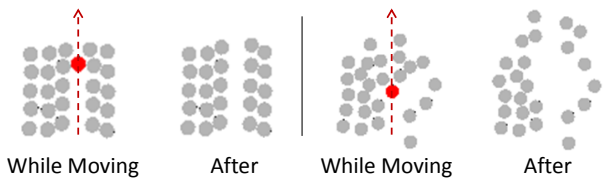
6 Results & Analysis

In this section, we highlight the performance of our algorithm in different scenarios. We also analyze the approach and compare it with other techniques.

6.1 Agent-Agent Interaction

We first demonstrate a few scenarios which highlight the effect of forces propagating in agent-agent interactions.

Running Through Scenario: We demonstrate a scenario where an agent runs at a high speed through a dense crowd of 25 agents that are standing still. Figure 6 compares the result of our method to those achieved using multi-agent simulation without any physical interactions.



(a) *without physical interactions* **(b)** *with physical interactions*

Figure 6: ***Rushing through still agents****: The red agent tries to rush through a group of standing agents, simulated (a) with only anticipatory collision avoidance and (b) with physical interactions. Using our method, the forces are propagated among the agents, resulting in a new distribution pattern (b).*

The left side of each image shows a pushing agent (red) passing through the crowd, and the right side of each image shows the position of all other agents in the crowd after the fast-moving agent has passed. As Fig. 6 demonstrates, agents simulated without physics-based interaction use minimal motion to avoid collisions. In contrast, agents simulated using our physically-based formulation resist the pushing motion (in an attempt to stand still) and propagate the effects of being pushed to other agents.

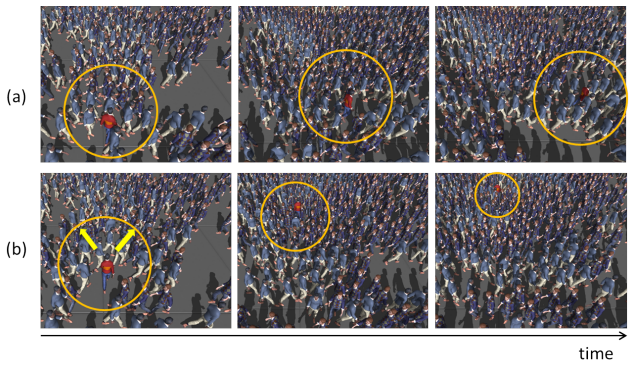


Figure 7: ***Pushing through dense crowd****: The red agent pushes through a dense crowd that moves perpendicular to its direction of travel. Agents are simulated using (a) ancipatory collision avoidance only, and (b) combination of anticipatory collision avoidance and physically-based interaction. In the latter case, the red agent can proceed to its goals quickly by pushing other agents through its path.*

Dense Crossing Scenario: In this case, an agent attempts to cross perpendicularly through a dense stream of crowd flow. Fig. 7 shows a comparison between our method and using no physical interactions.

As the figure shows, an agent who is only avoiding collisions (without pushing) cannot effectively cut through the crowd's flow, is eventually swept up with the crowd, as that motion avoids all impending collision. This is because moving with the crowd successfully avoids all impending collisions. However, the pushing force based on our approach allows an agent to clear its path and move freely.

Two Bottlenecks Scenario: In this scenario, long lines of closely spaced walking agents attempt to pass through two narrow bottlenecks, as illustrated in Fig. 8. The first bottleneck (shown as (2)) is about the width of two agents; the second is narrower, about wide enough for one agent (shown as (1)). A local navigation algorithm that performs collision avoidance frequently results in congestion at both the bottlenecks due to stable-arch formation of agents (highlighted with a yellow circle) in Fig. 8 (a). However, agents simulated by our physically-based method are able to break this congestion at the bottleneck area by pushing the blocking agents. The ability to break through bottlenecks also results in a quantitatively higher rate of flow for agents using our approach. After seconds, twice as many agents make it through both the bottlenecks, using our algorithm.

6.2 Agent-object Interaction

We can also demonstrate the effect of two-way coupling between dynamic objects and agents in multi-agent simulations. In the following scenarios, the Bullet Physics engine [AMD 2012] is used to compute the 3D rigid body dynamics, which in turn are used to

(a) *Multi-agent simulation with no physical interaction*

(b) *Physical interaction amongst agents and with the walls*

Figure 8: ***Two bottlenecks scenario*** *We simulate and compare crowd behavior at two narrow bottlenecks in these scenarios, (1) and (2), which are marked with red dotted lines. Bottleneck (1) is barely wide enough for one person to pass through; bottleneck (2) is about twice that width and allows two agents to pass through it at a time. The result from collision-avoidance-only simulation results in an arch-shaped arrangement of agents in the crowd (highlighted with a yellow circle), which causes congestion at the bottleneck. Our method breaks the congestion by allowing the agents to push one other in congested conditions.*

compute object motion (see Fig. 2). The effects of user interaction in these scenarios can be seen in the supplemental video[1].

Rolling Ball Scenario: In this scenario, a few agents interact with varying numbers of dynamically generated balls. A user can interact with the agents by moving around the dynamic obstacles, or by generating new balls. Agents attempt to avoid these dynamically moving balls and push them away when there is a collision.

Wall Breaking Scenario: In this scenario, long lines of agents come at a constant rate into the simulated region, which is blocked off with a movable wall made of 200 blocks glued together. This wall can be broken into separate blocks if a large external force is applied by the agents. Agents initially stop to avoid hitting the wall, but as other agents start to push from behind, the wall breaks apart and gets carried away with the agents. Fig. 1 shows stills from the simulation.

Cluttered Office Scenario: In this scenario, several decomposed 3d models - a table, a chair, and a shelf, and several rigid bodies (e.g. boxes) stacked on top of each other – are placed in the way of the agents. A long stream of agents attempts to navigate past the obstacles. Users can throw boxes, which push the agents and knock over objects in the environment. Fig. 9 shows a still from the simulation.

These scenarios demonstrate several features of our approach:

- *Dynamic Obstacle Avoidance:* Agents try to avoid collisions with other agents and with dynamic obstacles.
- *Agent-Object Interactions:* Our method takes into account the collisions which occur between the agents and the objects. The forces generated by these collisions affect both the objects and the agents.
- *User Interactions:* Our method is fast enough for real-time interactive simulation. Users can participate in the simulation by moving rigid bodies inside the scene; this movement dynamically changes the environment for the moving agent.

[1]Supplementary video can be found at
`http://gamma.cs.unc.edu/CrowdInteractions/`

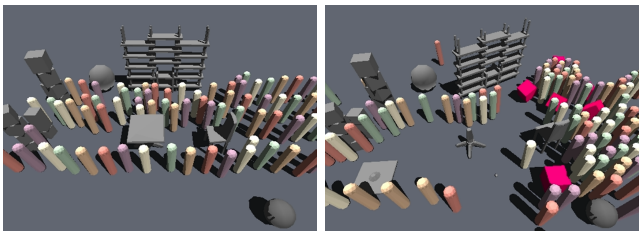


Figure 9: ***Office Scenario****. Agents navigate to avoid office furniture. As users insert flying pink boxes into the scene, the agents get pushed, collide into each other, and avoid falling objects (see video).*

6.3 Performance

We measured the simulation timings for the demos we presented (see Table 1). The timings were computed on a 3.4 GHz Intel i7 processor with 8GB RAM. Our method efficiently simulates large numbers of agents, and also exhibits interactive performance when integrated with the Bullet Physics Library.

Scenario	# Agents	# Dynamic Obstacles	# Static Obstacles	fps
Pushing Through	1600	0	0	229.6
Two Bottlenecks	1000	0	20	829.7
Rolling Balls	10	1000	2	1205.9
Wall Breaking	1200	200	2	50.1
Office	1200	65	0	69.0

Table 1: *Performance on a single core for different scenarios. Our algorithm can handle all of them at interactive rates.*

6.4 Analysis

Our approach is mainly designed for interactive applications that require plausible physical behavior (e.g. games or virtual worlds). By using a combination of force and navigation constraints that affect agents' behavior, our approach can simulate many use effects and emergent behaviors. For example, our formulation allows for intentionally uncooperative agents to physically push their way through a crowd by imparting physical forces to nearby agents. Additionally, agents can use navigation constraints to avoid collisions with dynamic obstacles as well as other agents. By expressing all interactions as linear velocity constraints, we can naturally combine the two different simulation paradigms of forces and navigation into a unified framework and compute the new velocity for each agent using linear programming. This is useful in generating physically plausible simulations of large numbers of agents.

Benefits of Our Method

Many techniques have been proposed in the literature for simulating large numbers of agents that display a wide variety of emergent behaviors. However, the primary emphasis of these methods is on collision avoidance – avoiding any physical contact between the

agents. In other words, they model how agents move around each other, but do not usually model explicit physical contacts, interactions, and external forces.

Force-based methods such as [Helbing and Molnár 1995] use forces to describe social factors (e.g. attraction and repulsion) between the agents, not physical interactions. Most closely related to our work are methods such as [Helbing et al. 2000; Yu and Johansson 2007; Pelechano et al. 2007], which model crowd turbulence or physical interactions among panicking agents by adding explicit physical force or by increasing repulsive forces. These methods are capable of reproducing some important emergent crowd phenomena, but do not account for the anticipation needed to efficiently avoid upcoming collisions with other agents and obstacles [Curtis et al. 2012].

Force-based methods can also suffer from stability issues in dense scenarios, which require careful tuning and small time steps in order to remain stable [Curtis et al. 2011]. Our method provides stable, anticipatory motion for agents while incorporating agent responses to forces. It can be easily combined with other velocity-based approaches. Our approach is also stable in terms of varying the size of time-steps. More details are given in [Kim et al. 2013].

Limitations

We use a physically-inspired approach to simulate the interactions between a high number of agents and the obstacles. However, it is only an approximation and may not be physically accurate. Secondly, we assume that agents are constrained to move along a 2D plane, and we use the projected positions of 3D dynamic objects to compute the interactions. Third, like other agent-based simulation methods, we use a rather simple approximation for each agent (a 2D circle). This means that we cannot accurately simulate physical interactions with human-like articulated models and 3D objects.

7 Conclusion and Future Work

We have proposed a novel method to combine physics-based interactions with anticipatory collision-avoidance techniques that use velocity-based formulation. Our method can generate many emergent behaviors, physically-based collision responses, and propagation of forces to the agent's nearby neighbors. In combination with the Bullet Physics library, we were able to simulate complex interactions between agents and dynamic obstacles in the environment. The resulting approach is useful for interactive large-scaled simulations and can generate physically plausible behaviors. This approach has been extended to model physical interactions between dense crowds and applied to Tawaf simulation [Kim et al. 2013].

In our future work, we would like to further explore our method by comparing the results with real-world crowd behaviors and by performing more validation. We would like also to extend our model to agents moving in 3D space or multi-layer frameworks, and to consider using more complex shapes, or even articulated body models, to represent agents, as this would allow for more accurate force computation. Finally, we would like to use more accurate physically-based modeling algorithms to generate appropriate behaviors.

Acknowledgements

We are grateful to the reviewers for their comments, we would like to thank Sean Curtis, Ming C. Lin and Ioannis Karamouzas for their help and feedback, and Karl Hillesland, Erwin Coumans, and Jason Yang from AMD for their support. This research is supported in part by ARO Contracts W911NF-10-1-0506, W911NF-12-1-0430, NSF awards 0917040, 0904990, 100057, and 1117127, AMD, and Intel.

References

AMD, 2012. Bullet Physics 2.80. http://bulletphysics.org.

BARAFF, D. 1997. An introduction to physically based modeling: Rigid body simulation i - unconstrained rigid body dynamics. In *In An Introduction to Physically Based Modelling, SIGGRAPH '97 Course Notes*, 97.

CURTIS, S., GUY, S. J., ZAFAR, B., AND MANOCHA, D. 2011. Virtual tawaf: A case study in simulating the behavior of dense, heterogeneous crowds. In *1st IEEE Workshop on Modeling, Simulation and Visual Analysis of Large Crowds*, 128–135.

CURTIS, S., ZAFAR, B., GUTUB, A., AND MANOCHA, D. 2012. Right of way. *The Visual Computer*, 1–16.

DURUPINAR, F., PELECHANO, N., ALLBECK, J., GÜ ANDDÜ ANDKBAY, U., AND BADLER, N. 2011. How the ocean personality model affects the perception of crowds. *Computer Graphics and Applications, IEEE 31*, 3 (may-june), 22 –31.

GUY, S. J., KIM, S., LIN, M. C., AND MANOCHA, D. 2011. Simulating heterogeneous crowd behaviors using personality trait theory. In *Symposium on Computer Animation*, ACM, 43–52.

GUY, S. J., CURTIS, S., LIN, M. C., AND MANOCHA, D. 2012. Least-effort trajectories lead to emergent crowd behaviors. *Phys. Rev. E 85* (Jan), 016110.

HELBING, D., AND MOLNÁR, P. 1995. Social force model for pedestrian dynamics. *Phys. Rev. E 51* (May), 4282–4286.

HELBING, D., FARKAS, I., AND VICSEK, T. 2000. Simulating dynamical features of escape panic. *Nature 407*, 6803 (Sept.), 487–490.

HUGHES, R. L. 2003. The flow of human crowds. *Annual Review of Fluid Mechanics 35*, 1, 169–182.

KARAMOUZAS, I., AND OVERMARS, M. 2012. Simulating and evaluating the local behavior of small pedestrian groups. *IEEE Trans. on Visualization and Computer Graphics 18*, 3, 394–406.

KIM, M., HYUN, K., KIM, J., AND LEE, J. 2009. Synchronized multi-character motion editing. *ACM Trans. Graph. 28*, 3 (July), 79:1–79:9.

KIM, S., GUY, S. J., MANOCHA, D., AND LIN, M. C. 2012. Interactive simulation of dynamic crowd behaviors using general adaptation syndrome theory. In *Symposium on Interactive 3D Graphics*, ACM, New York, NY, USA, I3D '12, 55–62.

KIM, S., GUY, S. J., ZAFAR, B., GUTUB, A., AND MANOCHA, D. 2013. Velocity-based modeling of physical interactions in multi-agent simulations in dense crowd. Tech. rep., Department of Computer Science, University of North Carolina at Chapel Hill.

LEE, K. H., CHOI, M. G., HONG, Q., AND LEE, J. 2007. Group behavior from video: a data-driven approach to crowd simulation. In *Symposium on Computer Animation*, 109–118.

LERNER, A., CHRYSANTHOU, Y., SHAMIR, A., AND COHEN-OR, D. 2009. Data driven evaluation of crowds. In *MIG*, 75–83.

MUICO, U., POPOVIĆ, J., AND POPOVIĆ, Z. 2011. Composite control of physically simulated characters. *ACM Transactions on Graphics 30*, 3.

MUSSE, S. R., AND THALMANN, D. 1997. A model of human crowd behavior: Group inter-relationship and collision detection

analysis. In *Proc. Workshop of Computer Animation and Simulation of Eurographics'97*, 39–51.

NARAIN, R., GOLAS, A., CURTIS, S., AND LIN, M. C. 2009. Aggregate dynamics for dense crowd simulation. *ACM Trans. Graph. 28*, 5 (Dec.), 122:1–122:8.

ONDŘEJ, J., PETTRÉ, J., OLIVIER, A.-H., AND DONIKIAN, S. 2010. A synthetic-vision based steering approach for crowd simulation. *ACM Trans. Graph. 29*, 4 (July), 123:1–123:9.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2007. Controlling individual agents in high-density crowd simulation. In *Symposium on Computer animation*, 99–108.

PETTRÉ, J., ONDŘEJ, J., OLIVIER, A.-H., CRETUAL, A., AND DONIKIAN, S. 2009. Experiment-based modeling, simulation and validation of interactions between virtual walkers. In *Symposium on Computer Animation*, ACM, SCA '09, 189–198.

REYNOLDS, C. 1999. Steering Behaviors for Autonomous Characters. In *Game Developers Conference 1999*.

SAKUMA, T., MUKAI, T., AND KURIYAMA, S. 2005. Psychological model for animating crowded pedestrians: Virtual humans and social agents. *Comput. Animat. Virtual Worlds 16*, 343–351.

SHAO, W., AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *Symposium on Computer animation*, 19–28.

SHAPIRO, A., PIGHIN, F., AND FALOUTSOS, P. 2003. Hybrid control for interactive character animation. In *Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, PG '03, 455–.

SHUM, H. P. H., KOMURA, T., AND YAMAZAKI, S. 2012. Simulating multiple character interactions with collaborative and adversarial goals. *IEEE Transactions on Visualization and Computer Graphics 18*, 5 (May), 741–752.

SOK, K. W., YAMANE, K., LEE, J., AND HODGINS, J. 2010. Editing dynamic human motions via momentum and force. In *Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '10, 11–20.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. In *ACM SIGGRAPH 2006*, ACM, 1160–1168.

ULICNY, B., AND THALMANN, D. 2002. Towards interactive real-time crowd behavior simulation. In *Computer Graphics Forum*, vol. 21, Wiley Online Library, 767–775.

VAN DEN BERG, J., GUY, S. J., LIN, M., AND MANOCHA, D. 2011. Reciprocal n-body collision avoidance. In *Robotics Research: 14th ISRR (STAR)*, vol. 70, 3–19.

YEH, H., CURTIS, S., PATIL, S., VAN DEN BERG, J., MANOCHA, D., AND LIN, M. 2008. Composite agents. In *Symposium on Computer Animation*, 39–47.

YU, W., AND JOHANSSON, A. 2007. Modeling crowd turbulence by many-particle simulations. *Phys. Rev. E 76* (Oct), 046105.

YU, Q., AND TERZOPOULOS, D. 2007. A decision network framework for the behavioral animation of virtual humans. In *Symposium on Computer animation*, 119–128.

ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. *ACM Trans. Graph. 24*, 3 (July), 697–701.

COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2011; **22**:151–158
Published online 12 April 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.403

SPECIAL ISSUE PAPER

Footstep navigation for dynamic crowds

Shawn Singh*, Mubbasir Kapadia, Glenn Reinman and Petros Faloutsos

Department of Computer Science, University of California, Los Angeles, USA

ABSTRACT

The majority of steering algorithms output only a force or velocity vector to an animation system, without modeling the constraints and capabilities of human-like movement. This simplistic approach lacks control over how a character should navigate. This paper proposes a steering method that uses *footsteps* to navigate characters in dynamic crowds. Instead of an oriented particle with a single collision radius, we model a character's center of mass and footsteps using a 2D approximation of an inverted spherical pendulum model of bipedal locomotion. We use this model to generate a timed sequence of footsteps that existing animation techniques can follow exactly. Our approach not only constrains characters to navigate with realistic steps but also enables characters to intelligently control subtle *navigation* behaviors that are possible with exact footsteps, such as side-stepping. Our approach can navigate crowds of hundreds of individual characters with collision-free, natural steering decisions in real-time. Copyright © 2011 John Wiley & Sons, Ltd.

KEYWORDS

crowds; footprints; footsteps; navigation; steering

***Correspondence**

Shawn Singh, Department of Computer Science, University of California, Los Angeles, Boelter Hall 4531F, Los Angeles, CA 90095, USA.

E-mail: shawnsin@cs.ucla.edu

1. INTRODUCTION

The majority of previous steering algorithms represent a character as an oriented particle that moves by choosing a force or velocity vector. Often, orientation is heuristically chosen to be the particle's velocity. This approach has the two key disadvantages:

- *Limited locomotion constraints*: Most steering algorithms do not account for constraints of real human movement. Trajectories may have discontinuous velocities, oscillations, awkward orientations, or may try to move a character unnaturally, and these side-effects make it harder to animate the character intelligently.
- *Limited navigation control*: It is common to assume that an animation system will know how to interpret a vector-based steering decision. In practice, a vector does not have enough information to indicate appropriate maneuvers, such as side-stepping versus reorienting the torso, stepping backwards versus turning around, planting a foot to change momentum quickly, or carefully placing steps in exact locations.

We propose to generate sequences of footsteps as the output of navigation. Since there are already several animation techniques that can animate a character to follow timed footsteps exactly [1–6], the main challenge and focus of our work is how to *generate* footsteps as the output of navigation. Footsteps are an intuitive abstraction for most locomotion tasks, and they provide precise, unambiguous spatial and timing information to animation.

In our system, each step is defined by a 2D parabolic trajectory that approximates the motion of a 3D inverted pendulum. The location, orientation, and timing of footsteps are derived from the these trajectories. We use a best-first search to plan a sequence of space–time parabolic trajectories and the associated footsteps that avoids time-varying collisions, satisfies footstep constraints for natural locomotion, and minimizes the effort to reach a local goal. Characters successfully avoid collisions with each other and choose steps that correspond to natural and fluid motion, including precise timing. Because the most significant biomechanics constraints are already taken into account in our model, integrating our results with an existing animation algorithm that follows footsteps is straightforward and results in navigation that is often richer and less awkward than vector-based navigation.

Contributions. This paper presents a new approach to steering in dynamic crowds that uses a simple biomechanically-based footstep model combined with space–time

planning. Our work demonstrates that a steering algorithm can have better navigation features than a vector interface, while still retaining fast performance. These features include: short-term space–time planning, dynamic collision bounds, appropriate movement constraints, and more precise navigation control. Because substantial work already exists to animate characters to follow exact footsteps including timing information, we focus on the navigation: how to generate biomechanically plausible footsteps for dynamic crowds.

2. RELATED WORK

Two widely accepted strategies are (1) the social forces model [7], which associates a small force field around agents and obstacles, and (2) the steering behaviors model [8], where forces are procedurally computed to perform desired functions such as seek, flee, pursuit, evasion, and collision avoidance. Many works are extensions or elaborations of these two ideas [9–17]. A more complete survey of collision avoidance, navigation, and crowd simulation work can be found in Reference [18]. The common theme in these works is the use of force or velocity vectors as navigation decisions, which has the limitations described above.

Only a few steering techniques take into account locomotion constraints that an animation system will have. Paris and Donikian [19] demonstrate a framework where the animation module can potentially tell steering that an action is not plausible. Musse and Thalmann [20] and Shao and Terzopoulos [21] both address higher-level aspects of pedestrians, and their navigation modules output a choice from a set of navigation behaviors that correspond directly to animations the character can produce. Van Basten and Egges [22] discuss problems of interfacing navigation with animation, proposing abstractions that reduce such discrepancies.

Another approach to navigation is to plan sequences of motion clips [23], demonstrated this is possible in real-time for crowds, by precomputing a tree of all possible sequences of motion clips. However, a large number of motion clips would be needed to emulate the versatility of far fewer stepping options. The technique of precomputing a search tree can also be applied to our footstep planner, but our approach is scalable even without this extension.

Footsteps. Several animation techniques, academic and commercial, can *follow* a given sequence of footsteps [1–6], and more. Animation methods in these works include forward and inverse kinematics, physically based control, and motion capture.

The challenge of *generating* footsteps has so far only been explored for single characters in static environments. Research in robotics [24–28] explores autonomous foot-placement to avoid obstacles while navigating towards a goal. Their focus is practical robot control, and so they do not consider issues of real human locomotion. Torkos and Van de Panne [1] generate footsteps to randomly wander, changing direction if nearby objects are too close, used to demonstrate their animation system. Chung and Hahn [2] input a trajectory, and generate footsteps by aligning each step to the orientation of the trajectory, with smaller footsteps around curves. Choi *et al*. [29] use roadmaps to plan sequences of steps, choosing from steps that are possible with the given motion clips and requiring costly roadmap construction and footstep verification. Zhang *et al*. [30] propose a hierarchical planning approach that computes full-body motion including footsteps for tasks in highly constrained environments. Recently several papers have considered footsteps as a way of guiding controllers for physically-based character animations [3,31].

3. FOOTSTEP MODEL

The primary data structure in our model is a *footstep*, which includes: (1) the position, velocity, and timing of the character's center of mass trajectory, (2) the location and orientation of the foot, and (3) the cost of taking the step. In this section, we describe these aspects of a footstep, as well as the constraints for choosing footsteps.

Center of Mass Trajectory. The analogy between human locomotion and the inverted pendulum is well known [32]; the pendulum pivot represents a point on or near a footstep, while the pendulum mass represents a character's center of mass. We define a 2D analytical approximation to the dynamics of an inverted spherical pendulum using parabolas. Piecewise parabolic curves are enough to capture the variety of trajectories that a human's center of mass will have: varying curvature, speed, and step sizes. Each step is a parabola defined with the following parameters in local space:

$$(x(t), y(t), \dot{x}(t), \dot{y}(t)) = \left(v_{x_0} t, \alpha t^2, v_{x_0}, 2\alpha t\right) \qquad (1)$$

such that both v_{x_0} and α are positive.

Equation 1 allows us to *analytically* evaluate the position and velocity of a character's center of mass at any time t. This makes it practical to search through many possible trajectories for many characters in real-time.

3.1. Footstep Actions

The state of the character $s \in \mathcal{S}$ is defined as follows (Figure 1a):

$$s = \{(x, y), (\dot{x}, \dot{y}), (f_x, f_y), f_\phi, I \in \{\mathrm{L}, \mathrm{R}\}\}$$

where (x, y) and $(\dot{x}, \dot{y})$ are the position and velocity of the center of mass of the character at the *end* of the step, (f_x, f_y) and f_ϕ are the location and orientation of the foot, and I is an indicator of which foot (left or right) is taking the step. The state space $\mathcal{S}$ is the set of valid states that satisfy the constraints described below.

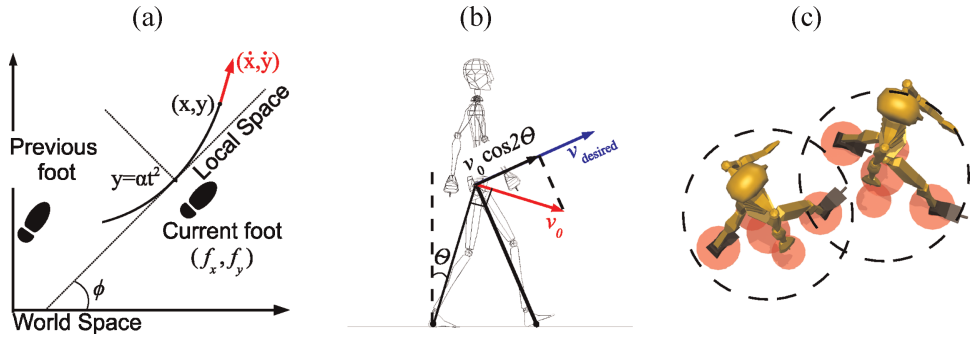


Figure 1. Our footstep model. (a) Depiction of state and action parameters. (b) A sagittal view of the pendulum model used to estimate energy costs. (c) The collision model uses five circles that track the torso and feet over time, allowing tighter configurations than a single coarse radius.

A *footstep action* determines the next parabolic trajectory, defined as $a \in \mathcal{A}$:

$$a = \{\phi, v_{\text{desired}}, T\}$$

where ϕ is the desired orientation of the parabola, v_{desired} is the desired initial speed of the center of mass, and T is the desired time duration of the step. The action space $\mathcal{A}$ is the set of valid footstep actions, where the input and output states are both valid. Note that when the character's previous step is fixed, varying ϕ directly affects the width of the parabolic trajectory, thus allowing a large variety of step choices.

A key aspect of the model is the transition function, $s' = \textbf{createFootstep}(s, a)$. This function receives a desired footstep action a and a state s and returns a new state s' if the action is valid. It is implemented as follows. First, ϕ, which indicates the orientation of the parabola, is used to compute a transform from world space to local parabola space. Then, the direction of velocity $(\dot{x}, \dot{y})$ from the end of the previous step is transformed into local space, normalized, and re-scaled by the desired speed v_{desired}. With this local desired velocity, there is enough information to solve for α, and then Equation 1 is used to compute (x, y) and $(\dot{x}, \dot{y})$ at the end of the next step. In local space, the foot location is always located at $(f_x, f_y) = (0, -d)$, where d is describes the distance between a character's foot and center at rest. Finally, all state information is transformed back into world space, which serves as the input to create the next footstep.

3.2. Locomotion Constraints

3.2.1. Biomechanical Properties.

Several properties of human locomotion are automatically enforced by the definition of our model. The piecewise parabola will be G-1 continuous, and the center of mass will remain between the two feet by enforcing the local-space parabola remains positive. Our footstep model offers a number of intuitive parameters with meaningful defaults and well-defined physical meaning. These parameters include the height of the character's center of mass, the min, max, and preferred step timing and stride length, the preferred and max velocities of the character's walk, the interval of valid foot orientations, etc. If these constraints are violated, the footstep is considered invalid. A user can modify these parameters to create new locomotion styles. For example, restricting the valid range of step timing and output velocity for one foot results in asymmetric limping, like an injured character.

3.2.2. Footstep Orientation.

Intuitively, it may seem that footstep orientations must be an additional control parameter when creating a footstep. However, the choice of footstep orientation has no direct effect on the dynamics of the center of mass trajectory; the foot orientation only constrains the options for current trajectory and future footsteps. This is a key aspect to our model's efficiency—instead of increasing the dimensionality of our search space to include foot orientation, we use orientation to constrain the search space of a lower dimensional system.

To implement this constraint, we compute an interval $[f_{\phi\text{inner}}, f_{\phi\text{outer}}]$ of valid foot orientations. This interval is constrained by the same interval from the previous step, and further constrained by the parabola orientation ϕ used to create the next footstep (Figure 2):

$$[f_{\phi\text{next\_inner}}, f_{\phi\text{next\_outer}}] = \left[f_{\phi\text{prev\_outer}}, f_{\phi\text{prev\_inner}} + \frac{\pi}{2}\right] \cap [\phi, \text{atan2}(\dot{y}, \dot{x})]$$

If this intersection becomes an empty set, that implies that no foot orientation can satisfy the step constraints, so the step is invalid. Note the ordering of bounds in these intervals; the next foot's outer bound is constrained by the previous foot's inner bound. In words, the interval $[\phi, \text{atan2}(\dot{y}, \dot{x})]$ describes two constraints: (1) the character would not choose a foot orientation that puts his center of mass on the outer side of the foot, (2) a human would rarely orient the next step more outwards than the direction of momentum; violating this constraint would put the

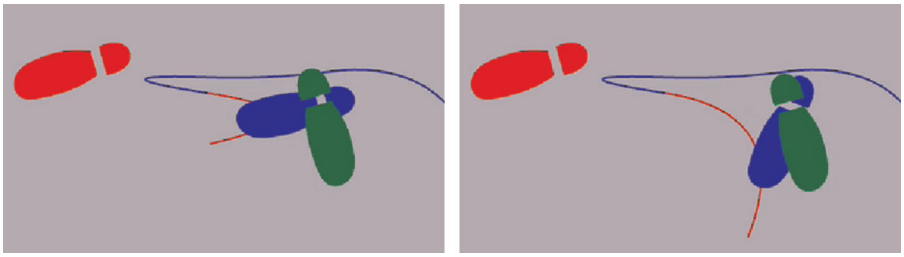


Figure 2. An interval of valid foot orientations (the blue and green feet) is maintained for each step, constrained by the previous step (red foot) and the chosen trajectory (red line).

character's center of mass on the wrong side of the foot. The exact orientation is chosen as a fast postprocess, described below.

3.2.3. Space–Time Collision Model.

For any given footstep, our model computes the *time-varying* collision bounds of the character at any exact time. To determine if a footstep causes a collision, we iterate over several time-steps within the footstep and query the collision bounds of nearby characters for that time. The collision bounds are five circles, depicted in Figure 1c. Each circle associated with a foot exists while the foot is planted on the ground. The three circles associated with the torso are placed on the center of mass, which moves along the parabola over several time-steps. If any of these circles collide with an obstacle or another character's circles, the footstep is considered invalid.

3.3. Cost Function

We define the cost of a given step as the energy spent to execute the footstep action. We model three forms of energy expenditure for a step: (1) ΔE_1, a fixed rate of energy the character spends per unit time, (2) ΔE_2, the work spent due to ground reaction forces to achieve the desired speed, and (3) ΔE_3, the work spent due to ground reaction forces accelerating the center of mass along the trajectory. The total cost of a footstep action transitioning a character from s to s' is given by:

$$c(s, s') = \Delta E_1 + \Delta E_2 + \Delta E_3 \tag{2}$$

3.3.1. Fixed Energy Rate.

The user defines a fixed rate of energy spent per second, denoted as R. For each step, this energy rate is multiplied by the time duration of the step T to compute the cost:

$$\Delta E_1 = R \cdot T \tag{3}$$

This cost is proportional to the amount of time it takes to reach the goal, and thus minimizing this cost corresponds to the character trying to minimize the time it spends walking to his goal. We found that good values for R are roughly proportional to the character's mass.

3.3.2. Ground Reaction Forces.

As a character pushes against the ground, the ground exerts equal and opposite forces on the character. We model three aspects of ground reaction forces that are exerted on the character's center of mass, from the study of biomechanics. The geometry and notation of the cost model is shown in Figure 1b. First, at the beginning of a new step (heel-strike), some of the character's momentum dissipates into the ground. We estimate this as an instantaneous loss of momentum along the pendulum shaft, reducing the character's speed from v_0 to $v_0\cos(2\theta)$. In order to resume a desired speed, the character actively exerts additional work on his center of mass, computed as:

$$\Delta E_2 = \frac{m}{2}\left|(v_{\text{desired}})^2 - (v_0\cos(2\theta))^2\right| \tag{4}$$

This cost measures the effort required to choose a certain speed. At every step, some energy is dissipated into the ground, and if a character wants to maintain a certain speed, it must actively add the same amount of energy back into the system. On the other hand, not all energy dissipates from the system after a step, so if the character wants to come to an immediate stop, the character also requires work to remove energy from the system. Minimizing this cost corresponds to finding footsteps that require less effort, and thus tend to look more natural. Furthermore, when walking with excessively large steps, $\cos(2\theta)$ becomes smaller, implying that more energy is lost per step.

It should be noted that there is much more complexity to real bipedal locomotion than this cost model. For example, the appropriate bending of knees and ankles and the elasticity of human joints can significantly reduce the energy lost per step, reducing the required work for a real human. While the model is not an accurate measurement of energy spent, it is sufficient for *comparing* the effort of different steps.

ΔE_2 captures only the cost of changing a character's momentum at the beginning of each step. The character's momentum may also change during the trajectory. For relatively straight trajectories, this change in momentum is mostly due to the passive inverted pendulum dynamics that requires no active work. However, for trajectories of high curvature, a character spends additional energy to change his momentum. We model this cost as the work required to

DOI: 10.1002/cav

change momentum (denoted as P) over the length of the step, weighted by constant w:

$$\Delta E_3 = w \cdot \frac{d\mathrm{P}}{dt} \cdot \text{length} = w \cdot m\alpha \cdot \text{length} \qquad (5)$$

Note that α is the same coefficient in Equation 1, the acceleration of the trajectory. α increases if the curvature of the parabola is larger, and also if the speed of the character along the trajectory is larger. Minimizing this cost corresponds to preferring straight steps when possible, and preferring to go slower (and consequently, taking smaller steps) when changing the direction of momentum significantly. The weight w can be adjusted to change whether it costs more energy to walk around an obstacle or to stop and wait for the obstacle to pass. We found good values of w to be between 0.2 and 0.5, meaning that 20 to 50 per cent of the curvature is due to the character's active effort, and the rest due to the passive inverted pendulum dynamics.

4. Generating Sequences of Steps

4.1. Discretizing Action Space

The choices for a character's next step are generated by discretizing the action space $\mathcal{A}$ described above, in all three dimensions and using the **createFootstep**(s, a) function to compute the new state and cost of each action. We have found that v_{desired} and T can be discretized extremely coarsely, as long as there are at least a few different speeds and timings. Further optimizations are made by observing that speed v_{desired} and step timing T have a slight inverse correlation, and so not all combinations of v_{desired} and T need to be generated. Most of the complexity of the action space lies in the choices for the parabola orientation, ϕ. The choices for ϕ are defined relative to the orientation of the velocity vector $(\dot{x}, \dot{y})$ from the end of the previous footstep, and the discretization of ϕ ranges from almost straight to almost U-turns. We note that the first choice that real humans would consider when navigating is to step directly toward the local goal. To address this, we create a special option for ϕ that would orient the character directly toward its goal. With this specialized goal-dependent option, we found it was possible to give fewer fixed options for ϕ, focusing on larger turns. Without this option, even with a large variety of choices for ϕ, the character appears to steer toward an offset of the actual goal and then takes an unnatural corrective step.

4.2. Short-Horizon Best-First Search

We use a best-first search planner for a sequence of footsteps that minimizes energy cost. The implementation of our planner is the same as an A* search, except for the *horizon*, described below.

The cost of taking a step is computed using Equations 2–5. The heuristic function used by the best-first search, $h(s)$, estimates the energy cost from the current state to a local goal:

$$h(s) = c_{\text{expected}} \times n \qquad (6)$$

where c_{expected} is the energy spent in taking one normal footstep action based on the character's user-defined parameters, and n is the number of steps it would take to travel directly to the goal.

The *horizon* of our planner is the maximum number of nodes to be expanded for a single search. In most cases, a path is found before this threshold. We limit the horizon so that difficult or unsolvable situations will not cause a significant delay. If the planner searches too many nodes without reaching the goal, we instead construct a path to a node from the closed list that had the best heuristic value (the same closed list used in A*). Intuitively, this means that if no path is found to the goal within the search horizon, the planner returns a path to the reachable state that had the most promise of reaching the goal. The short-horizon approach guarantees that we will have at least some path for the character to use, even in difficult or unsolvable planning problems. In worst case, if no good solution is found, the path will simply be a sequence of 'stop' actions. For example, this can occur when a character is stuck dense environment. Eventually when the density clears, the character will continue.

4.3. Local Goals and Collision Avoidance

To navigate through large environments, we first plan a path using A* (a traditional spatial path, not footsteps). Whenever a character needs to plan more footsteps, a local footstep goal is chosen, placed approximately 10 m ahead on the spatial path. This 10 m requirement is not strict; we experimented with other methods of choosing a local footstep goal, and they all worked decently well. Characters that are visible to each other can read each other's plans in order to predict their dynamic collision bounds at any given time. Visibility is determined by (1) having line-of-sight between the two characters, and (2) being within the character's visual field, modeled as a hemisphere centered around the character's forward-facing direction. This knowledge is analogous to the unspoken communication that occurs between real human pedestrians that makes human steering very robust. When a character re-plans, it does not try to avoid characters that it does not see, and therefore other characters, who are still executing old plans, may collide. The number of collisions can be drastically reduced by re-planning n steps in advance, before the previous plan is fully completed. This way there is always a 'buffer' of 2 or 3 steps that are guaranteed to be correct when a character predicts how to steer around another character. While deadlocks and collisions are still possible with this scheme, collisions are

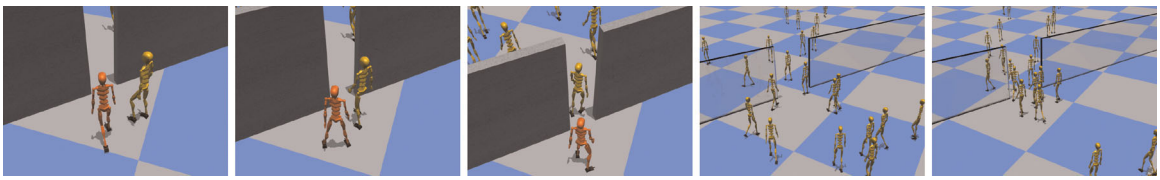


Figure 3. (Left) A character side-steps and yields to the other pedestrian, then precisely navigates through the narrow doorway. (Right) An egress simulation. Characters do not get stuck around the corners of the glass door.

very rare, and we have not yet encountered a deadlock in our experiments.

4.4. Choosing Exact Footstep Orientation

As described above, the planner maintains an interval of valid foot orientations for every step, constrained by the previous step's interval, as well as the trajectory of the current step. Once a sequence of footsteps has been planned, it is possible to choose exact footstep orientations. We constrain the interval of valid orientations once more using the *next* step's trajectory, now that this information is available. This computation relies on the same interval arithmetic described in Section 3. It is easy to see by contradiction that this process will not cause an invalid interval of orientations: if the interval becomes invalid during this postprocess, that would imply that no orientation of the current step could have produced a valid interval of the next step—but if this is true, that option would have already been pruned during planning and would not be encountered here. The exact orientation can be any value within this final interval; we found a good heuristic is to orient the foot as closely as possible to the orientation of the step's trajectory, with a special case for large turns.

5. RESULTS

For most results, characters are modeled with a center of mass 1 m above the ground, with a step length between 0.1 m and 1.0 m, step timing between 0.2 seconds and 0.8 seconds, and torso width of 60 cm.

Our short-horizon planner can solve challenging situations such as potential deadlocks in narrow spaces. Figure 3 depicts a challenging doorway situation. In many previous algorithms, characters would 'fight' at the doorway and may reach deadlock. In our method, the characters exhibit predictive cooperation, where one character steps aside. The doorway, 70 cm wide, is barely wide enough to fit a single pedestrian. In this tight situation, vector-based techniques would rely on collision prevention at the walls until the character eventually finds the door.

Our collision model allows tighter spacing in crowded conditions. An example is shown in Figure 3, where a group of characters squeeze through a glass door. With a single coarse collision radius, there would be many false-positive collisions. Instead, like real humans, these characters are comfortable placing their feet and shoulders close to others in the dense crowd.

Our planner works online, in real-time. Performance is shown in Table 1, measured on a Core 2 processor, using a single thread. Planning is fast is because of the scope of footsteps: a short horizon plan of 5–10 footsteps takes seconds to execute but only a few milliseconds to compute. The amortized cost of updating a character at 20 Hz is also shown in Table 1.

6. DISCUSSION AND FUTURE WORK

Footsteps are an appropriate form of control since they are the major contact point between a bipedal system and the external environment. By generating space–time sequences of footsteps, and by considering tighter dynamic collision bounds, our approach is able to control characters more precisely than existing crowd navigation techniques.

A 'stop' step is a specialized action in our planner. Being based on general planning, our technique can extend to use other specialized actions, such as running, jumping, even motion capture clips, as long as the action has well defined transitions, costs, and constraints. Existing steering techniques can also be emulated, for example, social forces models can be mapped to cost functions used by our planner.

There are some prominent aspects of bipedal locomotion which should be addressed in future work. Knee joints, ankle joints, muscles, angular momentum, and the center of pressure (pendulum pivot) shifting from heel-to-toe during a step—all of these affect the energy cost of real footsteps. We would also like to explore social and cognitive costs, where a character's objective may not necessarily be to minimize effort.

Table 1. Performance of our footstep planner for a character. The typical worst case plan generated up to 5000 nodes and expanded about 3000 nodes.

	Egress	2-way hall	700 boxes
	50 agents	200 agents	500 agents
Avg. # nodes generated	137	234	261
Avg. # nodes expanded	82	190	192
Planner performance	1.6 ms	4.4 ms	3 ms
Amortized cost 20 Hz	0.037 ms	0.1 ms	0.11 ms

Comp. Anim. Virtual Worlds 2011; **22**:151–158 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/cav

ACKNOWLEDGEMENTS

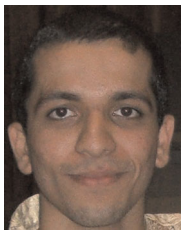
Authors thank Intel Corp. for their generous support through equipment and grants.

REFERENCES

1. van de Panne M. From footprints to animation. *Computer Graphics Forum* 1997; **16**(4): 211–223.
2. Chung S-K, Hahn JK. Animation of human walking in virtual environments. In *Computer Animation*, 1999; 4–15.
3. Coros S, Beaudoin P, Yin KK, van de Panne M. Synthesis of constrained walking skills. *ACM Transactions on Graphics* 2008; **27**(5): 1–9.
4. Wu C-C, Medina J, Zordan VB. Simple steps for simply stepping. In *ISVC (1)*, 2008; 97–106.
5. van Basten BJH, Peters PWAM, Egges A. *The Stepspace: Example-Based Footprint-Driven Motion Synthesis. In Computer Animation and Virtual Worlds, CASA 2010 Special Issue, Vol 21, Issue 3-4*, Chichester, UK: John Wiley and Sons Ltd., 2010.
6. Autodesk. *3ds Max*, 2010.
7. Helbing D, Molnár P. Social force model for pedestrian dynamics. *Physical Review E* 1995; **51**(5): 4282–4286.
8. Reynolds CW. Steering Behaviors for Autonomous Characters, in the proceedings of Game Developers Conference 1999 held in San Jose, California. Miller Freeman Game Group, San Francisco, California. 1999. 763–782.
9. Kapadia M, Singh S, Hewlett W, Faloutsos P. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 2009. 215–223.
10. Singh S, Kapadia M, Hewlett W, Reinman G, Faloutsos P. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 2011.
11. Pelechano N, Allbeck JM, Badler NI. Controlling individual agents in high-density crowd simulation. In *SC*A, Eurographics Association, 2007; 99–108.
12. Gayle R, Sud A, Andersen E, Guy SJ, Lin MC, Manocha D. Interactive navigation of heterogeneous agents using adaptive roadmaps. *IEEE Transactions on Visualization and Computer Graphics* 2009; **15**(1): 34–48.
13. Boulic R. Relaxed steering towards oriented region goals. In *MIG'08*, 2008; 176–187.
14. van den Berg JP, Lin MC, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA*, 2008; 1928–1935.
15. Lamarche F, Donikian S. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum* 2004, **23**(3): 509–518.
16. Paris S, Pettré J, Donikian S. Pedestrian reactive navigation for crowd simulation: a predictive approach. In *Eurographics 2007*, Vol. 26, 2007; 665–674. http://www.morganclaypool.com/doi/abs/10.2200/S00123ED1V01Y200808CGR008
17. Feurtey F. Simulating the collision avoidance behavior of pedestrians. *Master's Thesis*, The University of Tokyo, School of Engineering, 2000.
18. Badler N. *Virtual Crowds: Methods, Simulation, and Control*. Morgan and Claypool Publishers, 2008. DOI: 10.2200/S00123ED1V01Y200808CGR008 http://www.morganclaypool.com/doi/abs/10.2200/S00123ED1V01Y200808CGR008
19. Paris S, Donikian S. Activity-driven populace: a cognitive approach to crowd simulation. *IEEE Computer Graphics and Applications* 2009; **29**(4): 34–43.
20. Musse SR, Thalmann D. A model of human crowd behavior. In *Proceedings of the CAS'97*, Springer Verlag, Wien, 1997; 39–51.
21. Shao W, Terzopoulos D. Autonomous pedestrians. In *SCA*, 2005; 19–28.
22. van Basten BJH, Egges A. Path abstraction for combined navigation and animation. *MIG'09*, 5884/2009, 2009; 182–193.
23. Lau M, Kuffner JJ. Precomputed search trees: planning for interactive goal-driven animation. In *SCA*, September 2006; 299–308.
24. Kuffner,J.J., Jr., Nishiwaki K, Kagami S, Inaba M, Inoue H. Footstep planning among obstacles for biped robots. In *IEEE Intelligent Robots and Systems (IEEE/RSJ)*, Vol. 1, 2001; 500–505.
25. Nishiwaki Kh, Sugihara T, Kagami S, Inaba My, Inoue S. Online mixture and connection of basic motions for humanoid walking control by footprint specification. In *ICRA*, Vol. 4, 2001; 4110–4115.
26. Kuffner J, Nishiwaki K, Kagami S, Kuniyoshi Y, Inaba M, Inoue H. Online footstep planning for humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, September 2003.
27. Li T-Y, Chen P-F, Huang P-Z. Motion planning for humanoid walking in a layered environment. In *ICRA*, Vol. 3, 2003; 3421–3427.
28. Chestnutt J, Lau M, Cheung KM, Kuffner J, Hodgins JK, Kanade T. Footstep planning for the honda asimo humanoid. In *ICRA*, April 2005.
29. Choi MG, Lee J, Shin SY. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics* 2003; **22**(2): 182–203.
30. Zhang L, Pan J, Manocha D. Motion planning and synthesis of human-like characters in constrained environments. *MIG'09*, 5884/2009, 2009; 138–145.
31. Wu J-C, Popović Z. Terrain-adaptive bipedal locomotion control. *ACM Transactions on Graphics* 2010; **29**(4): 72:1–72:10.
32. Kuo AD. The six determinants of gait and the inverted pendulum analogy: a dynamic walking perspective. *Human Movement Science* 2007; **26**(4): 617–656.

AUTHORS' BIOGRAPHIES

Shawn Singh is currently working on his Ph.D. at the University of California, Los Angeles. He received his M.S. in computer science from the University of Southern California. His research includes real-time photon mapping, novel forms of computation, and robust virtual pedestrian steering behaviors.



Mubbasir Kapadia received his B.E. in Computer Engineering in 2007 from University of Mumbai, India. He is currently working on his M.S. at the University of California, Los Angeles. His current research is applying egocentric approaches to pedestrian simulation and the evaluation of agent steering behaviors.

Glenn Reinman is an assistant professor in the department of computer science at UCLA. He received his B.S. from MIT in 1996 and his PhD and M.S. in Computer Science from UCSD in 2001. His main area of research is microprocessor architecture, and he directs the MARS lab at UCLA.

Petros Faloutsos is an assistant professor at the Department of Computer Science at the University of California at Los Angeles. He received his PhD degree (2002) and his MSc degree in Computer Science from the University of Toronto, Canada and his BEng degree in Electrical Engineering from the National Technical University of Athens, Greece. Professor Faloutsos is the founder and the director of the graphics lab at the Department of Computer Science at UCLA. The lab, called MAGIX (Modeling Animation and GrafIX), performs state of the art research in all aspects of graphics, focusing on virtual actors, virtual reality, physics-based animation and motor control. Professor Faloutsos is also interested in computer networks and he has co-authored a highly cited paper on the topology of the Internet. Professor Faloutsos is a member of the Editorial Board of the Journal Of The Visual Computer and has served as a Progam Co-Chair for the 2005 ACM SIGGRAPH/Eurographics Synmposium on Computer Animation. He is a member of the ACM and the Technical Chamber of Greece.

Comp. Anim. Virtual Worlds 2011; **22**:151–158 © 2011 John Wiley & Sons, Ltd.
DOI: 10.1002/cav

Multi-Domain Real-time Planning in Dynamic Environments

Mubbasir Kapadia*[1], Alejandro Beacco†[2], Francisco Garcia‡[3], Vivek Reddy§[1], Nuria Pelechano¶[2], and Norman I. Badler‖[1]

[1]University of Pennsylvania
[2]Universitat Politècnica de Catalunya
[3]University of Massachusetts Amherst

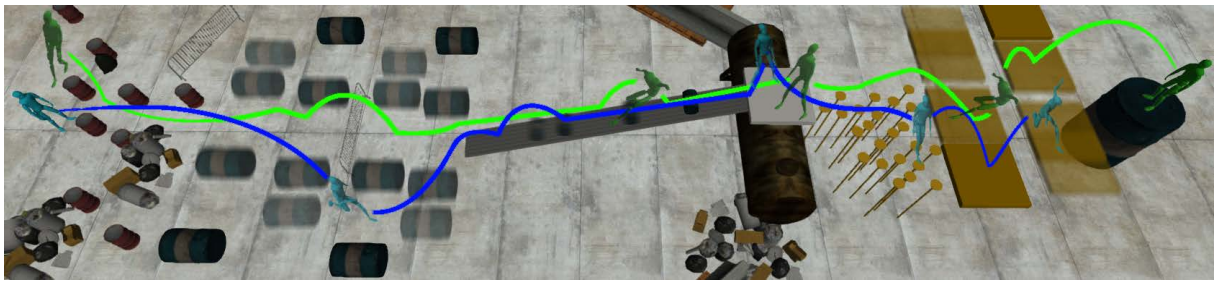


Figure 1: *Two agents navigating with space-time precision through a complex dynamic environment.*

Abstract

This paper presents a real-time planning framework for multi-character navigation that enables the use of multiple heterogeneous problem domains of differing complexities for navigation in large, complex, dynamic virtual environments. The original navigation problem is decomposed into a set of smaller problems that are distributed across planning tasks working in these different domains. An anytime dynamic planner is used to efficiently compute and repair plans for each of these tasks, while using plans in one domain to focus and accelerate searches in more complex domains. We demonstrate the benefits of our framework by solving many challenging multi-agent scenarios in complex dynamic environments requiring space-time precision and explicit coordination between interacting agents, by accounting for dynamic information at all stages of the decision-making process.

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

Keywords: real-time navigation, space-time planning, multiple problem domains, crowd simulation

1 Introduction

The next generation of interactive applications requires high fidelity navigation of interacting autonomous agents in non-deterministic, dynamic virtual worlds. The environment and agents are constantly affected by unpredictable forces (e.g., human input), making it impossible to accurately extrapolate the future world state to make optimal decisions. These complex domains require robust navigation algorithms that can handle partial and imperfect knowledge, while still making decisions which satisfy space-time constraints.

Different situations require different granularity of control. An open environment with no agents and static obstacles requires only coarse-grained control while cluttered dynamic environments require fine-grained character control with careful planned decisions that have spatial and temporal precision. Some situations (e.g., potential deadlocks) may require explicit coordination between multiple agents.

The problem domain of interacting autonomous agents in dynamic environments is extremely high-dimensional and continuous, with infinite ways to interact with objects and other agents. Having a rich action set, and a system that makes intelligent action choices, facilitates robust, intelligent virtual characters, at the expense of interactivity and scalability. Greatly simplifying the problem domain yields interactive virtual worlds with hundreds and thousands of agents that exhibit simple behavior. The ultimate, far-reaching goal is still a considerable challenge: a real-time system for autonomous character control that can handle many characters, without compromising control fidelity.

Previous work simulates crowds by decoupling global navigation [Sung et al. 2005; Kallmann 2010] and local collision avoidance [Pelechano et al. 2008], or demonstrates space-time planning for global navigation for a single character [Levine et al. 2011], while meeting real-time constraints. These approaches provide a tradeoff between number of agents, control fidelity, and environment complexity. To our knowledge, no proposed technique efficiently accounts for the dynamic nature of the environment at all levels of the decision-making process.

This paper proposes a real-time planning framework for multi-character navigation that uses multiple heterogeneous problem domains of differing complexities for navigation in large, complex, dynamic virtual environments. We define a set of problem domains (spaces of decision-making) which differ in the complexity of their state representations and the fidelity of agent control. These range from a static navigation mesh domain which only accounts for static objects in the environment, to a space-time domain that factors in

*mubbasir@seas.upenn.edu
†abeacco@lsi.upc.edu
‡fmaxgarcia@gmail.com
§vivreddy@seas.upenn.edu
¶npelechano@lsi.upc.edu
‖badler@seas.upenn.edu

SCA 2013, July 19 – 21, 2013, Anaheim, California.

dynamic obstacles and other agents at much finer resolution. These domains provide different trade-offs in performance and fidelity of control, requiring a framework that efficiently works in multiple domains by using plans in one domain to focus and accelerate searches in more complex domains.

A global planning problem (start and goal configuration) is dynamically decomposed into a set of smaller problem instances across different domains, where an anytime dynamic planner is used to efficiently compute and repair plans for each of these problems. Planning tasks are connected by either using the computed path from one domain to define a *tunnel* to focus searches, or using successive waypoints along the path as start and goal for a planning task in another domain to reduce the search depth, thereby accelerating searches in more complex domains. Using our framework, we demonstrate real-time character navigation for multiple agents in large-scale, complex, dynamic environments, with precise control, and little computational overhead.

2 Related Work

There is extensive research in multi-agent simulations with many proposed techniques that differ in domain complexity and control fidelity. Global navigation approaches [Sung et al. 2005; Sud et al. 2007; van den Berg et al. 2008b; Kallmann 2010] precompute a roadmap of the global environment which is used for making efficient navigation queries, but generally regard the environment to be static. Crowd approaches [Pelechano et al. 2008; Thalmann 2008] compromise on control fidelity in an effort to efficiently simulate a large number of agents in real-time. Continuum-based methods [Treuille et al. 2006] model macroscopic crowd flow, while agent-based approaches [Reynolds 1987; Lamarche and Donikian 2004; Loscos et al. 2003] model collision avoidance for goal-directed agents using rules. Predictive approaches [van den Berg et al. 2008a; Paris et al. 2007; Kapadia et al. 2009] approximate the trajectories of neighboring agents in choosing collision-free velocities, and the work in [Singh et al. 2011a] proposes a hybrid technique that combines reactive rules, predictions, and planning.

Planning based control of autonomous agents has demonstrated control of single agents with large action spaces [Choi et al. 2003; Fraichard 1999; Shapiro et al. 2007]. In an effort to scale to a large number of agents, meet real-time constraints, and handle dynamic environments, a large variety of methods [Pettré et al. 2008] have been proposed. The complexity of the domain is made simpler [Lau and Kuffner 2005; Lo and Zwicker 2008] to reduce the branching factor of the search, or the horizon of the search is limited to a fixed depth [Singh et al. 2011b; Choi et al. 2011]. Anytime planners [Likhachev et al. 2003; van den Berg et al. 2006] tradeoff optimality to satisfy strict time constraints, and have been successfully demonstrated for motion planning for a single character [Safonova and Hodgins 2007]. Randomized planners [Hsu et al. 2002; Shapiro et al. 2007] expand nodes in the search graph using sampling methods, greatly reducing search efforts to make it a feasible solution in high-dimensional, continuous domains. The work in [Hoff et al. 2000] exploits the use of graphics hardware to enable interactive motion planning in dynamic environments.

Hierarchical Planning. Hierarchical planners [Botea et al. 2004; Bulitko et al. 2007; Holte et al. 1996] reduce the problem complexity by precomputing abstractions in the state space, which can be used to speed up plan efforts. Given a discrete environment representation, neighboring states are first clustered together to precompute abstractions for high-level graphs. Different algorithms are proposed [Kring et al. 2010] which plan paths hierarchically by planning at the top level first, then recursively planning more detailed paths in the lower levels, using different methods [Lacaze 2002; Sturtevant and Geisberger 2010] to communicate information across hierarchies. These include using the plans in high-level graphs to compute heuristics for accelerating searches in low-level graphs [Holte et al. 2005], using the waypoints as intermediate goals, or using the high-level path to define a tunnel [Gochev et al. 2011] to focus the search in the low-level graph. The work in [Arikan and Forsyth 2002] demonstrates the use of randomized search in a hierarchy of motion graphs for interactive motion synthesis.

Comparison to Prior Work. Our work builds on top of excellent recent contributions [Levine et al. 2011; Lopez et al. 2012] showcasing the use of space-time planning for global navigation in dynamic environments, for a single agent. Levine *et al.* [2011] uses parameterized locomotion controllers to efficiently reduce the branching factor of the search and assumes that object motion have known trajectories, thus mitigating the need for replanning. Lopez *et al.* [2012] introduces a dynamic environment representation which is computed by deducing the evolution of the environment topology over time, thus enabling space-time collision avoidance with no prior knowledge of how the world changes. In contrast, we use multiple heterogeneous domains of control, and present a planning-based control scheme that reuses plan efforts across domains to demonstrate real-time, multi-character navigation, in constantly changing dynamic environments. Instead of automatically computing abstractions from a given representation, we develop a set of heterogeneous domains with different state and action representations that provide trade-offs in control fidelity and computational performance, and investigate different methods of communicating between domains to meet our application needs.

3 Overview

The problem domain of a planner determines its effectiveness in solving a particular problem instance. A complex domain that accounts for all environment factors such as dynamic environments and other agents, and has a large branching factor in its action space can solve more difficult problems, but at a larger cost overhead. A simpler domain definition provides the benefit of computational efficiency while compromising on control fidelity. Our framework enables the use of multiple heterogeneous domains of control, providing a balance between control fidelity and computational efficiency, without compromising either.

A global problem instance P_0 is dynamically decomposed into a set of smaller problem instances $\{P^{'}\}$ across different planning domains $\{\Sigma_i\}$. Section 4 describes the different domains, and Section 5 describes the problem decomposition across domains. Each problem instance $P^{'}$ is assigned a planning task $T(P^{'})$, and an anytime dynamic planner (Section 5.1) is used to efficiently compute and repair plans for each of these tasks, while using plans in one domain to focus and accelerate searches in more complex domains. Plan efforts across domains are reused in two ways. The computed path from one domain can be used to define a *tunnel* which focuses the search, reducing its effective branching factor. Each pair of successive waypoints along a path can also be used as start,goal pairs for a planning task in another domain, thus reducing the search depth. Both these methods are used to focus and accelerate searches in more complex domains, providing real-time efficiency without compromising on control fidelity. Section 6 describes the relationships between domains.

4 Planning Domains

A problem domain is defined as $\Sigma = \langle \mathbb{S}, \mathbb{A}, \mathrm{c}(s, s'), \mathrm{h}(s, s_{goal}) \rangle$, where the state space $\mathbb{S} = \{\mathbb{S}_{self} \times \mathbb{S}_{env} \times \mathbb{S}_{agents}\}$ includes the

internal state of the agent $\mathbb{S}_{self}$, the representation of the environment $\mathbb{S}_{env}$, and other agents $\mathbb{S}_{agents}$. $\mathbb{S}_{self}$ may be modeled as a simple particle with a collision radius. $\mathbb{S}_{env}$ can be an environment triangulation with only static information or a uniform grid representation with dynamic obstacles. $\mathbb{S}_{agents}$ is defined by the vicinity within which neighboring agents are considered. Imminent threats may be considered individually or just represented as a density distribution at far-away distances. The action space $\mathbb{A}$ defines the set of all possible successors $\texttt{succ}(s)$ and predecessors $\texttt{pred}(s)$ at each state s, as shown in Equation 1. Here, $\delta(s, i)$ describes the i^{th} transition, and $\Phi(s, s')$ is used to check if the transition from s to s' is possible. The cost function $\texttt{c}(s, s')$ defines the cost of transition from s to s'. The heuristic function $\texttt{h}(s, s_{goal})$ defines the estimate cost of reaching a goal state.

$$\texttt{succ}(s) = \{s + \delta(s, i) | \Phi(s, s') = \texttt{TRUE} \ \ \forall i\} \tag{1}$$

A problem definition $P = \langle \Sigma, s_{start}, s_{goal} \rangle$ describes the initial configuration of the agent, the environment and other agents, along with the desired goal configuration in a particular domain. Given a problem definition P for domain Σ, a planner searches for a sequence of transitions to generate a plan $\Pi(\Sigma, s_{start}, s_{goal}) = \{s_i | s_i \in \mathbb{S}(\Sigma)\}$ that takes an agent from s_{start} to s_{goal}.

4.1 Multiple Domains of Control

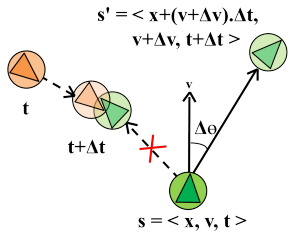
We define 4 domains which provide a nice balance between global static navigation and fine-grained space-time control of agents in dynamic environments. Figure 2 illustrates the different domain representations for a given environment.

Static Navigation Mesh Domain Σ_1. This domain uses a triangulated representation of free space and only considers static immovable geometry. Dynamic obstacles and agents are not considered in this domain. The agent is modeled as a point mass, and valid transitions are between connected free spaces, represented as polygons. The cost function is the straight line distance between the center points of two free spaces. Additional connections are also precomputed (or manually annotated) to represent transitions such as jumping with a higher cost definition. The heuristic function is the Euclidean distance between a state and the goal. Searching for an optimal solution in this domain is very efficient and quickly provides a global path for the agent to navigate. We use Recast [Mononen 2009] to precompute the navigation mesh for the static geometry in the environment.

Dynamic Navigation Mesh Domain Σ_2. This also uses triangulations to represent free spaces and coarsely accounts for dynamic properties of the environment to make a more informed decision at the global planning layer. The work in [van Toll et al. 2012] embeds population density information in environment triangulations to account for the movement of agents at the global planning layer. We adopt a similar method by defining a time-varying density field $\phi(t)$ which stores the density of moveable objects (agents and obstacles) for each polygon in the triangulation at some point of time t. $\phi(t_0)$ represents the density of agents and obstacles currently present in the polygon. The presence of objects and agents in polygons at future timesteps can be estimated by querying their plans (if available). The space-time positions of deterministic objects can be accurately queried while the future positions of agents can be approximated based on their current computed paths, assuming that they travel with constant speed along the path without deviation. $\phi(t)$ contributes to the cost of selecting a waypoint in Σ_2 during planning. The resolution of the triangulation may be kept finer than Σ_1 to increase the resolution of the dynamic information in this domain. Hence, a set of global waypoints are chosen in this domain which avoids crowded areas or other high cost regions.

Grid Domain Σ_3. The grid domain discretizes the environment into grid cells where a valid transition is considered between adjacent cells that are free (diagonal movement is allowed). An agent is modeled as a point with a radius (orientation and agent speed is not considered in this domain). This domain only accounts for the current position of dynamic obstacles and agents, and cannot predict collisions in space-time. The cost and heuristic are distance functions that measure the Eucledian distance between grid cells.

Space-Time Domain Σ_4. This domain models the current state of an agent as a space-time position with a current velocity $(\mathbf{x}, \mathbf{v}, t)$. The figure alongside illustrates the schematic illustration of the state and action space in Σ_4, showing a valid transition, and an invalid transition due to a space-time collision with a neighboring agent. The transition function $\delta(s, i)$ for Σ_4 is defined below:



$$\delta(s, i) = \{\Delta \mathbf{v}_i \cdot \Delta t | \Delta \mathbf{v}_i = (\Delta v_i \cdot \sin \Delta\theta_i, \Delta v_i \cdot \cos \Delta\theta_i) \forall i\}$$

where $\Delta v = \{0, \pm a\}$ is the possible speed changes and $\Delta\theta = \{0, \pm\frac{\pi}{8}, \pm\frac{\pi}{4}, \pm\frac{\pi}{2}\}$ is the possible orientation changes the agent can make from its current state. For example, $\Delta\mathbf{v} = a, \Delta\theta = \frac{\pi}{8}$ produces a transition where the agent accelerates by a for the duration of the timestep and rotates by $\frac{\pi}{8}$. The bounds of $\Delta\theta$ are limited between $\{-\frac{\pi}{2}, \frac{\pi}{2}\}$ to limit the maximum rate of turning. Transitions are also bound so that the speed and acceleration of an agent cannot exceed a given threshold. Jumps are additionally modeled as a high cost transition between two space-time points such that the region between them may be occupied or untraversable for that time interval. Inspite of the coarse discretization of $\Delta\theta$, the branching factor of this domain is much higher, providing greater degree of control fidelity with added computational overhead.

Σ_4 accounts for all obstacles (static and dynamic) and other agents. The traversability of a grid cell is queried in space-time by checking to see if moveable obstacles and agents occupy that cell at that particular point of time, by using their published paths. For space-time collision checks, only agents and obstacles that are within a certain region from the agent, defined using a foveal angle intersection, are considered. The cost and heuristic definitions have a great impact on the performance in Σ_4. We use an energy based cost formulation that penalizes change in velocity with a non-zero cost for zero velocity. Jump transitions incur a higher cost. The heuristic function penalizes states that are far away from s_{goal} in both space and time. This is achieved using a weighted combination of a distance metric and a penalty for a deviation of the current speed from the speed estimate required to reach s_{goal}.

The domains described here are *not* a comprehensive set and only serve to showcase the ability of our framework to use multiple heterogeneous domains of control in order to solve difficult problem instances at a fraction of the computation cost. Our framework can be easily extended to use other domain definitions (e.g., a footstep domain), as described in Section 7.4.

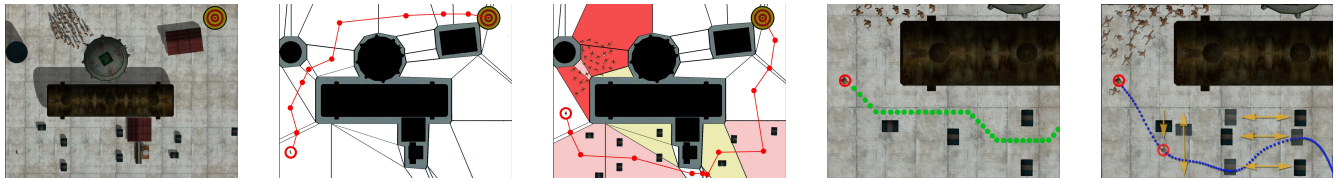


Figure 2: *(a) Problem definition with initial configuration of agent and environment. (b) Global plan in static navigation mesh domain Σ_1 accounting for only static geometry. (c) Global plan in dynamic navigation mesh domain Σ_2 accounting for cumulative effect of dynamic objects. (d) Grid plan in Σ_3. (e) Space-time plan in Σ_4 that avoids dynamic threats and other agents.*

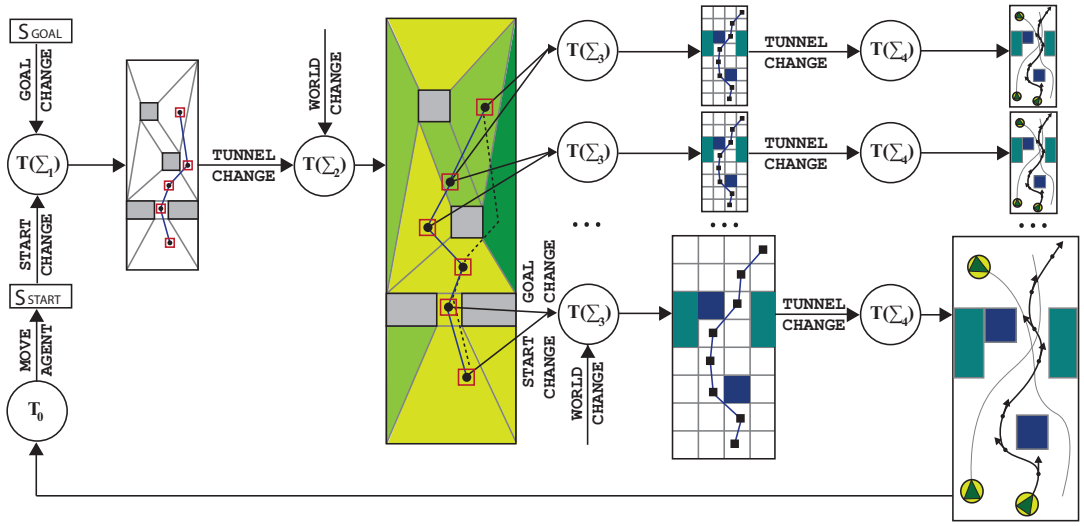


Figure 3: *Expanded illustration of domain relationship shown in Figure 4(b). A global problem instance (start and goal state) is decomposed into a set of smaller problem instances across multiple planning domains. Planning tasks $T(\Sigma)$ are assigned to each of these problems and scheduled using a dynamic priority scheme based on events from the environment and other tasks.*

5 Problem Decomposition and Multi-Domain Planning

Figure 4(a) illustrates the use of tunnels to connect each of the 4 domains, ensuring that a complete path from the agents initial position to its global target is computed at all levels. Figure 4(b) shows how Σ_2 and Σ_3 are connected by using successive waypoints in $\Pi(\Sigma_2)$ as start and goal for independent planning tasks in Σ_3. This relation between Σ_2 and Σ_3 allows finer-resolution plans being computed between waypoints in an independent fashion. Limiting Σ_3 (and Σ_4) to plan between waypoints instead of the global problem instance ensures that the search horizon in these domains is never too large, and that fine-grained space-time trajectories to the initial waypoints are computed quickly. However, completeness and optimality guarantees are relaxed as Σ_3, Σ_4 never compute a single path to the global target.

Figure 3 illustrates the different events that are sent between planning tasks to trigger plan refinement and updates for the domain relationship in Figure 4(b). Σ_1 is first used to compute a path from s_{start} to s_{goal}, ignoring dynamic obstacles and other agents. $\Pi(\Sigma_1)$ is used to accelerate computations in Σ_2, which refines the global path to factor in the distribution of dynamic objects in the environment. Depending on the relationship between Σ_2 and Σ_3, a single planning task or multiple independent planning tasks are

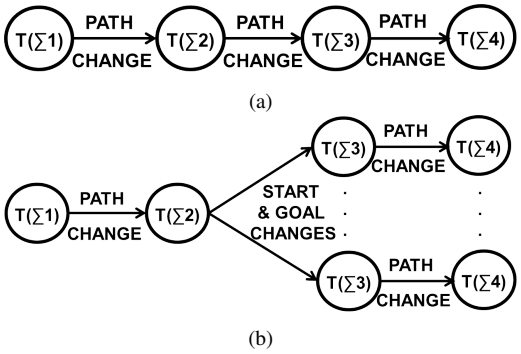


Figure 4: *Relationship between domains. (a) Use of tunnels to connect each of the 4 domains. (b) Use of successive waypoints in $\Pi(\Sigma_2)$ as start, goal pairs to instantiate multiple planning tasks in Σ_3 and Σ_4.*

used in Σ_3. Finally, the plan(s) of $T(\Sigma_3)$ are used to accelerate searches in Σ_4.

Changes in s_{start} and s_{goal} trigger plan updates in $T(\Sigma_1)$, which are propagated through the task dependency chain. $T(\Sigma_2)$ monitors

plan changes in $T(\Sigma_1)$ as well as the cumulative effect of changes in the environment to refine its path. Each $T(\Sigma_3)$ instance monitors changes in the waypoints along $\Pi(\Sigma_2)$ to repair its solution, as well as nearby changes in obstacle and agent position. Finally, $T(\Sigma_4)$ monitors plan changes in $T(\Sigma_3)$ (which it depends on) and repairs its solution to compute a space-time trajectory that avoids collisions with static and dynamic obstacles, as well as other agents.

Events are triggered (outgoing edges) and monitored (incoming edges) by tasks, creating a cyclic dependency between tasks, with $\mathtt{T}_0$ (agent execution) monitoring changes in the plan produced by the particular $T(\Sigma_4)$, which monitors the agents most imminent global waypoint. Tasks that directly affect the agent's next decision, and tasks with currently invalid or sub-optimal solutions are given higher priority. Given the maximum amount of time to deliberate t_{max}, the agent pops one or more tasks that have highest priority and divides the deliberation time across tasks (most imminent tasks are allocated more time). Task priorities constantly change based on events triggered by the environment and other tasks.

5.1 Planning Tasks

A task $T(P)$ is a planner which is responsible for generating and maintaining a valid (and ideally optimal) solution for a particular problem definition $P = \langle \Sigma, s_{start}, s_{goal} \rangle$ where s_{start}, s_{goal}, and the search graph may be constantly changing. There are 4 types of tasks, each of which solves a particular problem in the domains described in Section 4. An additional task $\mathtt{T}_0$ is responsible for moving the agent along the path, while enforcing steering and collision constraints.

Planning tasks constantly receive events from the environment and other tasks, which render the current plan invalid, forcing it to constantly update, refine, and repair its existing plan. For this purpose, we use the Anytime Dynamic A* planner [Likhachev et al. 2005] which combines the properties of incremental planners such as D* Lite [Koenig and Likhachev 2002] and anytime algorithms such as ARA* [Likhachev et al. 2003] to provide an algorithm which efficiently repairs its solutions to accommodate world changes and agent movement, while providing solution guarantees under strict time constraints. It performs repeated backward searches (from goal to start), reusing previous search efforts to iteratively produce solutions with improved bounds on optimality, like ARA*. This is done using an inflation factor ϵ which effectively weighs the contribution of the heuristic value in estimation of node costs, thus focusing the search towards the goal, expanding fewer nodes to produce ϵ sub-optimal solutions [Pearl 1984]. We provide an overview of the algorithmic details of the planning task in the supplementary document and refer the readers to a comprehensive review of the AD* algorithm here [Likhachev et al. 2005].

Plan Repair vs. Planning from Scratch. Note that there are often instances during the simulation when the start and goal changes of planning tasks change or when plans are invalidated due to obstacle movement. Plans are always recomputed for goal changes. AD* performs a backward search which allows it to efficiently update the search graph to accommodate agent movement along the path. For significant start changes or when the plan is invalidated due to obstacle movement, the choice between replanning or repairing a plan is a heuristic decision with tradeoffs in performance. Plan repair may expand lesser nodes in the current iteration but bloat the number of nodes visited, thus impacting performance in subsequent plan iterations. It is not uncommon to plan from scratch during the simulation. By resetting the inflation factor to a high value, we can quickly compute a valid sub-optimal plan while meeting time constraints and refine it in successive plan iterations.

5.2 Events and Task Priorities

Events are triggered and monitored by planning tasks in different domains, as illustrated in Figure 3. Changes in start and goal, or environment changes may potentially invalidate current plans, requiring plan refinement. Tasks that use tunnels to accelerate searches in more complex domains, monitor plan changes in other tasks. Finally, tasks observe the optimality status of their own plans to determine their task priority. The supplementary document describes the different events in more detail.

The priority of a task $p(\mathtt{T}_a)$ determines the tasks that are picked to be executed at every time step, with tasks having smallest $p(\mathtt{T}_a)$ chosen for execution ($p(\mathtt{T}_a)$ is short for $p(T(\Sigma_a))$). Task $\mathtt{T}_0$, which handles agent movement always has a priority of 1. Priority of other tasks is calculated as follows:

$$p(\mathtt{T}_a) = \begin{cases} 1 \text{ if } \mathtt{T}_a = \mathtt{T}_0 \\ \mu(\mathtt{T}_a, \mathtt{T}_0) \cdot \Omega(\mathtt{T}_a) \text{ else} \end{cases} \quad (2)$$

where $\mu(\mathtt{T}_a, \mathtt{T}_0)$ is the number of edge traversals required to reach $\mathtt{T}_0$ from $\mathtt{T}_a$ in the task dependency chain (Figure 3). $\Omega(\mathtt{T}_a)$ denotes the current state of the plan of $\mathtt{T}_a$ and is defined as follows:

$$\Omega(\mathtt{T}_a) = \begin{cases} 1 \text{ if } \texttt{SOLUTION\_INVALID} \\ \epsilon \text{ if plan inflation factor, } \epsilon > 1 \\ \infty \text{ if plan inflation factor, } \epsilon = 1 \end{cases} \quad (3)$$

where ϵ is the inflation factor used to determine the optimality bounds of the current plan for that task. The agent pops one or more tasks that have highest priority and divides the deliberation time available across tasks, with execution-critical tasks receiving more time. Tasks that have the same priority are ordered based on task dependency. Hence, $\mathtt{T}_0$ is always executed at the end of every update after all planning tasks have completed.

The overall framework enforces strict time constraints. Given an allocated time to deliberate for each agent (computed based on desired frame rate and number of agents), the time resource is distributed based on task priority. In the remote event that there is no action to execute, the agent remains stationary (no impact on frame-rate) for a few frames (fractions of a second) until a valid plan is computed.

6 Relationship between Domains

The complexity of the planning problem increases exponentially with increase in dimensionality of the search space – making the use of high-dimensional domains nearly prohibitive for real-time applications. In order to make this problem tractable, planning tasks must efficiently use plans in one domain to focus and accelerate searches in more complex domains. Section 6.1 describes a method for mapping a state from a low-dimensional domain to one or more states in a higher dimensional domain. Sections 6.2 and 6.3 describe two ways in which plans in one domain can be used to focus and accelerate searches in another domain.

6.1 Domain Mapping

We define a $1:n$ function $\lambda(s, \Sigma, \Sigma^{'})$ that allows us to maps states in $\mathbb{S}(\Sigma)$ to one or more equivalent states in $\mathbb{S}(\Sigma^{'})$.

$$\lambda(s, \Sigma, \Sigma^{'}) : s \rightarrow \{s' | s' \in \mathbb{S}(\Sigma^{'}) \wedge s \equiv s'\} \quad (4)$$

The mapping functions are defined specifically for each domain pair. For example, $\lambda(s, \Sigma_1, \Sigma_2)$ maps a polygon $s \in \mathbb{S}(\Sigma_1)$ to one or more polygons $\{s' | s' \in \mathbb{S}(\Sigma_2)\}$ such that s' is spatially contained in s. If the same triangulation is used for both Σ_1 and Σ_2, then there exists a one-to-one mapping between states. Similarly, $\lambda(s, \Sigma_2, \Sigma_3)$ maps a polygon $s \in \mathbb{S}(\Sigma_2)$ to multiple grid cells $\{s' | s' \in \mathbb{S}(\Sigma_3)\}$ such that s' is spatially contained in s. $\lambda(s, \Sigma_3, \Sigma_4)$ is defined as follows:

$$\lambda(s, \Sigma_3, \Sigma_4) : (\mathbf{x}) \rightarrow \{(\mathbf{x} + W(\Delta \mathbf{x}), t + W(\Delta t))\} \quad (5)$$

where $W(\Delta)$ is a window function in the range $[-\Delta, +\Delta]$. The choice of t is important in mapping Σ_3 to Σ_4. Since we use λ to effectively map a plan $\Pi(\Sigma_3, s_{start}, s_{goal})$ in Σ_3 to a tunnel in Σ_4, we can exploit the path and the temporal constraints of s_{start} and s_{goal} to define t for all states along the path. We do this by calculating the total path length and the time to reach s_{goal}. This allows us to compute the approximate time of reaching a state along the path, assuming the agent is traveling at a constant speed along the path.

6.2 Mapping Successive Waypoints to Independent Planning Tasks.

Successive waypoints along the plan from one domain can be used as start and goal for a planning task in another domain. This effectively decomposes a planning problem into multiple independent planning tasks, each with a significantly smaller search depth.

Consider a path $\Pi(\Sigma_2) = \{s_i | s_i \in \mathbb{S}(\Sigma_2), \forall i \in (0, n)\}$ of length n. For each successive waypoint pair (s_i, s_{i+1}), we define a planning problem $P_i = \langle \Sigma_3, s_{start}, s_{goal} \rangle$ such that $s_{start} = \lambda(s_i, \Sigma_2, \Sigma_3)$ and $s_{goal} = \lambda(s_{i+1}, \Sigma_2, \Sigma_3)$. Even though λ may return multiple equivalent states, we choose only one candidate state. For each problem definition P_i, we instantiate an independent planning task $T(P_i)$which computes and maintains path from s_i to s_{i+1} in Σ_3. Figure 4 illustrates this connection between Σ_2 and Σ_3.

6.3 Tunnels

The work in [Gochev et al. 2011] observes that a plan in a low dimensional problem domain can often be exploited to greatly accelerate high-dimensional complex planning problems by focusing searches in the neighborhood of the low dimensional plan. They introduce the concept of a tunnel $\tau(\Sigma_{hd}, \Pi(\Sigma_{ld}), t_w)$ as a sub graph in the high dimensional space Σ_{hd} such that the distance of all states in the tunnel from the low dimensional plan $\Pi(\Sigma_{ld})$ is less than the tunnel width t_w. Based on their work, we use plans from one domain in order to accelerate searches in more complex domains with much larger action spaces. A planner is input a low dimensional plan $\Pi(\Sigma_{ld})$ which is used to focus state transitions in the sub graph defined by the tunnel $\tau(\Sigma_{hd}, \Pi(\Sigma_{ld}), t_w)$.

To check if a state s lies within a tunnel $\tau(\Sigma_{hd}, \Pi(\Sigma_{ld}), t_w)$ without precomputing the tunnel itself, the low dimensional plan $\Pi(\Sigma_{ld})$ is first converted to a high dimensional plan $\Pi^{'}(\Sigma_{hd}, s_{start}, s_{goal})$ by mapping all states of Π to their corresponding states in $\Pi^{'}$, using the mapping function $\lambda(s, \Sigma_{ld}, \Sigma_{hd})$ as defined in Equation 4. Note that the resulting plan $\Pi^{'}$ may have multiple possible trajectories from s_{start} to s_{goal} due to the $1 : n$ mapping of λ. Next, we define a distance measure $\mathbf{d}(s, \Pi(\Sigma))$ which computes the distance of s from the path $\Pi(\Sigma)$. During a planning iteration, a state is generated if and only if $\mathbf{d}(s, \Pi(\Sigma_{hd})) \leq t_w$. This is achieved by redefining the $\texttt{succ}(s)$ and $\texttt{pred}(s)$ to only consider states that lie in the tunnel. Furthermore, node expansion can be prioritized to states that are closer to the path by modifying the heuristic function as shown in below.

$$h_t(s, s_{start}) = h(s, s_{start}) + |\mathbf{d}(s, \Pi(\Sigma))| \quad (6)$$

Note that the heuristic $h_t(s, s_{start})$ is an estimate of the distance from s to s_{start} since we use a backward search from s_{goal} to s_{start} to accomodate start movement. For spatial domains Σ_1, Σ_2, and Σ_3, $\mathbf{d}(s, \Pi(\Sigma))$ is the perpendicular distance between s and the line segment connecting the two nearest states in $\Pi(\Sigma)$. $\mathbf{d}(s, \Pi(\Sigma_4))$ will return a two-tuple value for spatial distance as well as temporal distance.

TunnelChangeUpdate. When the tunnel changes, previously visited nodes that are no longer within the new tunnel are assigned an infinite cost and the changes are propagated to their successors. Also, their heuristic values are updated to reflect the new tunnel distance using Equation 6, which re-prioritizes node expansion to nodes that are closer to the new path. The tunnel width $t_w$is inversely proportional to the inflation factor ϵ. Thus, a high ϵ focuses the search within a narrow tunnel, which is iteratively expanded when ϵ is reduced to increase the breadth of the search. Due to the extremely dynamic nature of the planning tasks, we find that a reasonably narrow tunnel allows solutions to be returned very quickly which can be improved, if time permits. If the tunnel is too narrow, however, no plan maybe returned, requiring a replan in a wider tunnel. The supplementary document provides the algorithmic details to handle tunnel changes that are sent between planning tasks in different domains.

Completeness and Optimality Guarantees. The use of tunnels enables AD* to leverage plans across domains in order to expedite searches in high-dimensional domains. However; by modifying the definition of $\texttt{succ}(s)$ and $\texttt{pred}(s)$ to prune nodes that lie outside the tunnel, we sacrifice the strict bounds on optimality provided by AD*, as nodes that lie outside the tunnel may lead to a more optimal solution. By iteratively expanding the tunnel width t_w, when the search is unsuccessful, we ensure that a solution will be found, if one exists. For practical purposes, we find that a constantly dynamic world mitigates the need for strict optimality bounds as solutions are constantly invalidated, before their use. In our experiments (Section 7.1), we find that the computational benefit of using tunnels far outweighs its drawbacks, providing an exponential reduction in the nodes expanded, while still producing reasonable quality solutions.

7 Results

7.1 Comparative Evaluation of Domain Relationships

We randomly generate 1000 scenarios of size $100m \times 100m$, with random configurations of obstacles (both static and dynamic), start state, and goal state and record the effective branching factor, number of nodes expanded, time to compute a plan, success rate, and quality of the plans obtained. The effective branching factor is the average number of successors that were generated over the course of one search. Success rate is the ratio of the number of scenarios for which a collision-free solution was obtained. Plan quality is the ratio of the length of the static optimal path and the path obtained. A plan quality of 1 indicates that the solution obtained was able to minimize distance without any deviations. Similar metrics for analyzing multi-agent simulations have been used in [Kapadia et al. 2011]. The aggregate metrics for the different domains and domain relationships are shown in Table 1. Rows 3 and 6 in Table 1 include the added time to compute plans in earlier domains for tunnel

search, to provide an absolute basis of comparison. All experiments were performed on a single-threaded 2.80 GHz Intel(R) Core(TM) i7 CPU.

Σ_1 and Σ_2 can quickly generate solutions but is unable to solve most of the scenarios as they don't resolve fine-grained collisions. The use of plans from Σ_1 accelerates searches in Σ_2 (Table 1, Row 3). However, the real benefit of using both Σ_1 and Σ_2 is evident when performing repeated searches across domains in large environments when an initial plan $\Pi(\Sigma_1)$ accelerates repeated refinements in Σ_2 (and other subsequent domains). Using Σ_3 in a large environment takes significantly longer to produce similar paths. Σ_4 is unable to find a complete solution for large-scale problem instances (we limit maximum number of nodes expanded to 10^4), and the partial solutions often suffer from local minima, resulting in a low success rate. The benefit of using tunnels is evident in the dramatic reduction of the effective branching factor and nodes expanded for Σ_4.

When using the complete global path from Σ_3 as a tunnel for Σ_4 (Figure 4(a) and Row 6 in Table 1), the effective branching factor reduces from 21.5 to 5.6, producing an exponential drop in node expansion and computation time, and enabling complete solutions to be generated in the space-time domain. This planning task is able to successfully solve nearly 92% of the scenarios that were generated. However, since s_{start} and s_{goal} are far apart, the large depth of the search prevents this from being used at interactive rates for many agents.

By using successive waypoints in $\Pi(\Sigma_2)$ as s_{start} and s_{goal} to create a series of planning tasks in Σ_3 and Σ_4 (Figure 4(b) and Row 7 in Table 1), we reduce the breadth *and* depth of the search, allowing solutions to be returned at a fraction of the time (6 ms), without significantly affecting the success rate. The tradeoff is that independent plans are generated between waypoints along the global path, creating a two-level hierarchy between the domains.

Domain	BF	N	T	S	Q
$T(\Sigma_1)$	3.7	43	3	0.17	0.76
$T(\Sigma_2)$	4.6	85	8	0.23	0.57
$T(\Sigma_2, \Pi(\Sigma_1))$	2.1	17	5	0.32	0.65
$T(\Sigma_3)$	7.4	187	18	0.68	0.73
$T(\Sigma_4)$	21.5	10^4	2487	0.34	0.26
$T(\Sigma_4, \Pi(\Sigma_3, \Sigma_2, \Sigma_1))$	5.6	765	136	0.92	0.64
$\sum T_i(\Sigma_4, \Pi(\Sigma_3, \Sigma_2, \Sigma_1))$	5.4	75	8	0.86	0.58

Table 1: *Comparative evaluation of the domains, and the use of multiple domains.* ***BF*** *= Effective branching factor.* ***N*** *= Average number of nodes expanded.* ***T*** *= Average time to compute plan (ms).* ***S*** *= Success rate of planner to produce collision-free trajectory.* ***Q*** *= Plan quality. Row 6,7 corresponds to the domain relationships illustrated in Figures 4(a) and (b) respectively.*

Conclusion. The comparative evaluations of domains shows that no single domain can efficiently solve the challenging problem instances that were sampled. The use of tunnels significantly reduce the effective branching factor of the search in Σ_3 and Σ_4, while mapping successive waypoints in $\Pi(\Sigma_2)$ to multiple independent planning tasks reduce the depth of the search in Σ_3 and Σ_4, without significantly impacting success rate and quality. For the remaining results in the paper, we adopt this domain relationship as it works well for our application of simulating multiple goal-directed agents in dynamic environments at interactive rates. Users may choose a different relationship based on their specific needs.

7.2 Performance

We measure the performance of the framework by monitoring the execution time of each task type, with multiple instances of planning tasks for Σ_3 and Σ_4. We limit the maximum deliberation time $t_{max} = 10$ ms, which means that the total time executing any of the tasks at each frame cannot exceed $10ms$. For this experiment, we limit the total number of tasks that can be executed in a single frame to 2 (including $\mathtt{T}_0$) to visualize the execution time of each task over different frames. Figure 6 illustrates the task execution times of a single agent over a 30 second simulation for the scenario shown in Figure 2(a). The execution task $\mathtt{T}_0$ which is responsible for character animation and simple steering takes approximately $0.4 - 0.5$ ms of execution time every frame. Spikes in the execution time correlate to events in the world. For example, a local non-deterministic change in the environment (Frames 31,157) triggers a plan update in $T(\Sigma_3)$, which in turn triggers an update in $T(\Sigma_4)$. A global change such as a crowd blocking a passage or a change in goal (Frames 39, 237,281) triggers an update in $T(\Sigma_2)$ or $T(\Sigma_1)$ which in turn propagates events down the task dependency chain.

Note that there are often instances during the simulation when the start and goal changes significantly or when plans are invalidated, requiring planning from scratch. However, we ensure that our framework meets real-time constraints due to the following design decisions: (a) limiting the maximum amount of time to deliberate for the planning tasks, (b) intelligently distributing the available computational resources between tasks with highest priority, and (c) increasing the inflation factor to quickly produce a sub-optimal solution when a plan is invalidated, and refining the plan in successive frames.

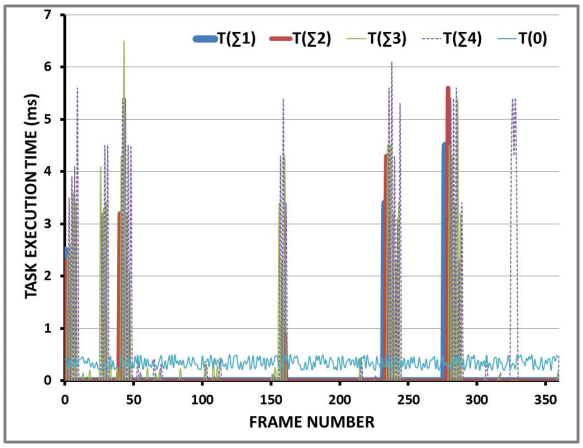
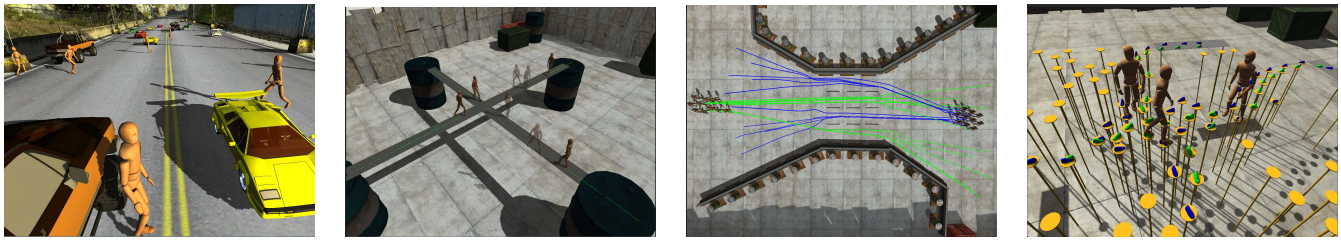


Figure 6: *Task execution times of the different tasks in our framework over the course of a* 60 *second simulation.*

Memory. $T(\Sigma_1)$ and $T(\Sigma_2)$ precomputes navigation meshes for the environment whose size depend on environment complexity, but are shared by all agents in the simulation. The runtime memory requirement of these tasks is negligible since it expands very few nodes. The memory footprint of $T(\Sigma_3)$ and $T(\Sigma_4)$ is defined by the number of nodes visited by the planning task during the course of a simulation. Since each planning task in Σ_3 and Σ_4 searches between successive waypoints in the global plan, the search horizon of the planners is never too large. On average, the number of visited nodes is 75 and 350 for $T(\Sigma_3)$ and $T(\Sigma_4)$ respectively with each node occupying $16 - 24$ bytes in memory. For 5 running instances of $T(\Sigma_3)$ and $T(\Sigma_4)$, this amounts to approximately $45KB$ of memory per agent. Additional memory for storing other plan containers such as `OPEN` and `CLOSED` are not considered in this calculation as they store only node references and are cleared after every plan iteration.

Scalability. Our approach scales linearly with increase in number of agents. The maximum deliberation time *for all* agents can be



(a) (b) (c) (d)

Figure 5: *Different scenarios. (a) Agents crossing a highway with fast moving vehicles in both directions. (b)* 4 *agents solving a deadlock situation at a 4-way intersection. (c)* 20 *agents distributing themselves evenly in a narrow passage, to form lanes both in directions. (d) A complex environment requiring careful foot placement to obtain a solution.*

chosen based on the desired frame rate which is then distributed among agents and their respective planning tasks at each frame. The cost of planning is amortized over several frames and all agents need not plan simultaneously. Once an agent computes an initial plan, it can execute the plan with efficient update operations until it is allocated more deliberation time. If its most imminent plan is invalidated, it is prioritized over other agents and remains stationary till computational resources are available. This ensures that the simulation meets the desired framerate.

7.3 Scenarios

We demonstrate the benefits of our framework by solving many challenging scenarios (Figure 5) requiring space-time precision, explicit coordination between interacting agents, and the factoring of dynamic information (obstacles, moving platforms, user-triggered changes, and other agents) at all stages of the decision process. All results shown here were generated at 30 fps or higher, which includes rendering and character animation. We use an extended version of the ADAPT character animation system [Johansen 2009] for the results shown in the video.

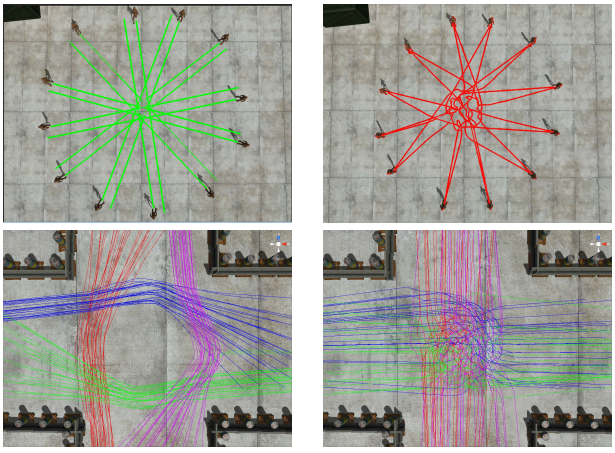
Deadlocks. Multiple oncoming and crossing agents in narrow passageways cooperate with each other with space-time precision to prevent potential deadlocks. Agents observe the presence of dynamic entities at waypoints along their global path and refine their plan if they notice potentially blocked passageways or other high cost situations. Crowd simulators deadlock for these scenarios, while a space-time planner does not scale well for many agents.

Choke Points. This scenario shows our approach handling agents arriving at a common meeting point at the same time, producing collision-free straight trajectories. Figure 7 compares the trajectories produced using our method with an off the shelf navigation and predictive collision avoidance algorithm in the Unity game engine. Our framework produces considerably smoother trajectories and minimizes deviation by using subtle speed variations to avoid collisions in space-time.

Unpredictable Environment Change. Our method efficiently repairs solutions in the presence of unpredictable world events, such as the user-placement of obstacles or other agents, which may invalidate current paths.

Road Crossing. The road crossing scenario demonstrates 40 agents using space-time planning to avoid fast moving vehicles and other crossing agents.

Lane Selection for Bi-directional Traffic. This scenario requires agents to make a navigation decision in choosing one of 4 lanes created by the dividers. Agents distribute themselves among the lanes, while bi-directional traffic chooses different lanes to avoid



Our framework Unity navigation and steering

Figure 7: *Trajectory comparison of our method with an off the shelf predictive steering algorithm in the Unity game engine. Our framework minimizes deviation and uses speed variations to avoid collisions in space-time.*

deadlocks. This scenario requires non-deterministic dynamic information (other agents) to be accounted for while making global navigation decisions. This is different from emergent lane formation in crowd approaches, which bottlenecks at the lanes and cause deadlocks without a more robust navigation technique.

Four-way Crossing We simulate 100 oncoming and crossing agents in a four-way crossing. The initial global plans in Σ_1 take the minimum distance path through the center of the crossing. However, Σ_2 predicts a space-time collision between groups at the center and performs plan refinement so that agents deviate from their optimal trajectories to minimize group interactions. A predictive steering algorithm only accounts for imminent neighboring threats and is unable to avoid mingling with the other groups (second row of Figure 7).

Space-Time Goals. We demonstrate a complex scenario where 4 agents in focus (additional agents are also simulated) have a temporal goal constraint, defined as an interval $(40 + / - 1second)$. Agents exhibit space-time precision while jumping across moving planes to reach their target and the temporal goal significantly impacts the decision making at all levels, where the space-time domain maybe unable to meet the temporal constraint and require plans to be modified in earlier domains. No other approach can solve this with real-time constraints.

Many of these scenarios *cannot* be solved by the current state of the

art in multi-agent motion planning, which is able to either handle a single agent with great precision, or simulate many simple agents that exhibit reactive collision avoidance.

7.4 Framework Extensibility

The potential of our framework lies in the ability to use multiple domains of control, and is not limited to the domains described in Section 4.1, which only serve as a sufficient set to showcase the benefits of our method. For example, the scenario shown in Figure 5(d) requires careful control over how the character chooses its footsteps and cannot be solved by Σ_4, which does not model bipedal locomotion.

We add a footstep domain Σ_5, which models the motion of the character's center of mass and feet placement using an inverted spherical pendulum model for bipedal locomotion. The agent state $s = (\mathbf{x}, \mathbf{v}, \mathbf{f}_x, f_\phi, \mathrm{I} \in \{\mathtt{L}, \mathtt{R}\})$ includes the center of mass position and velocity, the position and orientation of the current support foot, and an indicator function for the swing foot. An action is chosen by selecting the time period of the footstep, the orientation and speed of the center of mass at the end of the step, and the orientation of the foot plant. The space of possible values of these parameters, while satisfying the constraints enforced by the inverted pendulum model, defines the action space of Σ_5. We discretize this space to keep the set of possible footstep transitions at each state to approximately 35. For implementation details of this domain, please refer to [Singh et al. 2011b].

$\lambda(s, \Sigma_4, \Sigma_5)$ maps states in Σ_4 to one or more states in Σ_5 in order to define a tunnel $\tau(\Sigma_5, \Pi(\Sigma_4), t_w)$ around $\Pi(\Sigma_4)$. We start from a default double support configuration of the character at the start and assume that the character takes a left foot stride first. The COM position is used to define a set of valid positions of the support foot at each space-time waypoint, and $\mathbf{f}_x$ is constrained based on the future COM position (where the character turns to next).

8 Discussion

Choice of Domains. The domains described in this paper represent popular solutions that are used in both academia and industry. Navigation meshes (Σ_1) are a standard solution [Mononen 2009] for representing free spaces in arbitrarily large, complex, static environments with recent proposed extensions [van Toll et al. 2012] that account for dynamic information (Σ_2). A grid-based representation (Σ_3) provides a uniform discretization of the environment, and is widely used in robot motion planning [Koenig and Likhachev 2002; Likhachev et al. 2005]. The introduction of time as a third dimension (Σ_4) enables collision checks in the future, facilitating more robust collision resolution.

These domains provide a nice balance between global navigation and space-time planning, enabling us to showcase the strength of our framework: the ability to use multiple domains of control, and leverage solutions across domains to accelerate computations while still providing a high degree of control fidelity. Additional domains can be easily integrated (e.g., a footstep domain) to meet application-specific needs, or solve more challenging motion planning problems.

Relationship Between Domains. Domains can be connected by using the plan from one domain as a tunnel for the other, or by using successive waypoints along the plan as start and goal pair for multiple planning tasks in a more complex domain. We evaluated both domain relationships based on computational efficiency and coverage, as shown in Table 1. Using waypoints from the navigation mesh domain as start, goal pairs for planning tasks in the grid and space-time domain keeps the search depth for Σ_3 and Σ_4 within reasonable bounds. The tradeoff is that a space-time plan is never generated at a global level from an agent's start position to its target, thus sacrificing completeness guarantees. This design choice worked well for our experiments where the reduction in success rate of our framework when using this scheme was within reasonable bounds, while providing a considerable performance boost, making it suitable for practical game-like applications. Users may wish to opt for different domain relationships depending on the application.

Acknowledgements

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement # W911NF-10-2-0016. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. This work has been partially funded by the Spanish Ministry of Science and Innovation under Grant TIN2010-20590-C01-01. A. Beacco is also supported by the grant FPUAP2009-2195 (Spanish Ministry of Education).

References

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. In *SIGGRAPH*, ACM, 483–490.

BOTEA, A., MLLER, M., AND SCHAEFFER, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development 1*, 7–28.

BULITKO, V., STURTEVANT, N., LU, J., AND YAU, T. 2007. Graph abstraction in real-time heuristic search. *J. Artif. Int. Res. 30*, 1 (Sept.), 51–100.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph. 22*, 2 (Apr.), 182–203.

CHOI, M. G., KIM, M., HYUN, K., AND LEE, J. 2011. Deformable motion: Squeezing into cluttered environments. *Comput. Graph. Forum 30*, 2, 445–453.

FRAICHARD, T. 1999. Trajectory planning in a dynamic workspace: a'state-time space'approach. *Advanced Robotics 13 6*, 8, 7594.

GOCHEV, K., COHEN, B. J., BUTZKE, J., SAFONOVA, A., AND LIKHACHEV, M. 2011. Path planning with adaptive dimensionality. In *SOCS*.

HOFF, K., I., CULVER, T., KEYSER, J., LIN, M., AND MANOCHA, D. 2000. Interactive motion planning using hardware-accelerated computation of generalized voronoi diagrams. In *ICRA*, vol. 3, 2931–2937 vol.3.

HOLTE, R. C., PEREZ, M. B., ZIMMER, R. M., AND MACDONALD, A. J. 1996. Hierarchical A *: searching abstraction hierarchies efficiently. In *National conference on Artificial intelligence*, AAAI Press, AAAI, 530–535.

HOLTE, R., GRAJKOWSKI, J., AND TANNER, B. 2005. Hierarchical heuristic search revisited. In *Abstraction, Reformulation and Approximation*, vol. 3607 of *LNCS*. Springer Berlin Heidelberg, 121–133.

HSU, D., KINDEL, R., LATOMBE, J.-C., AND ROCK, S. 2002. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research 21*, 3, 233–255.

JOHANSEN, R. S. 2009. *Automated Semi-Procedural Animation for Character Locomotion*. Master's thesis, Aarhus University.

KALLMANN, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *ACM SIGGRAPH/Eurographics SCA*, 159–168.

KAPADIA, M., SINGH, S., HEWLETT, W., AND FALOUTSOS, P. 2009. Egocentric affordance fields in pedestrian steering. In *Symposium on Interactive 3D graphics and games*, ACM, I3D, 215–223.

KAPADIA, M., WANG, M., SINGH, S., REINMAN, G., AND FALOUTSOS, P. 2011. Scenario space: characterizing coverage, quality, and failure of steering algorithms. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA, 53–62.

KOENIG, S., AND LIKHACHEV, M. 2002. D* Lite. In *National Conf. on AI*, AAAI, 476–483.

KRING, A. W., CHAMPANDARD, A. J., AND SAMARIN, N. 2010. DHPA* and SHPA*: Efficient Hierarchical Pathfinding in Dynamic and Static Game Worlds. In *AIIDE*, The AAAI Press.

LACAZE, A. 2002. Hierarchical planning algorithms. In *SPIE Int. Symposium on Aerospace/Defense Sensing, Simulation, and Controls*.

LAMARCHE, F., AND DONIKIAN, S. 2004. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. In *Computer Graphics Forum 23.*

LAU, M., AND KUFFNER, J. J. 2005. Behavior planning for character animation. In *ACM SIGGRAPH/Eurographics SCA*, 271–280.

LEVINE, S., LEE, Y., KOLTUN, V., AND POPOVIĆ, Z. 2011. Space-time planning with parameterized locomotion controllers. *ACM Trans. Graph. 30* (May), 23:1–23:11.

LIKHACHEV, M., GORDON, G. J., AND THRUN, S. 2003. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In *NIPS*.

LIKHACHEV, M., FERGUSON, D. I., GORDON, G. J., STENTZ, A., AND THRUN, S. 2005. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *ICAPS*, 262–271.

LO, W.-Y., AND ZWICKER, M. 2008. Real-time planning for parameterized human motion. In *ACM SIGGRAPH/Eurographics SCA*, 29–38.

LOPEZ, T., LAMARCHE, F., AND LI, T.-Y. 2012. Space-time planning in changing environments : using dynamic objects for accessibility. *CAVW 23*, 2, 87–99.

LOSCOS, C., MARCHAL, D., AND MEYER, A. 2003. Intuitive crowd behaviour in dense urban environments using local laws. In *TPCG*, IEEE, 122.

MONONEN, M., 2009. Recast: Navigation-mesh construction toolset for games. http://code.google.com/p/recastnavigation/.

PARIS, S., PETTRÉ, J., AND DONIKIAN, S. 2007. Pedestrian reactive navigation for crowd simulation: a predictive approach. In *EUROGRAPHICS 2007*, vol. 26, 665–674.

PEARL, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2008. *Virtual Crowds: Methods, Simulation, and Control*. Synthesis Lectures on Computer Graphics and Animation.

PETTRÉ, J., KALLMANN, M., AND LIN, M. C. 2008. Motion planning and autonomy for virtual humans. In *ACM SIGGRAPH classes*, 1–31.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH*, 25–34.

SAFONOVA, A., AND HODGINS, J. K. 2007. Construction and optimal search of interpolated motion graphs. In *ACM SIGGRAPH*.

SHAPIRO, A., KALLMANN, M., AND FALOUTSOS, P. 2007. Interactive motion correction and object manipulation. In *ACM SIGGRAPH I3D*.

SINGH, S., KAPADIA, M., HEWLETT, B., REINMAN, G., AND FALOUTSOS, P. 2011. A modular framework for adaptive agent-based steering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D, 141–150 PAGE@9.

SINGH, S., KAPADIA, M., REINMAN, G., AND FALOUTSOS, P. 2011. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds 22*, 2-3, 151–158.

STURTEVANT, N., AND GEISBERGER, R. 2010. A comparison of high-level approaches for speeding up pathfinding. 76–82.

SUD, A., GAYLE, R., ANDERSEN, E., GUY, S., LIN, M., AND MANOCHA, D. 2007. Real-time navigation of independent agents using adaptive roadmaps. In *VRST*, ACM, 99–106.

SUNG, M., KOVAR, L., AND GLEICHER, M. 2005. Fast and accurate goal-directed motion synthesis for crowds. In *ACM SIGGRAPH/Eurographics SCA*, 291–300.

THALMANN, D. 2008. Crowd simulation. In *Wiley Encyclopedia of Computer Science and Engineering*.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. In *ACM SIGGRAPH*, 1160–1168.

VAN DEN BERG, J., FERGUSON, D., AND KUFFNER, J. 2006. Anytime path planning and replanning in dynamic environments. In *ICRA*, 2366 –2371.

VAN DEN BERG, J., LIN, M. C., AND MANOCHA, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of ICRA*, IEEE, 1928–1935.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of multiple agents in crowded environments. In *ACM SIGGRAPH I3D*, 139–147.

VAN TOLL, W. G., COOK, A. F., AND GERAERTS, R. 2012. Real-time density-based crowd simulation. *CAVW 23*, 1, 59–69.

How the Ocean Personality Model Affects the Perception of Crowds

Funda Durupınar ■ *Bilkent University*

Nuria Pelechano ■ *Universitat Politècnica de Catalunya*

Jan M. Allbeck ■ *George Mason University*

Uğur Güdükbay ■ *Bilkent University*

Norman I. Badler ■ *University of Pennsylvania*

This approach extends the HiDAC (High-Density Autonomous Crowds) system by providing each agent with a personality model based on the Ocean (openness, conscientiousness, extroversion, agreeableness, and neuroticism) personality model. Each personality trait has an associated nominal behavior. Specifying an agent's personality leads to an automation of low-level parameter tuning.

Simulating the behavior of animated virtual crowds is a challenge for the computer graphics community. To achieve realistic behavior in virtual crowds requires extensive study of the semantics underlying real crowds' motion. Psychologists study human nature to identify salient behavior characteristics. There has been extensive research on incorporating psychological models into the simulation of autonomous agents. Here, however, we're not interested in a person's personality, per se, but in incorporating a personality model into large groups of people. By changing the parameters, we examine how subgroups of people with different personality traits interact and, accordingly, how global crowd behavior is influenced. The user decides the percentage and distribution of the personality traits.

Personality is a pattern of a person's behavioral, temperamental, emotional, and mental traits. Considerable controversy exists in personality research about how many personality traits there are. However, one popular model is the Five Factor, or Ocean (openness, conscientiousness, extroversion, agreeableness, and neuroticism) model.[1] These five factors are orthogonal dimensions of the personality space. Openness describes the imaginative and creative aspect of human character. Conscientiousness determines to what level a person is organized and careful. Extroversion relates to how outgoing and sociable a person is. Agreeableness is friendliness, generosity, and the tendency to get along with other people. Finally, neuroticism refers to emotional instability and the tendency to experience negative emotions. Each factor is bipolar and has several traits, which essentially are adjectives used to describe people.[2]

We've mapped these trait terms to the set of behaviors in the HiDAC (High-Density Autonomous Crowds) crowd simulation system.[3] HiDAC models individual differences by assigning each person different psychological and physiological traits. Users normally set these parameters to model a crowd's nonuniformity and diversity. Our approach frees users of the tedious task of low-level parameter tuning and combines all these behaviors in distinct personality factors. To verify our mapping's plausibility, we evaluated users' perception of the personality traits in generated animations.

The System

By combining a standard personality model with a high-density crowd simulation, our approach creates plausible variations in the crowd and enables novice users to dictate these variations.[4]

 Published by the IEEE Computer Society

Table 1. Low-level parameters versus trait-descriptive adjectives.

Agent behavior	Personality factor	Ocean factor*
Leadership	Assertive, social, unsocial, calm, fearful	E, N
Trained or untrained	Informed, ignorant	O
Communication	Social, unsocial	E
Panic	Oversensitive, fearful, calm, orderly, predictable	N, C+
Impatience	Rude, assertive, patient, stubborn, tolerant, orderly	E+, C, A
Pushing	Rude, kind, harsh, assertive, shy	A, E
Right preference	Cooperative, predictable, negative, contrary, changeable	A, C
Avoidance or personal space	Social, distant	E
Waiting radius	Tolerant, patient, negative	A
Waiting timer	Kind, patient, negative	A
Exploring environment	Curious, narrow	O
Walking speed	Energetic, lethargic, vigorless	E
Gesturing	Social, unsocial, shy, energetic, lethargic	E

**The letters in this column stand for openness, conscientiousness, extroversion, agreeableness, and neuroticism.*

HiDAC

HiDAC addresses the simulation of local behaviors and the global wayfinding of crowds in a dynamically changing environment. It directs autonomous agents' behavior by combining geometric and psychological rules. Psychological attributes include impatience, panic, and leadership behaviors, which are determined by traits such as locomotion, energy levels, and maximum speed. Agents have skills such as navigation in complex environments, communication, learning, and certain kinds of decision making. Agents also have perception so that they can react to obstacles, other agents, and dynamic changes in the environment.

To achieve realistic behavior, HiDAC handles collisions through avoidance and response forces. Over long distances, the system applies collision avoidance so that agents can steer around obstacles. Over shorter distances, it applies collision response to prevent agents from overlapping with each other and the environment.

Besides the usual crowd behavior, agents might display pushing behavior or show that they can wait for other agents to pass first, depending on their politeness and patience. Pushing behavior arises from varying each agent's personal-space threshold. Impatient agents don't respect others' personal space and appear to push their way through the crowd. Relaxed agents temporarily stop when another agent moves into their path; impatient agents don't respond to this feedback and tend to push.

Integrating the Ocean Model into HiDAC

A crowd consists of subgroups with different personalities. Variations in the subgroups' characteristics influence emergent crowd behavior. The user can add any number of groups with shared personality traits and can edit these characteristics throughout an animation.

To model an agent's personality π, we use a five-dimensional vector, in which a personality factor Ψ_i represents each dimension. To model the factors' distribution in a group of people, we use a Gaussian distribution function N with mean μ_i and standard deviation σ_i:

$$\pi = <\Psi_O, \Psi_C, \Psi_E, \Psi_A, \Psi_N>,$$

$$\Psi_i = N\left(\mu_i, \sigma_i^2\right), \text{ for } i \in \{O, C, E, A, N\},$$

where $\mu \in [0, 1]$, $\sigma \in [-0.1, 0.1]$.

A person's overall behavior β is a combination of different behaviors. Each behavior is a function of personality:

$$\beta = (\beta_1, \beta_2, \ldots, \beta_n)$$

$$\beta_j = f(\pi), \text{ for } j = 1, \ldots, n.$$

Because each factor is bipolar, Ψ can have positive and negative values. For instance, a value of 1 for extroversion means that the person is highly extroverted, whereas –1 means that the person is highly introverted.

Personality-to-Behavior Mapping

We map agents' personality factors (adjectives) onto low-level parameters and onto the built-in behaviors in the HiDAC model (see Table 1). A positive factor takes values in the range [0.5, 1]; a negative factor takes values in the range [0, 0.5). A factor with no sign indicates that both poles apply to that behavior. For instance, E+ for

a behavior means that only extroversion is related to that behavior; introversion isn't applicable. As Table 1 shows, more than one personality dimension can define a behavior. The more adjectives of a certain factor that are defined for a behavior, the stronger that factor's impact on that behavior. We assign a weight to the factor's impact on a specific behavior. For instance, ω_{EL} is the weight of extroversion on leadership; it takes a value in the range [0, 1]. The sum of the weights for a specific type of behavior is 1.

Now, we show how our approach maps a personality dimension to a specific type of behavior. We define the behavior parameters for an agent *i* as follows.

Leadership. Leaders tend to have more confidence in themselves. They remain calm during emergencies. Each agent has a leadership percentage determined by extroversion and stability. We compute leadership behavior as

$$\beta_i^{\text{Leadership}} = \omega_{EL}\Psi_i^{E} + \omega_{NL}\left(1-\Psi_i^{N}\right),$$

where

$$\beta_i^{\text{Leadership}} \propto \text{E},\ \beta_i^{\text{Leadership}} \propto^{-1} \text{N, and } \beta_i^{\text{Leadership}} \in [0,1].$$

Trained. Trained agents have complete knowledge about the environment. Because being trained requires curiosity and because trained people are informed, we associate this parameter with openness.

Being trained is a Boolean parameter, so we use a probability function to represent it. As openness increases, the probability that the agent is trained increases:

$$P_i(\text{Trained}) = \Psi_i^{O}$$
$$\beta_i^{\text{Trained}} = \begin{cases} 1 & \text{if } P_i(\text{Trained}) \geq 0.5 \\ 0 & \text{otherwise} \end{cases},$$

where

$$P_i\ (\text{Trained}) \propto \text{O and } \beta_i^{\text{Trained}} \in \{0,\ 1\}.$$

Communication. This parameter determines whether agents communicate. Similar to being trained, communication depends on the probability of agent behavior. As extroversion increases, the probability that the agent communicates increases:

$$P_i(\text{Communication}) = \Psi_i^{E}$$
$$\beta_i^{\text{Communication}} = \begin{cases} 1 & \text{if } P_i(\text{Communication}) \geq 0.5 \\ 0 & \text{otherwise} \end{cases},$$

where

$$P_i(\text{Communication}) \propto \text{E and } \beta_i^{\text{Communication}} \in \{0,1\}.$$

Panic. In emergency situations, agents display panic behavior depending on their stability and conscientiousness. When they panic, their walking speed increases and they don't wait. We compute panic as

$$\beta_i^{\text{Panic}} = \omega_{NP}\Psi_i^{N} + \omega_{CP} f\left(\Psi_i^{C}\right)$$
$$f\left(\Psi_i^{C}\right) = \begin{cases} -2\Psi_i^{C} + 2 & \text{if } \Psi_i^{C} \geq 0 \\ 0 & \text{otherwise} \end{cases},$$

where

$$\beta_i^{\text{Panic}} \propto \text{N},\ \beta_i^{\text{Panic}} \propto^{-1} \text{C+, and } \beta_i^{\text{Panic}} \in [0,1].$$

Impatience. We implement this parameter by modifying the route selection dynamically on the basis of environmental changes. This parameter depends on an agent's politeness and assertiveness. We compute impatience as

$$\beta_i^{\text{Impatience}} = \omega_{EI} f\left(\Psi_i^{E}\right) + \omega_{AI}\left(1-\Psi_i^{A}\right) + \omega_{CI}\left(1-\Psi_i^{C}\right)$$
$$f\left(\Psi_i^{E}\right) = \begin{cases} 2\Psi_i^{E} - 1 & \text{if } \Psi_i^{E} \geq 0 \\ 0 & \text{otherwise} \end{cases},$$

where

$$\beta_i^{\text{Impatience}} \propto \text{E+},\ \beta_i^{\text{Impatience}} \propto^{-1} \text{A,C, and } \beta_i^{\text{Impatience}} \in [0,1].$$

Pushing. HiDAC can realistically simulate a person's respect for others. Agents can try to force their way through a crowd by pushing others, exhibit more respectful behavior when desired, make decisions about letting others walk first, and queue when necessary. Disagreeable agents tend to push others more because they're harsh and impolite. Similarly, extroverted agents display pushing behavior because they tend to be assertive. We compute pushing as

$$P_i(\text{Pushing}) = \omega_{EP}\Psi_i^{E} + \omega_{AP}\left(1-\Psi_i^{A}\right)$$
$$\beta_i^{\text{Pushing}} = \begin{cases} 1 & \text{if } P_i(\text{Pushing}) \geq 0.5 \\ 0 & \text{otherwise} \end{cases},$$

where

$$P_i(\text{Pushing}) \propto \text{E},\ P_i(\text{Pushing}) \propto^{-1} \text{A, and } \beta_i^{\text{Pushing}} \in \{0,1\}.$$

Right preference. People prefer to move toward the right side of the obstacle that they're about to encounter. This behavior shows the person's level of conformity to the rules. The right-preference behavior is a probability function. If an agent is disagreeable or nonconscientious, that agent can make a right or left preference with equal probability. On the other hand, an agent that prefers the right side increases the probability proportionally to the agent's agreeableness and conscientiousness values, if these values are positive. We compute right preference as

$$P(\text{Right}) = \begin{cases} 0.5 & \text{if } \Psi_i^{\text{A}} < 0 \text{ or } \Psi_i^{\text{C}} < 0 \\ \omega_{\text{AR}}\Psi_i^{\text{A}} + \omega_{\text{CR}}\Psi_i^{\text{C}} & \text{otherwise} \end{cases}$$

$$\beta_i^{\text{Right}} = \begin{cases} 1 & \text{if } P_i(\text{Right}) \geq 0.5 \\ 0 & \text{otherwise} \end{cases},$$

where

$P_i(\text{Right}) \propto \text{A, C, and } \beta_i^{\text{Right}} \in \{0,1\}$.

Personal space. Personal space determines the territory in which a person feels comfortable. Agents try to preserve their personal space when they approach other agents and when other agents approach from behind. However, the values for these two situations aren't the same. According to research, the average personal space in Western cultures is 0.7 meters in front and 0.4 meters behind.[5] The personal space of an agent i with respect to an agent j is

$$\beta_{i,j}^{\text{PersonalSpace}} = \begin{cases} 0.8 * f(i,j) & \text{if } \Psi_i^{\text{E}} \in \left[0\ \frac{1}{3}\right) \\ 0.7 * f(i,j) & \text{if } \Psi_i^{\text{E}} \in \left[\frac{1}{3}\ \frac{2}{3}\right] \\ 0.5 * f(i,j) & \text{if } \Psi_i^{\text{E}} \in \left(\frac{2}{3}\ 1\right] \end{cases}$$

$$f(i,j) = \begin{cases} 1 & \text{if } i \text{ is behind } j \\ 0.4 \,/\, 0.7 & \text{otherwise} \end{cases},$$

where

$\beta_{i,j}^{\text{PersonalSpace}} \propto^{-1} \text{E and } \beta_{i,j}^{\text{PersonalSpace}} \in \{0.5, 0.7, 0.8\}$.

Waiting radius. In an organized situation, people tend to wait for available space before moving. We call this space the *waiting radius*; it depends on a person's kindness and consideration—that is, the agreeableness dimension. We compute the waiting radius as

$$\beta_i^{\text{WaitingRadius}} = \begin{cases} 0.25 & \text{if } \Psi_i^{\text{A}} \in \left[0\ \frac{1}{3}\right) \\ 0.45 & \text{if } \Psi_i^{\text{A}} \in \left[\frac{1}{3}\ \frac{2}{3}\right] \\ 0.65 & \text{if } \Psi_i^{\text{A}} \in \left(\frac{2}{3}\ 1\right] \end{cases},$$

where

$\beta_i^{\text{WaitingRadius}} \propto \text{A and } \beta_i^{\text{WaitingRadius}} \in \{0.25, 0.45, 0.65\}$.

Waiting timer. If two people are heading in the same direction, each waits for the other to move first. The time they wait—that is, the duration during which they display patience toward each other—depends on their agreeableness. We compute the waiting time as

$$\beta_i^{\text{WaitingTimer}} = \begin{cases} 1 & \text{if } \Psi_i^{\text{A}} \in \left[0\ \frac{1}{3}\right) \\ 5 & \text{if } \Psi_i^{\text{A}} \in \left[\frac{1}{3}\ \frac{2}{3}\right] \\ 50 & \text{if } \Psi_i^{\text{A}} \in \left(\frac{2}{3}\ 1\right] \end{cases},$$

where

$\beta_i^{\text{WaitingTimer}} \propto \text{A and } \beta_i^{\text{WaitingTimer}} \in \{1, 5, 50\}$.

Exploring the environment. HiDAC assigns people specific behaviors. The number of actions they complete depends on their curiosity. Open people will more likely explore different experiences and perform more actions. The openness factor determines the time during which a person explores the environment. The number of actions that a person completes increases by the degree of openness. We compute the exploring parameter as

$$\beta_i^{\text{Exploring}} = 10\Psi_i^{\text{O}},$$

where

$\beta_i^{\text{Exploring}} \propto \text{O and } \beta_i^{\text{Exploring}} \in [0, 10]$.

Walking speed. A person's energy level determines that person's maximum walking speed. Because extroverts tend to be more energetic and introverts more lethargic, the extroversion trait controls this parameter. We compute the walking speed as

$$\beta_i^{\text{WalkingSpeed}} = \Psi_i^{\text{E}} + 1,$$

where

$\beta_i^{\text{WalkingSpeed}} \propto \text{E and } \beta_i^{\text{WalkingSpeed}} \in [1, 2]$.

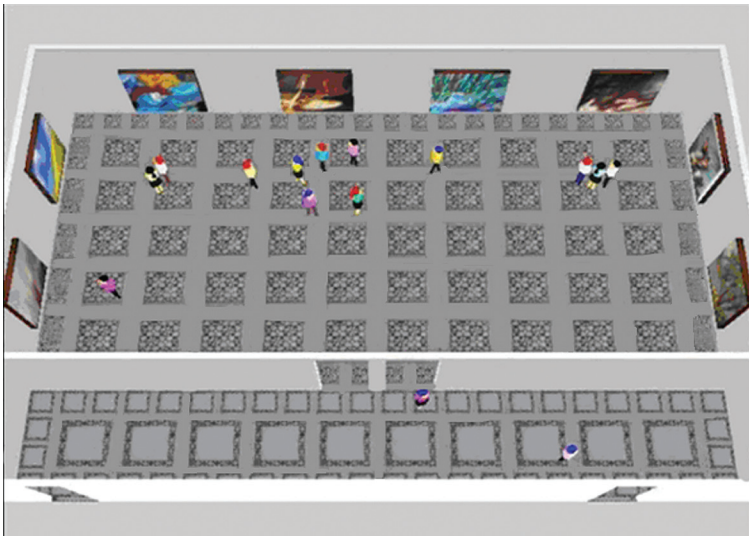


Figure 1. Openness tested in a museum. The most open people (red hair) stayed the longest. The least open people (blue hair) left the earliest.

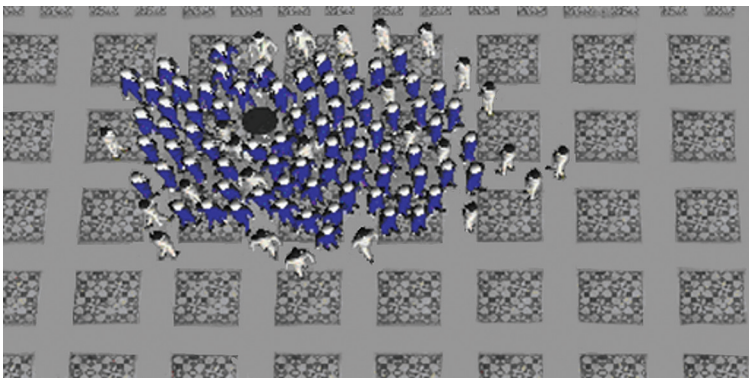


Figure 2. A ring formation example. Extroverts (blue suits) are inside and introverts are outside.

Gesturing. The amount of gesturing during a conversation indicates a person's sociability. Outgoing people use more gestures than shy people, which indicates extroversion. We compute the gesturing parameter as

$$\beta_i^{\text{Gesturing}} = 10\Psi_i^{\text{E}},$$

where

$$\beta_i^{\text{Gesturing}} \propto \text{E} \quad \text{and} \quad \beta_i^{\text{Gesturing}} \in [0,10].$$

Evaluation

To evaluate whether users will correctly perceive our suggested mappings, we conducted user studies. We created several animations to study how modifying subgroups' personality parameters affects global crowd behavior.

The Experiment's Design

We created 15 videos presenting the emergent behaviors of people in scenarios in which the settings assigned in the Ocean model drive crowds' behavior. The scenarios ranged from evacuation drills to cocktail parties or museum galleries.

We performed the mapping from HiDAC parameters to Ocean factors by using trait-descriptive adjectives. To validate our system, we determined the correspondence between our mapping and the users' perception of these trait terms in the videos. Our studies involved 70 participants (21 females and 49 males, ages 18 to 30). We showed the videos to them on a projected display and asked them to complete a questionnaire containing 123 questions—about eight questions per video. After each video, participants had time to answer the related questions. The participants had no previous knowledge of the experiment.

Questions assessed how much a person agreed with statements such as, "I think the people in this video are kind" or "I think the people with green suits are calm." We asked questions that included the adjectives describing each Ocean factor instead of asking directly about the factors. We used descriptive questions because the general public, being unfamiliar with the Ocean model, might have difficulty answering questions such as, "Do the people exhibit openness?" Although the participants were proficient in English, to prevent any misconceptions, we attached dictionary definitions of the adjectives to the questionnaires. Participants chose answers on a scale from 0 to 10, where 0 = totally disagree, 5 = neither agree nor disagree, and 10 = totally agree. We omitted the antonyms from the list of adjectives, for conciseness. The remaining adjectives were *assertive*, *calm*, *changeable*, *contrary*, *cooperative*, *curious*, *distant*, *energetic*, *harsh*, *ignorant*, *kind*, *orderly*, *patient*, *predictable*, *rude*, *shy*, *social*, *stubborn*, and *tolerant*.

Sample Scenarios

In the scenarios, novel, emergent formations and different behavior timings occurred.

The museum scenario tested the impact of openness. A key factor determining openness is the belief in the importance of art. Figure 1 shows a screenshot from the sample animation. This scenario tested the adjectives *curiosity* and *ignorance*. There were three groups of people, with openness values of 0, 0.5, and 1. We mapped the number of tasks that each agent must perform to openness, with each task requiring looking at a painting. The least open agents (with blue hair) left the museum first, followed by the agents with openness values of 0.5 (with black hair). The most open agents (with red hair) stayed the longest. We asked the participants how they perceived each group.

Another video assessed how the participants perceived extroverts and introverts according to their distribution around a point of attraction. Figure 2 shows a screenshot in which the agents in blue suits are extroverted (μ = 0.9 and σ = 0.1) and those in grey suits are introverted (μ = 0.1 and σ = 0.1). The ratio of introverts to extroverts in a society is 25 percent;[6] we assigned the initial number of agents according to this ratio. At the animation's end, introverts were outside the ring structure around the object of attraction. Because extroverts are faster, they approached the attraction point in less time. In addition, when other agents blocked their way, they tended to push them to reach their goal. The figure also shows the difference between the personal spaces of introverts and extroverts. This animation tested the adjectives *social*, *distant*, *assertive*, *energetic*, and *shy*.

To test whether the participants could distinguish the personalities of people who create congestion, we showed them two videos of the same duration and asked them to compare the characteristics of the agents in each video. Each video consisted of two groups of people moving through each other. The first video showed people with high agreeableness and conscientiousness values (μ = 0.9 and σ = 0.1 for both traits). The second video showed people with low agreeableness and conscientiousness values (μ = 0.1 and σ = 0.1 for both traits). In the first video, groups managed to cross each other, whereas in the second, congestion occurred after a fixed time period. Such behaviors emerged because people who are agreeable and conscientious are more patient; they don't push each other and are always predictable, because they prefer to move on the right side. Figure 3 shows how congestion occurred because of low conscientiousness and agreeableness. People were stuck at the center and refused to let other people move. They also were stubborn, negative, and uncooperative.

Figure 4 shows a screenshot from the animation demonstrating how neuroticism, nonconscientiousness, and disagreeableness affect panic. We simulated 13 agents. Five of them had neuroticism values of μ = 0.9 and σ = 0.1, conscientiousness values of μ = 0.1 and σ = 0.1, and agreeableness values of μ = 0.1 and σ = 0.1. The other agents, which were psychologically stable, had neuroticism values of μ = 0.1 and σ = 0.1, conscientiousness values of μ = 0.9 and σ = 0.1, and agreeableness values of μ = 0.9 and σ = 0.1. The agents in green suits are neurotic, less conscientious, and disagreeable. As the figure shows, they tend to panic more, push other agents, force their way through the crowd, and rush to the door. They aren't predictable, cooperative, patient, or calm; they're rude, changeable, negative, and stubborn.

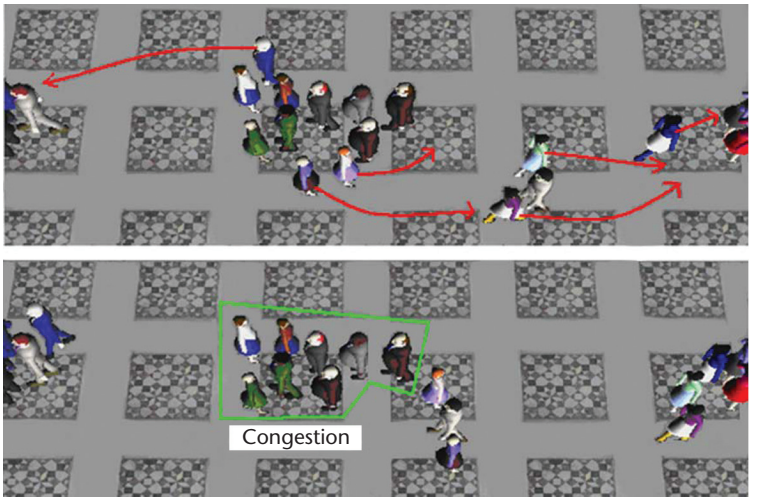


Figure 3. People with low conscientiousness and agreeableness cause congestion.

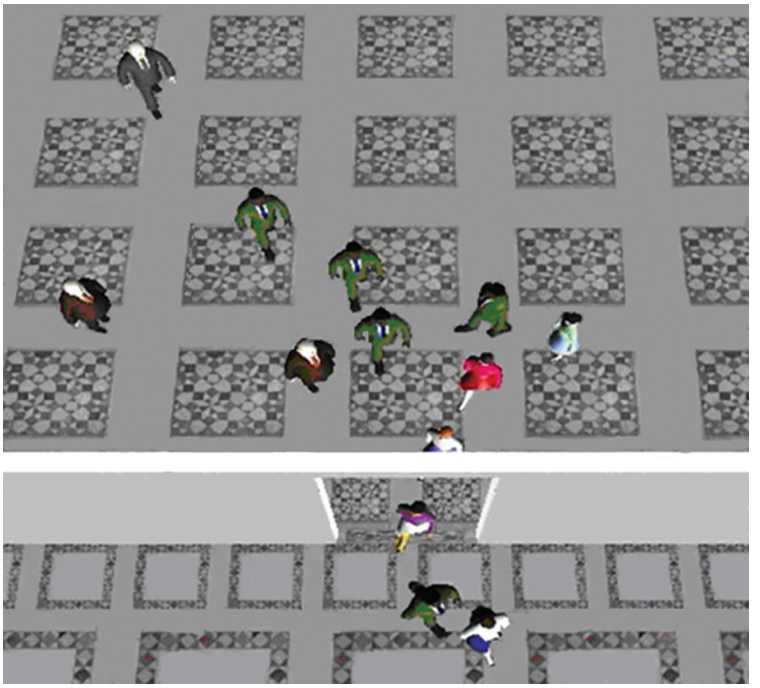


Figure 4. Neurotic, nonconscientious, and disagreeable agents (in green suits) display panic behavior.

Analysis

After collecting the participants' answers for all the videos, we organized the data for the adjectives. We classified each adjective by its question number, the simulation parameter, and the participants' answers to the corresponding question. We calculated the Pearson correlation (r) between the simulation parameters and the average of the subjects' answers for each question. For instance,

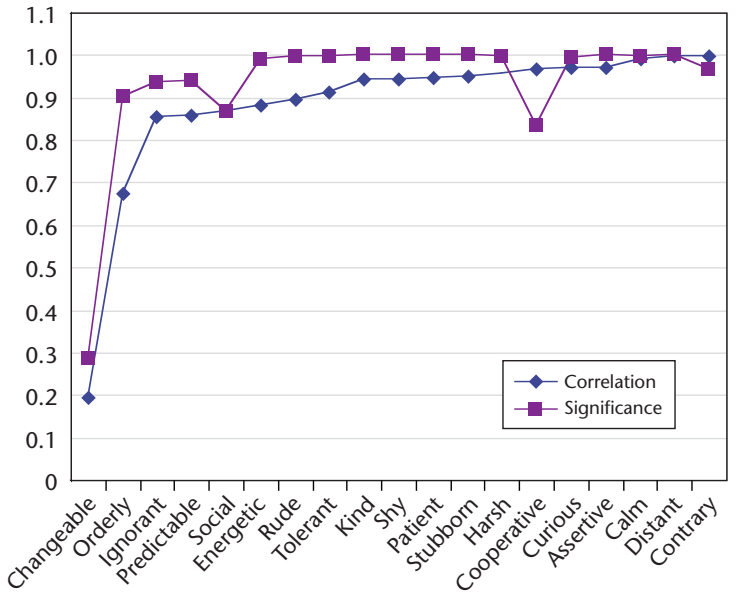


Figure 5. The correlation coefficients between the parameters and the subjects' answers for the descriptive adjectives (blue), and the significance values for the corresponding correlation coefficients (violet). Significance is low (<0.95) for changeable, orderly, ignorant, predictable, social, and cooperative.

eight questions included the adjective *assertive*, indicating a sample size of 8. We calculated the correlation coefficient between the parameters and the means of the participants' answers between these 16 values, eight for each group.

Table 2. Correlation coefficients and significance values for the 12 adjectives.

Adjective	Correlation	Significance
Changeable	0.199	0.288
Orderly	0.674	0.903
Ignorant	0.853	0.936
Predictable	0.870	0.938
Social	0.872	0.869
Energetic	0.882	0.992
Rude	0.897	0.997
Tolerant	0.912	0.998
Kind	0.943	1.000
Shy	0.945	1.000
Patient	0.948	1.000
Stubborn	0.950	1.000
Harsh	0.956	0.997
Cooperative	0.967	0.834
Curious	0.971	0.994
Assertive	0.971	1.000
Calm	0.988	0.999
Distant	0.998	1.000
Contrary	0.999	0.969

We grouped the relevant adjectives for each Ocean factor to assess the perception of personality traits. This evaluation was similar to the evaluation of adjectives, this time considering the questions for all the adjectives that corresponded to an Ocean factor. For instance, because openness is related to curiosity and ignorance, we took into account the answers for *curious* and *ignorant*. Again, we averaged the subjects' answers for each question. Then, we computed the correlation with the parameters and the mean throughout all the questions involving *curious* and *ignorant*.

To estimate the probability of having obtained the correlation coefficients by chance, we computed the correlation coefficients' significance. The significance is 1 – p, where p is the two-tailed probability, taking into account the sample size and the correlation value. Higher correlation and significance values suggest more accurate user perception.

Results and Discussion

Figure 5 shows the correlation coefficients and significance values for the adjectives; Table 2 shows the exact results. As the table shows, the significance is low (<0.95) for *changeable*, *orderly*, *ignorant*, *predictable*, *social*, and *cooperative*. For *changeable* and *orderly*, this is because of low correlation values. For *predictable*, *ignorant*, *social*, and *cooperative*, the correlation coefficients are high, but their significance is low because of the small sample size.

From the participants' comments, we determined that *changeable* is especially confusing. To understand why, consider the setting in which two groups of agents crossed each other. The participants identified the nonconscientious agents as rude but perceived them as persistent in their rudeness. This perception caused the participants to mark lower values for the question about changeability. The same problem held for predictable agents. One participant's comments suggested that if a person is in a rush, that person can be predicted to push others. However, a predictable agent has a higher correlation despite these comments, even though *predictable* implies the opposite of *changeable*. This meaning might be because of the relatively low significance for *predictable*. Participants perceived nonconscientious agents that cause congestion as less predictable, which indicates that changing right-preference and rude behavior decreases the perceived predictability.

Orderly is another weakly correlated adjective. Analyzing the results for each video, we found that agents in the evacuation drill scenario were orderly although they displayed panic behavior.

In these videos, even if the agents pushed each other and moved fast, some kind of order could be observed. This order was because of the crowd's smooth flow during evacuation. The crowd displayed collective synchrony, in which individuality was lost. Although people were impatient and rude, the overall crowd behavior appeared orderly.

We assigned the same goal to the entire crowd in the evacuation simulations because we aimed to observe disorganization locally. For instance, disorderly agents looked rushed; they pushed other agents and they didn't have solid preferences for selecting a direction when crossing another agent. However, they moved toward the same goal, which was to exit the building. The crowd would have appeared more disorderly if everyone ran in different directions and changed direction for no apparent reason.

Participants' answers suggest that they didn't recognize orderliness when the goal was the same for the whole crowd. On the other hand, in a scenario showing queuing behavior in front of a water dispenser, the participants could easily distinguish orderly versus disorderly people. Orderly agents waited at the end of the queue; disorderly agents rushed to the front. In this scenario, although the main goal was the same for all the agents (drink water), there were two distinguishable groups that acted differently.

Figure 6 and Table 3 show the correlation coefficients and their significance for the Ocean parameters. We computed these values by taking into account all the relevant adjectives for each Ocean factor. As the figure and table show, all the coefficients have high significance, with a probability of less than 0.5 percent of occurring by chance ($p < 0.005$). The significance is high because we took into account all the adjectives describing a personality factor, thereby achieving a sufficiently large sample.

The correlation coefficient for conscientiousness is comparatively low, showing that the participants correctly perceived only approximately 44 percent of the traits ($r^2 \approx 0.44$). To understand why, consider the relevant adjectives: *orderly*, *predictable*, *rude*, and *changeable*. Low correlation values for *orderly* and *changeable* reduce the overall correlation. If we consider only *rude* and *predictable*, the correlation increases by 18.6 percent. The results suggest that people can observe the politeness aspect in short-term crowd behavior settings more easily than the organizational aspects. This observation also explains why the perception of agreeableness correlates highly with the actual parameters.

Figure 6 and Table 3 also show that the participants perceived neuroticism the best. In this study, we've considered only neuroticism's calmness aspect, which is tested in emergency settings and building evacuation scenarios.

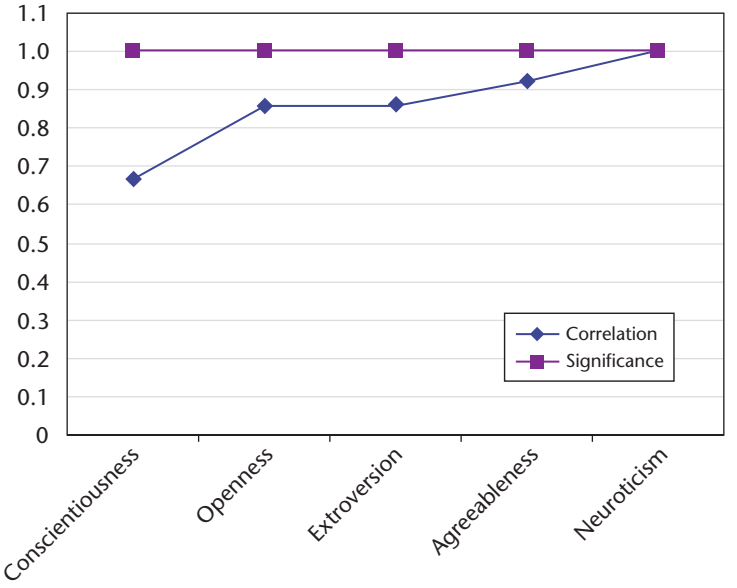


Figure 6. The correlation coefficients between actual parameters and subjects' answers for the Ocean factors (blue), and the two-tailed probability values for the corresponding correlation coefficients (violet). All the coefficients have high significance.

Table 3. The correlation coefficients and the significance values for the Ocean factors.

Factor	Correlation	Significance
Conscientiousness	0.665	1.000
Openness	0.859	0.999
Extroversion	0.860	1.000
Agreeableness	0.922	1.000
Neuroticism	0.990	0.999

Our results are promising; they indicate a high correlation between our parameters and the participants' perception of them. The low correlation for some adjectives is due to the terms' ambiguity.

Unlike the low-level parameter tuning process in previous research (see the sidebar), we let our users select from higher-level concepts related to human psychology. Our approach frees users from understanding the underlying methodologies used in HiDAC. Our mapping also decreases the number of parameters to set, from 13 to 5. Using a personality model let us move the user's focus to the agents' character instead of behavioral parameters, while providing us with a somewhat widely accepted structure for describing character. Certainly, you could create an interface that lets

Related Work in Crowd Simulations

Crowd simulation research has evolved from creating reactive techniques to implementing crowds consisting of more complex agents. Reactive methods are limited; they don't present any knowledge representation, learning ability, reasoning, or individual differences in the agents. For instance, flocking systems are rule-based and specify an animation as a distributed global motion with a local tendency.[1]

On the other hand, systems with cognitive control involve reasoning and planning to accomplish long-term tasks, and they concentrate on achieving full autonomy. A notable step toward creating more intelligent agents was Xiaoyuan Tu and Demetri Terzopoulos's artificial-life simulation, which equipped artificial fishes with synthetic vision and perception of the environment, as well as behavior and learning centers.[2] Soraia Musse and Daniel Thalmann proposed a crowd behavior model that implemented group interrelationships and introduced a multiresolution collision method specific to crowd modeling.[3] Wei Shao and Terzopoulos introduced a complex pedestrian animation system that combined rule-based and cognitive models; it incorporated perceptual, behavioral, and cognitive control components.[4]

Several studies have integrated emotion and personality models and roles into the simulation of autonomous agents, representing the individual differences through psychological states. Arjan Egges and his colleagues studied the simulation of the personality, emotions, and moods for conversational virtual humans.[5] Taihua Li and colleagues proposed a framework that, like ours, uses the Ocean (openness, conscientiousness, extroversion, agreeableness, and neuroticism) model of personality to define and formulate a pedagogical agent in a social learning environment.[6] However, these studies focused on individual agents, not crowds.

Only recently have researchers studied the perception of crowd variety. Christopher Peters and his colleagues evaluated pedestrians' perception.[7] They determined how the orientation and context rules for characters in static scenes affect perceived plausibility. Rachel McDonnell and her colleagues analyzed the perceptual impact of the cloning of virtual characters for simulating large crowds.[8]

References

1. C. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavior Model," *Proc. Siggraph*, ACM Press, 1987, pp. 25–34.
2. X. Tu and D. Terzopoulos, "Artificial Fishes: Physics, Locomotion, Perception, Behavior," *Proc. Siggraph*, ACM Press, 1994, pp. 43–50.
3. S.R. Musse and D. Thalmann, "A Model of Human Crowd Behavior," *Proc. Eurographics Workshop Computer Animation and Simulation*, Springer, 1997, pp. 39–51.
4. W. Shao and D. Terzopoulos, "Autonomous Pedestrians," *Graphical Models*, vol. 69, nos. 5–6, 2007, pp. 246–274.
5. A. Egges, S. Kshirsagar, and N. Magnenat-Thalmann, "A Model for Personality and Emotion Simulation," *Proc. Knowledge-Based Intelligence Information and Eng. Systems*, LNCS 2773, Springer, 2003, pp. 453–461.
6. T. Li et al., "Modelling Personality, Emotion, and Mood for a Pedagogical Agent," *Proc. 25th IASTED Int'l Multiconference: Artificial Intelligence and Applications* (AIAP 07), ACTA Press, 2007, pp. 272–277.
7. C. Peters et al., "Crowds in Context: Evaluating the Perceptual Plausibility of Pedestrian Orientations," *Proc. Eurographics, Short Papers*, Eurographics Assoc., 2008, pp. 33–36.
8. R. McDonnell et al., "Clone Attack! Perception of Crowd Variety," *ACM Trans. Graphics*, vol. 27, no, 3, 2008, article 26.

users create subgroups based on a set of adjectives instead of personality traits, but it would increase the number of parameters to set. Also, psychology and research on autonomous agents has linked personality models to other psychological, sociological, and cognitive models. Integrating a personality model into a crowd simulator will let us expand our simulator and explore how these other models affect crowd simulations.

We certainly could have used other psychological models. Autonomous-agent research has investigated emotion models. Future research might include adding emotion to the agents, but whereas personality is a behavior pattern (extended through time), emotions change according to the agent state and the situation. Emotions must evolve through the simulation and not be set by the animator. Certainly, personality affects emotional tendency and provides a foundation. Because personality is a behavior pattern, it might help a character's observers develop a sense of knowing that character. Thus, characters might become individuals instead of just another collection of anonymous computer characters.

Acknowledgments

The research described in this article was initiated while Funda Durupınar was visiting the University of Pennsylvania's Center for Human Modeling and Simulation. The Scientific and Technological Research Council of Turkey supported this research under International PhD Research Fellowship Programme 2214 and projects EEE-AG 104E029 and 105E065. The Spanish Government partially funded this research under grant TIN2010-20590-C01-01.

References

1. J.S. Wiggins, *The Five-Factor Model of Personality: Theoretical Perspectives*, Guilford Press, 1996.
2. L.R. Goldberg, "An Alternative 'Description of Personality': The Big-Five Factor Structure," *J. Personality and Social Psychology*, vol. 59, no. 6, 1992, pp. 1216–1229.
3. N. Pelechano, J.M. Allbeck, and N.I. Badler, "Controlling Individual Agents in High-Density Crowd Simulation," *Proc. ACM Siggraph/Eurographics Symp. Computer Animation* (SCA 07), ACM Press, 2007, pp. 99–108.
4. F. Durupınar et al., "Creating Crowd Variation with the Ocean Personality Model," *Proc. 7th Int'l Joint Conf. Autonomous Agents and Multiagent Systems* (AAMAS 08), Int'l Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 1217–1220.
5. E.T. Hall, *The Hidden Dimension*, Anchor Books, 1966.
6. K.C. McLean and M. Pasupathi, "Collaborative Narration of the Past and Extroversion," *J. Research in Personality*, vol. 40, no. 6, 2006, pp. 1219–1231.

Funda Durupınar *received her PhD from the Department of Computer Engineering at Bilkent University. Her research interests include crowd simulation with heterogeneous behaviors that incorporate psychological aspects of agents. Durupınar has an MS in computer engineering from Bilkent University. Contact her at fundad@cs.bilkent.edu.tr.*

Nuria Pelechano *is an associate professor of Llenguatges i Sistemes Informàtics at the Universitat Politècnica de Catalunya, where she's a member of the Moving and Event-Lab groups. Her research interests include simulation of crowds with heterogeneous behaviors, real-time 3D graphics, and human-avatar interaction in virtual environments. Pelechano has a PhD in computer and information science from the University of Pennsylvania. Contact her at npelechano@lsi.upc.edu.*

Jan M. Allbeck *is an assistant professor of computer science at George Mason University. Her research interests are at the crossroads of animation, artificial intelligence, and psychology in the pursuit of simulating humans, including functional, heterogeneous crowds. Allbeck has a PhD in computer and information science from the University of Pennsylvania. Contact her at jallbeck@gmu.edu.*

Uğur Güdükbay *is an associate professor in Bilkent University's Department of Computer Engineering. His research interests include human modeling and animation, crowd simulation and visualization, and physically based modeling. Güdükbay has a PhD in computer engineering and information science from Bilkent University. He's a senior member of IEEE and the ACM. Contact him at gudukbay@cs.bilkent.edu.tr.*

Norman I. Badler *is a professor of computer and information science at the University of Pennsylvania. He also directs the university's SIG Center for Computer Graphics and Center for Human Modeling and Simulation. His research interests center on computational connections between language and action. Badler has a PhD in computer science from the University of Toronto. Contact him at badler@seas.upenn.edu.*

Populations with Purpose

Weizi Li and Jan M. Allbeck

Laboratory for Games and Intelligent Animation
George Mason University
4400 University Drive, MSN 4A5
Fairfax, VA 22030
http://cs.gmu.edu/~gaia/

Abstract. There are currently a number of animation researchers that focus on simulating virtual crowds, but few are attempting to simulate virtual populations. Virtual crowd simulations tend to depict a large number of agents walking from one location to another as realistically as possible. The virtual humans in these crowds lack higher purpose. They have a virtual existence, but not a virtual life and as such do not reasonably depict a human population. In this paper, we present an agent-based simulation framework for creating virtual populations endowed with social roles. These roles help establish reasons for the existence of each of the virtual humans. They can be used to create a virtual population embodied with purpose.

Keywords: Crowd Simulation, Social Roles

1 Introduction

Military training and other applications desire simulations that establish normal human behavior for an area. Once normalcy is established, observers can be trained to recognize abnormal and possibly dangerous behaviors. This requires the simulation of longer periods of time including different times of day. The problem is how to select reasonable, purposeful behaviors for a population for such periods of time. Roles are, in part, expected patterns of behavior and therefore seem like an intuitive feature for authoring these scenarios. Furthermore, role switching would enable plausible variations in behaviors throughout a day, but requires mechanisms to initiate the switching. While admittedly not comprehensive, role switching based on schedules, reactions, and needs, seems like a good starting point.

This paper describes an agent-based simulation framework for creating virtual populations endowed with various social-psychological factors including social roles. These roles help establish reasons for the existence of the virtual humans and can be used to create a virtual population embodied with purpose. Human decisions and the behaviors that result, stem from a complex interplay of many factors. The aim of this work is not to try to replicate all of these factors. We have focused on social roles because they are so heavily linked to meaningful behaviors. From this starting point we have included other factors that are

linked to role and that can add reasonable variability to behaviors while still maintaining a framework where scenarios can be feasibly authored, modified, and controlled.

In addition, our framework focuses on higher level control mechanisms as opposed to lower level animation implementations. It also links roles and role switching to different action types such as reactions, scheduled actions, and need-based actions. As such the authoring of roles is largely just associating a set of these actions with a role. The techniques and methodologies used are adopted from a number of research disciplines including multi-agent systems, social psychology, ontologies, and knowledge representations, as well as computer animation.

2 Related Work

Crowd simulation research has been approached from several different perspectives. Some research groups are addressing how to simulate large crowds mainly through focusing on global path planning and local collision avoidance [18, 17, 19]. The behaviors in these simulations are for the most part limited to locomotion maneuvers.

Work has also been done on adding contextual behavioral variations through spatial patches [26, 11, 23]. The common theme in these works is defining regions in the virtual world and associating these regions with certain behaviors and interactions. The computer game, *The Sims*, might also be considered to incorporate spatially dependent behavior [25]. For example, if an agent is hungry and near a refrigerator, even if not being explicitly directed by the player, he would eat. While these certainly add richness to the virtual world, they still fall short of embodying consistent reasonable behaviors with purpose.

Most of the works described so far included few or no social psychology factors. There have, however, been some that do. The work of Pelechano et al. included the concept of role and other psychological factors, but the roles were limited to leaders and followers which along with the other factors influence only the navigation behavior of agents [20]. In [16], Musse and Thalmann describe a crowd simulation framework that includes sociological factors such as relationships, groups, and emotion, but again the behaviors are centered around locomotion actions.

In [21] Shao and Terzopoulos describe a virtual train station. Here they classified their autonomous pedestrians into a few categories, including commuters, tourists, performers, and officers. Each type of character is then linked to hand coded action selection mechanisms. Similarly in [27] the authors introduce a decision network which addresses agent social interaction based on probabilities and graph theory, however action selection is still manually coded.

Some research groups have worked directly on incorporating roles into virtual humans. Hayes-Roth and her collaborators were one of the first research groups to develop virtual roles [8]. Their interactive intelligent agent was instilled with the role of bartender and a set of actions were defined such that the user's

expectations would be met. There was, however, no switching of roles for this character and the behaviors were only related to communication acts.

Most recently work by Stocker et al. introduced the concept of priming for a virtual agent [22]. Here agents are primed for certain actions based on the other agents and events around them. While they do not address roles specifically, this concept of priming is somewhat similar to the role switching behaviors we will describe in this paper.

3 Approach Overview

In this section, we provide an overview of our approach and describe the social psychological models on which it is founded, including a definition of roles, factors affecting role switching, and action types.

3.1 Definition of Role

A role is *the rights, obligations, and expected behavior patterns associated with a particular social status* [1]. Ellenson's work [6] notes that each person plays a number of roles. Taking into consideration these descriptions as well as discussions from other social psychologists [13, 4], we conclude that roles are patterns of behaviors for given situations or circumstances. Roles can demand certain physical, intellectual, or knowledge prerequisites, and many roles are associated with social relationships.

3.2 Role Switching

People's priorities are set by a number of interplaying factors, including emotions, mood, personality, culture, roles, status, needs, perceptions, goals, relationships, gender, intelligence, and history, just to name a few. In this work, we have chosen a few factors that are related to roles and role switching that we believe will help endow virtual humans with meaningful, purposeful behaviors.

Switching from one role to another can be linked to time, location, relationships, mental status, and needs (See Figure 1). For example, one can imagine someone switching to a *businessman* role as the start of the work day approaches or as he enters his office or when he encounters his boss. Also, someone may need to shop for groceries to provide for his family. The *shopping* behavior would stem from a need and cause a switch in role to *shopper*. Elements of mental status, such as personality traits, can impact the selection and performance of these roles. For example, a non-conscientious person might not shop for groceries even if the need exists.

Action and role selection is further affected by a filtration of proposed actions according to an agent's *Conventional Practice* and *World Knowledge* [10, 4, 3]. *Conventional Practice* is a set of regulations and norms that each individual in the society should obey. *World Knowledge* indicates that certain physical, intellectual, and knowledge elements are required for specific roles.

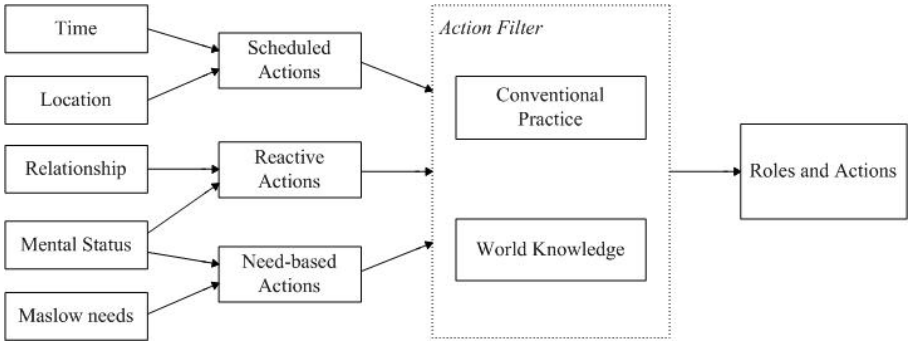


Fig. 1. System Diagram

Another perspective from which to consider selection is to examine what triggers various behaviors. Some actions are planned for such as going to work or attending a meeting. These actions, called *Scheduled* actions, tend to establish a person's daily routine and are often heavily coupled with their roles. Other actions are not so predetermined. Some actions arise to fulfill needs. Among these needs might be those depicted in *Maslow's Hierarchy of Needs*, including, food, water, excretion, friendship, family, employment, and creativity [12]. These type of actions, called *Need-based* actions, can also be linked to roles. For example, in order to maintain employment safety, a businessman might need to contact his clients are on regular basis. Still other actions, *Reactive actions*, are responses to agents, objects, or events in the world. Who and what we react to is at least partially determined by our roles. If we see a friend or co-worker as we are walking to work, we are likely to stop and greet them.

While we cannot claim that these three types of actions make a complete categorization of all behaviors, we believe that they can encompass a wide range of behaviors and provide strong ties to roles and purposeful behaviors. Another key factor is the ability to easily author or initiate these behaviors. Each requires a finite, straightforward amount of data:

Scheduled actions: $Sch = \langle P, A, L, T \rangle$, where P is the performer (an individual or group), A is the action to be performed (simple or complex), L is the location where the action is to be performed (based on an object or a location), and T is an indication of the time (i.e. start time and duration).

Reactive actions: $Rea = \langle P, S, A \rangle$, where P is the performer (an individual or group), S is the stimulus (an object, type of object, person, location, event, etc) and A is the action to be performed (simple or complex)

Need-based actions: $Nee = \langle P, N, D, C \rangle$, where P is the performer (an individual or group), N is the name of the need, D is the decay rate, and C is a set of tuples $\langle A, O, F \rangle$, where A is the action to be performed (simple or complex), O is a set of object types, and F is the fulfillment rate.

4 Implementation

The work presented here extends work previously reported [2]. Previous work included an implementation of different action types and very rudimentary roles, but was limited to a single role per character which is not realistic for day scale simulations.

To continue to ensure scenario authoring is feasible, we have extended our data-driven approach where all of the vital scenario data is stored in a database. This includes information about each agent such as conventional practices, mental status, world knowledge, and role sets. We also store a mapping of the relationships between agents. Information about the world, objects in the world, and actions are also stored in the database. This includes the specification of schedules, needs, and reactions. As such, scenarios can be authored entirely through the database and without any coding. In addition, our framework is built on an existing crowd simulator that provides navigation and collision avoidance for the agents [19].

We have extend the previous implementation including now a much richer definition of role. The most important component of a role is a set of actions [13]. This action set corresponds to the conventional practices associated with the role. These actions may be scheduled, need-based, or reactive actions as mentioned in the previous section. Furthermore, these actions may also be linked to parameters such as location, object participants, start-times, and durations. Interpersonal roles are also associated with relationships, which are a simple named linking of agents. Among agent parameters is a set of capabilities corresponding to actions that they can perform. These capabilities form the foundation of an agent's world knowledge.

As described in the previous section, factors influencing role selection include, time, location, relationships, mental status, and needs. In this section, we will describe the implementation of each of these factors and how they have been incorporated into the three action types. We will also discuss how actions and roles are further filtered by conventional practice and an agent's world knowledge (See Figure 1).

4.1 Action Types

A large part of the definition of a role includes a pattern of behaviors. Our framework associates each role with a set of actions. These actions can be of any of the three types of actions described earlier, namely *Scheduled*, *Reactive*, or *Need-based*.

Scheduled Actions Scheduled actions include time and location parameters [2] and can be used to establish an agenda for a day. Some roles are directly associated with scheduled actions. For example, a businessman may be scheduled to work in his office from 9am to 5pm. As 9am approaches, the framework will initiate processing of the scheduled work in office action and send the character to his office and the businessman role.

However, if an agent does not have a scheduled action to perform, they will perform a default action that is associated with their current role. Generally default actions are the actions most often performed by that role. For example, a businessman or administrator might work in an office. A shopkeeper might attend to the cash register. Just as in real life, scheduled actions can be suspended by higher priority need-based and reactive actions.

Need-based Actions Need-based actions are merely database entries associating a decay rate, actions, objects, and a fulfillment quotient. The examples described in this paper are based on *Maslow's Hierarchy of Needs* [12]. Conceptually, there is a reservoir that corresponds to each need for each agent. Currently the initial level of each reservoir is set randomly at the beginning of the simulation. At regular intervals, the reservoirs are decreased by the specified decay rates. When the level of a reservoir hits a predetermined threshold, the fulfilling action is added to the agent's queue of actions. Its priority will increase as the reservoir continue decreases eventually its priority will greater than all actions on the queue. Then the agent will perform the action, raising the level of the reservoir.

We have chosen to use *Mental Status* as an influence on needs (and reactions), because social scientists have linked it to roles and we feel it adds plausible variability. It includes several factors, but we focus on personality as it addresses an individual's long-term behavior. There are several psychological models of personality. One of the most popular is the Five-Factor or OCEAN model [24]. The five factors are: *Openness* (i.e. curious, alert, informed, perceptive), *Conscientiousness* (i.e. persistent, orderly, predictable, dependable, prompt), *Extroversion* (i.e. social adventurous, active, assertive, dominant, energetic), *Agreeableness* (i.e. cooperative, tolerant, patient, kind), and *Neuroticism* (i.e. oversensitive, fearful, unadventurous, dependent, submissive, unconfident).

We have based our implementation of personality on the work of Durupinar et al. [5]. An agent's personality π is a five-dimension vector, where each is represented by a personalty factor Ψ_i. The distribution of the personality factors in a populations of individuals is modeled by a Gaussian distribution function with mean μ_i and standard deviation σ_i:

$$\begin{aligned} &\pi = \langle \Psi^O, \Psi^C, \Psi^E, \Psi^A, \Psi^N \rangle \\ &\Psi^i = N(\mu_i, \sigma_i^2), for\, i \in O, C, E, A, N \\ &\text{where} \quad \mu \in [-10, 10],\, \sigma \in [-2, 2] \end{aligned}$$

Since each factor is bipolar, Ψ can take both positive and negative values. For instance, a positive for *Extroversion*, E+, means that the individual has extroverted character; whereas a negative value means that the individual is introverted. In Section 4.1, we will describe how personality dimensions affect the decay rates of need reservoirs, creating reasonable variations in behaviors from person to person.

Needs and priorities differ from person to person. We represent this variation by linking the personality traits just described with needs. This is, of course, a massive oversimplification, but one that leads to plausible variations. Table 1 shows our mapping from Maslow needs to OCEAN personality dimensions. It should be noted that just the personality dimension is represented, not the valence of the dimension. For example, neuroticism is negatively correlated with needs for security of employment and family. This mapping was formulated by examining the adjectives associated with the personality dimensions and the descriptions of the Maslow needs.

Reservoir Descriptions	**Personality Traits**
problem solving, creativity, lack of prejudice	O, A
achievement, respect for others	O, A
friendship, family	E
security of employment, security of family	C, N
water, food, excretion	

Table 1. Mapping between Maslow need reservoirs and personality dimensions.

In this work, we represent the decay rate as β. For example, $\beta^{friendship}$ indicates the decay rate of the *friendship* reservoir. For Maslow based needs, we also apply a correlation coefficient r which ranges from (0,1] to represent the relationship between decay rate and personality traits. More precisely, $r_{E,friendship}$ represents how strongly the *Extroversion* trait and *friendship*'s decay rate correlated. The closer r gets to 1 the faster reservoir empties (i.e. the decay rate is high indicating the agents strong need for friendship). Since a need can be affected by more than one personality trait, for those needs marked with multiple personality traits we assign weights to each one: $\omega_{E,friendship}$ meaning the impact of *Extroversion* on *friendship*. If a need has more than one trait's influence, its summation of ω should be equal to one and for simplicity in this work we assume each personality trait contributes the same weight.

Consequently, for ith agent the decay rates and their relationship with personality traits are shown below (here we list two examples):

***security of employment (se)*:**
$\beta_i^{se} = (\omega_{C,se}|r_{C,se} \times \Psi_i^C| + \omega_{N,se}|r_{N,se} \times \Psi_i^N|) \times 0.1$
where $\omega_{C,se} + \omega_{N,se} = 1$, $\beta_i^{se} \propto C$, N and $\beta_i^{se} \in [0, 10]$

***friendship (fr)*:**
$\beta_i^{fr} = (\omega_{E,fr}|r_{E,fr} \times \Psi_i^E|) \times 0.1$
where $\omega_{E,fr} = 1$, $\beta_i^{fr} \propto E$ and $\beta_i^{fr} \in [0, 10]$

Reactive Actions As a simulation progresses, agents make their way through the virtual world, attempting to adhere to their schedules and meet their needs. In doing so, agents encounter many stimuli to which a reaction might be warranted. Reactions play an important part in our implementation of roles. In [7], Merton states that a person might switch roles as a response to those around him. Relationships are a major impetus for reactive role switching. For example, if two agents encounter each other and are linked by a relationship such as friendship, they will switch to the friend role.

Since reactive actions must be performed soon after the stimulus is encountered, they are given a higher priority and generally result in the suspension of whatever other action might be being performed, though this is not always the case. For example, if an agent is fulfilling a high priority need, then they might not react to the people and things around him.

The duration of responses can vary according to the activity and characteristics of the agents. For example, an agent that is hurrying off to work or who is introverted, may not linger as long on the street to greet a friend, as someone strolling home from work or an extrovert would.

4.2 Action Filter

Once a set of actions and roles have been purposed, some may be eliminated due to conventional practice constraints or an agent's lack of necessary world knowledge.

Conventional Practice Social science researchers believe that when an agent plays a role in a given organizational (or social) setting, he must obey *Conventional Practices*, meaning *behavioral constraints* and *social conventions* (e.g. the businessman must obey the regulations that his company stipulates) [15, 14]. To be more specific, behavioral constraints are associated with the following factors: responsibilities, rights, duties, prohibitions and possibilities [15]. Role hierarchies include conventional roles (e.g. *citizen, businessman, mailman*) and interpersonal roles (e.g. *friends, lovers, enemies*). Figure 2 shows part of the conventional role hierarchy that we designed according to the taxonomy presented in [9]. We have linked each conventional practice norm with an impact factor (range [0, 1]) which reflects how strongly these norms are imposed on certain roles. Having 1 as an impact factor would indicate that it is the most powerful norm. For convenience, we have set all impact factors in our current simulations to be high enough to indicate that every agent would obey not only the conventional practice of current professions but also those inherited from upper levels of the hierarchy. However, users could choose whatever impact factors they would like according to the behaviors that they desire.

World Knowledge Some roles have physical or intellectual requirements and these requirements may be difficult to obtain. Also some people are just naturally more physically or intellectually gifted or have more talent in an area than

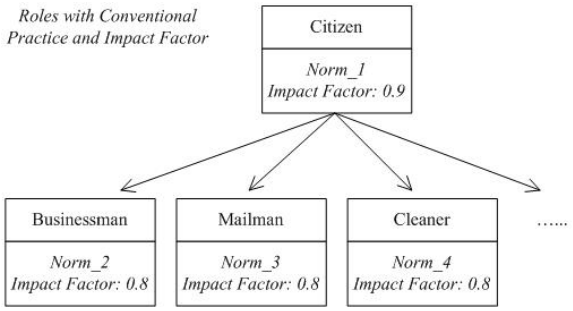


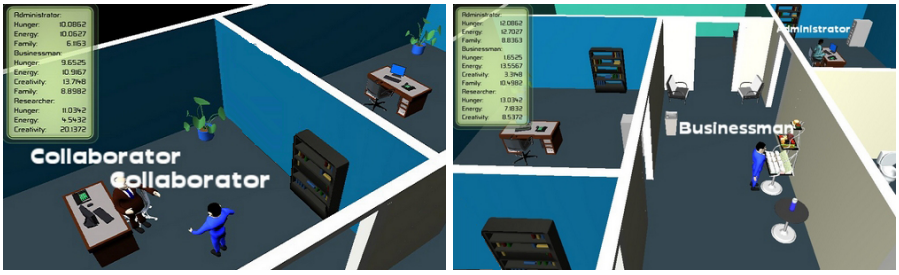
Fig. 2. Role Hierarchy with Conventional Practice and Impact Factor

others. These factors can put limitations on what roles a person can take on [4]. We represent world knowledge as capabilities. Agent capabilities are the set of actions that the agent can perform. Actions are categorized and placed in a hierarchy to lessen the work of assigning capabilities to agents and also checking to ensure that agents meet the capability conditions before performing an action.

5 Examples

To explore the effects of roles and more precisely role switching on virtual human behaviors, we have authored a typical day in a neighborhood. As with real humans, each virtual human is assigned a set of roles. Figure 3 demonstrates one agent taking on the role of *businessman* as he enters his office building (i.e. location-based role switching). Two other agents react to seeing each other by switching to *friend* roles (i.e. relationship-based role switching). Another agent reacts to trash in the street by starting her *cleaner* role (i.e. behavior selection-based role switching). These first examples of agents going to work demonstrate how time, location, and behavior selection impact role switching. They focus on role transitions caused by scheduled and reactive actions. The following office examples concentrate more on need-based actions. The top image of Figure

Fig. 3. Locations, relationships and behavior selection affect role switching.

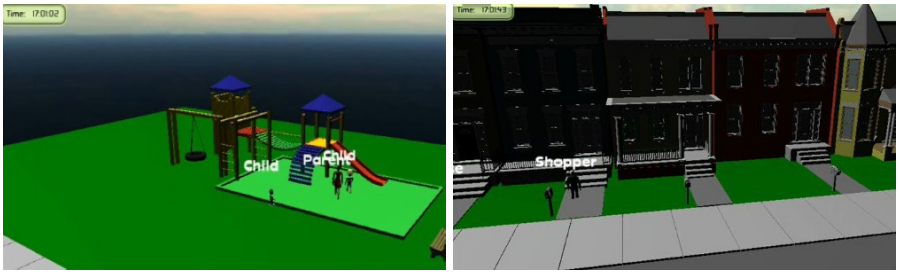


(a) (b)

Fig. 4. (a) The businessman's *creativity* need prompts him to speak to a co-worker, causing both to switch to *collaborator* roles and replenish their creativity reservoirs. (b) The businessman's role remains while eating to refill his hunger reservoir..

4a shows that the businessman's *creativity* reservoir is approaching the critical threshold (i.e. 2). When it reaches the threshold, he suspends his current action and starts a conversation with his co-worker. This exchange of ideas causes the *creativity* reservoirs of both men to refill.

Figure 4b shows that not all need-based actions cause role switching. The hunger need is associated with the role of being human, because *businessman* is a descendant of this role in the role hierarchy there is no need to switch. The final scene depicts a late afternoon in our neighborhood. Figure 5a shows a *businesswoman* becoming a *parent* when playing with her children. In Figure 5b, a man was headed home from work, but before he could reach his door, the *security of family* need prompted him to switch his role and instead he go to the grocery store.



(a) (b)

Fig. 5. (a) A businesswoman switches her role to parent when she spends time with her children. (b) A businessman is heading home after work, when his *security of family* need preempts this action and switches his role to *shopper*.

6 Discussion

In this paper, we have presented a framework for instilling virtual humans with roles and role switching to produce more typical virtual worlds where people's behaviors are purposeful. The methods presented are based on social psychology models and focus on approaches that facilitate authoring and modifications. As people's complicated lives rarely allow them to embody just a single role during the course of a day, role switching is important to creating reasonable virtual human behaviors. The framework presented can also be used to include abnormal behaviors. For example, one could author a subversive role for a character that includes reacting to pedestrians by robbing them or includes a strong need for drugs and alcohol.

There are numerous possible extensions to this work. First, we could illustrate the dynamics that stem from status hierarchies by experimenting with the concepts of power scale and social distance. We might also address situations where multiple roles could be adopted. For example, a man being approached by both his boss and his child. Here, social power scales and social distance might result in different social threats which would cause one role to be favored over the other. Finally, we could focus on agents that learn role definitions by observing the behaviors of others, enabling each agent to have customized definitions based on their own experiences.

Acknowledgments

Partial support for this effort is gratefully acknowledged from the U.S. Army SUBTLE MURI W911NF-07-1-0216. We also appreciate donations from Autodesk.

References

1. Webster's College Dictionary. Random House (1991)
2. Allbeck, J.M.: CAROSA: A tool for authoring NPCs. Motion in Games pp. 182–193 (2010)
3. Barker, R.G.: Ecological Psychology: Concepts and methods for studying the environment of human behavior. Stanford University Press (1968)
4. Biddle, B.J.: Role Theory: Concepts and Research. Krieger Pub Co (1979)
5. Durupinar, F., Pelechano, N., Allbeck, J.M., Gudukbay, U., Badler, N.I.: How the OCEAN personality model affects the perception of crowds. IEEE Computer Graphics and Applications 31(3), 22–31 (2011)
6. Ellenson, A.: Human Relations. Prentice Hall College Div; 2 edition (1982)
7. Fan, J., Barker, K., Porter, B., Clark, P.: Representing roles and purpose. In: International Conference on Knowledge Capture (K-CAP). pp. 38–43. ACM (2001)
8. Hayes-Roth, B., Brownston, L., van Gent, R.: Readings in Agents, chap. Multi-agent Collaboration in Directed Improvisation, pp. 141–147. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)

9. Hewitt, J.P.: Self and society: A symbolic interactionist social psychology. Oxford, England (1976)
10. Ickes, W., Knowles, E.: Personality, Roles, and Social Behavior. Springer; 1 edition (1982)
11. Lee, K.H., Choi, M.G., Lee, J.: Motion patches: Building blocks for virtual environments annotated with motion data. In: Proceedings of the 2006 ACM SIGGRAPH Conference. pp. 898–906. ACM, New York, NY, USA (2006)
12. Maslow, A.: A theory of human motivation. Psychological Review 50, 370–396 (1943)
13. McGinnies, E.: Perspectives on Social Behavior. Gardner Press, Inc. (1994)
14. Merton, R.K.: Social Theory and Social Structure. Free Press (1998)
15. Moulin, B.: The social dimension of interactions in multiagent systems. Agents and Multi-agent Systems, Formalisms, Methodologies, and Applications 1441/1998 (1998)
16. Musse, S.R., Thalmann, D.: A model of human crowd behavior: Group interrelationship and collision detection analysis. In: Workshop Computer Animation and Simulation of Eurographics. pp. 39–52 (1997)
17. Narain, R., Golas, A., Curtis, S., Lin, M.C.: Aggregate dynamics for dense crowd simulation. In: Proceedings of the 2009 ACM SIGGRAPH Asia Conference. pp. 122:1–122:8. ACM, New York, NY, USA (2009)
18. Ondřej, J., Pettré, J., Olivier, A.H., Donikian, S.: A synthetic-vision based steering approach for crowd simulation. In: Proceedings of the 2010 ACM SIGGRAPH 2010 Conference. pp. 123:1–123:9. ACM, New York, NY, USA (2010)
19. Pelechano, N., Allbeck, J.M., Badler, N.I.: Controlling individual agents in high-density crowd simulation. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 99–108. Eurographics Association (2007)
20. Pelechano, N., O'Brien, K., Silverman, B., Badler, N.I.: Crowd simulation incorporating agent psychological models, roles and communication. In: First International Workshop on Crowd Simulation. pp. 21–30 (2005)
21. Shao, W., Terzopoulos, D.: Autonomous pedestrians. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 19–28. ACM, New York, NY, USA (2005)
22. Stocker, C., Sun, L., Huang, P., Qin, W., Allbeck, J., Badler, N.: Smart events and primed agents. Intelligent Virtual Agents 6356, 15–27 (2010)
23. Sung, M., Gleicher, M., Chenney, S.: Scalable behaviors for crowd simulation. Computer Graphics Forum 23(3), 519–528 (2004)
24. Wiggins, J.: The Five-Factor Model of Personality: Theoretical Perspectives. The Guilford Press, New York (1996)
25. Wright, W.: The Sims (2000)
26. Yersin, B., Maïm, J., Pettré, J., Thalmann, D.: Crowd patches: Populating large-scale virtual environments for real-time applications. In: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games. pp. 207–214. ACM, New York, NY, USA (2009)
27. Yu, Q., Terzopoulos, D.: A decision network framework for the behavioral animation of virtual humans. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 119–128. Eurographics Association (2007)

ADAPT: The Agent Development and Prototyping Testbed

Alexander Shoulson* Nathan Marshak† Mubbasir Kapadia‡ Norman I. Badler§

University of Pennsylvania, Philadelphia, PA, USA

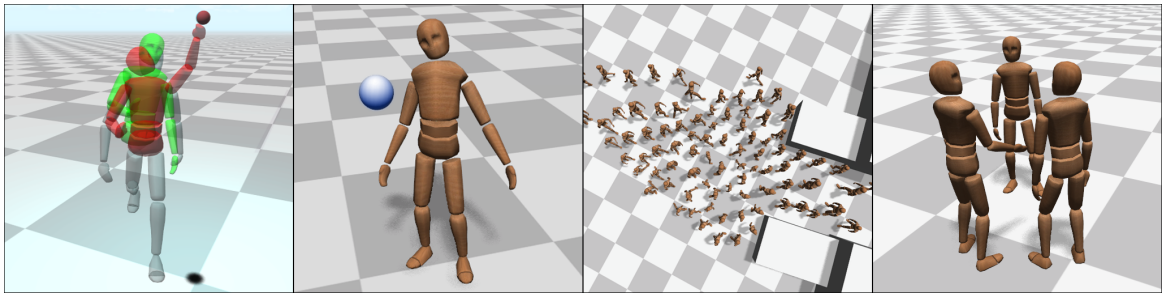


Figure 1: *Demonstrating the capabilities of ADAPT. Visualizing multiple choreographers that blend to produce a pose for the display model; an agent reacting to the impact force of a ball; a crowd of* 100 *agents resolving a bottleneck; three characters engaged in a conversation.*

Abstract

We present ADAPT, a flexible platform for designing and authoring functional, purposeful human characters in a rich virtual environment. Our framework incorporates character animation, navigation, and behavior with modular interchangeable components to produce narrative scenes. Our animation system provides locomotion, reaching, gaze tracking, gesturing, sitting, and reactions to external physical forces, and can easily be extended with more functionality due to a decoupled, modular structure. Additionally, our navigation component allows characters to maneuver through a complex environment with predictive steering for dynamic obstacle avoidance. Finally, our behavior framework allows a user to fully leverage a character's animation and navigation capabilities when authoring both individual decision-making and complex interactions between actors using a centralized, event-driven model.

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

Keywords: Virtual Humans, Character Animation, Behavior Authoring, Crowd Simulation

1 Introduction

Animating interacting virtual humans in real-time is a complex undertaking, requiring the solution to numerous tightly coupled problems such as steering, path-finding, full-body character animation (e.g. locomotion, gaze tracking, and reaching), and behavior authoring. This complexity is greatly amplified as we increase the number and degree of sophistication of characters in the environment. Numerous solutions for character animation, navigation, and behavior design exist, but these solutions are often tailored to specific applications, making integration between systems arduous. Integrating multiple character control architectures requires a deep understanding of each controller's design so that they may communicate with one another; otherwise character controllers will conflict at overlapping parts of the body and produce visual artifacts by naïvely overwriting one another. Directly modifying arbitrary character controllers to cooperate with one another and respond to external behavior commands can be costly and time-consuming. Monolithic, feature-rich character animation systems do not commonly support modular access to only a subset of their capabilities, while simpler systems lack control fidelity. Realistically, no sub-task of character control has a "perfect" solution. An ideal character animation system would allow a designer to choose between preferable techniques for producing a particular action or animation, leveraging the wealth of established systems already produced by the character animation research community and interfacing with robust frameworks for behavior and navigation.

We present a modular system that allows for the seamless integration of multiple character animation controllers on the same model, without requiring any controller to drastically change or accommodate any other. Rather than requiring a tightly-coupled set of character controllers, ADAPT uses a system for blending arbitrary poses in a user-authorable dataflow pipeline. Our system couples these animation controllers with an interface for path-finding and steering, as well as a comprehensive behavior authoring structure for authoring both individual decision-making and complex interactions between groups of characters. Our platform generalizes to allow the addition of new character controllers and behavior routines with minimal integration effort. Since controllers do not need to be fundamentally redesigned to work with one another, we avoid the combinatorial effect of having to modify each pre-existing controller to adjust for the change. Our system for character control contributes to our core goal of providing a platform for experimentation in character animation, navigation, and behavior authoring. We allow researchers to rapidly iterate on character controller designs with visual feedback, compare their results with other es-

*e-mail:ashoulson@gmail.com
†e-mail:nmarshak@seas.upenn.edu
‡e-mail:mubbasir.kapadia@gmail.com
§e-mail:badler@seas.upenn.edu

I3D 2013, Orlando, FL, March 21 – 23, 2013.
© 2013 ACM 978-1-4503-1956-0/13/0003 $15.00

tablished systems on the same model, and use features from other packages to provide the functionality they lack without the need to deeply integrate or reinvent known techniques.

2 Related Work

There exists a wealth of research [Pelechano et al. 2008] in virtual human simulation that separately addresses the problems of character animation, steering and path-finding, and behavior authoring.

Character Animation. Data-driven approaches [Kovar et al. 2002] use motion-capture data to animate a virtual character. Motion data can be manipulated by warping [Witkin and Popovic 1995] or blending [Menardais et al. 2004] to enforce parametric constraints on a recorded action. Interactive control of virtual characters can be achieved by searching through motion clip samples for desired motion as an unsupervised process [Lee et al. 2002], or by extracting descriptive parameters from motion data [Johansen 2009]. Procedural methods are used to solve specific tasks such as reaching, and can leverage empirical data [Liu and Badler 2003], example motions [Feng et al. 2012b], or hierarchical inverse kinematics [Baerlocher and Boulic 2004] for more natural movement. Physically-based approaches [Faloutsos 2002; Yin et al. 2007] derive controllers to simulate character movement in a dynamic environment. We refer to Pettré et. al. [2008] for a more extensive summary of work in these areas.

Steering and Path-finding. For navigation, the environment itself is often described and annotated as a reduction of the displayed geometry to be used in path planning. Probabilistic roadmaps superimpose a stochastic connectivity structure between nodes placed in the maneuverable space [Kavraki et al. 1996]. Navigation meshes [Kallmann 2010] provide a triangulated surface upon which agents can freely maneuver. Steering techniques use reactive behaviors [Reynolds 1999] or social force models [Helbing and Molnar 1995] to perform goal-directed collision avoidance in dynamic environments. Predictive approaches [Paris et al. 2007; van den Berg et al. 2008; Kapadia et al. 2009; Singh et al. 2011a] enable an agent to avoid others by anticipating their movements. Recast [Mononen 2009] provides an open-source solution to generating navigation meshes from arbitrary world geometry by voxelizing the space, and the associated Detour library provides path planning and predictive steering on the produced mesh. Pelechano et. al. [2008] provide a detailed review of additional work in this field.

Behavior Authoring. Animating behaviors in virtual agents has been addressed using multiple diverse approaches, particularly with respect to how behaviors are designed and animated. Early work focuses on imbuing characters with distinct, recognizable personalities using goals and priorities [Loyall 1997] along with scripted actions [Perlin and Goldberg 1996]. Our system makes use of parameterized behavior trees [Shoulson et al. 2011] to coordinate interactions between multiple characters. The problem of managing a character's behavior can be represented with decision networks [Yu and Terzopoulos 2007], cognitive models [Fleischman and Roy 2007], and goal-oriented action planning [Young and Laird 2005; Kapadia et al. 2011]. Very simple agents can also be simulated on a massive scale using GPU processing [Erra et al. 2010].

Multi-Solution Platforms. End-to-end commercial solutions [Massive Software Inc. 2010; Autodesk, Inc. 2012] combine multiple diverse character control modules to accomplish simultaneous tasks on the same character, incorporting navigation, behavior, and/or robust character animation. SteerSuite [Singh et al. 2009] is an open-source platform for developing and evaluating steering algorithms. SmartBody [Shapiro 2011] is an open-source system that combines steering, locomotion, gaze tracking, and reaching. These tasks are accomplished with 15 controllers working in unison to share control of parts of the body. SmartBody's controllers are hierarchically managed [Kallmann and Marsella 2005] where multiple animations, such as gestures, are displayed on a virtual character using a scheduler that divides actions into phases and blends those phases by interpolation. The controllers must either directly communicate and coordinate, or fix cases where their controlled regions of the body overlap and overwrite one another, making the addition of a new controller a process that affects several other software components. SmartBody also provides a navigation system with dynamic obstacle avoidance. Our platform shares some qualities with SmartBody, but also differs in several fundamental ways. While we do provide a number of character controllers for animating a virtual human, our work focuses more on enabling high-level behavioral control of multiple interacting characters, the modularity of these character controllers, and the ease with which a user can introduce a new animation repertoire to the system without disturbing the other controllers already in place.

3 Framework

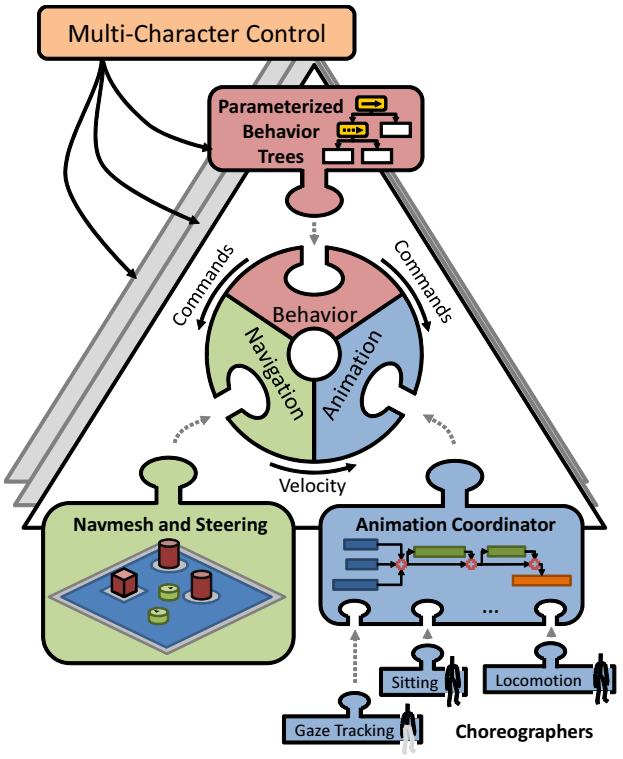


Figure 2: *Overview of ADAPT, illustrating the structure for controlling an individual character and all of the characters in an environment. Every character has a core interface for behavior, navigation, and animation, each of which connects to more specific modular components. Top-level narrative control communicates with each character through the behavior interface.*

ADAPT operates at multiple layers with interchangeable, lightweight components, and we focus on minimizing the amount of communication and interdependency between modules (Figure 2). The animation system performs control tasks such as locomotion, gaze tracking, and reaching as independent modules, called choreographers, that can share parts of the same character's body without explicitly communicating or negotiating with one another. These modules are managed by a coordinator, which acts as a cen-

tral point of contact for manipulating the virtual character's pose in real-time. The navigation system performs path-finding with predictive steering and we provide a common interface to allow users to interchange the underlying navigation library without affecting the functionality of the rest of the framework. The behavior level is split into two tiers. Individual behaviors are attached to each character and manipulate that character using the behavior interface, while a centralized control structure orchestrates the behavior of multiple interacting characters in real-time. The ultimate product of our system is a pose for each character at an appropriate position in the environment, produced by the animation coordinator and applied to a rendered virtual character in the scene each frame.

3.1 Full-Body Character Control

Controlling a fully-articulated character is traditionally accomplished using a series of interwoven subcomponents responsible for various parts of the body. Without prior knowledge of other systems, a designer creating a character controller will generally do so with the assumption that no other systems are acting on the rigged model at the same time. If a controller sets the orientation or position of a character's joint, it does so expecting no other controller to overwrite that orientation or position in the current frame. If two controllers conflict and overwrite one another, the constant changes cause visual artifacts such as jitter as the character rapidly shifts between the two settings for its joints. Controllers can be made to share control of a single body either by negotiating with one another, or by dividing the body into sections and controlling those sections alone. However, this requires that the controllers be specifically designed to coordinate, which requires additional effort on either the designer or the user of the control system. The addition of new functionality also becomes more difficult as all of the previous body controllers must be modified to communicate with any new components and share control of the body's joints.

To address this issue, we divide the problem of character animation into a series of isolated, modular components called *choreographers* attached to each character. Each choreographer operates on a *shadow*, which is an invisible clone of the character skeleton, and has unmitigated control to manipulate the skeletal joints of its shadow. Each frame, a choreographer produces an output pose consisting of a snapshot of the position and orientation of each of the joints in its private shadow. A *coordinator* receives the shadow poses from each choreographer and performs a weighted blend to produce a final pose that is applied to the display model for that frame. Since each choreographer has its own model to manipulate without interruption, choreographers do not need to communicate with one another in order to share control of the body or prevent overwriting one another. This allows a single structure, the coordinator, to manage the indirect interactions between choreographers using a simple, straightforward, and highly authorable process centered around blending the shadows produced by each choreographer. This system is discussed in more detail in Section 4.

3.2 Steering and Path-finding

We use a navigation mesh approach for steering and path-finding with dynamic obstacle avoidance. Each display model is controlled by a point-mass system, which sets the root positions (usually the hips) of the display model and each shadow every frame. We use a common interface for navigation with basic commands such as setting a goal position in the world. Character choreographers do not directly communicate with the navigation layer. Instead, choreographers are made aware of the position and velocity of the character's root, and will react to that movement on a frame-by-frame basis. A character's orientation can follow several different rules, such as facing forward while walking, or facing in an arbitrary direction, and we handle this functionality outside of the navigation system itself. ADAPT supports both the Unity3D built-in navigation system and the Recast/Detour library [Mononen 2009] for path-finding and predictive goal-directed collision avoidance, and users can easily experiment with alternate solutions, such as navigation graphs.

3.3 Behavior

ADAPT is designed to accommodate varying degrees of behavior control for its virtual characters by providing a diverse set of choreographers and navigation capabilities. Each character has a capability interface with commands like `ReachFor()`, `GoTo()`, and `GazeAt()` that take straightforward parameters like positions in space and send messages to that character's navigation and animation components. To invoke these capabilities, we use Parameterized Behavior Trees (PBTs) [Shoulson et al. 2011], which present a method for authoring character behaviors that emphasizes simplicity without sacrificing expressiveness. Having a single, flat interface for a character's action repertoire simplifies the task of behavior authoring, with well-described and defined tasks that a character can perform. One advantage of the PBT formalism is that they accommodate authoring behavior for multiple actors in one centralized structure. For example, a conversation between two characters can be designed in a single data structure that dispatches commands to both characters to take turns playing sounds or gestural animations. For very specific coordination of characters, this approach can be preferable over traditional behavior models where characters are authored in isolation and interactions between characters are designed in terms of stimuli and responses to triggers. ADAPT also generalizes across traditional or experimental new ways of modeling behavior to cover cases where PBTs are not the most appropriate. The behavior system is discussed in more detail in section 5.

4 Shadows in Full-Body Character Animation

Model rendering systems describe a virtual human as a skinned mesh with a hierarchical skeletal structure underneath. The movement of the body is determined by altering the position and orientation of each joint in the skeleton "rig", which in turn affects the position and orientation of that joint's children in the hierarchy. General character controllers are systems designed to manipulate the character by setting the positions and orientations of that character's joints, either via animations or procedurally with physical models or inverse kinematics. We address the problem of coordination between these controllers by allocating each character controller its own private character model, a replica of the skeleton or a subset of the skeleton of the character being controlled. Our modular controllers, called choreographers, act exactly the same way as traditional character controllers, but do so on private copies of the actual rendered character model. These skeleton clones (shadows), match the skeletal hierarchy, bone lengths, and initial orientations of the final rendered character (display model), but have no visual component in the scene. This is illustrated and described in figure 3. The general process of our character animation system has two interleaving steps. First, each choreographer manipulates its personal shadow and outputs a snapshot (called a shadow pose) describing the position and orientation of that shadow's joints at that time step. Then, we use a centralized controller to blend the shadow pose snapshots into a final pose for the rendered character. For clarity, note that "shadow" refers to the invisible articulated skeleton allocated to each choreographer to manipulate, while a "shadow pose" is a serialized snapshot containing the joint positions and orientations for a shadow at a particular point in time.

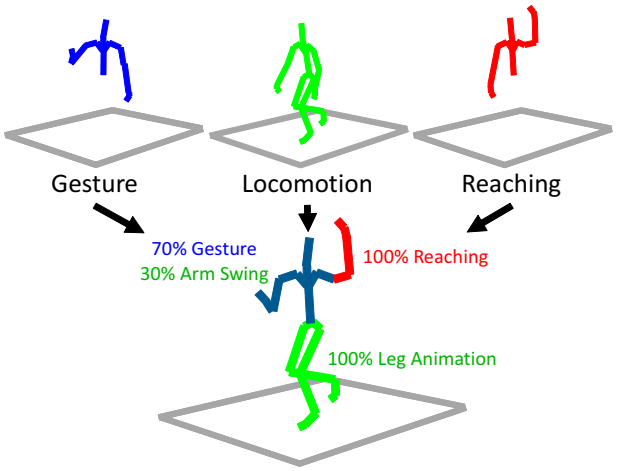


Figure 3: *Blending multiple character shadows to produce a final output skeleton pose. As an example, we combine the pose of the locomotion choreographer (green, full-body) during a walk cycle with the reaching choreographer (red, upper-body) extending the left arm towards a point above the character's head, and the gesture choreographer (blue, upper-body) playing a waving animation. The generated poses are projected, either wholly or partially, on different sections of the displayed body during any particular frame. The partial blend is represented with a mix of colors in the RGB space.*

4.1 Choreographers

The shadow pose of a character at time t is given by $\mathbf{P}_t \in \mathbb{R}^{4\times|J|}$. where $\mathbf{P}_t^j$ where is the configuration of the j^{th} joint at time t. A choreographer is a function $C(\mathbf{P}_t) \longrightarrow \mathbf{P}_{t+1}$ which produces the next pose by changing the configuration of the shadow joints for that time step. Using these definitions, we define two classes of choreographers:

Generators. Generating choreographers produce their own shadow pose each frame, requiring no external pose data to do so. Each frame, the input shadow pose $\mathbf{P}_t$ for a generator C is the pose $\mathbf{P}_{t-1}$ generated by that same choreographer in the previous frame. For example, a sitting choreographer requires no external input or data from other choreographers in order to play the animations for a character sitting and standing, and so its shadow's pose is left untouched between frames. This is the default configuration for a choreographer.

Transformers. Transforming choreographers expect an input shadow pose, to which they apply an offset each frame. Each frame, the input shadow pose $\mathbf{P}_t$ to a transformer C is an external shadow pose $\mathbf{P}'_{t+1}$ from another choreographer C', computed for that frame. The coordinator sets its shadow's pose to $\mathbf{P}'_{t+1}$ and applies an offset to the given pose during its execution, to produces a new pose $\mathbf{P}_{t+1}$. For example, before executing, the reach choreographer's shadow is set to the pose of a previously-updated choreographer's shadow (say, the locomotion choreographer with swinging arms and torso movement). The reach choreographer then solves the reach position from the base of the arm based on the torso position it was given, and overwrites its shadow's arm and wrist joints to produce a new pose. Without an input shadow, the reach choreographer would not be aware of other choreographers moving the torso, and would not be able to accommodate different torso positions when solving a reaching problem. Note that this is accomplished without the choreographers directly communicating or even being fully aware of one another. A transforming choreographer can receive an input pose, or blend of input poses, from any choreographer that has already been updated in the current frame.

4.2 The Coordinator

During runtime, our system produces a pose for the display model each frame, given the character choreographers available. This is a task overseen by the coordinator. The coordinator is responsible for maintaining each choreographer, organizing the sequence in which each choreographer performs its computation each frame, and reconciling the shadow poses that each choreographer produces by sending them between choreographers and/or blending them together. The coordinator's final product each frame is a sequence of weighted blends of each active choreographer's shadow pose. We compute this product using the *pose dataflow graph*, which dictates the order of updates and the flow of shadow poses between choreographers. Generators pass data to transformers, which can then pass their data to other transformers, until a final shadow pose is produced, blended with others, and applied to the display model.

Blending is accomplished at certain points in the pose dataflow graph denoted by *blend nodes*, which take two or more input shadows and produce a weighted blend of their transforms. If the weights sum to a value greater than 1, they are automatically normalized.

$$B(\{(\mathbf{P}_i, w_i) : i = 1..n)\}) \longrightarrow \mathbf{P}' \quad (1)$$

Designing a dataflow graph is a straightforward process of dictating which nodes pass their output to which other nodes in the pipeline, and the graph can be modified with minimal effort. The dataflow graph for a character is specified by the user during the design and authoring process, connecting choreographers with blend nodes and one another. The weights involved in blending are bound to edges in the graph and then controlled at runtime by commands from the behavior system. The order of the pose dataflow graph roughly dictates the priority of choreographers over one another. Choreographers closer to the final output node in the graph have the authority to overwrite poses produced earlier in the graph, unless bypassed by the blending system. Changing the order of nodes in the dataflow graph will affect these priorites, and so we generally design the graph so that choreographers controlling more parts of the body precede those controlling fewer.

Blended poses are calculated on a per-joint basis using each joint's position vector and orientation quaternion. The weighted average we produce accommodates cases where parts of a shadow's skeleton have been pruned or filtered from the blend (such as an upper-body shadow missing the character's legs). The blend function produces a new shadow pose that can be passed to other transformers, or be applied to the display model's skeleton. Taking a linear weighted average of vectors is a solved problem, but such is not the case with the problem of quickly averaging $n > 2$ weighted quaternions. We discuss the techniques with which we experimented, and the final calculation method we decided to use in Appendix A. In addition, Feng et. al. [2012a] provide a detailed review of more sophisticated motion blending techniques than our linear approach.

Figure 4 illustrates a sample dataflow graph, incorporating generating and transmuting choreographers, as well as four blend nodes. Three generating choreographers (blue) begin the pipeline. The gesture choreographer affects only the upper body, with no skeleton information for the lower body. Increasing the value of the gesture weight w_g places this choreographer in control of the torso, head, and arms. The sitting and locomotion choreographers can affect the entire body, and the user controls them by raising and lowering

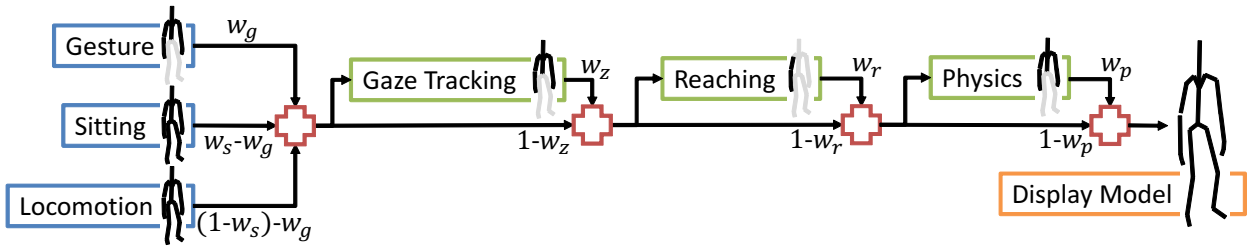


Figure 4: *A sample dataflow graph we designed for evaluating ADAPT. Generating choreographers appear in blue, transmuting choreographers appear in green, and blend nodes appear as red crosses. The final display model node is highlighted in orange. The sitting weight w_s, gesture weight w_g, gaze weight w_z, reach weight w_r, and physical reaction weight w_p are all values between some very small positive ϵ and $1 - \epsilon$.*

the sitting weight w_s. If w_g is set to $1 - \epsilon$, the upper body will be overridden by the gesture choreographer, but since the gesture choreographer's shadow has no legs, the lower body will still be controlled by either the sitting or locomotion choreographer as determined by the value of w_s. The first red blend node combines the three produced poses and sends the weighted average pose to the gaze tracker. The gaze tracking choreographer receives an input shadow pose, and applies an offset to the upper body to achieve a desired gaze target and produce a new shadow pose. The second blend node can bypass the gaze tracker if the gaze weight w_z is set to a low value (ϵ). The reach and physical reaction choreographers receive input and can be bypassed in a similar way. The final result is sent and applied to joints of the display model, and rendered on screen. The dataflow graph accommodates the addition of new choreographers in a generalizable fashion, allowing a user to insert new nodes and blend between the poses they create. Rather than designing animation modules to explicitly negotiate, the coordinator seamlessly fades control of parts of the body between arbitrary choreographers in an authorable pipeline.

4.3 Using Choreographers and the Coordinator

The dataflow graph, once designed, does not need to be changed during runtime or to accommodate additional characters. Instead, the coordinator provides a simple interface comprising messages and exposed blend weights for character animation. Messages are commands (e.g., `SitDown()`) relayed by the coordinator to its choreographers, making the coordinator a single point of contact for character control, as illustrated in Figure 2. In addition to messages, the weights used for blending the choreographers at each blend node in the dataflow graph are exposed, allowing external systems to dictate which choreographer is active and in control of the body (or a segment of the body) at a given point of time. For example, in Figure 4, lowering w_s will transfer control of the body to the locomotion choreographer, while raising its value will give influence to the sitting choreographer. Both choreographers are still manipulating their shadows each update, but only one choreographer's shadow pose is displayed on the body at a given time, with smooth fading transitions between the two where necessary.

For gesturing, we raise w_g, which takes control of the arms and torso away from both the locomotion and sitting choreographers and stops the walking animation's arm swing. Given sole control, the gesture choreographer plays an animation on the upper body, and then is faded back out to allow the walking arm-swing to resume. Since the gesture choreographer's shadow skeleton has no leg bones, it never overrides the sitting or locomotion choreographer, so the lower body will still be sitting or walking while the upper body gesture plays. All weight changes are smoothed over several frames to prevent jitter and transition artifacts. Note that the controllers are never in direct communication to negotiate this exchange of body control. The division of roles between the coordinator and choreographers centralizes character control to a single externally-facing character interface, while leaving the details of character animation distributed across modular components are isolated from one another and can be easily updated or replaced.

Shadow Pose Post-Processing. Since shadow poses are serializations of a character's joints, additional nodes can be added to the pose dataflow graph to manipulate shadows as they are transferred between choreographer nodes or blend nodes. For instance, special filter nodes can be added to constrain the body position of a shadow pose, preventing joints from reaching beyond a comfortable range by clamping angles, or preventing self-collisions by using bounding volumes. Nodes can be designed to broadcast messages based on a shadow's pose, such as notifying the behavior system when a shadow is in an unbalanced position, or has extended its reach to a certain distance. The interface for adding new kinds of nodes to a pose dataflow graph is highly extensible. This affords the user another opportunity to quickly add functionality to a coordinator without directly modifying any choreographers.

4.4 Example Choreographers

ADAPT provides a diverse array of character choreographers for animating a fully articulated, expressive virtual character. Some of these choreographers were developed specifically for ADAPT, while others were off-the-shelf solutions used to highlight the ease of integration with the shadow framework. ADAPT is designed to "trick" a well-behaved character control system into operating on a dedicated shadow model rather than the display model of the character, and so the process of modifying an off-the-shelf character control library to a character choreographer often requires modifying only a few lines of code. Since shadows replicate the structure and functionality of a regular character model, no additional considerations are required once the choreographer has been retargeted to the shadow. Note that the choreographers presented here are largely baseline examples. The focus of ADAPT is to allow a user to add additional choreographers, experiment with new techniques, and easily exchange generic choreographers with more specialized alternatives.

Locomotion. ADAPT uses a semi-procedural motion-blending locomotion system for walking and running released as a C# library with the Unity3D engine [Johansen 2009]. The system takes in animation data, analyzes those animations, and procedurally blends them according to the velocity and orientation of the virtual character. We produced satisfactory results on our test model using five motion capture animation clips. Additionally, the user can annotate the character model to indicate the character's legs and feet,

which allows the locomotion library to use inverse kinematics for foot placement on uneven surfaces. We extended this library to work with the ADAPT shadow system, with some minor improvements.

Gaze Tracking. We use a simple IK-based system for attention control. The user defines a subset of the upper body joint hierarchy which is controlled by the gaze tracker, and can additionally specify joint rotation constraints and delayed reaction speeds for more realistic results. These parameters can be defined as functions of the characters velocity or pose, to produce more varied results. For instance, a running character may not be permitted to rotate its torso as far as a character standing still. Integrating the gaze tracker into ADAPT required minimal changes to the existing library.

Upper Body Gesture Animations. We dedicate a shadow with just the upper body skeleton to playing animations such as hand gestures. We can play motion clips on various parts of the body to blend animations with other procedural components.

Sitting and Standing. The sitting choreographer maintains a simple state machine for whether the character is sitting and standing, and plays the appropriate transition animations when it receives a command to change state. This choreographer acts as an alternative to the locomotion choreographer when operating on the lower body, but can be smoothly overridden by choreographers acting on the upper body, such as the gaze tracker.

Reaching. We implemented a simple reaching control system based on Cyclic Coordinate Descent (CCD). We extended the algorithm to dampen the maximum angular velocity per frame, include rotational constraints on the joints, and apply relaxation forces in the iteration step. During each iteration of CCD (100 per frame), we clamp the rotation angles to lie within the maximum extension range, and gently push the joints back towards a desired "comfortable" angle for the character's physiology. These limitations and relaxation forces are based on an empirical model for reach control based on human muscle strength [Slonneger et al. 2011]. This produces more realistic reach poses than naïve CCD, and requires no input data animations. The character can reach for an arbitrary point in space, or will try to do so if the point is out of range.

Physical Reaction. By allocating an upper-body choreographer with a simple ragdoll, we can display physical reactions to external forces. Once an impact is detected, we apply the character's last pose to the shadow skeleton, and then release the ragdoll and allow it to buckle in response to the applied force. By quickly fading in and out of the reeling ragdoll, we can display a physically plausible response and create the illusion of recovery without requiring any springs or actuators on the ragdoll's joints.

SmartBody Integration. To access its locomotion and procedural reaching capabilities, we integrated the ICT SmartBody framework into our platform, using SmartBody's Unity interface and some modifications. Since our model's skeleton hierarchy differed from that of the default SmartBody characters, sample animations had to be retargeted to use on our model. Additionally, our animation interface needed to interact with SmartBody using BML. Since our coordinator is already designed to relay messages from the behavior system, changing those messages to a BML format was a straightforward conversion. Overall, the SmartBody choreographer blends naturally with other choreographers we have in the ADAPT framework, though SmartBody has other features that we do not currently exploit. This process demonstrates the efficacy of integrating other available libraries and/or commercial solutions.

5 Behavior

The navigation and shadow-based character animation system provides a number of capability functions, including:

Commands	Description
`ReachFor(target)`	Activates the reaching choreographer, and reaches towards a position.
`GazeAt(target)`	Activates the gaze choreographer, and gazes at a position.
`GoTo(target)`	Begins navigating the character to a position.
`Gesture(name)`	Activates the gesture choreographer for the duration of an animation.
`SitDown()`	Activates the sitting choreographer and sits the character down.
`StandUp()`	Stands the character up and then disables the sitting choreographer.

Passing an empty target position will end that task, stopping the gaze, reach, or navigation. The locomotion choreographer will automatically react to the character's velocity, and move the legs and arms to compensate if the character should be turning, walking, side-stepping, backpedaling, or running. Note that only sitting and navigating are mutually exclusive. All other commands can be performed simultaneously without visual artifacts.

5.1 Adding a New Behavior Capability

Adding a new behavior capability with a motion component, such as climbing or throwing an object, requires a choreographer capable of producing that motion. Choreographers can be designed to perform animation tasks based on animation data, procedural techniques, or physically-driven models. Since choreographers operate on their own private copies of the character's skeleton, they can be designed in isolation and integrated into the system separately. Once the choreographer is developed, the process of adding a new behavior capability to take advantage of the choreographer requires two steps. First, the choreographer must be authored into the pose dataflow graph, either as a generating or transforming node, with appropriate connections to blend nodes and other choreographers. Next, the behavior interface can be extended with new functions that either modify the blend weights relevant to the new choreographer, or pass messages to that choreographer by relaying them through the coordinator. The sophistication of character choreographers varies, but the process of integrating a functioning choreographer into the behavior and animation pipeline for a character is authorable and generalizable.

5.2 Multi-Character Interactions

Using this behavior repertoire, we can produce more sophisticated actions as characters interact with one another and the environment. Authoring complex behaviors requires an expressive and flexible behavior authoring structure granting the behavior designer reasonable control over the characters in the environment. To accomplish this task, we use parameterized behavior trees (PBTs). PBTs are an extension of the behavior tree formalism that allow behavior trees to manage and transmit data within their hierarchical structure without the use of a blackboard. A useful advantage of PBTs is the fact that they can simultaneously control multiple characters in a single reusable structure called an *event*. Events are pre-authored behavior trees that sit uninitialized in a library until invoked at runtime. When instantiated, an event takes one or more actors as parameters, and is temporarily granted exclusive control over those

characters' actions. While in control, an event treats these characters as limbs of the same entity, dispatching commands for agents to navigate towards and interact with one another. Once the event ends, control is yielded to the characters' own individual decision processes, which can also be designed using PBTs or with some other technique. Events are a convenient formalism to use for interactions with a high degree of interchange and turn-taking, such as conversations. A conversation event can be authored as a simple sequential and/or stochastic sequence of commands directing agents to face one another and take turns playing gesture animations or exchanging physical objects. ADAPT provides a fully-featured scheduler for managing and updating both the personal behavior trees belonging to each character and higher-level event behavior trees encompassing multiple characters.

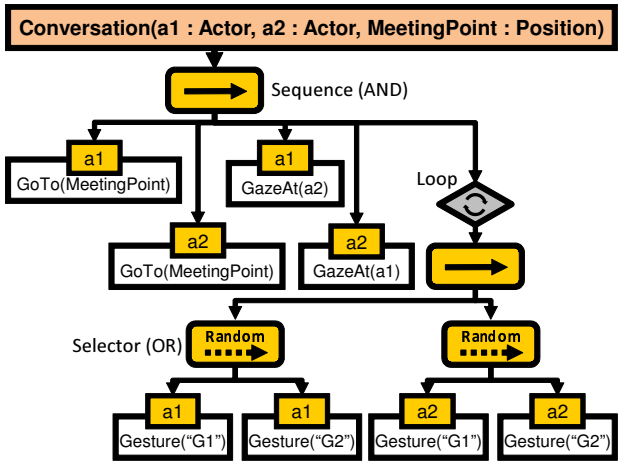


Figure 5: *A simple conversation PBT event controlling two characters,* `a1` *and* `a1`*, with one additional* `MeetingPoint` *parameter.*

Figure 5 illustrates a sample behavior tree event conducting two characters through a conversation using our action repertoire. The characters, `a1` and `a2`, are passed as parameters to the tree, along with the meeting position. Using our action interface, the tree directs the two characters to approach one another at the specified point, face each other, and alternatively play randomly selected gesture animations. The gesturing phase lasts for an arbitrary duration determined by the configuration of the loop node in the tree. After the loop node terminates, the event ends, reporting success, and the two characters return to their autonomous behaviors. Note that this tree can be reused at any time for any two characters and any two locations in the environment in which to stand. This framework can be exploited to create highly sophisticated interactions involving crowds of agents, and its graphical, hierarchical nature makes subtrees easier to describe and encapsulate.

6 Results

We demonstrate the features of ADAPT in isolation, as well as a final scene showcasing animation, navigation, and behavior working together to produce a narrative sequence (Figure 1). Using our system, we can create a character that can simultaneously reach, gaze, walk, and play gesture animations, as well as activate other functionality like sitting and physically reacting to external forces. ADAPT characters can intelligently maneuver an environment avoiding both static obstacles and one another. These features are used for authoring sequences like exchanging an object between actors, wandering while talking on a phone, and multiple characters holding a conversation.

Multi-Actor Simulation. The concluding narrative sequence shown in the video is simulated using several reusable authored events, which are activated using spatial and temporal triggers. Events once active, can be successfully executed or interrupted by other triggers due to dynamic events, or user input. This produces a rich interactive simulation where virtual characters can be directed with a high degree of fidelity, without sacrificing autonomy or burdening the user with authoring complexity.

In the beginning, an event ensues where a character is given a phone and converses while wandering through the scene, gazing at objects of interest. The phone conversation event successfully completes and the character hands back the phone. Spotting nearby friends invokes a conversation, which is an extension of the event illustrated in Figure 5. The conversation is interrupted when a ball is thrown at one of the characters. The culprit flees from the scene of the crime, triggering a chasing event where the group runs after the child. The chase fails as the child is able to escape through a crossing crowd of characters, which are participating in a group event to navigate to the theater and find a free chair to sit. We illustrate some of the trees used for this sequence in greater detail in a supplemental document.

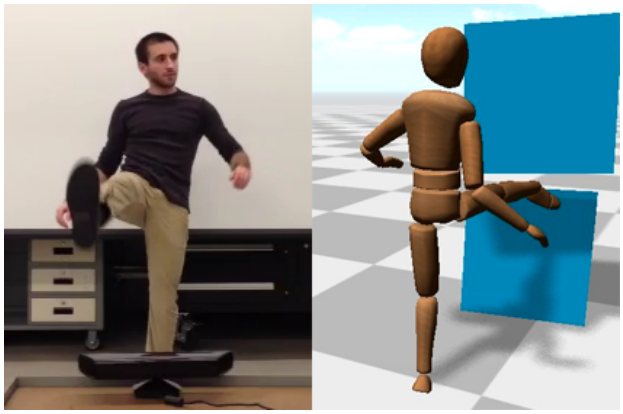


Figure 6: *Controlling a character in ADAPT and physically interacting with the environment using the Kinect.*

Adding a Kinect Choreographer. As an example of our system's extensibility, we created an additional choreographer to interface with the Microsoft Kinect and control a character with gesture input. To do so, we allocated a full-body choreographer to the input of the Kinect, applying the captured skeleton from the Kinect's framework directly to the joints of the dedicated shadow. This is demonstrated in Figure 6. Blending this choreographer with others allowed us to expand the character's agency in the world. When the character stands idle, we give full upper and lower body control to the Kinect input. When the user wishes to make the character move, we blend the legs of the locomotion choreographer on top of the Kinect input, displaying appropriate walking or running animations and foot placement while still giving the Kinect control of the upper body. This is a feasible compromise for allowing a user to retain correct leg animation when exploring a virtual environment larger than the Kinect's capture area. The process of interfacing the Kinect skeleton input with a new choreographer and a blending coordinator was very fast and straightforward.

Performance. ADAPT supports approximately 150 agents with full fidelity at interactive frame rates. Figure 7 displays the update frequency for the animation and navigation system (for our scenes, the computational cost of behavior was negligible). This varies with the complexity of the choreographers active on each character. The ADAPT animation interface and the pose dataflow graph has little impact on performance, and the blend operation is linear in num-

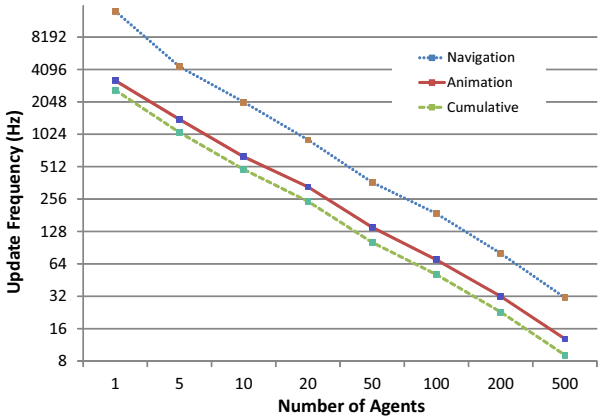


Figure 7: *Update frequency for the character animation and navigation components in ADAPT.*

ber of choreographers. Each joint in a shadow is serialized with 7 4-byte float values, making each shadow 28 bytes per joint. For 26 bones, the shadow of a full-body character choreographer has a memory footprint of 728 bytes. For 200 characters, the maximum memory overhead due to shadows is less than 1 MB. In practice, however, most choreographers use reduced skeletons with only a limb or just the upper body, making the actual footprint much lower for an average character.

Separating character animation into discrete modules and blending their produced poses as a post-processing effect also affords the system unique advantages with respect to dynamic level-of-detail (LOD) control. Since no choreographer is architecturally dependent on any other, controllers can be activated and deactivated arbitrarily. Deactivated controllers can be smoothly faded out of control at any time, and their nodes in the dataflow graph can be bypassed using the already-available blend weights. This drastically reduces the number of computed poses, and conserves processing resources needed for background characters that do not require a full repertoire of actions. The system retains the ability to re-activate those choreographers at any time if a specific complex action is suddenly required. Since choreographers are not tightly coupled, no choreographer needs to be made aware of the fact that any other choreographer has been disabled for LOD purposes.

7 Conclusions

ADAPT is a modular, flexible platform which provides a comprehensive feature set for animation, navigation, and behavior tools needed for end-to-end simulation development. By allowing a user to independently incorporate a new animation choreographer or steering system, and make those components immediately accessible to the behavior level without modifying other existing systems, characters can very easily be expanded with new capabilities and functionality. We are releasing ADAPT as an open-source project not only to provide animation, steering, and behavior package to community, but to provide a platform that is designed to allow users to tailor the system to fit their own personal needs, to rapidly iterate on experimental designs, and to compare their results against other established techniques. The library, assets, and documentation are available at `http://cg.cis.upenn.edu/ADAPT`

7.1 Limitations and Complications

The ADAPT platform has some surmountable issues that arise from blending poses as a post-process. Solutions to these complications either already exist in the system or could be introduced with future work.

Cross-Choreographer State Awareness. At times, choreographers may need to be aware of major state changes in the character's pose caused by another choreographer. For example, we may wish to restrict the degree to which the character can rotate its torso for gaze tracking while the character is running. We accomplish this using the message broadcast system integrated into the coordinator. When a character reaches a certain speed, the locomotion choreographer can broadcast to all other choreographers that the character is in an `IsRunning` state. The gaze tracking choreographer can receive this message and restrict its maximum torso rotation accordingly. This allows choreographers to cooperate without being explicitly aware of one another, and is a more extensible paradigm than deep integration of controllers.

Foot-Placement Artifacts. Interpolation between arbitrary poses generally produces smooth results in our system, with the exception of blends that linearly translate the position of a character's feet. This situation arises with our sitting choreographer, where the placement of a character's feet while standing may not coincide with the foot placement in the transition animation between standing and sitting. A linear blend here results in an unrealistic sliding of the foot despite ground contact. This issue could be easily solved with a slightly more robust locomotion system that allowed arbitrary foot placement, so that we could use a special case to adjust the feet to step to the proper position before blending from the locomotion to the sitting choreographer. Since each choreographer is aware of which parts of the body it uses, and how it wants to pose each joint, this solution could be generalized for transitioning between any choreographers that use the lower body of the character.

7.2 Future Work

Moving forward, we will continue to expand the animation and authoring capabilities supported by ADAPT. For example, the design of the pose dataflow graph is one possible avenue for improvement. Currently, a designer manually organizes the choreographer and blend nodes in the structure of the coordinator. While this is a conceptually simple task because the dataflow graph is so easy to visualize, we have yet to develop a scripting or graphical interface to make the process more accessible to a completely untrained user. More importantly, however, we believe the process of authoring a dataflow graph can be completely automated based on which parts of the body each choreographer uses.

Another main development effort is the production of more capable choreographers for use in the ADAPT framework. We would like to develop or incorporate a better locomotion system with the ability to control an autonomous character with footstep-level precision [Singh et al. 2011b]. One major advantage in this effort is the ability to directly integrate other developed systems into the ADAPT framework and seamlessly blend them with the rest of our choreographers, as we have done with the SmartBody package. In addition to choreographers described here, we want our platform to provide an array of options for different kinds of motor skills, including jumping, climbing, and carrying objects with weight.

Finally, we are also interested in improving the virtual environment and developing extensible ways for characters to interact with the environment on a behavioral level. To ease the authoring burden, we are currently creating an interface similar to smart objects for annotating the environment and describing the ways that characters

can interact with it. We are particularly interested in extending the ADAPT platform to develop solutions for the automated scheduling of events to follow global narrative arcs. All of these improvements will allow us to apply our platform to other areas research, as ADAPT is uniquely suited for producing the next generation of narrative-driven simulations.

Acknowledgments

We acknowledge Dr. Ari Shapiro and Dr. Ben Sunshine-Hill for their discussions and contributions to the ADAPT project. The research reported in this document was performed in connection with Contract Numbers W911NF-07-1-0216 and W911NF-10-2-0016 with the U.S. Army Research Laboratory. The views and conclusions contained in this document are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of the U.S. Army Research Laboratory, or the U.S. Government unless so designated by other authorized documents. Citation of manufacturers or trade names does not constitute an official endorsement or approval of the use thereof. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation heron.

Appendix A: On Quaternion Blending

While vectors can be easily averaged using addition and scalar multiplication in $\mathbb{R}^3$, interpolating between quaternions is not as simple due to the spherical surface of the unit-quaternion space. For our coordinator to be able to blend shadow poses between choreographers, we require a fast method for blending the quaternions describing the rotations of the character's joints in the unit-quaternion manifold. Our ultimate goal is to develop a function $\text{Blend}((q_1, w_1), (q_2, w_2), \ldots, (q_n, w_n)) = q$ taking n weighted unit quaternions and producing an average quaternion q. To create this function, we experimented with a number of different techniques, and felt it valuable to document our efforts here.

Slerp. Spherical linear interpolation (Slerp) [Shoemake 1985] is a constant-time operation for interpolating between two unit quaternions q_1 and q_2, using an interpolation weight w. Slerp is defined equivalently as follows:

$$\begin{aligned} \text{Slerp}(q_1, q_2; w) &= q_1(q_1^{-1}q_2)^w && (2)\\ &= q_2(q_2^{-1}q_1)^{1-w} && (3)\\ &= (q_1q_2^{-1})^{1-w}q_2 && (4)\\ &= (q_2q_1^{-1})^w q_1 && (5) \end{aligned}$$

Slerp has the ideal properties of being a closed form solution, and generally taking the shortest path between two quaternions. Because of this, we considered using a chain of Slerp operations. For instance, with three quaternions q_1, q_2, and q_3 and two blend weights w_0, w_2, we could perform:

$$\begin{aligned} &\text{Blend}_{SLERP}(q_1, q_2, q_3; , w_1, w_2)\\ &\quad = \text{Slerp}(\text{Slerp}(q_1, q_2; w_1), q_3; w_2) \end{aligned} \tag{6}$$

Ultimately, we abandoned the idea primarily because operation would not be commutative in all cases, which could create unexpected and confusing results in the authoring process, especially as the number of input quaternions grew.

Angular Velocities. Another alternative is to treat each of the character's joint as a ball-and-socket joint with three rotational degrees of freedom. This allows to compute the angular velocities applies to each joint between frames, and to average those velocities when blending between each choreographer's produced motions. This had two problems. First, the process did not properly handle blending into static poses from dynamic ones. Second, this essentially converted each joint quaternion into a set of Euler angles, which suffer from the problem of gimbal lock.

Iterative Solutions. Multiple iterative solutions exist for constrained interpolation. Pennec [1998], Johnson [2003], and Buss et. al. [2001] all describe iterative techniques for finding the weighted average of quaternions on the spherical surface. While these provide accurate results and generally blend over the shortest distance, we looked for a solution with a closed form.

Because our blending is spread out over numerous frames, our quaternion blending function is usually only interpolating between very short distances. As a result, we experienced success with a naïve algorithm.

Data: Unit quaternions $q_1, \ldots, q_n$
Data: Normalized weights $w_1, \ldots, w_n$
Result: An average quaternion q
$q = [0, 0, 0, 0]$;
for $i = 1 \rightarrow n$ **do**
  if $i > 1$ **and** $q_1 \cdot q_i < 0$ **then**
    // negate every element of the quaternion;
    $q_i = [-q_i^0, -q_i^1, -q_i^2, -q_i^3]$;
  end
  for $j = 0 \rightarrow 3$ **do**
    $q^j += q_i^j * w_i$
  end
end
return $\text{Normalize}(q)$
Algorithm 1: The computation of $\text{Blend}((q_1, w_1), \ldots, (q_n, w_n))$

So long as the blend distances are short, we can naively treat the quaternion space as $\mathbb{R}^4$, and take a weighted average of each element of the quaternion. This will work provided the quaternion is normalized after the calculation. To ensure that we generally take the shortest distance between angles, we pick an arbitrary quaternion q_p and calculate the dot product of that quaternion with each other quaternion $q_{i \neq p}$. If the dot product is negative, we negate each term in q_i. Note that we negate each term rather than inverting the quaternion. This, again, is a quick approximation for blending over short distances. In practice, this quick method produces good visual results, and is computationally cheaper than multiple slerps or an iterative solution. For blends across more significant distances, we could opt for a more complicated solution, but so far have seen no need to do so.

References

AUTODESK, INC., 2012. Autodesk gameware - artificial intelligence middleware for games.

BAERLOCHER, P., AND BOULIC, R. 2004. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *Vis. Comput. 20*, 6 (Aug.), 402–417.

BUSS, S. R., AND FILLMORE, J. P. 2001. Spherical averages and applications to spherical splines and interpolation. *ACM Trans. Graph. 20*, 2 (Apr.), 95–126.

ERRA, U., FROLA, B., AND SCARANO, V. 2010. Behavert: a gpu-based library for autonomous characters. In *Motion in Games*, MIG'10, 194–205.

FALOUTSOS, P. 2002. *Composable Controllers for Physics-Based Character Animation*. PhD thesis, University of Toronto.

FENG, A. W., HUANG, Y., KALLMANN, M., AND SHAPIRO, A. 2012. An analysis of motion blending techniques. In *The Fifth International Conference on Motion in Games*.

FENG, A. W., XU, Y., AND SHAPIRO, A. 2012. An example-based motion synthesis technique for locomotion and object manipulation. I3D, 95–102.

FLEISCHMAN, M., AND ROY, D. 2007. Representing intentions in a cognitive model of language acquisition: Effects of phrase structure on situated verb learning. In *AAAI '07*, AAAI, 7–12.

HELBING, D., AND MOLNAR, P. 1995. Social force model for pedestrian dynamics. *PHYSICAL REVIEW E 51*, 42–82.

JOHANSEN, R. S. 2009. *Automated Semi-Procedural Animation for Character Locomotion*. Master's thesis, Aarhus University.

JOHNSON, M. P. 2003. *Exploiting Quaternions to Support Expressive Interactive Character Motion*. PhD thesis, Massachusetts Institute of Technology.

KALLMANN, M., AND MARSELLA, S. 2005. Lncs '05. ch. Hierarchical motion controllers for real-time autonomous virtual humans, 253–265.

KALLMANN, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *Proceedings of the Eurographics / SIGGRAPH Symposium on Computer Animation (SCA)*.

KAPADIA, M., SINGH, S., HEWLETT, W., AND FALOUTSOS, P. 2009. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '09, 215–223.

KAPADIA, M., SINGH, S., REINMAN, G., AND FALOUTSOS, P. 2011. A behavior-authoring framework for multiactor simulations. *Computer Graphics and Applications, IEEE 31*, 6 (nov.-dec.), 45 –55.

KAVRAKI, L., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE 12*, 4 (aug), 566 –580.

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

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. *ACM TOG 21*, 3, 491–500.

LIU, Y., AND BADLER, N. I. 2003. Real-time reach planning for animated characters using hardware acceleration. CASA, 86–93.

LOYALL, A. B. 1997. *Believable Agents: Building Interactive Personalities*. PhD thesis, Carnegie Mellon University.

MASSIVE SOFTWARE INC., 2010. Massive: Simulating life. www.massivesofware.com.

MENARDAIS, S., MULTON, F., KULPA, R., AND ARNALDI, B. 2004. Motion blending for real-time animation while accounting for the environment. CGI, 156–159.

MONONEN, M., 2009. Recast/Detour navigation library.

PARIS, S., PETTR, J., AND DONIKIAN, S. 2007. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Computer Graphics Forum 26*, 3, 665–674.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2008. *Virtual Crowds: Methods, Simulation, and Control*. Synthesis Lectures on Computer Graphics and Animation.

PENNEC, X. 1998. Computing the Mean of Geometric Features Application to the Mean Rotation. Tech. Rep. RR-3371, INRIA, Mar.

PERLIN, K., AND GOLDBERG, A. 1996. Improv: a system for scripting interactive actors in virtual worlds. SIGGRAPH, 205–216.

PETTRÉ, J., KALLMANN, M., AND LIN, M. C. 2008. Motion planning and autonomy for virtual humans. In *ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, SIGGRAPH '08, 42:1–42:31.

REYNOLDS, C., 1999. Steering behaviors for autonomous characters.

SHAPIRO, A. 2011. Building a character animation system. MIG, 98–109.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph. 19*, 3 (July), 245–254.

SHOULSON, A., GARCIA, F., JONES, M., MEAD, R., AND BADLER, N. I. 2011. Parameterizing Behavior Trees. In *Proceedings of the 4th International Conference on Motion in Games (MIG '11)*, Springer, 144–155.

SINGH, S., KAPADIA, M., FALOUTSOS, P., AND REINMAN, G. 2009. An open framework for developing, evaluating, and sharing steering algorithms. In *Proceedings of the 2nd International Workshop on Motion in Games*, Springer-Verlag, Berlin, Heidelberg, MIG '09, 158–169.

SINGH, S., KAPADIA, M., HEWLETT, B., REINMAN, G., AND FALOUTSOS, P. 2011. A modular framework for adaptive agent-based steering. In *Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '11, 141–150 PAGE@9.

SINGH, S., KAPADIA, M., REINMAN, G., AND FALOUTSOS, P. 2011. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds 22*, 2-3, 151–158.

SLONNEGER, D., CROOP, M., CYTRYN, J., JR., J. T. K., RABBITZ, R., HALPERN, E., AND BADLER, N. I. 2011. Human model reaching, grasping, looking and sitting using smart objects international symposium on digital human modeling. Proc. International Ergonomic Association Digital Human Modeling.

VAN DEN BERG, J., LIN, M. C., AND MANOCHA, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA*, IEEE, 1928–1935.

WITKIN, A., AND POPOVIC, Z. 1995. Motion warping. SIGGRAPH, 105–108.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: simple biped locomotion control. *ACM TOG 26*, 3.

YOUNG, R. M., AND LAIRD, J. E., Eds. 2005. Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA, AAAI Press.

YU, Q., AND TERZOPOULOS, D. 2007. A decision network framework for the behavioral animation of virtual humans. SCA, 119–128.

DOI: 10.1111/j.1467-8659.2011.02065.x

COMPUTER GRAPHICS *forum*
Volume 0 (2011), number 0 pp. 1–13

A Flexible Approach for Output-Sensitive Rendering of Animated Characters

A. Beacco[1], B. Spanlang[2], C. Andujar[1] and N. Pelechano[1]

[1]MOVING Research Group, Universitat Politècnica de Catalunya, Spain
{abeacco, andujar, npelechano}@lsi.upc.edu
[2]Dept. de Personalitat, Avaluació i Tractament Psicològic, Universitat de Barcelona, Spain
bspanlang@ub.edu

Abstract

Rendering detailed animated characters is a major limiting factor in crowd simulation. In this paper we present a new representation for 3D animated characters which supports output-sensitive rendering. Our approach is flexible in the sense that it does not require us to pre-define the animation sequences beforehand, nor to pre-compute a dense set of pre-rendered views for each animation frame. Each character is encoded through a small collection of textured boxes storing colour and depth values. At runtime, each box is animated according to the rigid transformation of its associated bone and a fragment shader is used to recover the original geometry using a dual-depth version of relief mapping. Unlike competing output-sensitive approaches, our compact representation is able to recover high-frequency surface details and reproduces view-motion parallax effectively. Our approach drastically reduces both the number of primitives being drawn and the number of bones influencing each primitive, at the expense of a very slight per-fragment overhead. We show that, beyond a certain distance threshold, our compact representation is much faster to render than traditional level-of-detail triangle meshes. Our user study demonstrates that replacing polygonal geometry by our impostors produces negligible visual artefacts.

Keywords: crowd rendering, relief mapping, level-of-detail, animated characters

ACM CCS: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Colour, shading, shadowing, and texture.

1. Introduction

Real-time crowd rendering is a key ingredient in many applications, from urban planning and emergency simulation, to video games and entertainment. Crowd simulations typically require hundreds or thousands of agents, each one with its own individual behaviour. Real-time rendering of detailed animated characters in such simulations is still a challenging problem in computer graphics.

Detailed characters are often represented as textured polygonal meshes which provide a high-quality representation at the expense of a high rendering cost. The animation of polygonal meshes is usually achieved through skeletal animation techniques: a set of geometric transformations are applied to the character's skeleton, and a weighted association between the mesh vertices and the skeleton bones (skinning) defines how these transformations modify the mesh geometry. Polygonal meshes are suitable for simulations involving a relatively small number of agents, but not for large-scale crowd simulations, as the rendering cost of each animated character is roughly proportional to the complexity of its polygonal representation.

A number of techniques have been proposed to accelerate rendering of animated characters. Besides view-frustum and occlusion culling techniques, related work has focused mainly on providing level-of-detail (LOD) representations so that agents located far away from the viewpoint are rendered in a more efficient way with little or no impact on the visual quality of the resulting images. A typical approach is to store, for each animated character, a small subset of independent polygonal meshes, each one representing the character at a

different level of detail. Unfortunately, most surface simplification methods are devoted to simplifying static geometry and do not work well with dynamic articulated meshes. As a consequence, the simplified versions of each character are often created manually. Moreover, these simplified representations either retain a large number of vertices, or suffer from a substantial loss of detail, which is particularly noticeable along character silhouettes.

Image-based pre-computed impostors [DHOO05, TC00, TLC02] provide a substantial speed improvements by rendering distant characters as a textured polygon, but suffer from two major limitations: all animations cycles have to be known in advance (and thus animation blending is not supported), and resulting textures are huge (as each character must be rendered for each discrete animation frame and view angle); otherwise characters appear pixelized.

Using separate impostors for different body parts provides a much more memory-efficient approach. Polypostors [KDC*08] subdivide each animated character into a collection of pieces, each one represented using two-dimensional (2D) polygonal impostors. Unfortunately, the representation is view-dependent, the animation sequence still has to be known at construction time, and character decomposition is done manually.

Relief mapping has been proven to be a powerful tool to encode detailed geometry and appearance information. Most importantly, since relief maps support efficient random-access, impostors based on relief mapping are output sensitive, i.e. their rendering cost is roughly proportional to the area of their screen projection. This feature makes relief impostors especially suitable for accelerating the rendering of scenes involving a huge number of objects.

In this paper we present a new representation for animated characters (Figure 1) which uses relief impostors to represent the different body parts of the character delimited by the bones of the skeleton. Each character is encoded through a collection of OBBs, where each box represents the geometry influenced by a particular bone. Textures are projected orthogonally onto the six faces of each box. For each face we store two textures encoding colour, normal and depth values. During animation the bounding boxes are transformed rigidly by a vertex shader according to the transformation of the associated bone in the animated skeleton. A fragment shader efficiently recovers the details of the avatar's skin and clothing using an adapted version of relief mapping.

Unlike competing output-sensitive approaches, our compact representation has no pre-processing requirements (construction can be performed at load time) and does not require us to pre-define the animation sequences or to select a subset of discrete views. Our performance experiments show a significant improvement with respect to geometry rendering. We have also conducted user perception tests validating our technique for rendering agents at middle and far distances from the observer.

2. Related Work

2.1. Crowd rendering acceleration

Rendering a large number of highly realistic animated characters can become a major bottleneck if we render the full geometry of all characters with animation and skinning. To achieve highly realistic populated scenes in real time several techniques have been developed. A well-known solution to this problem involves applying LOD for the characters depending on their distance to the camera [PPB*97]. Ciechomski *et al.* [CSMT05] avoid computing the deformation of a character's mesh by storing pre-computed deformed meshes for each key-frame of animation, and then carefully sorting these meshes to take cache coherency into account.

Impostors have been suggested to avoid rendering the 3D geometry during simulation time. Aubel *et al.* [ABT98] described dynamic impostors, where a multi-resolution virtual human was constructed to be rendered off-screen, from a view-angle and with its animated position, into a buffered texture. The texture was then mapped into a 3D polygon oriented towards the camera. This texture is only refreshed when necessary. Pre-generated impostors were first used by Tecchia *et al.* [TC00] by rendering each character from several viewpoints and for every animation frame of a simple animation cycle. The images were stored in a single texture atlas, and each crowd agent was rendered as a single polygon with suitable texture coordinates according to the view angle and frame. Pre-generated impostors with improved shading have also been used [TLC02]. Impostors can render crowds consisting of tens of thousands of agents, but require a large amount of memory, and at close distances they appear pixelated.

Dobbyn *et al.* [DHOO05] introduced the first hybrid system that presented impostors on top of a full, geometry-based human animation system, and switch between the two representations with minimal popping artefacts. Coic *et al.* [CLM07] described a similar hybrid system but with three LODs, by introducing a volumetric layered based impostor between flat impostor and geometry to help achieve continuity during transitions.

In order to reduce the memory requirements of impostors, while keeping a high level rendering efficiency, 2D polygonal impostors have been used [KDC*08], where an impostor is used per body part and viewing direction. When the character is animated, dynamic programming shifts the vertices of the 2D polygon to approximate the actual rendered image as closely as possible.

Pettre *et al.* [PCM*06] described a three-LOD approach, combining the animation quality of dynamic meshes with the

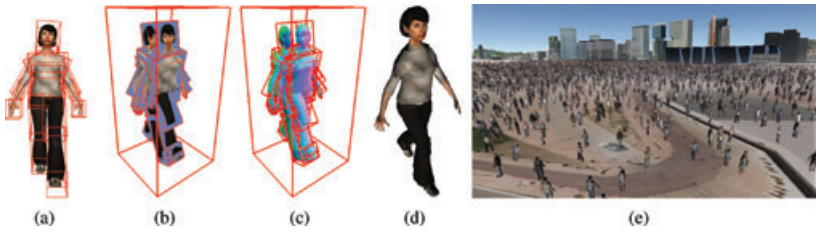


Figure 1: *Overview of our approach: A bounding box is created for each articulated part of an animated character (a), colour (b), normal (c) and depth information is projected onto the box faces, which are rendered through relief mapping (d). Image (e) shows a crowd with about 5000 agents, all of them rendered with our relief impostors.*

high performance offered by static meshes and impostors. A GPU acceleration crowd rendering is presented in [MR06], alternating the use of a single impostor per agent with pseudo-instancing of polygonal meshes.

2.2. Relief mapping

Among image-based techniques, relief mapping [POC05] has proven to be useful for recovering high-frequency geometric and appearance details. Relief maps store surface details in the form of a heightfield. Typically the RGB channels encode a normal or a colour map, while the alpha channel stores quantized depth values. The programmability of modern GPUs allows us to recover the original geometry by a simple ray-heightfield intersection algorithm executed in the fragment shader [POC05]. Acceleration techniques for computing the ray-heightfield intersection include, among others, linear search plus binary search refinement [POC05], varying sampling rates [Tat06], pre-computed distance maps [BD06], cone maps [PO07] and quadtree relief-mapping [SG06].

Other recent techniques adopt a relief mapping approach to encode details in arbitrary 3D models with minimal supporting geometry [BD06, ABB*07]. Unfortunately, these output-sensitive approaches are limited to static geometry.

Only a few works attempt to animate geometry encoded as relief impostors. In [PON08] the animator is requested to create an animation by manually defining and moving a few control points in texture space. Radial basis functions are used to warp the original image by texture coordinate modification. The above method suffers from two major limitations: control points defining the animation are just moved in 2D, providing only image-warp animation, and it does not support standard skeletal animation.

3. Our Approach

3.1. Overview

We aim at increasing the number of simulated agents in real-time crowd simulations by reducing the rendering cost of individual agents. This involves using a simple representation for animated characters supporting output-sensitive rendering, so that rendering times are roughly proportional to the number of rendered fragments, instead of depending on the complexity of the underlying surface. Therefore only characters that are very close to the observer are rendered as polygonal meshes, while the rest of the agents are rendered using our new relief impostor method. We assume the input character conforms to the *de facto* standard in character animation and thus consists of a textured polygonal mesh (skin), a hierarchical set of bones (skeleton) and vertex weights. We assume that both the skin and the skeleton have been designed in a reference pose. The nodes of the skeleton represent joints and the edges represent the bones. Since each bone can be easily identified by its origin, we can use the term joint interchangeably. The transformations affecting joints in the hierarchy are assumed to be rigid. The vertex weights describe the amount of influence of each joint on each vertex.

Since we want to keep pre-processing and memory costs at a minimum while still supporting real-time mixing of animation sequences, we use a separate relief impostor for each animated part of the articulated character. Our representation for distant characters consists of a collection of OBBs, one for each bone in the skeleton, along with a collection of textures projected into the OBB faces, encoding colour, normal and depth values (Figure 1). The OBB will be transformed in the same way as the bones of the skeleton, giving the impression that our impostor character is animated.

Our approach differs from previous work in several aspects. First, we do not attempt to animate a single relief impostor representing a whole character [PON08], but to provide relief impostors representing an already animated character. Second, we require much less memory than competing image-based approaches which require pre-rendering the character for every possible animation frame for a large set of view angles. Third, compared to previous image-based techniques, the cost of adding new characters is drastically lower as new animations can be added at no cost at all. Furthermore, our technique allows blending animations and also running animations at arbitrary speeds (including

slow-motion) since we are not limited to a discrete set of animation frames Finally, our method provides a detailed rendering for any character, viewpoint and animation sequence.

Our implementation relies on the Halca animation library [Spa09, GS10] to draw the animated characters from which we create our impostors. Halca is a hardware-accelerated library for character animation which is based on the Cal3D XML file format [cal] to describe skeleton weighted meshes, animations and materials. Our current implementation works with any animated avatar and any animation that can be exported to the Cal3D format.

3.2. Construction

The construction of our relief impostors from a given 3D character proceeds through the following steps, described in detail below:

1. Associate mesh triangles with impostors.
2. Select a suitable pose for capturing the impostors.
3. Compute the bounding boxes with the chosen pose.
4. Capture the textures of each bounding box.

Step 1. We start by assigning mesh triangles with impostors, where each impostor corresponds to a joint of the articulated character. We assume that each input vertex v_i is attached to joints $J_1, \dots J_n$ with weights $w = (w_1, \dots w_n)$. Now the problem is, given a triangle with vertices v_1, v_2, v_3, to decide which impostors the triangle will be attached to. This determines which triangles will be captured by the impostor. Since we want to keep pre-processing tasks at a minimum, we only tested simple, automatic solutions.

One extreme option is to allocate mesh triangles to joints, attaching each triangle to a single joint. More specifically, each triangle is attached to the joint having the largest influence over the triangle (the influence of a joint over a triangle is computed as the sum of the joint weights over the triangle's vertices). It turns out that this partition tends to produce visible gaps in the joint boundaries during animation; the higher the deviation with respect to the reference pose, the larger the resulting gaps.

The opposite approach would consist of attaching each triangle to a bone if at least one of its vertices is influenced by the bone, regardless of the corresponding weight. Therefore some triangles (those around joints) would be attached to a variable number of impostors, resulting in overlapping parts among joints. These overlapping parts produce protruding geometry when the character is rendered in a pose other than the capture pose.

Therefore we propose an optimized strategy where triangles are assigned to joints based on a given user-defined threshold ϵ. In this approach a triangle is attached to a joint if any of its vertices has a weight above ϵ. This approach assigns triangles to one or more joints, except for triangles where none of the weights are above the threshold. In this particular case we fall back to the first approach, i.e. the triangle is assigned to the joint with the highest influence. On the one hand high values for ϵ result in less protruding artefacts, but on the other hand low values for ϵ result in less cracks. Experimentally, we found that a threshold $\epsilon = 0.5$ worked well on all our test characters, minimizing both cracks and protruding artefacts (Figure 5).

Notice that all the strategies above only use vertex weights and thus are pose-independent.

Step 2. The second step is to choose a suitable pose for capturing the impostors. Triangles are captured according to the chosen pose, i.e. after mesh vertices have been blended according to the pose (using linear blend skinning). This choice of pose affects the captured geometry. Ideally, we should select a pose representing a somewhat average pose of the animation sequence.

For example, if the animation sequence shows a character walking with the arms in a rest position, it is better to capture the triangles around the shoulder with the arms in such a position rather than when stretching arms out sideways. Since impostors will undergo only a rigid transformation, choosing a pose corresponding to a walking animation keyframe tends to minimize artefacts around joints. Our current implementation uses a pose from a walking animation sequence, rather than the reference pose. Notice that the above choice only affects triangles influenced by multiple joints; triangles influenced by a single joint will be reconstructed in their exact position regardless of the selected pose.

Step 3. Once a suitable pose has been chosen, we deform the mesh accordingly by applying linear blend skinning to the mesh vertices, i.e. the transformed vertex v' is computed as $v' = \sum w_i M_{Ji} v$, where M_{Ji} is the rigid transformation matrix from the reference-pose of joint J_i to its actual position in the chosen posture. The bounding box of each impostor is then computed as the oriented bounding box (OBB) of the (transformed) triangles attached to the impostor.

Step 4. The last step is to render the deformed mesh to capture the relief maps corresponding to each one of the six faces of its bounding box. For each bounding box face, we set up an orthographic camera with its viewing direction aligned with the face's normal vector, and then render the triangles attached to the corresponding impostor. We capture the following RGBA textures (Figure 2):

- Colour map: the RGB channels encode the colour, and the alpha channel encodes the minimum (front) depth value z_f.

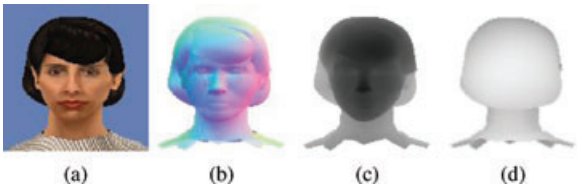


Figure 2: *Colour (a), normal (b), front depth (c) and back depth (d) values are encoded as two RGBA textures.*

- Normal map: the RGB channels encode the normal vector, and the alpha channel encodes the maximum (back) depth value z_b.

Front depth values are captured by rendering the attached triangles with the default GL_LESS depth comparison function. Likewise, back depth values are captured by clearing the depth buffer with a zero value (instead of the default unit value) and switching depth comparison to GL_GREATER. Although storing both depth values is redundant (front depth values of a face equal one minus back depth values of the opposing face), we have chosen this option to improve the locality of texture fetches during rendering.

In order to speed up rendering we reorganized the textures to have both depth values in the same texture. We still keep only two textures, but now the four channels of the first texture encode (R, G, z_f, z_b) and the second texture encodes (B, n_x, n_y, n_z). This reduces to one half the number of texture fetches during the ray-heightfield intersection step of relief mapping.

The vertices of all bounding boxes of a character are stored in a single Vertex Buffer Object (VBO), which is shared by all the instances of the same character.

Colour and normal maps of each character are stored in texture arrays to avoid texture switching while rendering the instances of the same articulated character.

Since a typical animated character for crowd simulation consists of about 21 bones, this accounts for storing $21 \times 6 \times 2 = 252$ RGBA textures per character. This is quite reasonable, considering that competing output-sensitive approaches need to capture the character for each view angle (typically 136 discrete view directions are sampled) and for each animation frame (typically sampled at 10 Hz). So for 1 s of animation, 64×64 textures (which provides a resolution of about 1 cm per texel for geometry, colours and normals) and 4 bytes per pixel, it would require about $136 \times 64 \times 64 \times 10 \times 4 = 22$ MB approximately. With our technique each character requires only about 4 MB of storage.

3.3. LOD for relief impostors

As the characters move away from the camera, we can further speed up rendering by having a hierarchical representation of our relief impostors. We construct a new representation with fewer boxes by merging boxes, i.e. a father node absorbs its child nodes. The OBB associated with the father node is recomputed to include the geometry of the child nodes. New textures are captured for the new OBBs and for this representation all the geometry included in an OBB will undergo the rigid transformation applied to the father. For instance, if we enclose the hand, fore-arm and upper-arm inside a single OBB, then the hand will not move other than following the upper-arm transformations. Notice that once the user selects the target number of boxes for each LOD and the bones associated with each of them, the task of creating OBBs and capturing textures is fully automatic. For the experiments we used relief impostors with 21, 7 and 1 boxes (see Figure 4). The 1-bone LOD has obviously no deformations, which is appropriate only for characters very far away from the camera.

3.4. Real-time rendering

Our current prototype uses multiple LOD representations for each character type; a textured polygonal mesh which is used for agents close to the viewpoint, and the multi-resolution impostor set described earlier for the rest of agents. We first render nearby polygonal agents (grouped by character type to minimize rendering state changes) and then the rest of the agents as impostors (again grouped by character type and LOD).

Each character is rendered through an adapted version of relief mapping over the fragments produced by the rasterization of the transformed bounding boxes. The CPU-based part of the rendering algorithm proceeds through the following steps:

1. Bind the corresponding texture arrays (colour and normal maps) into different texture units, and bind also the VBO with the geometry of the bounding boxes in the pose used to capture the impostors. These steps are performed only once per character type.
2. Draw the bounding box associated with each bone, to ensure that a fragment will be created for any viewing ray intersecting the underlying geometry.

The vertex shader multiplies the incoming vertices of the bounding boxes by the corresponding rigid transformation matrix so that they follow the original skeleton animation. The vertex shader also transforms the variables encoding the location and orientation of each relief map, as these will be used in the fragment shader.

The most relevant part of the rendering relies on the fragment shader, which uses the depth values stored in the colour and normal maps to find the intersection P of the fragment's viewing ray with the underlying geometry. For this particular task any ray-heightfield intersection algorithm can be

Figure 3: *Test data set. Each mesh contains between 4 K and 6 K polygons.*

adopted. Pyramidal displacement mapping [OKL06] is particularly suitable as it guarantees finding the correct intersection on any heightfield and viewing condition. Our current prototype though is based on the simpler relief mapping algorithm described in [POC05].

The fragment shader receives as input the following information:

- World space viewpoint coordinates E.
- World space fragment coordinates C.
- The origin o of the face, i.e. the vertex whose texture coordinates are (0, 0).
- An orthonormal basis of the bounding box face, consisting of a normal vector n and two vectors (u, v) aligned along the horizontal and vertical sides of the transformed face.

The fragment shader computes the intersection of the fragment's viewing ray $r = (C - E)$ with the heightfield encoded by the displacement values stored in the relief map. If no intersection is found, the fragment is discarded. As in [POC05], we use first a linear search by sampling the ray r at regular intervals to find a ray sample inside the object, and then a binary search to find the intersection point. This allows us to retrieve the diffuse colour of the fragment being processed, along with a normal vector to compute per-fragment lighting. Unlike classic relief mapping, we use two depth values z_f and z_b per texel. During the search process, a sample along the ray with depth z is classified as interior to the object iff $z_f \leq z \leq z_b$.

4. Results

We have implemented the construction and rendering algorithms described in C++ and OpenGL 3.2.

The algorithms have been tested on a collection of detailed human characters from the aXYZ Design's Metropoly 2 data set (Figure 3). When converted to Cal3D format, the triangle meshes had 4–6 K triangles, and used 2048 × 2048 texture atlases for diffuse colour and normal data. All models were initially rigged to 67-bone skeletons. Reported results have been measured on an Intel Core2 Quad Q6600 PC equipped with a GeForce 8800 GT.

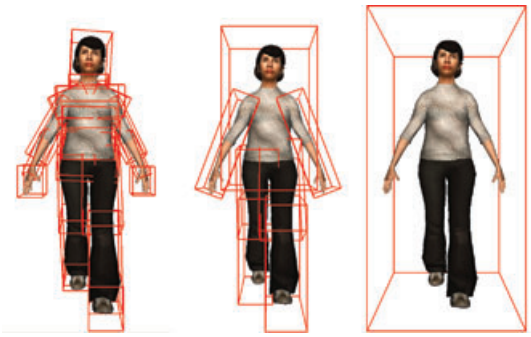


Figure 4: *Relief impostors consisting of 21, 7 and 1 boxes.*

4.1. Impostor creation

The conversion of the input character meshes into a multi-resolution collection of relief impostors took on average 859 ms on the test hardware, including all steps detailed in Section 3.2. We created three LOD representations with 21, 7 and 1 boxes, respectively (Figure 4).

All relief textures (diffuse, normal and depth maps) were captured at 64 × 64 pixels and stored in a single texture array shared by all the instances of the same character. This resulted in $21 \times 6 \times 64^2 \times 8 = 3.95$ MB for the finest LOD, 1.31 MB for the intermediate LOD and 192 KB for the coarsest LOD, i.e. about 5.25 MB per character type.

Although our image-based representation is a bit redundant, both within a LOD level (a single surface point is often captured by 1–3 box faces) and across levels (each LOD has its own collection of relief maps), it is still several orders of magnitude more efficient, in terms of memory space, than competing image-based approaches requiring a separate image for each view direction and animation frame.

4.2. Image quality

Our relief impostor representation aims at accelerating the rendering of animated characters at the expense of some image quality loss. Image artefacts in the resulting images may fall into the following categories (the surface associated to a particular bone will be referred to as a surface patch):

Surface undersampling due to non-height field patches. We represent each surface patch with six orthogonal relief maps, where each relief map stores a single depth value per texel. Therefore we assume that surface patches look like a heightfield when seen from any of these six directions. More formally, we assume that for any axis-aligned ray there is at most one frontface and at most one backface intersection with the surface

patch. If this assumption fails for some axis-aligned direction, the extra intersections between the nearest one at z_n and furthest one at z_f will be ignored, causing the relief mapping algorithm to reconstruct the surface as if the whole segment from z_n and z_f were interior to the object. This could be dealt with by storing multiple depth values per texel, as in [PO06]. Fortunately, our surface patches for the 21-bone skeleton correspond to simple body parts such as upper and lower leg, upper and lower arm, chest and head, for which the heightfield condition earlier typically holds. Therefore the assumption above produces no artifacts without requiring the storage of additional depth values.

Texel-to-pixel ratio. Since our impostors are image-based, the accuracy of the geometric and appearance details is obviously limited by the texel-to-pixel ratio [DHOO05]. Therefore we must ensure that textures are large enough to keep the texel-to-pixel ratio above 1:1 for all the viewing distances associated to the textures. Since our 64×64 textures guarantee the above ratio, no image undersampling artefacts appear in the final images.

Depth quantization. The relief mapping algorithm relies on depth values to find the intersection of per-fragment viewing rays with the underlying heightfield. We quantize such depth values (which are relative to the box dimensions) using 8-bit integers. Since our boxes have moderate sizes (with the longest edge typically below 50 cm), 8-bit quantization results in (at least) 0.2 mm accuracy, which is sufficient to prevent any visible quantization artefact.

Missing ray–surface intersections. Some relief mapping implementations do guarantee that correct ray–surface intersections are always found [OKL06, SG06] whereas others do not [POC05]. This issue has been extensively discussed in the literature and can be dealt with in multiple ways (see e.g. [SG06]).

Lack of geometric skinning. This issue is by far the major limiting factor when considering the valid distance range for our relief impostors. Recall that we animate each relief impostor using the rigid animation of the associated bone. This contrasts with geometric skinning techniques (such as linear blending and dual-quaternion blending) typically applied when animating geometry-based characters, where some vertices are influenced by more than one bone. In our case, each surface patch is fully influenced by its corresponding bone. This obviously results in some artefacts around joints (triangles influenced by a single bone are reconstructed correctly though). These artefacts might include cracks (e.g. if mesh triangles are assigned to a single bone) or overlapping parts (e.g. if each mesh triangle is assigned to all the bones influencing the triangle, regardless of the weights). Fortunately, our optimized construction results in much less artefacts around joints (Figure 5).

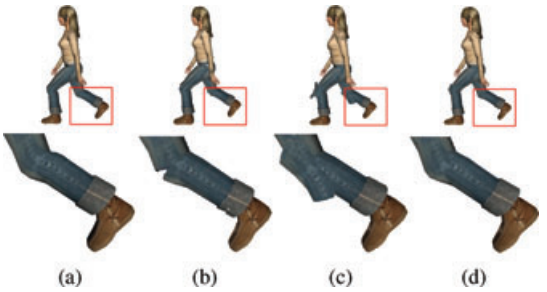


Figure 5: *Artefacts due to lack of geometric skinning: (a) original mesh, (b) impostors created by assigning each triangle to a single joint, (c) impostors created by assigning to each joint all the triangles influenced by the joint and (d) impostors created by our threshold-based strategy.*

Figure 6 shows multiple views of one character rendered with our 21-bone impostors. Note that these artefacts are hardly noticeable for moderate viewing distances (see also the accompanying video). Figure 7 shows several animation frames of the characters in the test data set rendered with our 21-bone impostors. Although the images show that artefacts may appear around the joints, these are very hard to perceive in the context of a crowd simulation. Figure 8 compares renders using 21-bone and 7-bone representations, respectively. The 1-bone LOD obviously supports no deformations and thus it is reserved for characters very far away from the camera.

One of the features of our approach is the joint handling of geometric and appearance details, encoded through displacement, diffuse and normal maps. The effects of reducing the size of the texture maps is illustrated in Figure 9. A side benefit of this approach is that we can use a mipmap pyramid for better minification filtering with no colour bleeding artefacts. This is a feature often lacking in polygonal characters, which typically use texture atlases with multiple disconnected patches, thus hindering mipmapping rendering.

4.3. Mesh versus impostor rendering

Comparing the performance of our impostors with that of the full-resolution mesh is clearly unfair, as in a real-world application, each character instance would be rendered using an appropriate LOD chosen according to, among other factors, its distance to the viewpoint. We thus compare our approach with a discrete collection of LOD meshes. We use the following notation. LOD representations using *relief impostors* will be denoted as R_j, j being the number of bones/boxes. As stated earlier, we constructed three representations R_{21}, R_7 and R_1 with 21, 7 and 1 bones, respectively. LOD representations using textured polygonal meshes will be denoted as M_i, i being the percentage of original polygons, M_{100} denoting the full-resolution mesh. We simplified the input mesh to

Figure 6: *An animated character rendered using our 21-box relief impostor representation.*

generate LODs $M_{90}, M_{85}, \ldots M_5$ and $M_{2.5}$ (Figure 10). Mesh simplification was accomplished using the *Optimize* filter of Autodesk 3DS MAX 2010.

We are now interested in a criterion to measure the quality of each representation, which will be used both to compare mesh- and impostor-based representations, and to select the appropriate LOD according to the distance to the viewer. Note that a measure of the geometric approximation error only makes sense for static polygonal meshes, and that it would ignore visual errors due to distortions of the diffuse and normal maps. We thus adopt an image-space error metric, computed using multiple animation frames and view directions.

Let L be a particular LOD representation of an animated character, using either mesh geometry or relief impostors (i.e. $L \in \{M_{100} \ldots M_{2.5}, R_{21}, R_7, R_1\}$). Let $\phi(L, d)$ be the average image difference resulting from rendering a character at distance d using the representation L instead of the full-resolution mesh M_{100}.

For the sake of clarity, we can consider distance d rather than screen-projected area or subtended solid angle, provided that we fix some viewing conditions. For all the discussion later, we used a $21'$ LCD monitor with a 1024×1024 viewport. The field of view of the camera was set to 60°, and the viewing distance from the LCD monitor was set to 60 cm. Note that these typical viewing conditions for a desktop user can be used to determine both the viewing angle subtended by an animated character at some particular distance from the camera, as well as its screen-projected area. Thus from now on we will refer only to character-to-camera distances.

Since the image difference obviously depends on both the animation frame and the viewing angle, we can compute $\phi(L, d)$ by selecting a representative set of animation frames and a

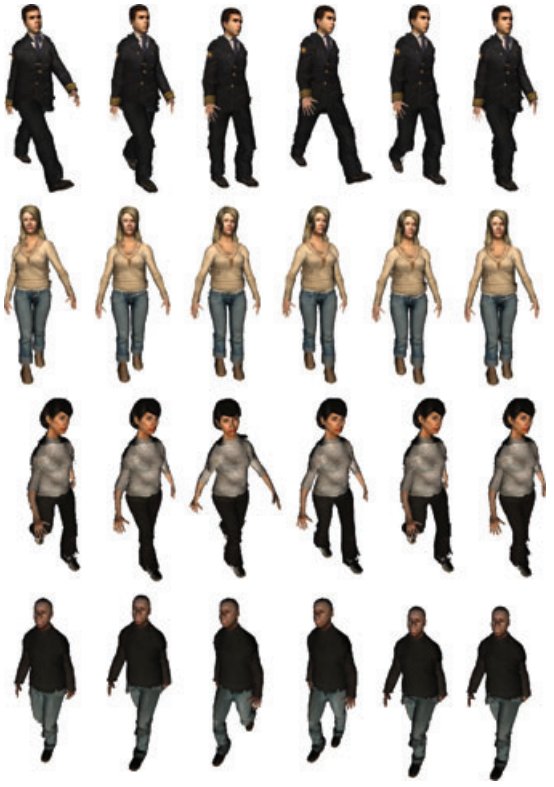


Figure 7: *Relief impostors corresponding to the test data set.*

Figure 8: *Rendering relief impostors with 21 bones (top) and 7 bones (bottom). Note that in the 7-bone representation the head bone has been collapsed with the trunk and thus both undergo the same transformation.*

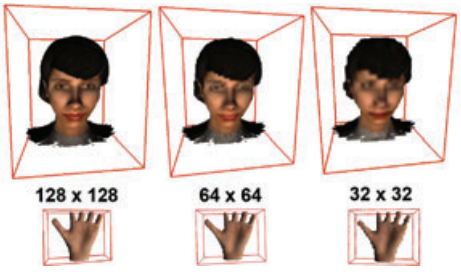


Figure 9: *Relief impostors with decreasing texture sizes.*

sufficiently dense discretization of the view directions in the Gauss sphere, and averaging the resulting image differences. We chose the root mean square (RMS) error as an objective measure to compare the image differences. We could have adopted perceptual-based image metrics [YN04], which integrate factors of the human visual system that reduce the sensitivity to errors, but these metrics are more appropriate for comparing two final, complete images rather than renders of individual agents. This is because high-level HVS models go beyond simple models of brightness and contrast and consider for example masking effects, i.e. decreased visibility of a signal due to background contrast. These effects can be measured only on complete images, where each pixel has a well-defined context. In our case we aim at comparing the rendering of individual characters (thus only a part of the final image) without prior knowledge on the context/background. This is why we discarded HVS-based metrics for comparing the renderings of single, isolated characters, and we chose the broadly adopted RMS error for this purpose. We thus computed $\phi(L, d)$ by averaging (in the L^2 norm sense) the RMS image differences along four equally spaced frames and 10 view directions uniformly distributed on the upper half-sphere surrounding the character. This amounts to 40 samples for computing $\phi(L, d)$, which gives quite reliable results.

Given a certain distance d, we are interested in the simplest mesh level M_i and the simplest relief level R_j the rendering of which produces an image error below some threshold ε. In other words, we want to compute $\min_i\{\phi(M_i, d) < \varepsilon\}$ and $\min_j\{\phi(R_j, d) < \varepsilon\}$. We call the resulting pair M_i, R_j the *optimal* representation for distance d.

We conducted an informal user study to decide a proper error threshold, considering the viewing conditions detailed earlier. Nine users (aged 23–35) participated in the experiment. Users were presented a video showing an animated character side-by-side, one side rendered using detailed polygonal meshes, and the other side using impostors. Every 10 s we doubled the distance from the character to the camera, thus decreasing its screen-projected area. Users were requested to stop the movie as soon as they perceived no difference between both sides of the image. We recorded the resulting RMS error. We observed that the average RMS error was 0.004, and that none of the users were able to find any difference for an RMS error below 0.003. Therefore we set $\varepsilon = 0.003$ to prevent users from perceiving any visual

Figure 10: *From left to right we show the original mesh M_{100} and some examples of the simplified meshes M_{75}, M_{55}, M_{35}, M_{20} and $M_{2.5}$.*

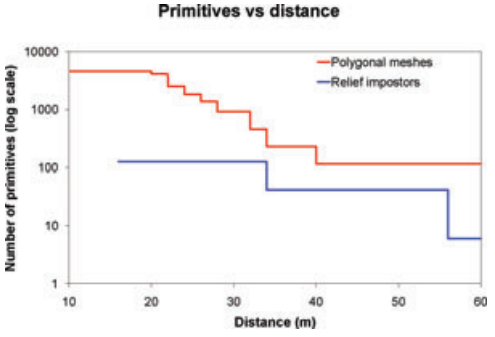


Figure 11: *Minimum number of primitives (triangles/quads) to be drawn (per character instance) to keep the RMS error below 0.003 for one of the test characters. Triangle mesh rendering requires one order of magnitude more primitives (notice the log-scale).*

difference between the original and the simplified representation.

The resulting number of primitives for this error threshold are shown in Figure 11. Note that triangle mesh rendering requires drawing about one order of magnitude more primitives than impostor rendering, under matching quality conditions. For example, for a character at 15 m, a 4 k triangle mesh (8 k vertices) is needed to keep the RMS error below $\varepsilon = 0.003$, whereas the matching impostor has 126 quads (168 vertices). In terms of per-vertex processing, the polygonal mesh requires about 33 k matrix operations for skinning, whereas our impostors requires just 168 operations. For a character at 40 m, the mesh requires 916 matrix operations whereas the impostors only 48. For close-up characters ($d <$ 15 m), the relief-based representation leads to an error above the threshold and thus we fall back to polygonal rendering.

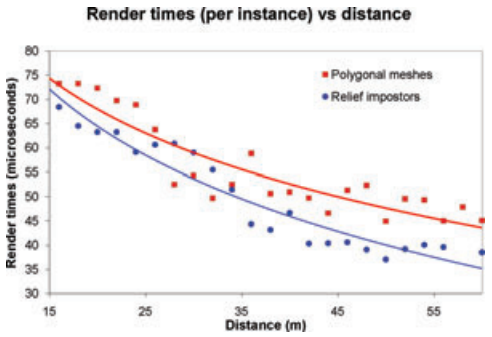


Figure 12: *Render times for the polygonal-based and relief-based LODs that keep the RMS error below 0.003.*

4.4. Choosing the fastest representation

Beyond a given distance (15 m for the chosen threshold), we can choose to render the characters using either the optimal mesh-based or the optimal relief-based representation. Since both provide images with similar quality, it makes sense to choose the appropriate representation according to its performance.

For each distance value d, we measured render times for the optimal mesh-based and the optimal relief-based representations computed earlier. Render times were measured using OpenGL's timer queries, which provide accurate timings. Each query block enclosed the OpenGL drawing commands that need to be executed for each character instance.

For mesh rendering, we used the hardware-accelerated Halca animation library [Spa09, GS10]. Render times are shown in Figure 12. As with image differences, times were averaged for multiple animation frames and view directions, taking 40 samples per distance value.

When rendering polygonal meshes, the bottleneck is likely to be in the vertex processing stage due to the large amount of matrix operations needed to implement skinning. This makes rendering times quite insensitive to the number of fragments produced. However, since for increasing distances we use a more simplified mesh (see Figure 11), the rendering times decrease accordingly.

Relief impostors achieve a drastic reduction in the number of primitives to be drawn, and each vertex is influenced by a single bone. This results in a very small number of per-vertex computations when compared to the equivalent level using mesh geometry. On the downside, fragment processing is more involved due to relief mapping computations. As a consequence, rendering times for impostors also decrease with increasing distances, but this time the shape of the resulting curve can be attributed more to the smaller number of fragments rather than to the reduced number of primitives.

Table 1: *Frame rates for different crowds and camera settings.*

Setting	Agents	Polygonal mesh	Our approach	Speed up
Camera 1	2000	21 fps	50 fps	2.4×
Camera 2	2000	23 fps	47 fps	2.0×
Camera 3	2000	25 fps	49 fps	2.0×
Camera 1	4000	10 fps	38 fps	3.8×
Camera 2	4000	10 fps	36 fps	3.6×
Camera 3	4000	12 fps	34 fps	2.8×
Camera 1	10 000	4 fps	16 fps	4.0×
Camera 2	10 000	4 fps	15 fps	3.7×
Camera 3	10 000	4 fps	14 fps	3.5×

Note that for characters beyond 15 m, using the relief-based representation results in a performance gain.

4.5. Crowd rendering performance

The results provide optimal switch distances for individual agents, but do not show actual frame rates when rendering a complete crowd. Therefore we also measured the frame rate using two different strategies for selecting the appropriate LOD: (1) using only mesh-based levels M_{100}, $M_{95}, \ldots M_{2.5}$, chosen according to Figure 11, and (2) using the full-resolution mesh M_{100} or the optimal relief-based level, whichever is fastest. According to the results discussed, we used the following criteria in option (2) to choose the appropriate representation for each agent: full-resolution mesh for $d < 15$, R_{21} for $15 \leq d < 34$, R_7 for $34 \leq d < 56$ and R_1 for $d > 56$.

Besides the hardware and the number of simulated agents, the actual frame rate depends on many factors, including population density (the higher the density, the higher the number of instances requiring fine LOD levels and thus the lower the frame rate), camera field of view (the higher the fov, the higher the perspective distortion, thus allowing coarser LOD levels), screen resolution and number of agents actually visible.

For the following comparison we used a crowd with a varying number of agents rendered into a 1024×600 viewport (see accompanying movies). Table 1 shows the resulting frame rates for the different crowd scenarios shown in Figure 13 . Note that our approach provides a speed up between $2\times$ and $4\times$.

4.6. User study

We conducted a user study to validate our impostor-based approach in terms of image quality. The main goal of the experiment was to evaluate whether users perceive any image quality loss when using our impostors instead of polygonal meshes. For this purpose, we rendered an animated crowd with two different strategies: (1) using the full-resolution mesh for all characters, and (2) using for each character the optimal representation (mesh or relief impostors), chosen according to the criteria discussed in Section 4.5. We produced a 25 s movie for different crowd settings (Figure 13). We used exactly the viewing conditions detailed in Section 4.3. In order to assess image quality with respect to the reference image, we grouped these movies in pairs, stacking horizontally the movie using impostors with the one using full-resolution meshes (see accompanying videos). The movie using impostors was stacked on the left/right randomly.

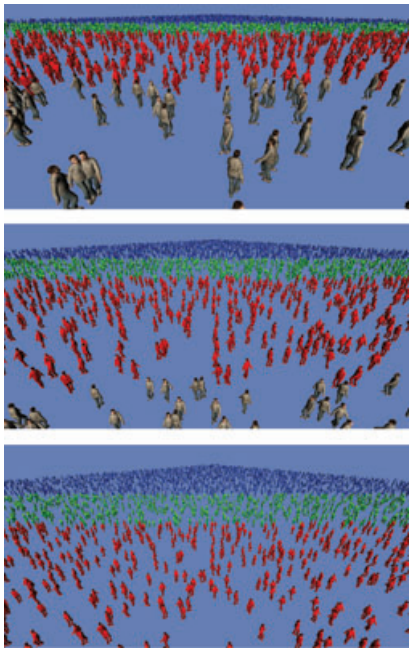


Figure 13: *Camera settings for the performance test.*

Nine subjects (aged 23–35) participated in the experiment. Users were requested to watch five pairs of videos (which were presented in a random order) and to decide which of the two sides (left/right) had better image quality than the other, if any. This yields a total of $N = 45$ trials. Let f_i be the (relative) frequency of users choosing the side with impostors as the best movie. Likewise, let f_g be the frequency of users choosing the side with geometry, and f_u the frequency of users unable to decide which side is better. The absolute frequencies we observed from the 45 samples of our user study where $n_i = 15$, $n_g = 14$ and $n_u = 16$. We consider the null hypothesis to be 'giving the answers by chance' which implies that all conditions should be chosen with equal probability, i.e. H_0: $f_i = f_g = f_u = 1/3$. The corresponding significance levels for a two-sided test against the null hypothesis that each proportion is 1/3 are 0.5, 0.7 and 0.8, therefore the null

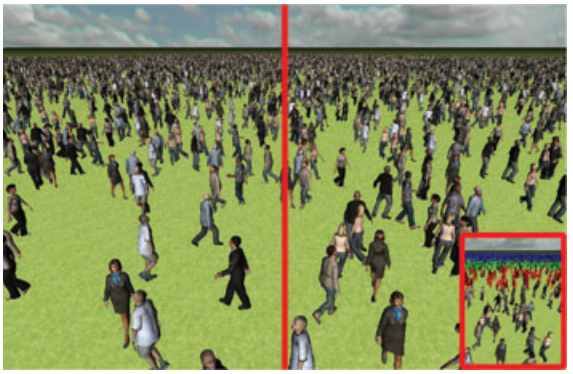


Figure 14: *Image rendered with full-resolution polygonal meshes (left) and our approach (right).*

hypothesis cannot be rejected. This means that the choices of the subjects were equivalent to random choices, and thus our impostor-based technique can be used for rendering acceleration with negligible visual artifacts (see Figure 14).

5. Conclusions and Future Work

We have presented a new method to accelerate the rendering of crowds by using static relief impostors on rigidly animated bounding volumes. Our method allows for real time rendering of thousands of agents. Compared to previous work where impostors were used, our method provides the advantage of being independent from both the viewing direction and the animation clips available. These two advantages offer not only important savings in terms of the memory required to store the impostors, but also that the library of animations can be increased on-the-fly without the need for capturing new impostors. Unlike previous image-based approaches, our approach does support blending between animation clips and even real-time motion capture data.

Our technique could be used in combination with other GPU-based acceleration techniques such as geometry instancing and matrix palette skinning. Geometry instancing [Dud07a] provides performance gains on the CPU side by minimizing drawing calls, and thus both polygonal-based and impostor-based rendering can benefit from this technique. On the downside, geometry instancing poorly supports LOD rendering and frustum culling. Unfortunately, since all visible characters belonging to a given LOD are drawn with a single API call, these tasks have to be completed for all characters before actual rendering starts, effectively sequentializing rather than parallelizing CPU-GPU work. As future work we will explore the potential of these GPU acceleration techniques to further speed up our rendering.

Both mesh- and impostor-based rendering can benefit also from matrix palette skinning [Dud07b] to avoid sending to the GPU the transformation matrices for each bone and character instance. In matrix palette skinning, bone matrices for each frame and for each animation are stored in graphics memory. This allows each agent to have its own distinct pose and animation [Dud07a]. Note however that palette skinning only saves memory bandwidth; it does not affect the number of matrix operations in the vertex shader. Thus the benefits of our approach over mesh rendering in terms of skinning (less vertices to be transformed, and a single bone influencing each vertex) still hold. Other avenues for future work include conducting psychophysical studies supporting the LOD selection instead of the current metric based on RMS. This will evaluate not only the quality of static images but also the impact of visualizing animated characters through relief impostors.

Acknowledgments

This research has been partially funded by the Spanish Government Grant TIN2010-20590-C01-01 and the TRAVERSE ERC Advanced Grant 227985.

References

[ABB*07] Andujar C., Boo J., Brunet P., Fairen M., Navazo I., Vazquez P., Vinacua A.: Omni-directional relief impostors. *Computer Graphics Forum 26*, 3 (September 2007), 553–560.

[ABT98] Aubel A., Boulic R., Thalmann D.: Animated impostors for real-time display of numerous virtual humans. In *VW '98: Proceedings of the First International Conference on Virtual Worlds* (London, UK, 1998), Springer-Verlag, pp. 14–28.

[BD06] Baboud L., Décoret X.: Rendering geometry with relief textures. In *GI '06: Proceedings of Graphics Interface 2006* (Toronto, Ont., Canada, Canada, 2006), Canadian Information Processing Society, pp. 195–201.

[cal] Cal3d. 3d character animation library. http://home.gna.org/cal3d/. Accessed 21 September 2011.

[CLM07] Coic J., Loscos C., Meyer A.: *Three LOD for the Realistic and Real-Time Rendering of Crowds with Dynamic Lighting*. Research Report RN/06/20, Université Claude Bernard, LIRIS, France, April 2007.

[CSMT05] Ciechomski P. D. H., Schertenleib S., Maïm J., Thalmann D.: Reviving the roman odeon of aphrodisias: Dynamic animation and variety control of crowds in virtual heritage. In *Proc. 11th International Conference on Virtual Systems and Multimedia (VSMM 05)* (Ghent, Belgium, 2005), pp. 601–610.

[DHOO05] Dobbyn S., Hamill J., O'conor K., O'sullivan C.: Geopostors: A real-time geometry/impostor crowd rendering system. In *I3D '05: Proceedings of the 2005*

symposium on Interactive 3D graphics and games (New York, NY, USA, 2005), ACM, pp. 95–102.

[Dud07a] DUDASH B.: Animated Crowd Rendering. In *GPU Gems 3*. H. Nguyen (Ed.). Addison-Wesley Professional (2007), pp. 39–52.

[Dud07b] DUDASH B.: Skinned Instancing. In *NVIDIA SDK 10* (2007), http://developer.download.nvidia.com/SDK/10/direct3d/screenshots/samples/SkinnedInstancing.html. Accessed 21 September 2011.

[GS10] GILLIES M., SPANLANG B.: Real-time character engines comparing and evaluating real-time character engines for virtual environments. *Special Issue on Presence 19* (2010).

[KDC*08] KAVAN L., DOBBYN S., COLLINS S., ŽÁRA J., O'SULLIVAN C.: Polypostors: 2d polygonal impostors for 3d crowds. In *I3D '08: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2008), ACM, pp. 149–155.

[MR06] MILLAN E., RUDOMIN I.: Impostors and pseudo instancing for gpu crowd rendering. In *GRAPHITE '06: Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia* (New York, NY, USA, 2006), ACM, pp. 49–55.

[OKL06] OH K., KI H., LEE C.-H.: Pyramidal displacement mapping: A GPU based artifacts-free ray tracing through an image pyramid. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (2006), VRST '06, pp. 75–82.

[PCM*06] PETTRÉ J., CIECHOMSKI P., MAÏM J., YERSIN B., LAUMOND J., THALMANN D.: Real-time navigating crowds: scalable simulation and rendering: Research articles. *Computer Animation and Virtual Worlds 17*, 3–4 (2006), 445–455.

[PO06] POLICARPO F., OLIVEIRA M. M.: Relief mapping of non-height-field surface details. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (2006), I3D '06, pp. 55–62.

[PO07] POLICARPO F., OLIVEIRA M.: Relaxed cone stepping for relief mapping. In *GPU Gems 3: Programming Techniques for High-Performance Graphics and General-Purpose Computation* (2007), Addison-Wesley Professional, pp. 409–428.

[POC05] POLICARPO F., OLIVEIRA M., COMBA J.: Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05: Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2005), ACM, pp. 155–162.

[PON08] PAMPLONA V., OLIVEIRA M., NEDEL L.: Animating Relief Impostors Using Radial Basis Functions Textures. In *Game Programming Gems VII*. Scott Jacobs (Ed.). Charles River Media, Inc., Hingham, Massachusetts (2008), pp. 401–412.

[PPB*97] PRATT D., PRATT S., BARHAM P., BARKER R., WALDROP M., EHLERT J., CHRISLIP C.: Humans in large-scale, networked virtual environments. *Presence 6*, 5 (1997), 547–564.

[SG06] SCHRODERS M., GULIK R.: Quadtree relief mapping. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (2006), ACM, pp. 61–66.

[Spa09] SPANLANG B.: *HALCA Hardware Accelerated Library for Character Animation*. Tech. Rep., Universitat de Barcelona, 2009.

[Tat06] TATARCHUK N.: Dynamic parallax occlusion mapping with approximate soft shadows. In *I3D '06: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2006), ACM, pp. 63–69.

[TC00] TECCHIA F., CHRYSANTHOU Y.: Real-time rendering of densely populated urban environments. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (London, UK, 2000), Springer-Verlag, pp. 83–88.

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Image-based crowd rendering. *IEEE Computer Graphics and Applications 22*, 2 (2002), 36–43.

[YN04] YEE Y. H., NEWMAN A.: A perceptual metric for production testing. In *SIGGRAPH'04: ACM SIGGRAPH 2004 Sketches* (New York, NY, USA, 2004), ACM, p. 121.

Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2011)
A. Bargteil and M. van de Panne (Editors)

Simulating Heterogeneous Crowd Behaviors Using Personality Trait Theory

Stephen J. Guy†, Sujeong Kim, Ming C. Lin, Dinesh Manocha

Department of Computer Science, UNC - Chapel Hill

Abstract

We present a new technique to generate heterogeneous crowd behaviors using personality trait theory. Our formulation is based on adopting results of a user study to derive a mapping from crowd simulation parameters to the perceived behaviors of agents in computer-generated crowd simulations. We also derive a linear mapping between simulation parameters and personality descriptors corresponding to the well-established Eysenck Three-factor personality model. Furthermore, we propose a novel two-dimensional factorization of perceived personality in crowds based on a statistical analysis of the user study results. Finally, we demonstrate that our mappings and factorizations can be used to generate heterogeneous crowd behaviors in different settings.

Categories and Subject Descriptors (according to ACM CCS): I.2.11 [Computer Graphics]: Distributed Artificial Intelligence—Multiagent systems

1. Introduction

Modeling the behavior of large, heterogeneous crowds is important in various domains including psychology, robotics, transport engineering and virtual environments. Heterogeneous crowds consist of dissimilar types of groups, each with potentially independent behavior characteristics and goals [LB97]. According to *Convergence Theory*, crowd behavior is not a product of the crowd itself, rather it is carried into the crowd by the individuals [TK87]. As a result, it is important to accurately model the behavior and interactions among the individuals to generate realistic, heterogeneous crowd behaviors.

In terms of modeling the behavior of individuals within a crowd, even simple tasks, such as walking toward a given destination, involve several complex decisions such as what route to take and the various ways to avoid collisions with obstacles and other individuals. As a result, different people will achieve the same goal in different manners. While there are many factors that govern people's overall behaviors, such as biological and developmental variations, we focus on capturing the portion of these variations that are due to differences in underlying personality.

In general, categorizing the variety of personalities that humans exhibit is a difficult and multifaceted task. While many psychologists have proposed different models to organize this variation in personality, there are limitations in their ability to capturing all types of human personality using a single classifying model [HMS95, RWC00]. In fact, personality can be defined as the interplay between maintaining goal-directness while responding to the demands of the current situation [Per03]. Rather than trying to directly encode this complex interplay by hand, we attempt to characterize these personalities based on data from our user study which asked participants to describe the perceived behaviors of individual agents in computer-generated crowds.

In this paper, we focus on the problem of generating heterogeneous crowd behaviors by adjusting the simulation parameters to emulate personality traits of individuals within a crowd and evaluate the effects of individual personalities on the overall crowd simulation. Our approach is based on Personality Trait Theory, which proposes that complex variations in behavior are primarily the result of a small number of underlying traits. We draw on established models from Trait Theory to specify these variations for each individual. We use the well-known Eysenck 3-Factor personality model [EE85] to establish the range of personality variation. This is a biologically-based model of three independent factors of personality: Psychoticism, Extraversion, and Neuroticism. This so-called PEN model has inspired other similar personality models, most famously the Big-5 or OCEAN personality model [CM92], which proposes five independent axes of personality based on a factor analysis of user responses. The

† contact: sjguy@cs.unc.edu
http://gamma.cs.unc.edu/personality/

OCEAN model has been previously used as framework for exploring variations in crowd simulations [DAPB08].

Our main result is an efficient approach to create and control the perceived personalities of agents in a crowd simulation. We present a mapping between the low-level simulation parameters and high level behavior descriptors. This mapping is used to control the extent that agents exhibit various degrees of aggressive, shy, tense, assertive, active, and impulsive behaviors. We also place these parameters in the context of the PEN personality model. Additionally, we propose a novel two-dimensional factorization of personality traits derived from our empirical study results on perceived personalities in computer-generated crowds. These mappings are used to generate heterogeneous crowd simulations with different, predictable perceived agent personalities.

The rest of the paper is organized as follows. In Section 2 we highlight related work in crowd simulation and behavior modeling. Section 3 gives a brief overview of established personality models and Trait Theory. We describe our user study on perceived personalities in Sec. 4, and Sec. 5 uses the results to compute the mappings. Section 6 demonstrates the resulting behavior of agents simulated using our approach.

2. Previous Work

2.1. Crowd Simulation

Several techniques have been proposed for local collision avoidance and interaction among various agents in crowd simulations. Boids, the seminal work of Reynolds [Rey87], provided a simple method based on forces that push individuals away from each other when they get too close, along with additional forces to provide cohesion in the crowd. The general Boids approach can be extended to simulate more complex crowd behaviors by adding more forces [Rey99].

Other techniques for local navigation also use force-based models, including the Social Force Model [HFV00] and HiDAC [PAB07]. These approaches use complex forces between agents to accurately model local interactions among the agents. Geometric formulations based on (Reciprocal) Velocity Obstacles (RVO) [vdBGLM09] have also been used to model local collision avoidance behavior and generate emergent crowd phenomena [GCC*10].

2.2. Human Behavior Modeling

Many researchers have proposed approaches to simulate crowds that can closely model human behavior. Funge et al. [FTT99] proposed using Cognitive Modeling to allow agents to plan and perform high level tasks. Shao and Terzopoulos [ST05] proposed an artificial life model with several components, that enabled agents to make decisions at both the reactive/behavioral and proactive/cognition levels of abstraction. Yu and Terzopoulos [YT07] introduced a decision network framework for behaviorally animated agents that was capable of simulating interactions between multiple agents and modeling the effect of different personalities.

Other approaches have directly incorporated personality models into crowd simulations. Durupinar et al. [DAPB08] suggested a method to vary the parameters of the HiDAC simulation model based on the OCEAN personality model by choosing a plausible mapping between OCEAN personality factors. Salvit and Sklar [SS11] created a testbed world based on termites collecting food where they demonstrated a variety of food-gathering patterns based on varying parameters of the MBTI personality model.

Perceptual or user studies have been used to improve crowd behaviors and rendering. McDonnell et al. [MLH*09] utilized perceptual saliency to identify important features that need to be varied to add visual variety to the appearance of avatars. McHugh et al. [MMON10] investigated the effect of an agent's body posture on their perceived emotional state. Durupinar et al. [DPA*11] evaluated their method to model the OCEAN personality with a user study.

2.3. Modeling Crowd Styles

Previous approaches have used data-driven methods to produce simulated crowds which behaved with a certain trait or "style". These methods commonly train models for crowd based on input video data. For example, Lee et al. [LCHL07] used data-driven methods to match recorded motion from videos by training a group behavior model. Ju et al. [JCP*10] also proposed a data-driven method which attempts to match the style of simulated crowds to those in a reference video.

3. Personality Models and Trait Theory

Psychologists have proposed various ways of characterizing the spectrum of personalities exhibited by humans. Several theories focus on aspects of personality that show cross-situational consistency, i.e. behavior aspects that are relatively consistent over time and across various situations. While there are many sources of variety in behavior, psychologists have proposed methods to categorize and organize these variations. Our work builds on Trait Theories of personality, a broad class of theories which categorizes people's behavior based on a small number of personality traits [Per03].

3.1. Trait Theory

A personality trait is an habitual pattern of behavior, thought or emotion. While humans display a vast number of different traits, a small number of these traits are believed to be central to an individual's basic personality. Trait theories identify these primary traits, which can be used to describe variations in personality; an individual's personality is described based on a score of how strongly or weakly they exhibit each of these primary traits.

One of the most well established trait theories is the Eysenck 3-factor model [EE85]. This model identifies three major factors which categorize personality: Psychoticism, Extraversion, and Neuroticism (commonly referred to as

PEN). An individual's personality is identified according to what extent they exhibit each of these three traits. The Psychoticism factor is a measure of a person's aggression and egocentricity. The Extraversion factor is a measure of social interest and higher levels of extroversion are associated with more active, assertive and daring behaviors. Finally, the Neuroticism factor is a measure of emotional instability which can correspond to shyness and anxiety [EE77]. Each of Eysenck's three PEN traits have been linked to biological basis, such as the levels of testosterone, serotonin and dopamine present in one's body.

3.2. Factor Analysis

The Eysenck 3-factor model is one of several different trait theories. Other theories have used different methods for classifying the fundamental dimensions of human personality. A particularly successful method of identifying basic personality traits comes from applying *factor analysis* to various user studies where participants use common personality adjectives to describe the behaviors of themselves or others in various situations [CE72]. Factor analysis is the process for determining which small number of unobserved latent variables can describe the behavior of a large number of observed variables. In the context of personality trait theory, the observed variables are the many different adjectives that people use to describe personalities, while the latent variables are a smaller number of axes which explain the correlation in the way people use these personality describing adjectives. An example latent variable might be extraversion, which is associated with the uses of the adjectives outgoing, active, and assertive.

Costa & McCrae [CM92] applied factor analysis to data collected from various personality studies and suggested five primary factors of personality which they dubbed: "Openness to experience", "Conscientiousness", "Extraversion", "Agreeableness", and "Neuroticism" (commonly referred to as OCEAN). While the OCEAN model is very popular, other researches have applied factor analysis to similar user studies and found different factors or different numbers of factor (e.g. the 16 Personality Factor model [CE72]). Additionally, many studies have shown that the five OCEAN factors are not fully orthogonal (i.e. not independent from each other) [DK95]. Furthermore, OCEAN, along with other models such as PEN, deals with personality in the context of general human behaviors. In this work, we seek to study personality specifically within the context of crowd simulations. To that end, we apply a similar factor analysis technique to user responses about personalities perceived in our computer-generated crowds.

4. Behavior Perception User Study

Our goal is to understand how varying parameters in a crowd simulation affects the perceived behavior of agents in the crowd. To this end, we investigated several low-level parameters commonly used in crowd simulations: preferred speed, effective radius (how far away an agent stays from other agents), maximum number of neighbors affecting the local behavior of an agent, maximum distance of neighbors affecting the agent, and planning horizon (how far ahead the agent plans). Many agent-based crowd simulation methods use these or similar parameters to compute the mutual interaction between agents.

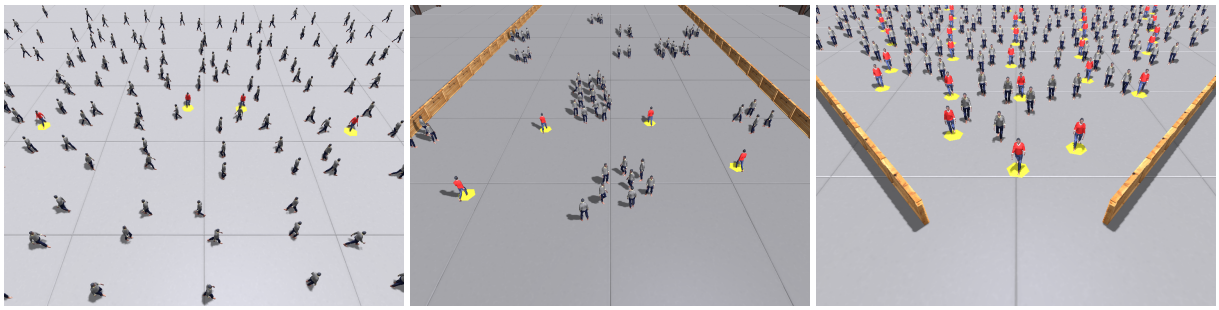
We adopt a data-driven approach and derive a mapping between simulation parameters and perceived agent behaviors based on the results of this perceptual study. Our approach has at least two advantages over trying to hand-tune a plausible mapping. First, it ensures that the perceived personality results are based on the input of a wide range of study participants. Second, it allows for richer, more complex mappings than would otherwise be possible with hand-tuning plausible parameters.

In designing the study, we developed an approach which would satisfy multiple goals. First, the ability to produce mappings to several common adjectives used to describe individuals in crowds, such as "shy", "assertive" and "aggressive". Second, the ability to produce a mapping from simulation parameters to an established psychological theory, such as the Eysenck's PEN model. Finally, the gathered data should be sufficiently rich enough to support a factor analysis that enables us to extract underlying latent variables describing the space of personality seen in crowd simulations.

4.1. Method

To achieve the above stated goals, we designed a user study, which allowed participants to describe behavior in crowd simulations using several adjectives. Our study involved 40 participants (40% female) between 24 and 64 years old, with an average age of 33 years (std. dev. of 12 years). In this study, participants were asked to view three different scenarios of computer generated crowds. In each video, several agents were highlighted to be the focus of user questions. Animations of these scenarios can be seen in the supplementary video. All simulations were created using the publicly available RVO2 Library for multi-agent simulation [vdBGLM09].

Fig. 1 shows a still from each of the scenarios used in the study. The first scenario was the Pass-Through scenario, where four highlighted agents move through a cross-flow of 400 agents. Second was the Hallway scenario where four highlighted agents move through a hallway past 66 other agents, who are in several small groups. Lastly, was the Narrowing Passage scenario where 40 highlighted agents walk alongside 160 other agents towards a narrowing exit. In all cases, the non-highlighted agents were given the default parameters from the simulation library, which mostly results in homogeneous behaviors of the agents in the simulation. The highlighted agents all share the same simulation parameters, that are randomly chosen for each question given to the participants.



(a) Pass-Through Scenario (b) Hallway Scenario (c) Narrowing Passage Scenario

Figure 1: Three crowd simulation scenarios. (a) Four highlighted agents move through crowd. (b) Four highlighted individuals move through groups of still agents. (c) 20 highlighted individuals compete with others to exit through a narrowing passage.

In all scenarios, the highlighted agents are displayed wearing a red shirt with a yellow disc beneath them to allow them stand out in the crowd. Each participant was shown several videos for each scenario with randomly chosen simulation parameters for the highlighted agents. Each video was shown side-by-side with a reference video in which all the agents were simulated using the default set of parameters of the library. This "reference video" was the same for each question involving the same scenario to provide a consistent baseline for comparison.

The participants were asked to rate how the highlighted agents behaved in comparison to those in the reference video. Participants were asked to describe the differences in behavior as being more or less "Aggressive", "Shy", "Assertive", "Tense", "Impulsive" and "Active". These particular six adjectives were chosen both because they are useful in describing behaviors of individuals in crowds, and can span the space covered by the PEN model, with at least two adjectives for each PEN trait [Per03]. Participants then rated each crowd video in terms of all six personality adjectives on a scale from 1-9, with 9 meaning, for example, "much more assertive" than the references video, 5 meaning "about as assertive" and 1 meaning "much less assertive". The participants were allowed to re-watch the videos as many times as they felt necessary, and could go back and forth between questions within a section and revise their answers if desired.

To generate the highlighted agents in the question video the following simulation parameters were randomly chosen: maximum distance to avoid neighbors, maximum number of neighbors to avoid, planning horizon, agent radius, and preferred speed. The random parameter values were shared by all the highlighted agents in each video. The range of the sampled values is shown in Table 1.

For this study, approximately 100 videos were pre-generated for the 3 different scenarios with random values for each of the 5 simulation parameters. Each subject was asked to rate behaviors in several videos randomly chosen from this pool. To keep subjects engaged, the number of videos shown to each participant was limited to 6 randomly chosen clips from each of the 3 different scenarios (18 videos total); users were given the option to skip videos and watched an average of 15 video each. Each video was accompanied with 6 questions, which resulted in a total of approximately 3,600 data points mapping each set of input parameters to perceived levels of various personality traits.

Parameter	Min	Max	Unit
Max. neighbors dist.	3	30	m
Max. num. neighbors	1	100	(n/a)
Planning horizon	1	30	s
Agent radius	0.3	2.0	m
Preferred speed	1.2	2.2	m/s

Table 1: Range of simulation parameters.

5. Data Analysis

Given the large number of data points from the study, we are able to derive a mapping of the relationship between crowd simulation parameters and the perceived personality of the agents. We derive a linear model for the mapping, though other forms of regression are possible.

5.1. Mapping Perceived Behaviors

Using a QR decomposition with column pivoting, we found a linear regression between simulation parameters and perceived behaviors. As input to the regression, we use the difference between the given agents' parameters and those of the agents in the reference video. This removes the need to compute an offset as part of the regression. We also normalized the input by dividing each parameter by half of its min-to-max range to increase the numerical stability of the linear regression.

Our mapping then takes the following form:

$$\begin{pmatrix} Aggressive \\ Assertive \\ Shy \\ Active \\ Tense \\ Impulsive \end{pmatrix} = A_{adj} \begin{pmatrix} \frac{1}{13.5}(Neighbor\ Dist - 15) \\ \frac{1}{49.5}(Max.\ Neighbors - 10) \\ \frac{1}{14.5}(Planning\ Horiz. - 30) \\ \frac{1}{0.85}(Radius - 0.8) \\ \frac{1}{0.5}(Pref.\ Speed - 1.4) \end{pmatrix}$$

Using a linear least-squares approach on the user study data we found the following 6-by-5 matrix A_{adj}:

$$A_{adj} = \begin{pmatrix} -0.02 & 0.32 & 0.13 & -0.41 & 1.02 \\ 0.03 & 0.22 & 0.11 & -0.28 & 1.05 \\ -0.04 & -0.08 & 0.02 & 0.58 & -0.88 \\ -0.06 & 0.04 & 0.04 & -0.16 & 1.07 \\ 0.10 & 0.07 & -0.08 & 0.19 & 0.15 \\ 0.03 & -0.15 & 0.03 & -0.23 & 0.23 \end{pmatrix}$$

Though A_{adj} is a not a square matrix, we can compute a mapping from high-level behaviors specified by the adjectives to simulation parameters by taking its pseudoinverse A^{+}_{adj}. In this way, we can predict the perceived change in behavior of an agent as we adjust the simulation parameters to achieve the desired behavior for each agent.

5.2. Mapping Parameters for the PEN Model

Rather than building a mapping for each of the six personality adjectives individually, we can use a similar procedure to build a mapping for the 3-factor PEN model. The adjectives from the user study can be mapped to the three PEN factors. We use the correspondence of adjective to PEN factors found in Pervin [Per03], summarized in Table 2.

Trait	Adjectives
Psychoticism	Aggressive, Impulsive
Extraversion	Assertive, Active
Neuroticism	Shy, Tense

Table 2: Excerpt from the mapping between adjectives and PEN factors given in [Per03] and used create A_{pen}.

Like the personality adjectives, we can determine a linear mapping for the PEN model, where:

$$\begin{pmatrix} Psychoticism \\ Extraversion \\ Neuroticism \end{pmatrix} = A_{pen} \begin{pmatrix} \frac{1}{13.5}(Neighbor\ Dist - 15) \\ \frac{1}{49.5}(Max.\ Neighbors - 10) \\ \frac{1}{14.5}(Planning\ Horiz. - 30) \\ \frac{1}{0.85}(Radius - 0.8) \\ \frac{1}{0.5}(Pref.\ Speed - 1.4) \end{pmatrix}$$

Based on a linear regression of the study data, A_{pen} was found to be

$$A_{pen} = \begin{pmatrix} 0.00 & 0.08 & 0.08 & -0.32 & 0.63 \\ -0.02 & 0.13 & 0.08 & -0.22 & 1.06 \\ 0.03 & -0.01 & -0.03 & 0.39 & -0.37 \end{pmatrix}$$

Again, this mapping lets us predict expected PEN values from any given simulation parameters.

5.3. Factor Analysis

Analyzing the various features of the A_{pen} matrix, we can observe the strong correlations between the different PEN factors. Psychoticism and Extraversion show a strong positive correlation with each other and both are negatively correlated with Neuroticism. Likewise in the A_{adj} matrix we see a correlation between several factors such as Aggressive and Assertive, which have a Pearson r-squared value of 0.45 in the data collected from our user study. These correlations suggest that a few underlying latent factors might be able to explain the perceived behaviors in the simulations.

Similar to the original OCEAN studies [CM92], we can find these few primary factors using factor analysis methods. By performing a Principal Component Analysis (PCA) on A_{adj}, we found two factors that can explain over 95% of the linear relationship between the simulation parameters and behaviors. This result suggests that low-dimension models such as the PEN model offers sufficiently rich dimensions to characterize personality traits in crowd navigation. The two Principal Component found through factor analysis on our user study data are:

$$\begin{pmatrix} PC1 \\ PC2 \end{pmatrix} = \begin{pmatrix} 0 & -0.04 & 0.04 & 0.75 & 0.66 \\ 0.14 & 0.5 & 0.8 & 0.15 & -0.19 \end{pmatrix}$$

We observe that PC1 primarily has the effect of increasing an agent's radius and speed. PC2 primarily makes agents plan further ahead and consider more agents for local avoidance. For these reasons, we suggestively refer to PC1 as "Extraversion" and PC2 as "Carefulness". Figure 2 shows which personality adjective is most affected, as PC1 and PC2 are jointly varied. The chart indicates that as "Extraverted" agents become more "Careful", they move from appearing Aggressive to Assertive to Active. Likewise, agents who are

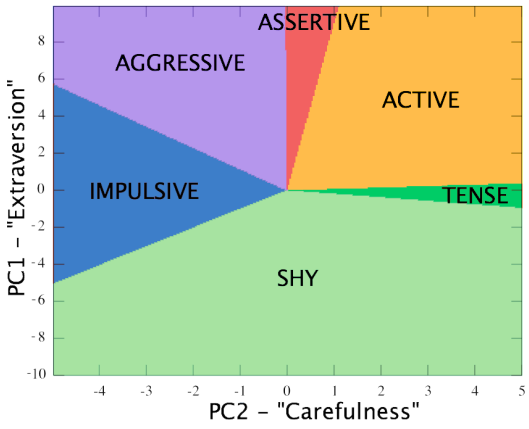
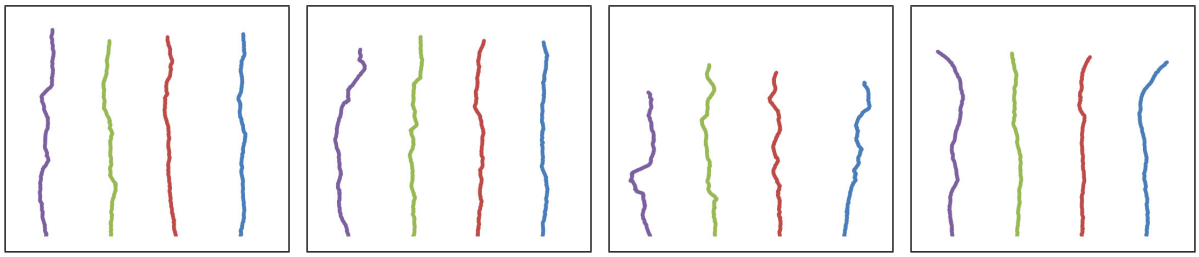


Figure 2: This chart shows which behavior adjective has the largest change as the two principal components are varied.



(a) Aggressive (b) Impulsive (c) Shy (d) Tense

Figure 3: **Pass-through Scenario.** Paths of agents trying to push through a crowd in various simulations. The agent's parameters correspond to various personalities. All paths are displayed for an equal length of time. (a) Aggressive agents make the most progress with the straightest paths. (b) Impulsive agents move quickly but take less direct routes. (c) Shy agents are diverted more easily in attempts to avoid others (d) Tense agents take less jittery paths, but are easily deflected by the motion of others.

not "Extraverted" appear Shy, as long as they are "Careful" enough to avoid looking impulsive. Furthermore, agents who are too "Careful" appear to be Tense. We believe these two principal components cover the personality space in an interesting and intuitive fashion.

6. Simulation Results and Validation Study

Using the above mappings of A_{pen} and A_{adj}, we are able to perform crowd simulations in which certain agents appear to exhibit high levels of the different PEN traits, or appear to display high levels of one or more of the studied personality adjectives. In this section, we show the resulting trajectory of agents displaying various personalities in several different scenarios. We also present the results of a second user study, designed to validate the ability of our approach to generate agents with a given personality using the derived mappings from the user study (see Sec. 5).

For the purpose of this validation study, we clamped the agents' preferred velocities to the range [1.35,1.55] m/s. We chose this range for two reasons. First, this is the range of normal walking velocities observed in crowds [Sti00], which focuses our study on normal behaviors rather than extreme ones. Second, inspecting the columns of A_{adj} and A_{pen} suggests that perceived personalities are most dependent on preferred velocities, by limiting this range we can better highlight the effect of other simulation parameters. Given these constraints on preferred velocity, we then used our mappings to find simulation parameters for various adjectives and traits covered in the user study. Again, to limit unnatural or extreme behaviors, we chose parameters that change behavior by only one "unit" (on the 1-9 scale described in Sec 4.1). The parameters used are summarized in Table 3.

Trait	Neigh. Dist	Num. Neigh.	Plan. Horiz.	Radius	Speed
Psych.	15	40	38	0.4	1.55
Extrav.	15	23	32	0.4	1.55
Neuro.	15	9	29	1.6	1.25
Aggres.	15	20	31	0.6	1.55
Assert.	15	23	32	0.5	1.55
Shy	15	7	30	1.1	1.25
Active	13	17	40	0.4	1.55
Tense	29	63	12	1.6	1.55
Impul.	30	2	90	0.4	1.55

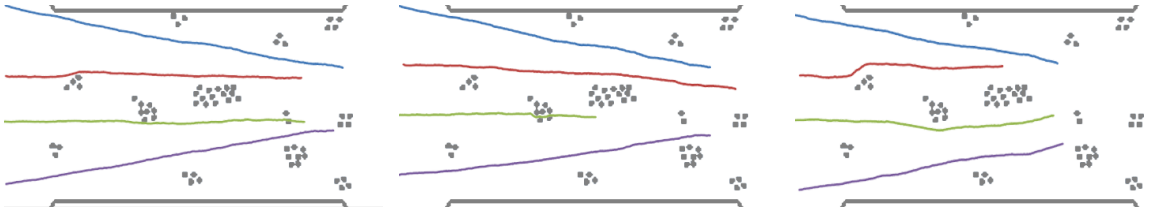
Table 3: Simulation parameters for various personality traits.

6.1. Simulation Results

We now show the results of agents with various personalities in different scenarios. Figure 3 shows paths taken by the highlighted agents in the Pass-Through scenario. The Aggressive agents can be seen to be taking fairly direct paths. The Impulsive agents still move quickly, but tend to take less direct routes. Shy agents avoid others more often, so progress more slowly. Tense agents take the least jittery paths, but are deflected by the crowds more than aggressive agents.

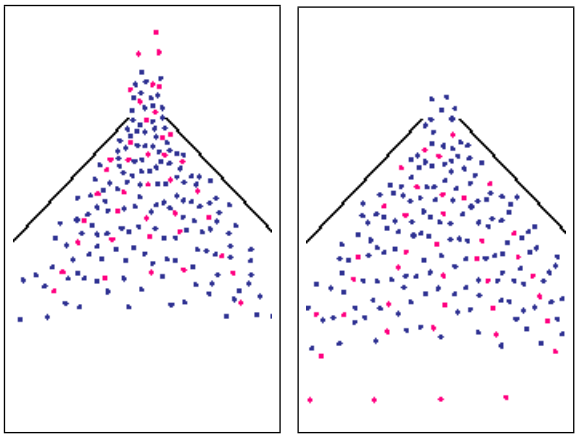
We can also choose agent behaviors based on the Eysnek 3-factor personality model by using A_{pen}. Figure 4 shows the Hallway scenario with agents that have a high level of "Psychoticism" (P-factor), agents with a high level of "Extraversion" (E-factor), and agents with a high level of "Neuroticism" (N-factor). The agents with a high level of Eysnek's P-factor take fast and direct paths coming close to other agents. The agents with a high level of Eynsek's E-factor also move quickly, but take more daring paths, sometimes attempting to weave through the other agents in the crowd. The agents with a high level of Eysnek's N-factor take slower less direct paths and move farther away to avoid the static gray agents.

In the Narrowing Passage scenario, agents also show a variety of behaviors for different personalities. Figure 5 shows the same time-step from two different simulations. In the left simulation, the light red agents are assigned a personality of



(a) High Psychoticism (P) (b) High Extraversion (E) (c) High Neuroticism (N)

Figure 4: **Hallway Scenario.** A comparison between (a) agents with high levels of "Psychoticism", (b) "Extraversion" and (c) "Neuroticism". Each of the four agents' paths is colored uniquely. The high P-factor agents repeatedly cut close to others taking the most direct paths. The high E-factor agents take faster and occasionally "daring" paths, the high N-factor agents take more indirect paths and keep their distance from others.



(a) Aggressive (b) Shy

Figure 5: **Narrowing Passage Scenario.** A comparison between dark-blue default agents and light-red Aggressive agents (a) and light-red Shy agents (b). The Aggressive agents exited more quickly, while several Shy agents stay back from the exit causing less congestion.

Aggressive. In the right simulation, the light red agents are Shy. At this point, a few seconds into the simulation, many more Aggressive agents have moved through the exit than the Shy agents. Furthermore, several of the Shy agents can be seen to be holding back away from the exit causing less congestion.

A comparison of the rate at which the agents of various personalities passed through the exit is shown in Fig. 6. Shy and Tense agents were the slowest to pass through the exit, as they moved less quickly and packed in less tightly than the Aggressive and Assertive agents who made it out fastest.

The evacuation results change when too many of the agents are acting aggressively. Figure 7 shows how the average speed of the Aggressive agents in the scenario varies as the percent of Aggressive agents increases. As the graph shows, our Aggressive agents exhibit the well known "faster-is-slower" behavior associated with panic in crowds [HFV00]. Once a critical threshold of too many aggressive agents is reached, the aggressive agents actually exit the room slower than a non-aggressive agents would.

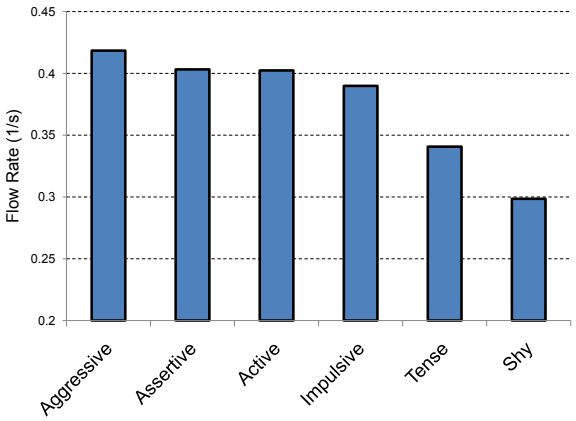


Figure 6: **Exit Rate.** Rate at which agents of various personalities exit in the Narrowing Passage scenario.

6.2. Heterogeneous Crowds

Using the mappings derived from experimental study, we can easily generate different simulations that map to different high-level personality specifications. We can use this capability to create interesting variations in complex, heterogeneous crowd simulations. Here, we chose an evacuation scene, where 215 agents simultaneously compete for space as they leave a room through the same exit. Using the personality-to-parameters mapping, our work can easily create a wide variety of specific behaviors during the evacuation, as shown in Fig. 8. We color code agents' shirts by their personalities, for example agents with red shirts are aggressive and those with brown shirts are shy. The agents behave as expected with aggressive ones exiting first, active agents

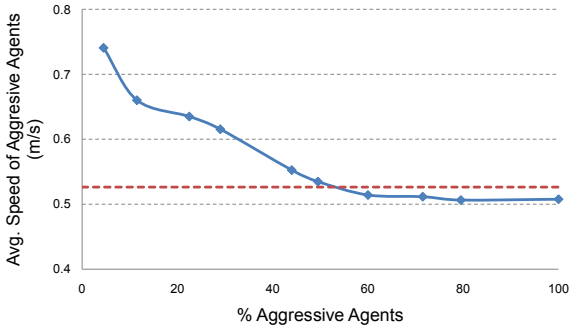


Figure 7: **Faster-is-slower behavior.** This graph shows the speed of Aggressive agents exiting in the Narrowing Passage scenario (solid blue line). As a larger percentage of agents become aggressive, their ability to exit quickly is reduced to the point where they exit more slowly than less Aggressive agents with the same preferred speed (dashed red line). This result is consistent with the well-known "faster-is-slower behavior" [HFV00].

darting around slow agents in front of them and shy agents hanging back. A rendering of this scenario can be seen in the supplementary video.

6.3. Timing Results

Because the behavior mapping can be computed as a preprocessing step, our method adds no overhead to the overall simulation runtime. Table 4 shows the execution time for simulating agents in several different scenarios, the timings were computed on a 3.2 GHz Intel i7 processor. In all cases, the simulation ran at interactive rates.

Scenario	Agents	Obstacles	Time (msec)
Hallway	70	2	0.4
Narrowing Passage	200	2	1.9
Pass Through	404	0	1.4
Evacuation	215	125	4.5

Table 4: Performance timings per frame.

6.4. Validation Study

To validate our personality mappings, we performed a follow-up user study where we asked questions targeted at evaluating how well our model performed at producing simulations with the expected behavior. The study was taken by 19 participants (39% female, average age 37±16), 72% of whom had participated in the original study. This follow-up study consisted of three sections. This validation study used entirely new videos to reduce participant bias. In the first two sections, a personality trait was selected at random,

(a) Initial Conditions

(b) Mid Simulation

Figure 8: **Evacuation Scenario.** 200 agents evacuating a building. Shy agents (brown shirts) hold back while Aggressive agents (red shirts) dart forward. The other personalities also display a variety of behaviors such as quick maneuvers, overtaking and pushing through.

and a pair of videos were generated: one showing a simulation of that trait using the values in Table 3, and one chosen to contrast the selected trait. Participants were asked to choose which of the two videos better showed the personality trait in question. The first section of the study evaluated the six personality adjectives (aggressive, assertive, shy, active, impulse, and tense). The second section evaluated the PEN traits after a brief explanation of each of their meanings to the participants. These sections were intended to measure how well a given personality attribute could be reproduced by our method.

In a third section, participants were shown a video where agents were chosen to display a high level of one adjective while maintaing no increase in another one (e.g. Active, but not Aggressive). Participants were then asked to choose

which of the two adjectives better described the video. This task was intentionally chosen to be challenging, as it explores to what degree our mapping can model each adjective independent of the others. Some combinations (such as "Impulsive, but not Active") were not used in the study as the mapping suggested the adjectives were too strongly correlated to be independently varied within the domain of allowed velocities.

The results of the three sections are summarized in Table 5. For all three sections the model predicted the perceived personalities correctly at a statistically significant rate ($p<.05$). For all results, the statistical p-values were calculated using a a one-tailed test with an exact binomial calculation of probability. The low p-values provide strong evidence these results are due capturing a mapping of traits to parameters and not just statistical noise.

Sec.	Description	Accuracy	p-value
1	Chose video from adjective	87%	1e-7
2	Chose video from PEN trait	96%	1e-11
3	Chose adjective from video	72%	1e-7

Table 5: Performance on validation study

We can further break down the results of the study by analyzing the results for each adjective separately. In the first section, users perform with a 100% success rate at identifying which videos corresponded to Assertive, Shy, and Active. Aggressive and Impulsive were also identified at a high, statistically significant, rate of 80% and 85% respectively.

When combined with the more difficult task of separating two simultaneous personalities constraints (such as Shy, but not Impulsive) the overall success rate drops. However, participants were still able to correctly identify most adjectives at a statistically significant rate. Figure 9 shows a graph of the breakdown of the overall success rate for all questions involving each of the six adjectives. An asterisk next to the adjective indicates a statistically significant result ($p<.05$).

This data suggests the traits of "Aggressive" and "Impulsive" were hard to vary independently without affecting the perceived levels of other traits, such as Assertiveness and Shyness. This result is consistent with the high correlations seen between these adjectives in the initial user study.

Our method also performed well at generating the specific PEN personality traits. Figure 10 shows the success rate for questions involving the PEN values. The high success rate indicates participants were easily able to apply the high level concepts behind the PEN model to evaluating various behaviors in the simulations.

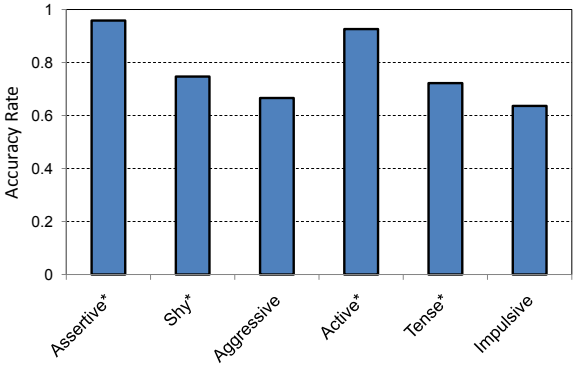


Figure 9: **Adjective Success Rate.** Rate at which user responses matched the indented adjective for all questions involving the six personality adjectives studied. *indicates statistically significant ($p<.05$).

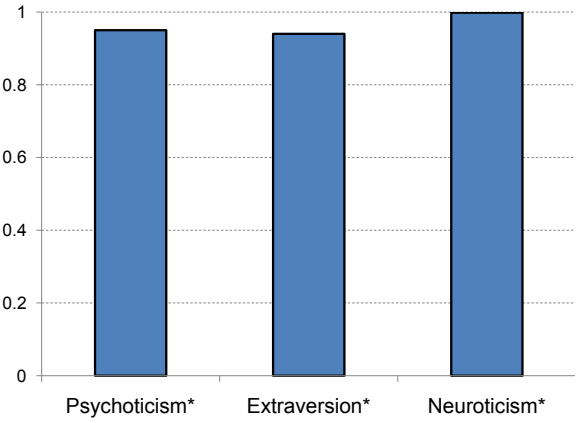


Figure 10: **PEN Success Rate.** Rate at which user responses matched the intended personality trait for questions involving the PEN traits. *indicates statistical significant ($p<.05$).

7. Limitations and Conclusions

7.1. Limitations

Our approach has some limitations. Our current implementation only explores variation allowed by the RVO2 library, we would like to use this approach with other collision avoidance and simulation methods to see if more drastic variation in behavior is possible. Moreover, we focused mainly on local behaviors and interactions between agents. However, the longer-term decision-making process includes global navigation and path-planning which are not modeled adequately by the simulation parameters used in this work. Given the large difference in approach between local and global planning, it is possible other personality models such as the Myers-Briggs Type Indicator [MMQH99] might be more appropriate to capture such behaviors.

Additionally, we compute a generic mapping between simulation parameters and personality traits which is intended to hold across a wide variety of scenarios. By focusing on more specific scenarios, we may be able to find more precise mappings for those particular scenarios. Finally, it may be useful to take into account other aspects of cognitive

modeling to derive mappings such as mapping the effect of internal weightings in decision networks.

7.2. Conclusion

We have presented a perceptually driven formulation to model the personality of different agents in a crowd simulations. Our approach can successfully generate crowd simulations in which agents appear to depict specific, user-specified personalities, such as assertive, shy, and impulsive. Furthermore, we have shown that our approach can successfully generate simulations where agents appear to have various levels of the established PEN personality traits. Finally, we proposed two novel factors (PC1 and PC2) which are highly orthogonal, and are able to capture more than 95% of the linear correlation captured in our experimental data. To the best of our knowledge, this is the first factor-model specifically targeted at analyzing various perceived personalities in crowd simulations.

In the future, we would like to evaluate our approach with other crowd simulation and collision avoidance techniques, including cellular automata and social-force models. We would further like to adopt the same data-driven techniques to build mappings from simulation parameters to other personality trait theories, such as the OCEAN model. We would also like to investigate the extent that our proposed two-factor model is appropriate for human behaviors in real-world crowds (perhaps based on video footage). Additionally, we have focused only on computing the trajectory of the agents. Other aspects of virtual agents such as posture, facial expression, and walking style can provide clues to an agent's personality and we would like to take them into account in our future evaluations.

Acknowledgments This work was supported in part by ARO Contract W911NF-10-1-0506, NSF awards 0917040, 0904990 and 1000579, and Intel.

References

[CE72] CATTELL R., EBER H.: The 16 personality factor questionnaire. *Institute for Personality and Ability Testing* (1972).

[CM92] COSTA P., MCCRAE R.: *Revised NEO Personality Inventory (NEO PI-R) and Neo Five-Factor Inventory (NEO-FFI)*. Psychological Assessment Resources, 1992.

[DAPB08] DURUPINAR F., ALLBECK J., PELECHANO N., BADLER N.: Creating crowd variation with the OCEAN personality model. In *Autonomous agents and multiagent systems* (2008).

[DK95] DRAYCOTT S. G., KLINE P.: The big three or the big five-the epq-r vs the neo-pi: a research note, replication and elaboration. *Personality and Individual Differences* (1995).

[DPA*11] DURUPINAR F., PELECHANO N., ALLBECK J., GUDUKBAY U., BADLER N.: How the ocean personality model affects the perception of crowds. *IEEE Computer Graphics and Applications 31*, 3 (2011), 22–31.

[EE77] EYSENCK S., EYSENCK H.: The place of impulsiveness in a dimensional system of personality description. *The British journal of social and clinical psychology 16*, 1 (1977), 57.

[EE85] EYSENCK H., EYSENCK M.: *Personality and individual differences: A natural science approach*. Plenum Press New York, 1985.

[FTT99] FUNGE J., TU X., TERZOPOULOS D.: Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *SIGGRAPH* (1999), ACM Press, pp. 29–38.

[GCC*10] GUY S. J., CHHUGANI J., CURTIS S., LIN M. C., DUBEY P., MANOCHA D.: Pledestrians: A least-effort approach to crowd simulation. In *Symposium on Computer Animation* (2010), ACM.

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamical features of escape panic. *Nature 407* (2000).

[HMS95] HARVEY R. J., MURRY W. D., STAMOULIS D. T.: Unresolved issues in the dimensionality of the myers-briggs type indicator. *Educational and Psych. Measurement* (1995).

[JCP*10] JU E., CHOI M. G., PARK M., LEE J., LEE K. H., TAKAHASHI S.: Morphable crowds. *ACM Trans. Graph. 29*, 6 (2010), 140.

[LB97] LE BON G.: *The crowd: A study of the popular mind*. Macmillian, 1897.

[LCHL07] LEE K. H., CHOI M. G., HONG Q., LEE J.: Group behavior from video: a data-driven approach to crowd simulation. In *Symposium on Computer Animation* (2007), pp. 109–118.

[MLH*09] MCDONNELL R., LARKIN M., HERNÁNDEZ B., RUDOMIN I., O'SULLIVAN C.: Eye-catching crowds: saliency based selective variation. *ACM Transactions on Graphics (TOG)* (2009).

[MMON10] MCHUGH J., MCDONNELL R., O'SULLIVAN C., NEWELL F.: Perceiving emotion in crowds: the role of dynamic body postures on the perception of emotion in crowded scenes. *Experimental brain research* (2010).

[MMQH99] MYERS I., MCCAULLEY M., QUENK N., HAMMER A.: *MBTI manual*. Consulting Psychologists Press, 1999.

[PAB07] PELECHANO N., ALLBECK J., BADLER N.: Controlling individual agents in high-density crowd simulation. In *Symposium on Computer Animation* (2007).

[Per03] PERVIN L.: *The Science of Personality*. Oxford University Press, Oxford, 2003.

[Rey87] REYNOLDS C.: Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH* (1987).

[Rey99] REYNOLDS C. W.: Steering behaviors for autonomous characters. *Game Developers Conference* (1999).

[RWC00] REISE S. P., WALLER N. G., COMREY A. L.: Factor analysis and scale revision. *Pshycological Assesment* (2000).

[SS11] SALVIT J., SKLAR E.: Toward a Myers-Briggs Type Indicator Model of Agent Behavior in Multiagent Teams. *Multi-Agent-Based Simulation XI* (2011), 28–43.

[ST05] SHAO W., TERZOPOULOS D.: Autonomous pedestrians. In *Symposium on Computer animation* (2005), ACM, pp. 19–28.

[Sti00] STILL G.: Crowd dynamics, phd thesis. *Coventry, UK: Warwick University* (2000).

[TK87] TURNER R. H., KILLIAN L. M.: *Collective Behavior*. Prentice Hall, 1987.

[vdBGLM09] VAN DEN BERG J., GUY S. J., LIN M., MANOCHA D.: Reciprocal n-body collision avoidance. In *Inter. Symp. on Robotics Research* (2009).

[YT07] YU Q., TERZOPOULOS D.: A decision network framework for the behavioral animation of virtual humans. In *Symposium on Computer animation* (2007), pp. 119–128.

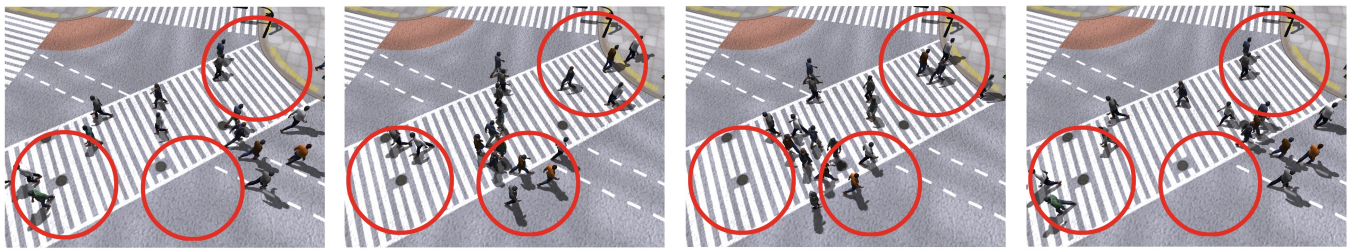
A Statistical Similarity Measure for Aggregate Crowd Dynamics

Stephen J. Guy*
University of Minnesota

Jur van den Berg
University of Utah

Wenxi Liu, Rynson Lau
City University of Hong Kong

Ming C. Lin, Dinesh Manocha
UNC-Chapel Hill



(a) Real-world Data (b) Entropy Metric: 4.7 (c) Entropy Metric: 3.8 (d) Entropy Metric: 2.7

Figure 1: *A comparison between a rendering of real-world crowd data (a), and stills from three different simulation algorithms applied to the same scenario (b-d). Our entropy metric is used to measure the similarity of simulation algorithm to real-world data. A small value of the metric, as in (d), indicates a better match to the data. Differences between the simulations are highlighted with circles.*

Abstract

We present an information-theoretic method to measure the similarity between a given set of observed, real-world data and visual simulation technique for aggregate crowd motions of a complex system consisting of many individual agents. This metric uses a two-step process to quantify a simulator's ability to reproduce the collective behaviors of the whole system, as observed in the recorded real-world data. First, Bayesian inference is used to estimate the simulation states which best correspond to the observed data, then a maximum likelihood estimator is used to approximate the prediction errors. This process is iterated using the EM-algorithm to produce a robust, statistical estimate of the magnitude of the prediction error as measured by its entropy (smaller is better). This metric serves as a simulator-to-data similarity measurement. We evaluated the metric in terms of robustness to sensor noise, consistency across different datasets and simulation methods, and correlation to perceptual metrics.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.6.4 [Simulation and Modeling]: Model Validation and Analysis; I.2.10 [Artificial Intelligence]: Vision and Scene Understanding—Perceptual Reasoning.

Keywords: crowd simulation, validation, data-driven simulations

Links: DL PDF WEB VIDEO

*Email: {sjguy,lin,dm}@cs.unc.edu. The first author is currently an Assistant Professor at the University of Minnesota. This work was done while the second and the third authors were at the University of North Carolina (UNC-Chapel Hill).

1 Introduction

Visual simulation of aggregates systems, including human crowds, animal herds, and insect swarms is a growing area of interest in computer graphics, with applications in diverse areas such as social sciences, swarm intelligence, and city planning. For applications in entertainment, providing artists and animators with high-level control while maintaining visual plausibility of motion is often sufficient. However, for many other training and planning applications, such as virtual reality based training, fire-safety planning, and crowd control and management, it is often critical to model accurate motion, in addition to producing a compelling visual rendering. In this context, we define a measure of a simulator's accuracy based on the similarity of the motion from the simulator to the motion captured in real-world observations. While some previous work has studied the visual plausibility of simulation techniques, we present a new metric for quantifying the similarity between a set of real-world observations and any algorithm designed to simulate the aggregate crowd dynamics captured in the data.

Evaluating the correctness or predictability of the results from a crowd simulation method presents several interesting challenges, many of which arise from the inherent nature of a crowd as a *complex system*. Complex systems are systems composed of several components or elements that interact to exhibit emergent patterns that cannot be easily predicted from the properties of the individual components alone [Schadschneider et al. 2011; Gallagher and Appenzeller 1999]. Because of issues inherent in these systems, such as uncertainty and non-determinism, the study of complex systems generally must draw on techniques from the fields of statistics, information theory, and non-linear dynamics. We likewise draw on inspiration from these fields, in proposing a new method to compare aggregate simulation methods with real-world data that accounts for these challenges.

Real-world data of crowds is becoming increasingly common, driven in part by recent improvement in sensor technology, such as LiDAR and GPS; the proliferation of high-resolution cameras; and advances in computer vision and motion tracking. However, several aspects of a crowd and its aggregate motions make it difficult to directly compare such data against any simulation results. For example, given two very similar initial states, a small crowd can reach two very different configurations after just a few seconds, because the effects of small changes in states can quickly compound

into large differences in the resulting crowd behaviors and motion patterns. This problem is exacerbated by the fact that any data on the motion of aggregate phenomena always comes with noise and uncertainty, making it impossible to know the true state of a crowd with complete accuracy. Additionally, even when presented with the same situation, different individuals can make different decisions. An individual's decisions can also vary under different emotional states (e.g., happy vs. sad) and other subtle factors. Because of the combined effect of all these uncertainties, it is necessary to treat any real-world data on crowd movements as a noisy sample of possible motions rather than an absolute ground truth, and perform a statistical analysis on the motions and behaviors represented by the observed, example motion of the crowd.

Main result: We introduce the *Entropy Metric* to evaluate the predictability of crowd simulation techniques in terms of similarity to real-world crowd data. Our metric is defined broadly and can be applied to any time-series simulation of aggregate motions in continuous space. In this paper, we focus our discussion on and illustrate results for data of human crowds.

The Entropy Metric is an *ensemble* measurement of the prediction errors of a given simulation technique relative to a given example set of crowd motions. At a high level, it works based on a two-stage process. First, we estimate a distribution of simulation states which best represents the observed data. Second, the simulator being evaluated is used to predict each subsequent state from the proceeding one. The smaller this prediction error, the better the simulator's ability to reproduce the motion of real-world crowd system represented by the example data. Because these two steps can depend on each other, we use the Expectation Maximization algorithm (EM-algorithm) to interleave these two steps and iterate until convergence.

By only computing prediction error across small simulation timesteps and by maintaining a distribution of likely simulation states, our formulation fundamentally accounts for noise in the measured data, as well as non-determinism in motion, and unmodeled effects of the given crowd simulation method. Moreover, we show that the Entropy Metric is *rankable, predictable, discriminative*, and *robust with respect to sensor noise*. Furthermore, we demonstrate a correlation between the values of the Entropy Metric and perceived motion similarity (as measured by a perceptual study). The Entropy Metric can be used to automatically select a set of appropriate simulation parameters for data-driven crowd simulation to achieve the desired motion and behavior patterns.

The rest of this paper is organized as follows: Section 2 gives a broad characterization of crowd simulation algorithms and introduces our notation. Section 3 describes the theoretical basis of the Entropy Metric and presents an efficient algorithm to compute it. Section 4 demonstrates the application of the metric to several example crowd simulation algorithms by evaluating them on different sets of real-world data. Section 5 analyzes properties of the metric such as its robustness to noise and correlation to user perceptions.

2 Background and Notation

In this section, we give a brief review of algorithms for crowds simulation and data-capture. We also introduce the notation used in the rest of the paper.

Notation We use the following notational conventions throughout this paper: Variables a printed in italics denote scalars or functions, variables $\mathbf{a}$ printed in boldface denote vectors, and variables $\mathbb{A}$ printed in blackboard bold denote vector spaces. Variables A printed in capitals denote (covariance) matrices, and variables $\mathcal{A}$ printed in calligraphic typeface denote probability distributions.

2.1 Simulation State

In the context of this paper, we use the term "crowd" to refer to an aggregate of entities (e.g. people or agents) whose behaviors and dynamics evolve over time. We define a crowd simulation state as follows: for a given simulator, and a given point in time k, the state $\mathbf{x}_k$ of a crowd contains all information about a crowd that is needed to compute its evolution over time. We denote the space of all crowd states by $\mathbb{X}$. For instance, for a crowd consisting of n agents, being simulated by a technique that is based on the position, velocity, and orientation of each agent on a 2D plane, the crowd state space is $\mathbf{x}_k \in \mathbb{X} = \mathbb{R}^{5n}$. Other time-varying aspects, such as the mental state of the agents or dynamic behavior parameters may also be part of the state. We make no specific assumptions about the representation of a crowd state. In addition to the crowd state, simulators may also use constant information, such as constant parameters shared across all agents, and obstacles that define the environment.

2.2 Crowd Simulation and Aggregate Dynamics

At a broad level, the field of crowd simulation can cover many facets of generating human motions and behaviors (such as full-body biomechanics, facial expressions and gestures, and motion dynamics based on the laws of physics). This makes modeling and analyzing all aspects of a crowd highly challenging and can quickly lead to a combinatorial explosion of potential variations. To increase the tractability of the problem, we focus primarily on the aspect of crowd simulation that corresponds to motion data related to the aggregate dynamics of the crowd.

Aggregate Dynamics: Crowds typically are in constant motion, with individual paths changing over time. Formally, we say that if the state of the crowd at time k is $\mathbf{x}_k \in \mathbb{X}$, the crowd has evolved into a state $\mathbf{x}_{k+1} \in \mathbb{X}$ one unit of time later. The rules defining crowd dynamics (that is, the actual rules governing the human motion and behavior) are unknown and likely cannot be defined by simple mathematical models nor derived from first principles. However, we can characterize these unknown dynamics, using an abstract function $f : \mathbb{X} \to \mathbb{X}$, such that:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k). \tag{1}$$

It should be noted that f is abstract and unknown, and we only use such a formulation to describe crowd evolution. We view a simulator *as an approximation to this function* f; the more accurate and predictive the simulator the better the approximation.

There has been extensive work on computing the pedestrian dynamics or aggregate movement of human-like agents as part of a crowd for more than three decades. These include force-based methods [Helbing and Molnar 1995; Pelechano et al. 2007; Karamouzas et al. 2009], boids and steering models [Reynolds 1987; Reynolds 1999], techniques based on velocity-based reasoning and geometric optimization [van den Berg et al. 2009; Pettre et al. 2009; Ondrej et al. 2010; Guy et al. 2010], field-based and flow-based models [Sung et al. 2004; Treuille et al. 2006; Narain et al. 2009; Patil et al. 2011], cognitive models and decision networks [Funge et al. 1999; Yu and Terzopoulos 2007], and example-driven crowd simulation [Lee et al. 2007; Lerner et al. 2007; Pettre et al. 2009]. These methods model different aspects of crowds, including collision avoidance between agents, emergent phenomena, path navigation, high-level cognition and behaviors. All of these methods share a common formulation though, of computing continuous trajectories for each agent to determine the collective dynamics of the motion in the crowd. This commonality leads to the following abstraction of a crowd simulator:

We formulate a simulator as a function $\hat{f} : \mathbb{X} \to \mathbb{X}$ that attempts to approximate the function f:

$$\hat{f}(\mathbf{x}_k) \approx f(\mathbf{x}_k). \tag{2}$$

That is, simulator $\hat{f}$ takes in a state $\mathbf{x}_k$ of the crowd at time k and produces an estimate of the state $\mathbf{x}_{k+1}$ of the crowd at one unit of time later (henceforth referred to as a timestep). We assume that the underlying crowd simulator works in a continuous space over time and our approach may not be applicable to approaches in discretized space (e.g., techniques based on cellular automata). In Section 4.1, we describe the detailed representation of function $\hat{f}$ for some of the commonly used methods.

2.3 Real-World Crowd Data

Empirical datasets of human crowd motion from videos, LiDAR, and GPS sensors are becoming increasingly available, aided by research in computer vision, robotics and pedestrian dynamics on extracting crowd trajectories from sensors and cameras [Seyfried et al. 2010; Lee et al. 2007; Rodriguez et al. 2009; Kratz and Nishino 2011; Pettre et al. 2009]. A recent trend in research has been the combination of crowd tracking algorithms with crowd dynamics models to extract more accurate trajectories or detect abnormal crowd behaviors [Pellegrini et al. 2009; Mehran et al. 2009].

Most of these tracking algorithms represent the position data or the trajectory as time-stamped vectors $\mathbf{z}_k, \mathbf{z}_{k+1}, \ldots$ that provide a partial (and potentially noisy) projection of the true crowd state $\mathbf{x}_k, \mathbf{x}_{k+1}, \ldots$ at the corresponding moment in time. We assume that the relation between the crowd state $\mathbf{x}_k$ and the data $\mathbf{z}_k$ available of the crowd at time k is given by a known function h:

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{q}_k, \qquad \mathbf{q}_k \sim \mathcal{Q}, \tag{3}$$

where $\mathbf{q}_k$ represents the noise or uncertainty in the real world data, drawn from a constant distribution $\mathcal{Q}$. We assume in this paper that $\mathcal{Q}$ (the sensor uncertainty) is known.

Because our metric measures a simulator's predictability with respect to the set of observed examples, it is important to choose representative data. As our method does not compare a simulator's output to a given set of trajectory data, but rather to the decisions and behaviors captured by the data, it is important to use observed examples which are representative of the behaviors of the real-world crowd. In Sections 4 and 5, we highlight the similarity results on a variety of crowd datasets.

2.4 Validating Crowd Simulators

Crowd simulators have previously been evaluated in terms of perceptual fidelity and other metrics. For example, the work of [Pelechano et al. 2008] evaluates crowd simulations based on quantifying presence in virtual environments. Similarly, [Ennis et al. 2011; Jarabo et al. 2012] measure the perceptual effects of factors such as illumination, camera position and orientation on the perceived fidelity of movement in crowds. In a similar spirit, we also perform a pilot study to measure the correlation between our numerical similarity metric and the perceptual similarity of crowd motion to the validation data.

Other approaches, such as [Singh et al. 2009; Kapadia et al. 2011], present a set of evaluation metrics directly based on the paths generated by a simulator, including path smoothness, number of collisions, or total path length. These metrics are designed to compare the results of different simulations in synthetic environments, but they are not applicable to the evaluation of the similarity between a given simulator and real-world crowd data. Because they provide a different type of motion analysis, these methods should be viewed as complementary to our similarity metric.

2.4.1 Data-driven Crowd Evaluation

Many researchers have proposed methods to measure how closely a simulator matches experimental data. For example, [Pettre et al. 2009] creates a simulation with the same initial conditions as the data and measures the error as a function of the deviation from the recorded trajectories. This approach works well in practice for small numbers of agents, but may not scale to medium or large scenes because of the accumulated, chaotic effect of errors over time.

Other approaches, such as the density measure of [Lerner et al. 2009] and fundamental diagram based comparisons such as in [Seyfried et al. 2010], suggest comparing measures based on crowd densities in the output of a simulator with the observed densities in the experimental data. While density-based metrics are applicable to many simulations, densities are not well defined for sparse scenarios, and metrics based on density will be sensitive to noise for these sparse scenarios and those with a small numbers of agents.

In contrast to previous approaches, our metric is applicable to data with both small and large numbers of agents, and to sparse and dense scenarios. Additionally, because our method is based on a robust, statistical interpretation of the validation data as samples of crowd behavior, the entropy metric can account for sensor noise, handle differences in states between simulators, and account for uncertainty in human motion.

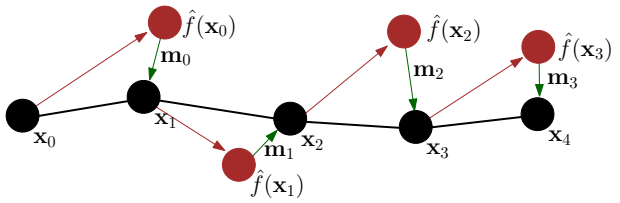
3 Entropy Metric

In this section, we present our Entropy Metric and an efficient algorithm to compute the metric. Fundamentally, we seek to measure the size of the prediction error for a given simulator. That is, given the state of a real crowd $\mathbf{x}_k$, how close does any given simulator $\hat{f}$ comes to predicting the subsequent crowd state $\mathbf{x}_{k+1}$ one timestep later. We denote the error in prediction of the state as the vector $\mathbf{m}_k$ (see Figure 2a). We refer to the distribution of all these error vectors across the entire validation dataset as $\mathcal{M}$ (see Figure 2b). To summarize:

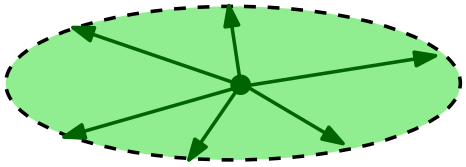
$$\mathbf{x}_{k+1} = f(\mathbf{x}_k) = \hat{f}(\mathbf{x}_k) + \mathbf{m}_k, \qquad \mathbf{m}_k \sim \mathcal{M}. \tag{4}$$

The intuition behind our metric lies in the fact that the distribution $\mathcal{M}$ depends on the underlying simulator $\hat{f}$ and encompasses all error and unmodeled effects in the simulator $\hat{f}$, along with potential non-determinism in the function f. A larger value of this distribution $\mathcal{M}$ implies a larger deviation of the simulator from the states represented in the real world data. This implies a higher dissimilarity between the simulator and real world data. Our proposed similarity metric is therefore the size of $\mathcal{M}$, with a smaller $\mathcal{M}$ implying a more accurate simulation with respect to the data.

In order to quantify the size of the error distribution $\mathcal{M}$, we use the notion of *entropy* from information theory as a measure of the unpredictability of a vector $\mathbf{m}$ from the distribution. The *entropy* of the distribution $\mathcal{M}$ measures the amount of information that is missing from the simulator $\hat{f}$ that would be needed to completely model the function f and capture true crowd motion. As a result, given two simulators, $\hat{f}_1$ and $\hat{f}_2$, the algorithm for which the entropy of $\mathcal{M}$ is lower for some given data is regarded as a better match for that dataset. This leads to the following definition.



(a) *Computing the prediction error each timestep*



(b) *Distribution of all errors over all timesteps*

Figure 2: *(a) For each timestep, there is some error* $\mathbf{m}$ *(green arrow) between the predicted crowd state from the simulator* $\hat{f}(\mathbf{x}_k)$ *(red dots) and the actual crowd state* $\mathbf{x}_{k+1}$ *(black dots). (b) The collection off all errors (green arrows) over all timesteps is denoted as* $\mathcal{M}$ *(green ellipse). The size of this error distribution, as measured by its entropy, forms our metric (smaller is better).*

Entropy Metric: *The entropy of the distribution* $\mathcal{M}$ *of errors between the evolution of a crowd predicted by a simulator* $\hat{f}$ *and by the function* f (lower is better).

A lower value of this entropy implies a smaller error distribution and better similarity with respect to that dataset.

Given this definition of the entropy metric, the underlying challenge is to determine the series of true crowd states $(\mathbf{x}_1 \dots \mathbf{x}_t)$. Because the true states are unknown, we need to estimate them from noisy, real-world crowd data $(\mathbf{z}_0, \dots, \mathbf{z}_t)$. We note that for any given data there are multiple possible crowd simulation states. Instead of inferring one true state $\mathbf{x}_k$, we infer a distribution of likely states $\mathcal{X}_k$. This procedure naturally accounts for data noise and simulator uncertainty. The remainder of this section describes our procedure for estimating the simulations states $\mathcal{X}$ and the error distribution $\mathcal{M}$.

3.1 Computing the Entropy Metric

We compute the Entropy Metric using a two phase process. Firstly, we estimate the crowd states $\mathcal{X}$ from the given validation data. Secondly, for each transition between inferred crowd states, from $\mathcal{X}_k$ to $\mathcal{X}_{k+1}$, we then compute the distribution of prediction errors $\mathbf{m}_k = \mathcal{X}_{k+1} - \hat{f}(\mathcal{X}_k)$ using a maximum likelihood estimator.

To reiterate, the true crowd states and transitions are unknown and must be inferred from the real-world validation data $\mathbf{z}_0, \dots, \mathbf{z}_t$. We estimate the simulator states using *Bayesian Inference* [McLachian and Krishnan 1996]. This is a process which takes observed data $(\mathbf{z})$, a model of how states evolve over time $(\hat{f})$, and an estimate of this model's accuracy $(\mathcal{M})$ to produce an estimate of the likely distribution of true simulation states $(\mathcal{X})$, as in Figure 3.

Unfortunately, this process creates a circular dependency: to estimate the prediction error $\mathcal{M}$ we must know the true crowd states $\mathcal{X}$, and to estimate the true crowd states $\mathcal{X}$ from the data we must know the prediction error $\mathcal{M}$. We solve this problem by taking an iterative approach, first using our best (most recent) guess of $\mathcal{M}$ to infer $\mathcal{X}$, then using this guess of $\mathcal{X}$ to infer $\mathcal{M}$, and continuing to alternate between these two steps until convergence (Figure 3).

This iterative approach to estimation is known as the *EM-algorithm*, and is guaranteed to converge in a coordinate-ascent manner to a locally optimal estimate of the distributions $\mathcal{X}_k$ and $\mathcal{M}$ (in terms of their *likelihood*) given the observed data $\mathbf{z}_0, \dots, \mathbf{z}_t$. This process is summarized in Figure 3 and discussed in detail bellow. This will directly estimate the error distribution $\mathcal{M}$ whose entropy serves as the evaluation metric for the crowd simulator $\hat{f}$. Further discussion of the theoretical foundations of EM and convergence conditions can be found in [McLachian and Krishnan 1996].

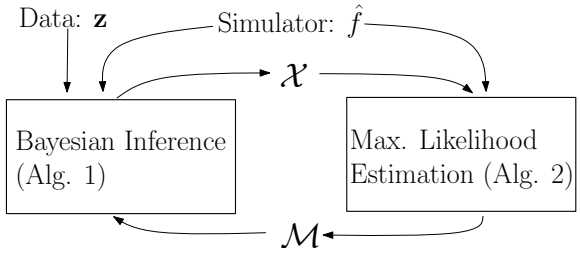


Figure 3: *We estimate the error distribution* $\mathcal{M}$ *for a crowd simulatior via an iterative process based on the EM-algorithm. We use Bayesian Inference to estimate the true crowd states* $\mathcal{X}$ *given data* $\mathbf{z}$*, simulator* $\hat{f}$*, and error distribution* $\mathcal{M}$*. We then compute the maximum likelihood estimate of* $\mathcal{M}$ *given the simulator and estimated state distributions* $\mathcal{X}$*. This process is repeated until convergence.*

3.2 Simplifying Assumptions

There are many difficulties in computing the Entropy Metric exactly, arising from both theoretical and practical issues related to the underlying complexity and non-linear aspects of crowd dynamics, combined with the general non-parametric nature of the distribution $\mathcal{M}$. This makes it necessary to find appropriate approximations and simplifying assumptions which allow us to compute an approximated value of the Entropy metric.

Our most important assumption is that all relevant distributions for computing the metric can be modeled as Gaussians. We must represent the distributions $\mathcal{M}$ and $\mathcal{X}_k$ in some parametric form, and it is natural to choose the first two moments (mean and variance) of the distribution as the relevant parameters. Given only the mean and variance this distribution, the maximum entropy principle suggests a Gaussian distribution as it imposes the least additional constraints on the distribution. Additionally, the Central Limit Theorem suggests that if these error distributions are the results of the combination of many independent sources of error, they can be well modeled as a Gaussian.

We therefore represent the distribution $\mathcal{X}_k$ of the state at time k and the error distribution $\mathcal{M}$ as Gaussian. Furthermore, we assume that a crowd state $\mathbf{x}_k$ is composed of the states of each of the n individual agents within the crowd. Hence, if the state of a single agent has dimension d, then the dimension of the composite crowd state is nd. We make three further assumptions regarding $\mathcal{M}$: (1) The crowd simulator has no systemic bias in the error of its predictions; (2) The crowd simulator is not systemically more accurate for some agents within a crowd than for others; (3) There is no systemic covariance between the prediction errors of different agents within the crowd. Hence, we can assume that the distribution $\mathcal{M}$ has a zero mean, and that its covariance matrix is block-diagonal;

$$\mathcal{M} = \mathcal{N}(\mathbf{0}, \begin{bmatrix} M & 0 & \cdots & 0 \\ 0 & M & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & M \end{bmatrix}), \tag{5}$$

where M is a $d \times d$ covariance matrix which appears n times along the diagonal. In this case, M models the per-agent error variance of the crowd simulator, and the distribution $\mathcal{M}$ is fully defined by the covariance matrix M. This representation also allows for the use of datasets of crowds of different sizes to evaluate a crowd simulator.

3.3 Computing Crowd State Distributions

Computing the Entropy metric involves estimating the prediction error distribution $\mathcal{M}$. As discussed in Section 3.1, given real-world data $\mathbf{z}_0, \ldots, \mathbf{z}_t$, and a simulator $\hat{f}$, we simultaneously estimate the crowd's simulation state $\mathcal{X}$ and error distribution $\mathcal{M}$ using the EM-algorithm. We first describe the mathematical details of estimating the true simulation states from the data.

To estimate the true crowd states $\mathcal{X}$ from the data $\mathcal{Z}$, we use a Bayesian estimation technique based on a variant of the well-known Kalman filter. A Kalman filter provides an optimal estimate of the true state of a model given noisy data, assuming that the model is linear and the noise is Gaussian. While we make a Gaussian noise assumption, we know that crowd models are highly non-linear. Additionally, a Kalman filter estimates the true state at any timestep based only on past data. However, we typically have data both before and after each timestep and can use both to improve the estimate of the true state at any timestep.

Given the above considerations, we use a method know as Ensemble Kalman Smoothing (EnKS) [Evensen 2003]. The EnKS inference algorithm represents the simulation state using a collection of several samples of possible simulator states (called an ensemble). Each sample is updated based on the non-linear motion model of the crowd simulator, and iteratively modified to correspond to the past and future observed data in a manner consistent with the model error $\mathcal{M}$ and data uncertainty $\mathcal{Q}$. The result is a robust estimate of the true simulation state of each agent for each timestep, based on the global data over all agents over all past and future timesteps.

The EnKS algorithm is known to work particularly well for high-dimensional state spaces and non-linear dynamics [Evensen 2003]. In our case, each distribution $\mathcal{X}_k$ is represented by an ensemble of m samples: $\mathcal{X}_k = \{\hat{\mathbf{x}}_k^{(1)}, \ldots, \hat{\mathbf{x}}_k^{(m)}\}$, and we assume an initial ensemble $\mathcal{X}_0$ is given. The method then proceeds as shown in Algorithm 1. This computes a representation for $\mathcal{X}_0, \ldots, \mathcal{X}_t$, given a current estimate of $\mathcal{M}$ and the trajectory data $\mathbf{z}_0, \ldots, \mathbf{z}_t$.

3.4 Computing the Variance M

The second step of the EM-algorithm consists of computing the maximum-likelihood estimate of the distribution $\mathcal{M}$, given the current estimates of the distributions $\mathcal{X}_0, \ldots, \mathcal{X}_t$ of the crowd states as estimated in the first step.

Recall that we do not estimate a single crowd state, but rather a distribution of likely states. Therefore, rather than compute $\mathcal{M}$ directly, we must instead find the most likely value for $\mathcal{M}$ given the inferred distributions of $\mathcal{X}_k$. Since $\mathcal{M}$ is fully defined by the variance M, this is equivalent to maximizing the expected likelihood of M (see Eqn. 5). Further, it is mathematically convenient to maximize the expected *log-likelihood* $\ell\ell(M)$ (as the logarithm cancels against the exponent in the probability density function of a Gaussian distribution), which is equivalent since the logarithm is a monotonic function.

We denote the part of the nd-dimensional crowd state $\mathbf{x}_k$ that contains the state of agent $j \in 1 \ldots n$ as $\mathbf{x}_k[j]$. The expected log-likelihood of variance matrix M is given by:

$$\mathrm{E}(\ell\ell(M)) = -\sum_{k=0}^{t-1}\sum_{j=1}^{n} \mathrm{E}\big((\mathbf{x}_{k+1}[j] - \hat{f}(\mathbf{x}_k)[j])^T M^{-1} \cdot (\mathbf{x}_{k+1}[j] - \hat{f}(\mathbf{x}_k)[j])\big), \qquad \mathbf{x}_k \sim \mathcal{X}_k. \quad (6)$$

Combining Eqn. 6 with the ensemble representations of the distributions $\mathcal{X}_k$, we can compute the maximum likelihood variance M using Algorithm 2.

Algorithm 1: EnKS for estimating crowd states

Input: Measured crowd data $\mathbf{z}_1 ... \mathbf{z}_k$, Crowd Simulator $\hat{f}$, Estimated error variance M
Output: Estimated crowd state distributions $\mathcal{X}_1 ... \mathcal{X}_k$
foreach $k \in 1 \ldots t$ **do**
  `// Predict`
  foreach $i \in 1 \ldots m$ **do**
    Draw $\mathbf{m}_{k-1}^{(i)}$ from $\mathcal{M}$
    $\hat{\mathbf{x}}_k^{(i)} = \hat{f}(\hat{\mathbf{x}}_{k-1}^{(i)}) + \mathbf{m}_{k-1}^{(i)}$
    Draw $\mathbf{q}_k^{(i)}$ from $\mathcal{Q}$
    $\hat{\mathbf{z}}_k^{(i)} = h(\hat{\mathbf{x}}_k^{(i)}) + \mathbf{q}_k^{(i)}$
  $\bar{\mathbf{z}}_k = \frac{1}{m}\sum_{i=1}^{m} \hat{\mathbf{z}}_k^{(i)}$
  $Z_k = \frac{1}{m}\sum_{i=1}^{m} (\hat{\mathbf{z}}_k^{(i)} - \bar{\mathbf{z}}_k)(\hat{\mathbf{z}}_k^{(i)} - \bar{\mathbf{z}}_k)^T$
  `// Correct`
  foreach $j \in 1 \ldots k$ **do**
    $\bar{\mathbf{x}}_j = \frac{1}{m}\sum_{i=1}^{m} \hat{\mathbf{x}}_j^{(i)}$;
    $\Sigma_j = \frac{1}{m}\sum_{i=1}^{m} (\hat{\mathbf{x}}_j^{(i)} - \bar{\mathbf{x}}_j)(\hat{\mathbf{z}}_k^{(i)} - \bar{\mathbf{z}}_k)^T$
    foreach $i \in 1 \ldots m$ **do**
      $\hat{\mathbf{x}}_j^{(i)} = \hat{\mathbf{x}}_j^{(i)} + \Sigma_j Z_k^{-1} (\mathbf{z}_k - \hat{\mathbf{z}}_k^{(i)})$

Algorithm 2: Maximum Likelihood Estimation

Input: Estimated crowd state distributions $\mathcal{X}_1 ... \mathcal{X}_k$, Crowd simulator $\hat{f}$
Output: Estimated error variance M
$M = 0$;
foreach $k \in 0 \ldots t-1$ **do**
  foreach $i \in 1 \ldots m$ **do**
    foreach $j \in 1 \ldots n$ **do**
      $M \mathrel{+}= (\hat{\mathbf{x}}_{k+1}^{(i)}[j] - \hat{f}(\hat{\mathbf{x}}_k^{(i)})[j])(\hat{\mathbf{x}}_{k+1}^{(i)}[j] - \hat{f}(\hat{\mathbf{x}}_k^{(i)})[j])^T$
$M \mathrel{/}= tmn$

The EM-algorithm is initialized with an initial guess for M and $\mathcal{X}_0$, and both steps are repeatedly performed until convergence. The resulting M is a (local-)maximum-likelihood estimate of the error variance of crowd simulator $\hat{f}$.

3.5 Computing the Entropy of $\mathcal{M}$

Given the per-agent variance M as computed above, it remains to compute the entropy of the Gaussian distribution $\mathcal{M}$ of Eqn. (5). This entropy is given by:

$$e(\mathcal{M}) = \frac{1}{2} n \log((2\pi e)^d \det(M)), \quad (7)$$

where n is the number of agents in the crowd, and d is the dimension of the state of a single agent. In order to make our metric independent of the number of agents in the crowd, we normalize the above equation by dividing by n. This gives the entropy of the normal distribution $\mathcal{N}(\mathbf{0}, M)$ that models the per-agent error of the crowd simulator $\hat{f}$. We note that this value is proportional to the log of the determinant of the per-agent variance M, meaning the Entropy Metric follows a log-scale.

4 Implementation and Evaluation

In this section, we demonstrate the application of the Entropy Metric to several crowd simulation algorithms. For each simulation method we evaluate the Entropy Metric on several different sets of simulation parameters, across several different scenarios, each with data gathered from different participants in the scenario. The simulation methods, validation scenarios, and data gathering techniques are described below. The entropy scores for all combinations of parameters, scenarios, and simulators are summarized in Table 1. Because the entropy metric is logarithmic, linear difference in scores corresponds to an exponential difference in performance.

4.1 Simulation Models

We chose three popular simulation methods to test the metric with: a rules-based steering approach, a social-forces model, and a predictive planning approach. Many variants and extensions of all these models have been proposed and widely used in different applications.

Steering Simulator: Steering based simulation approaches use a discrete set of rules to choose agent velocities. We chose a simulation technique based on the classical steering method proposed by Reynolds [1999]. Each agent follows three simple behavior-based rules: steer towards the goal, steer away from the nearest obstacle, and steer away from the nearest person. When obstacles are very close by or when collision are imminent, the avoidance rules are given precedence over the goal-following behavior.

Social Forces Simulator: Social force simulation models use potential fields defined by neighboring agents to impart an acceleration to each agent. We chose a simulation technique based on Helbing's Social Force Model (SFM) [Helbing et al. 2000]. SFM computes the trajectory of each agent by applying a series of forces to each agent that depend on the relative positions and velocities of nearby agents. An agent A receives a repulsive force pushing it away from each neighbor B, denoted as f_{AB}. Moreover, each agent experiences a force pushing it perpendicularly away from the walls or obstacles, denoted as f_W. The magnitude of these forces decreases exponentially with the distance. Each agent also has a goal velocity $\mathbf{v}_A^{pref}$, which is used to compute the desired speed and direction. The simulation function $\hat{f}$ can be summarized as:

$$\mathbf{f}_A^{new} = \frac{\mathbf{v}_A^{pref} - \mathbf{v}_A}{\alpha} + \sum_{A \neq B} f_{AB} + \sum_W f_W \tag{8}$$

where α controls the rate of acceleration.

Predictive Planning Simulator: Predictive, planning based simulators attempt to anticipate collisions based on neighboring agents' positions and velocities and determine new paths which avoid these collisions. We chose a velocity-based formulation called *Reciprocal Velocity Obstacle* as implemented in the RVO2 library [van den Berg et al. 2009]. Each agent navigates by constraining its velocity to those which will avoid collisions with nearby neighbors and obstacles for at least τ seconds. The set of velocity constraints imposed by all the neighbors of an agent A is denoted as RVO_A^τ. An agent is also assumed to have a desired velocity $\mathbf{v}_A^{pref}$. The resulting simulation function $\hat{f}$ can be expressed as:

$$\mathbf{v}_A^{new} = \underset{\mathbf{v} \in RVO_A^\tau}{\operatorname{argmin}} \|\mathbf{v} - \mathbf{v}_A^{pref}\|. \tag{9}$$

The avoidance computation is performed using linear programming, and all agents are assumed to reciprocate (share the responsibility) in avoiding collisions.

For each simulation method, three different sets of parameters were chosen which varying collision radii, preferred speeds, and other internal simulation constants. The resulting simulations that use these parameters are referred to as Steer-1, Steer-2, and Steer-3 for the steering-based approach, SFM-1, SFM-2, and SFM-3 for the social-forces based approach, and RVO-1, RVO-2, and RVO-3 for the predictive planning based approach, respectively.

Further information regarding the implementation of each algorithm are given in Appendix B in the supplementary materials, which details the specific parameters used in each simulation. Additional, this appendix B further describes other implementation details including the specific form of the state vectors, validation data and observation function used to obtain the results in this section.

4.2 Real-World Crowd Data

In order to evaluate the Entropy Metric, we use several sources of data. They correspond to different real-world scenarios (both indoor or outdoor) and have varying number of agents. Each dataset was captured using different sensing hardware (see Table 2).

Lab Setting: This data comes from a study performed in a controlled setting in a motion capture lab. In this scene, two people were placed about 6m apart and were asked to swap their positions [Moussaïd et al. 2011] (see Fig. 4a). We label this benchmark with two agents as *Lab*.

Street Crossing: This data comes from a video of pedestrians walking on a street, which was captured using an overhead camera. The trajectories of each agent were extracted using multi-object tracking [Pellegrini et al. 2009] (see Fig. 4b).

Importantly, we use data from two different capture sessions involving different groups of people crossing the same street. This allows us to test for correlation in the results of metric between different groups of people for the same scenario. The datasets from the two groups are labeled as *Street-1* and *Street-2*.

Narrow Passageway: This data comes from a large indoor experiment designed to capture human exiting behavior through passages of varying sizes [Seyfried et al. 2010]. The experiment involved use of markers and optical tracking equipment to gather high-quality data corresponding to subjects' positions near the entrance of the passageway. The experiment was performed with different exit widths, with each run consisting of hundreds of participants, about 50 of whom were in the tracked area at any given time (see Fig. 4c).

Again, we use data from two different runs of this experiment, the first with a passage of very narrow width of 1m and the second with a wider passage of width 2.5m, denoted as *Passage-1* and *Passage-2*, respectively. This is used to analyze correlation between similar scenarios.

For all five scenarios, we assume that the goal position for each agent is the last tracked position in the dataset and compute $\mathbf{v}^{pref}$ accordingly. However, in some scenarios (such as people moving

Scenario	RVO-1	RVO-2	RVO-3	SFM-1	SFM-2	SFM-3	Steer-1	Steer-2	Steer-3
Passage-1	3.048	2.329	3.400	6.576	6.581	6.579	6.403	6.490	6.435
Passage-2	1.991	0.690	1.990	5.430	5.458	5.451	4.713	4.748	4.764
Street-1	2.744	3.156	2.800	4.500	4.707	4.665	2.979	3.569	3.838
Street-2	2.709	2.564	2.520	3.793	3.885	3.780	2.660	2.744	3.060
Lab	1.920	1.610	1.230	2.538	2.523	2.509	1.871	1.847	2.305

Table 1: *Entropy Metric for different simulation algorithms on various real-world datasets (lower is better).*

(a) *Lab (small)* **(b)** *Street (medium)* **(c)** *Passage (large)*

Figure 4: *Our rendering of real-world crowd trajectories used for evaluation.*

in a maze) this assumption may not hold. In such cases $\mathbf{v}^{pref}$ can be considered as part of the state and inferred along with other parameters using Bayesian inference.

Scenario	Agents	Density	Capturing technique
Passage-1	40	2.76	optical tracking+camera
Passage-2	59	2.38	optical tracking+camera
Street-1	18	0.42	overhead camera
Street-2	11	0.37	overhead camera
Lab	2	-	motion capture

Table 2: *Real-world crowd datasets used by our evaluation algorithm. We report the average number of agents per frame and the density of the agents over the tracked area.*

4.3 Results

As can be seen in Table 1, different simulators vary in their ability to capture the motion characteristics of different datasets. Furthermore, for a given simulation method, different parameter sets also score better or worse. This suggests that maximizing the similarity to the data involves choosing not only the right simulator, but the right parameters. Some scenarios resulted in a relatively high Entropy Metric value across all simulators. For example, the best score for Passage-1 (the narrower passage) was worse than the worst score for the Lab scenario. This suggests that all the tested simulators performed poorly in terms of capturing the complex behaviors pedestrians exhibited in the narrow passage scenario.

5 Analysis

While Section 4 provides the results of the Entropy Metric on different simulators and validation datasets, this section analyzes the metric itself. Specifically, we analyze the metric in terms of predictiveness, consistency, robustness to noise, correlation with perceptual similarity, and other important properties. To begin with, we highlight several useful properties of the metric which follow directly from its mathematical definition.

Rankable results: For a given validation dataset, the Entropy Metric provides unique, global rankable results because it computes a single number in $\mathcal{R}$. The result can be ranked uniquely when there are no ties. If the Entropy Metric for $\hat{f}_1$ is lower than the Entropy Metric for $\hat{f}_2$, this implies that simulator $\hat{f}_1$ better captures the aggregate dynamics in that dataset than $\hat{f}_2$.

Discriminative: The data presented in Table 1 highlights the discriminative nature of the Entropy Metric. In contrast to approaches which test a simulator against a discrete set of benchmarks, the Entropy Metric returns a real number, eliminating the risk of ties. This allows us to generate a clear quantitative ranking of different simulators.

Generality: The Entropy Metric makes very few assumptions about the underlying simulator and the real-world data. This is because the Bayesian inference framework is capable of estimating the complete simulation state (position, velocity, orientation, etc.) based on only partial validation data (e.g., only positions). This allows to us to compare simulators that use only position and velocities to others that also account for orientation and accelerations, or other simulation specific parameters.

Because the entropy measure directly compares a simulation to reference data, it directly reflects optimizations made to increase the accuracy of simulations. Appendix A in the supplemental material provides a case study showing a correlation between improvements to RVO, and a decrease in the Entropy scores of the resulting simulations. These results can also be seen in the supplemental video.

5.1 Consistency

It is important that the Entropy Metric provides consistency in terms of results across similar datasets. Based on the empirical results presented in Table 1, we can determine that the Entropy Metric has this property. Specifically, the results on similar benchmarks are well correlated with each other. For example, the ordering from best to worst simulators for the benchmarks Passage-1 and Passage-2 does not differ significantly even though the data changes. This suggests the metric can reliably capture some inherent aspect of a simulator's ability to reproduce the movement through a passage.

We can numerically measure the correlation between scenarios using Pearson's correlation coefficient r, which measures correlation

Scenario	Correlation
Passage-1 & Passage-2	.975
Street-1 & Street-2	.917
Street-1 & Passage-2	.585
Passage-1 & Street-2	.414

Table 3: *The Entropy Metric results on similar datasets such as (Passage-1,Passage-2) or (Street-1,Street-2) are highly correlated. The metric shows lower correlation for different dataset pairs.*

on a scale from 0 (uncorrelated) to 1 (exactly correlated). The results from computing the Entropy Metric on two different datasets from the same scenario are highly correlated ($r > .9$), implying a consistency in the metric across similar datasets. The results from different datasets (e.g. Street vs Passage) are much less correlated. This is because different simulators have different abilities to capture different types of motion, which is reflected in the metric. These correlation results are summarized in Table 3.

5.2 Robustness to Noise

Any data-driven analysis of a simulator needs to consider the effects of noise present in the data. Even in controlled lab settings, there are small amounts of sensor noise, the effects of which can be magnified over time. These effects are even more pronounced when working with data captured from outdoor natural scenes. This data is normally produced by video processing techniques, which can have large amounts of errors compared to data gathered in a controlled lab setting.

We can analyze the effect of noise on the Entropy Metric by artificially adding noise to the validation datasets. In particular, we highlight the results on the data from outdoor street crossing scenario. We add uniformly distributed noise to the data of up to a half meter in size (while keeping $\mathcal{Q}$ constant). We then compute the Entropy Metric for three different simulators (RVO-1, SFM-2, and Steer-3) by varying the noise. The results are shown in Fig 5.

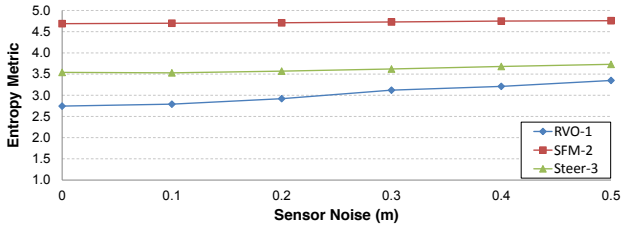


Figure 5: *We evaluate the impact of adding artificial error (uniformly distributed) to the Street-1 benchmark for different simulators. The Entropy Metric is relatively stable to this error and the relative ranking of different simulators does not change.*

As expected, the Entropy score gets worse for all simulators as more noise is added because the error from the noise is being attributed to the simulation. However, the metric handles the noise robustly, with the value of the metric changing in a slow, continuous fashion as large amounts of noise are added. Even with an extreme value of .5m of noise being added randomly every timestep, the metric maintains the same relative ranking between the simulators.

5.3 Similarity between Two Simulators

While the Entropy Metric is designed to compare a simulator to a validation dataset, we can also use it to compare two simulators. This comparison is performed by running a simulator to generate paths and using these paths as the validation dataset for the second simulator. In this way we can compute a score measuring the similarity of two simulators, with a lower score implying a better match between the aggregate dynamics of two simulators. Table 4 shows the results of this comparison on three different simulators. The results are asymmetric mainly due to differences in how sensitive each simulator is to the noise. A symmetric comparison can be achieved by averaging the pairwise differences.

Simulator	RVO-1	SFM-2	Steer-3
RVO-1	**0.19**	3.17	3.16
SFM-2	4.34	**1.38**	1.58
Steer-3	2.26	1.66	**0.90**

Table 4: *Results from using the Entropy Metric to compare two simulators As expected, the Entropy Metric is the smallest when a simulator is compared to itself.*

As expected, the similarity matrix is minimized along the diagonal, indicating that each simulation is most closely related to itself. Additionally, the match between RVO and the other two models is much worse than the match between SFM and Steer (i.e., $\sim$3 vs $\sim$1.5). We speculate that this is due to the fact that RVO uses a predictive approach to collision avoidance, while the other two approaches both use a model based on reactive, distance-based forces.

5.4 Data-driven Crowds and Behavior Modeling

Data-driven approaches are becoming more common in crowd simulation. In part, this is because of their ability to capture complex or subtle behaviors that to not directly map to simulation parameters. Here, we explore the Entropy Metric as it applies to a data-driven crowd simulation designed to capture high-level personality differences. Specifically, we look at simulators based on high-level personality models such as the Five-factor Trait Theory or OCEAN model. This model classifies a personality based on its level of Openness, Conscientiousness, Extraversion, Agreeableness and Neuroticism [Costa and McCrae 1992]. Recent crowd work such as [Durupinar et al. 2008] and [Guy et al. 2011] have proposed data-driven models for each of these personality traits.

In order to evaluate our metric, we analyze one trait (Extraversion), similar results also hold for other data-driven models of high level behaviors. We use a validation dataset consisting of five agents walking past each other with a high degree of Extraversion (as reported by [Guy et al. 2011]), and vary the simulator to have differing amounts of Extraversion. The Entropy results match well when we choose a simulator trained to match a high Extraversion dataset, and the Entropy metric was worse when using a simulator trained for low amounts of perceived Extraversion. As we interpolate between the two simulators using the perceptually linear personality space defined in [Guy et al. 2011], we see a linear improvement in the Entropy metric scores as the simulator moves from less Extraversion to more, see Figure 6. The linearity of the match suggests the Entropy Metric may also be well correlated with a perceptual notion of similarity, this is further explored in Section 5.5.

5.5 Comparison to Perceptual Evaluation

We conducted a user study to analyze how well the numerical similarity of the Entropy metric corresponds to perceptual similarity. This study involved 36 participants (22 males) and had two sections; the first was designed to directly investigate the correlation between the Entropy metric and perceived similarity, and the second section was designed to analyze how well the metric can predict a user's perceived similarity. When studying perceptual evaluation in crowd videos it is important to note the effect that rendering choices, such as cloned appearance and motion of individuals, can

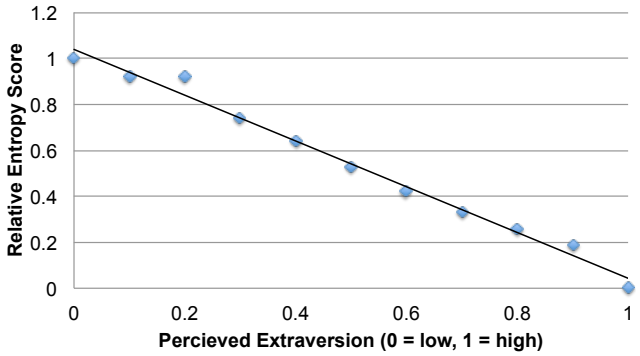


Figure 6: *Comparison of varying levels of the Extraversion personality trait to the Entropy score given an dataset of Extraverted motion. As the data-driven simulation is modified from a less Extraverted model, towards a more extraverted one the Entropy metric with respect to a highly Extraverted dataset decreases.*

affect a user's perception of simulated crowds [McDonnell et al. 2008]. To mitigate this effect, we rendered all motion users saw with the same visual crowd models and rendering parameters.

In the first section of the study, users were shown two videos. The first was a rendering of the simulation and the second was a rendering of the real-world data. The real-world motion was re-rendered with the crowd rendering system used for the simulated motion. Users were asked to rate the pairs of videos in terms of their similarity to each other on a Likert Scale of 0 (not at all similar) to 10 (very similar); this was done across four different simulators, each with a different entropy score compared to the real-world data. The results are shown in Fig 7.

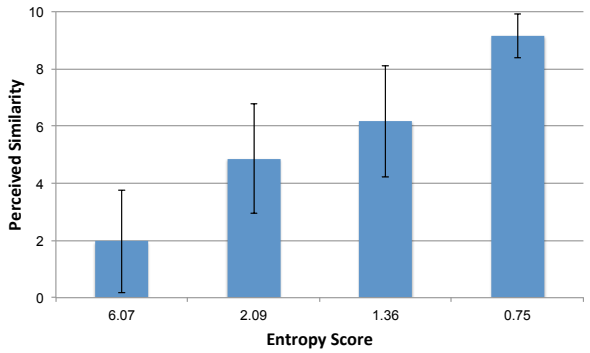


Figure 7: *Comparison of Entropy score (lower is better) to perceived similarity (higher is better) across four different simulators, error bars represent mean absolute deviation in user scores. Simulators with a lower Entropy Metric were consistently given a higher score in terms of perceived similarity to the source video by users.*

As can be seen in Fig 7, when the entropy score is very large (6.07), users gave the simulation a low score for similarity, generally ranging from 0 to 4. When the entropy score was very small (0.75), users gave the simulation a high score for similarity, generally ranging from 8 to 10. For entropy scores in-between, the users gave correspondingly intermediate similarity scores. Numerically, the Entropy Metric and user reported perceived similarity have a Pearson's correlation coefficient of .91, which indicates the Entropy Metric is strongly correlated with perceived similarity.

The second section of the user study was structured using the two-alternative forced choice (2AFC) procedure. For each question, a user was shown videos from two different simulations along with reference video from real-humans walking in the same environment. The users were asked two questions: which simulator matched the real-world data better and whether the two simulators had similar or different behaviors.

A priori, we would expect users to choose the simulator with a lower Entropy score as the one which matched the data better, as this is what the Entropy score seeks to measure. Figure 8 shows the accuracy of this prediction versus the relative differences in Entropy score between the two videos. When the Entropy score was greater than 0.1, the metric correctly predicted the user response with a high accuracy rate, and at statistically significant level ($p \leq .01$).

When the relative difference in Entropy scores between two simulations were very small (less than 0.1), the metric failed to correctly predict user preferences at a statistically significant level. However, for these scenarios the metric correctly classifies the simulators as "very similar", with users classifying these simulations as similar 94.3% of the time.

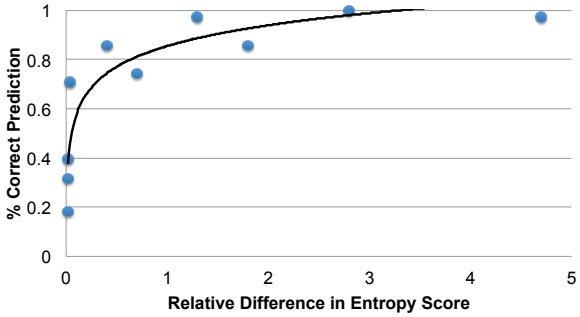


Figure 8: *Relative entropy differences versus user preferences. When the entropy score difference between two simulators was more than 0.1, users agreed with the score's prediction at a statistically significant rate.*

To summarize, the Entropy Metric correlates well with perceived similarity. When the metric indicated a large difference between two simulators, the users agreed with the metric at a statically significant rate (i.e. a rate much greater than chance). When the metric indicated very small differences, users also agreed that the difference was small at a statistically significant rate. This result is significant for those using this metric to design data-driven simulators and for those who want to evaluate simulators used in applications such as games and special effects. By finding simulation parameters that minimize the Entropy Metric, the resulting simulations will visually appear progressively closer to the target data without the need for a human in the loop to evaluate the intermediate results.

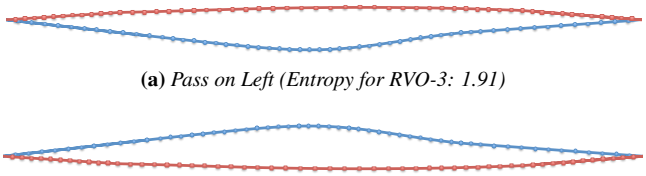
These results also provide a meaningful scale for the metric. The sharp inflection in Fig 8 around 0.1 suggests that this value may be a good estimate of the just-noticeable-difference level in the metric. Likewise, Figure 7 suggests that simulations with an Entropy score less than 1 should be considered very visually similar to the source data and those with a score greater than 6 visually very different.

5.6 Motion Uncertainty

As discussed in Section 2.4, many approaches have been suggested for evaluating a simulators ability to match data. The main novelty of the Entropy Metric comes from its ability to effectively handle sensor noise, its correlation with perceptual similarity, and its robust treatment of the uncertainty in human motion. The first two of these features have been discussed above; here, we analyze the effect of motion uncertainty.

When two people interact, their paths can vary for a variety of reasons. However, small variations early in a path can lead to large deviations later on. These deviations make it hard to compare two trajectories directly. This is what makes comparing trajectories an inappropriate similarity metric. To illustrate this, we used data corresponding to two people swapping their positions in the Lab scenario and mirrored the paths so the participants pass on the right rather than on the left. Both of these paths are equally valid ways of getting between the given start and end positions, and ideally would result in similar validation scores for a simulation.

Since the Entropy Metric does not a compare a simulator to the trajectory data directly, but rather to the decisions latent in that data, it computes nearly identical scores for the two datasets. For example, the RVO-3 simulator scores 1.91 for the original and 1.92 for the mirrored data. However, if average trajectory differences were used as the metric, the score would change significantly. This is because one simulator can predict passing either on the left or the right, but not both. In contrast, because the Entropy Metric is computed over a *distribution* of likely simulator states, this results in similar scores for two datasets.



(a) *Pass on Left (Entropy for RVO-3: 1.91)*

(b) *Pass on Right (Entropy for RVO-3: 1.92)*

Figure 9: *Two agents (red and blue) switch positions. Regardless of whether the agents pass each other on their left sides (a) or their right sides (b), the Entropy Metric for a given simulator remains almost unchanged.*

6 Assumptions and Limitations

When deriving our approach, we have made some basic assumptions that can help inform the appropriate use of the Entropy Metric for simulation evaluation. Most importantly, we assume that there exists some corpus of representative crowd data for a simulator to be compared to. Because the metric evaluates at the level of individual motion decisions it is important that the validation dataset contains motion which is representative of the types of motions considered realistic. Additionally, the basic formulation of the entropy metric does not directly measure characteristics arising from emergent phenomena, such as density, lane formation, and overall flow rate, though such quantities may be indirectly inferred through the motion of individuals in the crowd and would be of interesting topics for future investigation.

The formulation of the entroy metric assumes that all uncertainties are known and can be modeled with zero-meaned gaussian distributions. Such uncertainties include uncertainty in the validation data, errors in the estimation of the environment, senor noises, etc. This assumption extends to the error of a given simulation technique, which is what the Entropy metric seeks the measure. If any of these errors do not fit this model (for example, a simulation technique which has a systematic bias), our method will naturally account for these by estimating a larger variance for the error distribution. Likewise, any errors in modeling the environment or sensor uncertainty will also increase the estimated error of a simulation (see Section 5.2). By explicitly estimating model uncertainty, the Entropy metric seeks to account for the aggregate effect of the non-determinism in human motion, errors in sensors, and unmodeled sources of error. In this way, our metric can gracefully handle a breakdown of the mathematical assumptions underlying its derivation.

While our metric is simple and provides a consistent, rankable measure of similarity to source data as a single real number, the metric is not invariant to changes in timestep and measurement units. We also assume that the sensing accuracy $\mathcal{Q}$ is known, though it may be possible to extend our approach to unknown distributions.

In contrast to other methods that measuring density, collision counts, and other quantities, this approach instead estimates the error between the motion computations performed by a given simulator and with those captured in the validation dataset. Fundamentally, this type of analysis is applicable to where there is known ground truth motion data, along with knowledge about the environment. Therefore, the Entropy Metric is only defined for a given set of real-world crowd data and cannot directly compare different simulators in the absence of such data. It cannot measure the plausibility of a simulator in the absence of real-world 'ground truth', nor can it be directly applied to scenarios without data to validate against.

7 Conclusion and Future Work

We have introduced an Entropy Metric for evaluating crowd simulators against real-world crowd data. Our metric provides a meaningful quantification that can be used to rank various crowd simulators in a predictive and consistent manner. To the best of our knowledge, this is the first attempt at designing an objective, quantitative evaluation metric that measures the similarity of crowd simulation results with respect to real-world data and explicitly accounts for sensor noise, motion uncertainty, and non-determinism. We have used it to evaluate the performance of steering behaviors, force-based models, and predictive crowd simulation algorithms on different real-world datasets.

In the future, we hope to extend this work to analyze other widely used crowd simulation techniques, such as continuum techniques, vision-based steering, data-driven approaches, foot-step planners, and cognitive models – all of them can be described using continuous formulations introduced in this paper. Furthermore, we would like to apply our metric to a wider variety of real-world crowd data sets that measure many different aspects of motion, including local density, individual behaviors, and apparent personalities. We are also interested in exploring applications of this metric, such as automatic optimization of crowd simulations to data.

Acknowledgements We are grateful comments from the reviewers, and thankful for help from Sujeong Kim and Sean Curtis. This research is supported in part by ARO Contract W911NF-04-1-0088, NSF awards 0917040, 0904990, 100057 and 1117127, and Intel.

References

COSTA, P., AND MCCRAE, R. 1992. *Revised NEO Personality Inventory (NEO PI-R) and Neo Five-Factor Inventory (NEO-FFI)*. Psychological Assessment Resources.

DURUPINAR, F., ALLBECK, J., PELECHANO, N., AND BADLER, N. 2008. Creating crowd variation with the OCEAN personality model. In *Autonomous agents and multiagent systems*.

ENNIS, C., PETERS, C., AND O'SULLIVAN, C. 2011. Perceptual effects of scene context and viewpoint for virtual pedestrian crowds. *ACM Trans. Appl. Percept. 8*, 10:1–10:22.

EVENSEN, G. 2003. The ensemble kalman filter: theoretical formulation. *Ocean Dynamics 55*, 343–367.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Proc. of ACM SIGGRAPH*, 29–38.

GALLAGHER, R., AND APPENZELLER, T., Eds. 1999. *Science Magazine*, vol. 284. AAAS.

GUY, S., CHUGGANI, J., CURTIS, S., DUBEY, P., LIN, M., AND MANOCHA, D. 2010. Pledestrians: A least-effort approach to crowd simulation. *Proc. of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 119–128.

GUY, S. J., KIM, S., LIN, M., AND MANOCHA, D. 2011. Simulating heterogeneous crowd behaviors using personality trait theory. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, The Eurographics Association, 43–52.

HELBING, D., AND MOLNAR, P. 1995. Social force model for pedestrian dynamics. *Physical Review E 51*, 4282.

HELBING, D., FARKAS, I., AND VICSEK, T. 2000. Simulating dynamical features of escape panic. *Nature 407*, 487–490.

JARABO, A., EYCK, T. V., SUNDSTEDT, V., BALA, K., GUTIERREZ, D., AND O'SULLIVAN, C. 2012. Crowd light: Evaluating the perceived fidelity of illuminated dynamic scenes. *Proc. of Eurographics*. to appear.

KAPADIA, M., WANG, M., SINGH, S., REINMAN, G., AND FALOUTSOS, P. 2011. Scenario space: characterizing coverage, quality, and failure of steering algorithms. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 53–62.

KARAMOUZAS, I., HEIL, P., BEEK, P., AND OVERMARS, M. 2009. A predictive collision avoidance model for pedestrian simulation. *Proc. of Motion in Games*, 41–52.

KRATZ, L., AND NISHINO, K. 2011. Tracking pedestrians using local spatio-temporal motion patterns in extremely crowded scenes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 99, 1–1.

LEE, H., CHOI, M., HONG, Q., AND LEE, J. 2007. Group behavior from video: a data-driven approach to crowd simulation. In *Proc. of Symposium on Computer Animation*, Eurographics Association, 109–118.

LERNER, A., CHRYSANTHOU, Y., AND LISCHINSKI, D. 2007. Crowds by example. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3.

LERNER, A., CHRYSANTHOU, Y., SHAMIR, A., AND COHEN-OR, D. 2009. Data driven evaluation of crowds. In *Proceedings of the 2nd International Workshop on Motion in Games*, 75–83.

MCDONNELL, R., LARKIN, M., DOBBYN, S., COLLINS, S., AND O'SULLIVAN, C. 2008. Clone attack! perception of crowd variety. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 26.

MCLACHIAN, G., AND KRISHNAN, T. 1996. *The EM Algorithm and Extensions*. John Wiley and Sons.

MEHRAN, R., OYAMA, A., AND SHAH, M. 2009. Abnormal crowd behavior detection using social force model. In *Proc. of Computer Vision and Pattern Recognition*, 935–942.

MOUSSAÏD, M., HELBING, D., AND THERAULAZ, G. 2011. How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences 108*, 17, 6884.

NARAIN, R., GOLAS, A., CURTIS, S., AND LIN, M. C. 2009. Aggregate dynamics for dense crowd simulation. *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH Asia) 28*, 5, 122.

ONDREJ, J., PETTRE, J., OLIVIER, A., AND DONIKAN, S. 2010. A synthetic-vision based steering approach for crowd simulation. *ACM Trans. on Graphics 29*, 4, 123:1–123:9.

PATIL, S., VAN DEN BERG, J., CURTIS, S., LIN, M. C., AND MANOCHA, D. 2011. Directing crowd simulations using navigation fields. *IEEE Trans. on Vis. and Comp. Graphics 17*, 2, 244–254.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2007. Controlling individual agents in high-density crowd simulation. *Proc. of Symposium on Computer Animation*, 99–108.

PELECHANO, N., STOCKER, C., ALLBECK, J., AND BADLER, N. 2008. Being a part of the crowd: towards validating vr crowds using presence. In *Proc. of 7th Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 136–142.

PELLEGRINI, S., ESS, A., SCHINDLER, K., AND VAN EOOL, L. 2009. You'll never walk alone: Modeling social behavior for multi-target tracking. In *Proc. of Int. Conf. on Computer Vision*, 261–268.

PETTRE, J., ONDREJ, J., OLIVIER, A., CRETUAL, A., AND DONIKIAN, S. 2009. Experiment-based modeling, simulation and validation of interactions between virtual walkers. In *Proc. of Symposium on Computer Animation*, ACM, 189–198.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. *Proc. of ACM SIGGRAPH 21*, 25–34.

REYNOLDS, C. W. 1999. Steering behaviors for autonomous characters. *Game Developers Conference*.

RODRIGUEZ, M., ALI, S., AND KANADE, T. 2009. Tracking in unstructured crowded scenes. In *Computer Vision, 2009 IEEE 12th International Conference on*, 1389–1396.

SCHADSCHNEIDER, A., CHOWDHURY, D., AND NISHINARI, K. 2011. *Stochastic Transport in Complex Systems: From Molecules to Vehicles*. Elsevier.

SEYFRIED, A., BOLTES, M., KÄHLER, J., KLINGSCH, W., PORTZ, A., RUPPRECHT, T., SCHADSCHNEIDER, A., STEFFEN, B., AND WINKENS, A. 2010. Enhanced empirical data for the fundamental diagram and the flow through bottlenecks. *Pedestrian and Evacuation Dynamics 2008*, 145–156.

SINGH, S., KAPADIA, M., REINMANN, G., AND FALOUTSOS, P. 2009. Steerbench: A benchmark suite for evaluating steering behaviors. *Computer Animation and Virtual Worlds 20*, 533–548.

SUNG, M., GLEICHER, M., AND CHENNEY, S. 2004. Scalable behaviors for crowd simulation. *Computer Graphics Forum 23*, 3 (Sept), 519–528.

TREUILLE, A., COOPER, S., AND POPOVIC, Z. 2006. Continuum crowds. *Proc. of ACM SIGGRAPH*, 1160 – 1168.

VAN DEN BERG, J., GUY, S. J., LIN, M. C., AND MANOCHA, D. 2009. Reciprocal n-body collision avoidance. *Proc. of International Symposium on Robotics Research (ISRR)*, 3–19.

YU, Q., AND TERZOPOULOS, D. 2007. A decision network framework for the behavioral animation of virtual humans. In *Proc. of Symposium on Computer animation*, 119–128.

Appendices

A Case Study: Optimizing RVO

The Entropy metric seeks to capture how well a simulator captures the aggregate dynamics of the motion observed in real-world data. For simple scenarios it maybe possible to perform other, simple data analysis, such as measuring average speed or minimum distance between the agents to reliably indicate a failure to capture the recorded motion. Examples include measuring average speed or minimum distance between the agents. This type of analysis is commonly used to improve the simulator and lower the entropy score.

As an example, we chose the Lab scenario of two people passing each other. First we evaluated the Entropy metric with an intentionally poor set of parameters, labeled RVO-A, which resulted in a large Entropy score of 6.07. Next, we improve the simulator by measuring the closest distance between the two participants and using that parameter to set the radius (RVO-B). The simulator's similarity can be further improved by setting the preferred agent speed to be the average speed of the two participants (RVO-C). Finally, we increase the time horizon over which agents plan their collision avoidance to improve the anticipation in motion (RVO-D). As shown in Table 5, each change in the parameters lowers the Entropy score.

The simulations generated from each step of changing the parameters can be seen in the accompanying video. We note that because of the logarithmic scale of the Entropy metric, a reduction in score from 6.07 to 0.75 indicates more than an order of magnitude in reduction of error. The correlation between hand-tuned results and the Entropy metric provide an indication of the metric's ability to capture meaningful errors in terms of aggregate dynamics.

RVO	Metric	Rad.	Pref Speed	Time Horiz	Improvement Method
A	6.07	1.0m	0.9 m/sec	0.1 sec	-
B	2.09	.25m	0.9 m/sec	0.1 sec	Match radius from data
C	1.36	.25m	1.2 m/sec	0.1 sec	Match speed from data
D	0.75	.25m	1.2 m/sec	1.0 sec	Increase planning horizon

Table 5: *As we improve the similarity of the RVO benchmark based on the data from Lab benchmark, the entropy metric decreases. RVO-A is the worst algorithm for this benchmark and RVO-D is the best algorithm (as shown in video). We see a direct correlation between the entropy metric and similarity of the simulation algorithm.*

This result also indicate that the Entropy metric can be used to design data-driven crowd simulation algorithms. Given a motion data from a target set of behaviors, changing the simulator parameters to optimize the Entropy metric will result in a simulator that closely matches the desired behavior.

B Implementation Details

For the results presented in Section 4 we used the same crowd state vector format ($\mathcal{X}_k$), the same observed validation data ($\mathbf{z}_0, \ldots, \mathbf{z}_t$), and the same observation function (h) for all simulators tested. We varied only the simulator ($\hat{f}$). We briefly give details of each below.

State Vector ($\mathcal{X}_k$) The state of each agent was defined as a four dimensional vector of 2D position and 2D velocity.

Validation Data ($\mathbf{z}_0, \ldots, \mathbf{z}_t$) The validation data used was a series of 2D positions for each person being tracked, the data was generally between 10-15Hz.

Observation Function (h) The observation function must be defined to convert an instance of the state vector to the format found in the validation data. In our case, the function h simply returns the position component of the state vector $h((pos.x, pos.y, vel.x, vel.y)) = (pos.x, pos.y)$.

Simulation Function ($\hat{f}$) The details of this function varies between different simulators and each are discussed in more detail below. All simulators share some common features: all agents were restricted to a maximum velocity of 2.5m/s and all simulators considered only the 10 closest neighbors in velocity computations.

B.1 Social Force Simulation Details

Our implementation of a social force model (SFM) approach to simulation is based on the approach detailed by Helbing et al. [2000]. Each agent experiences an avoidance force from all neighboring agents with decreases exponential with distance from the agent, along with frictional and pushing forces that effect agents in contact with each other. All of the *SFM* simulators shared the same force balancing constants: a social scaling force of 2000N, an agent reaction time of 0.5s, a repulsive pushing spring constant of $120,000\ kg/s^2$, and a sliding friction constant of $240,000\ kg/s^2$.

Simulator	Pref Speed (m/s)	Radius (m)	Mass (kg)
SFM-1	1.40	0.16	80
SFM-2	1.10	0.31	80
SFM-3	1.20	0.20	80

Table 6: *Parameters used in various social force simulators.*

B.2 RVO Simulation Details

Our implementation of a predictive planning approach to simulation is based on the Reciprocal Velocity Obstacle (RVO) based approach detailed by Van den Berg et al. [2009]. Each agent chooses a velocity towards its goal which will minimize it's expected collision with all neighboring agents. The *RVO* simulators vary in the radius and preferred speed of the simulated agents as well as the time horizon agents plan over and the maximum distance at which a neighbor can effect planning. The various simulators are detailed below:

Simulator	Pref Speed (m/s)	Radius (m)	Tim Horiz. (s)	Neighbor Dist. (m)
RVO-1	1.44	0.20	1.7	41
RVO-2	1.10	0.20	1000	41
RVO-3	1.20	0.25	1000	200

Table 7: *Parameters used in various RVO simulators.*

B.3 Steering Simulation Details

Our implementation of a steering approach to simulation is based on the approach in OpenSteer by Reynolds [1999]. The final action of each agent is decided by a combination of a force towards the goal, and a force away from the nearest obstacle and pedestrian. All of the *Steer* simulators shared the same blending factor between goal-directed and avoidance forces, they differed in the preferred speed and radius of the agents.

Simulator	Pref Speed (m/s)	Radius (m)
Steer-1	1.44	0.20
Steer-2	1.00	0.30
Steer-3	0.85	0.20

Table 8: *Parameters used in various steering simulatiors.*

Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2011)
A. Bargteil and M. van de Panne (Editors)

Scenario Space: Characterizing Coverage, Quality, and Failure of Steering Algorithms

Mubbasir Kapadia[1,2], Matt Wang[1], Shawn Singh[1,3], Glenn Reinman[1], Petros Faloutsos[1]

[1]University of California Los Angeles
[2]University of Pennsylvania
[3]Google Inc.

Abstract

Navigation and steering in complex dynamically changing environments is a challenging research problem, and a fundamental aspect of immersive virtual worlds. While there exist a wide variety of approaches for navigation and steering, there is no definitive solution for evaluating and analyzing steering algorithms. Evaluating a steering algorithm involves two major challenges: (a) characterizing and generating the space of possible scenarios that the algorithm must solve, and (b) defining evaluation criteria (metrics) and applying them to the solution. In this paper, we address both of these challenges. First, we characterize and analyze the complete space of steering scenarios that an agent may encounter in dynamic situations. Then, we propose the representative scenario space and a sampling method that can generate subsets of the representative space with good statistical properties. We also propose a new set of metrics and a statistically robust approach to determining the coverage and the quality of a steering algorithm in this space. We demonstrate the effectiveness of our approach on three state of the art techniques. Our results show that these methods can only solve 60% *of the scenarios in the representative scenario space.*

Categories and Subject Descriptors (according to ACM CCS): I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Multiagent Systems I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.6.6 [Simulation and Modeling]: —Simulation Output Analysis

1. Introduction

Immersive virtual worlds have quickly come to the forefront in both industry and academia with their applicability being realized in a wide variety of areas from education, collaboration, urban design, and entertainment. A key aspect of immersion in virtual environments is the use of autonomous agents to inject life into these worlds. Autonomous agents require efficient, robust algorithms for navigation and steering in large, complex environments where the space of all possible situations an agent is likely to encounter is intractable. The rich set of scenarios and corresponding steering choices have resulted in a large variety of techniques that are focused on tackling a subset of this problem. To our knowledge, there exists no definitive measure of the ability of a steering algorithm to successfully handle the space of all possible scenarios that it is likely to encounter in complex environments. This greatly limits future researchers and end-users in objectively evaluating and analyzing the current state of the art before choosing their own direction of exploration.

There are two key requirements to doing a comprehensive evaluation of a steering technique. First, we must be able to sufficiently sample the representative set of challenging situations that an agent is likely to encounter. Next, we need a measure of scoring success for an algorithm for a particular scenario that has meaning on its own as well as in comparison with the scores for other approaches.

Previous approaches have addressed these issues with

SCA 2011, Vancouver, British Columbia, Canada, August 5 – 7, 2011.
© 2011 ACM 978-1-4503-0923-3/11/0008 $10.00

small sets of manually designed test cases, and ad hoc, scenario-dependent criteria. In this paper, we address both of these challenges with rigorous, statistically-based approaches.

We examine the complete space of possible scenarios that a steering algorithm may need to solve given a set of user defined parameters, such as the size of the agents. After showing that an exhaustive sampling of this space is not practical, we propose the *representative scenario* and an associated sampling method. Both the representative scenario space and the sampling method are constrained to produce test sets that favor complexity, and avoid easy to solve cases. To evaluate a steering algorithm on a single scenario we propose a set of metrics that can be normalized with respect to ideal values so as to become scenario independent. Based on these metrics, we then propose the concepts of *coverage, average quality and failure set* and show how they can be computed over the representative scenario space. Computing these concepts over an entire scenario space provides a rigorous, statistical view of an algorithm, and can be used to evaluate a single approach or compare different approaches. In our opinion, our work is the first attempt to evaluate steering techniques in an automated and statistically sound fashion.

This paper makes the following contributions:

- We propose three concepts to statistically evaluate steering algorithms over a scenario space: coverage, average quality and failure set.
- We define the space of all possible scenarios that an agent could encounter while steering and navigating in dynamically changing environments. In addition, we present a method of sufficiently sampling the representative scenarios in this space in order to effectively compute average quality and coverage for a particular steering algorithm.
- We provide a method of automatically determining a *failure set* for an algorithm – a subset of scenarios where the algorithm performs poorly based on some criteria. This provides an invaluable tool for users and AI developers in evaluating their own steering techniques.
- We demonstrate the effectiveness of our framework in analyzing four agent-based techniques: three state of the art [KSHF09, SKHF11, vdBLM08], and one simple baseline algorithm that only reacts to the most immediate threat.

2. Related Work

There are three broad categories when it comes to the analysis and evaluation of crowd simulations: (1) comparing simulations to real world data, (2) performing user-studies to determine if the desired qualities of the simulation have been met and to manually detect the presence of anomalous behaviors, and (3) using statistical tools to analyze simulations. The real world and its real human characters are extremely complex, which makes it very difficult to compare a simulation to real events. Manual inspection of simulations is prone to human error and personal inclinations. Surveys [LS02, McF06] show that automated evaluation, especially for autonomous characters, is yet to be fully realized in the games industry. Hence, the focus of this work is in the use of computational methods and statistical tools to analyze, evaluate, and test crowd simulations.

Section 2.1 reviews the traditional methods adopted in sampling the space of scenarios. Section 2.2 describes the metrics used for evaluation. Section 2.3 reviews some of the popular techniques used for steering. Section 2.4 describes our method in relation to prior work.

2.1. Benchmarks for Evaluation

Steering approaches, outlined in Section 2.3, are generally targeted at specific subsets of human steering behaviors and use their own custom test cases for evaluation and demonstration. The work in [SKN*09] proposes a standard suite of test cases that represent a large variety of steering behaviors and is independent of the algorithm used. In addition, [SKFR09] provides a suite of tools and helper functions to allow AI developers to quickly get started with their own algorithms. However, even the 42 test cases described here still cannot capture the large space of possible situations an agent will encounter in dynamic environments of realistic complexity.

2.2. Metrics for Evaluation

Prior work has proposed a rich set of application-specific metrics to evaluate and analyze crowd simulations. The work of [PSAB08] uses *presence* as a metric for crowd evaluation. Number of collisions and effort are often used as metrics to minimize when developing steering algorithms [ST05, GCC*10]. The work in [HFV00] uses "rate of people exiting a room" to analyze evacuation simulations. [LCSCO10] presents a data-driven approach for evaluating the behaviors of individuals within a simulated crowd. [RP07] describes a set of task-based metrics to evaluate the capability of a motion graph across a range of tasks and environments. The work in [SKN*09, KSA*09] proposes a rich set of derived metrics that provide an empirical measure of the performance of an algorithm. However, the values of these metrics (e.g. path length, total kinetic energy, total change in acceleration, etc.) are tightly coupled with the the length and complexity of a scenario, which prevents users from interpreting these metrics in a scenario-independent fashion.

2.3. Steering Approaches

Since the seminal work of [Rey87, Rey99], there has been a growing interest in pedestrian simulation with a wide array of techniques being tested and implemented. A comprehensive overview of the related work in steering and navigation techniques can be found here [PAB08].

Centralized techniques [MRHA98, Lov94, Hen71] focus on the system as a whole, modeling the characteristics of the flow rather than individual pedestrians. Centralized approaches usually model a broader view of crowd behaviors as flows rather than focusing on individual specialized agent behaviors.

De-centralized approaches model the agent as an independent entity that performs collision avoidance with static obstacles, reacts to dynamic threats in the environment, and steers its way to its target. Particle based approaches [Rey87, Rey99] model agents as particles and simulate crowds using basic particle dynamics. The social force model [HBJW05, BMOB03, BH97] solves Newton's equations of motion to simulate forces such as repulsion, attraction, friction and dissipation for each agent to simulate pedestrians. Rule-based approaches [LD04, LMM03, Rey99, RMH05, PAB07, SGA*07, vdBPS*08] use various conditions and heuristics to identify the exact situation of an agent. Data-driven methods use existing video data or motion capture to derive steering choices that are then used in virtual worlds (e.g., [LCHL07, LCL07]). The works of [Feu00, PPD07] use predictions in the space-time domain to perform steering in environments populated with dynamic threats. Predicting potential threats ahead of time results in more realistic steering behaviors.

We use three state of the art steering techniques to serve as the basis for the analysis results shown in this paper. In addition, we also evaluate a purely reactive approach to steering to demonstrate the efficacy of our framework over a variety of steering approaches.

- **Egocentric.** The work in [KSHF09] proposes the use of egocentric affordance fields to model local variable-resolution perception of agents in dynamic virtual environments. This method combines steering and local space-time planning to produce realistic steering behaviors in challenging local interactions as well as large scale scenarios involving thousands of agents.
- **PPR.** The work in [SKHF11] presents a hybrid framework that combines reaction, prediction and planning into one single framework.
- **RVO.** The work in [vdBLM08] proposes the use of reciprocal velocity obstacles to serve as a linear model of prediction for collision avoidance in crowds.
- **Reactive.** This steering technique employs the use of a simple finite state machine of rules to govern the behavior of an autonomous agent in a crowd. This technique is purely reactive in nature and does not employ the use of any form of predictive collision avoidance. A description of the implementation of this technique can be found in [SKHF11].

2.4. Comparison to Related Work

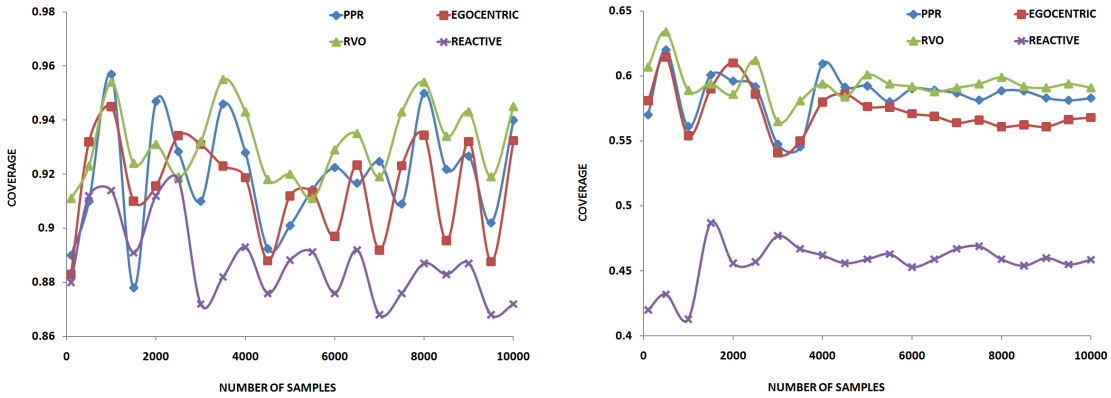
Our work leverages was inspired by SteerBench [SKN*09] and [RP07]. The work in [RP07] presented a method of calculating coverage of motion graphs for a set of animation and navigation benchmarks. SteerBench proposed an objective set of test cases and an ad hoc, automatic method of scoring the performance of steering algorithms. The approximately 42 test cases provide a fixed and very sparse sampling of the scenario space. In this paper, we take a large step along this direction. First, we characterize the entire scenario space, and propose a sampling based approach to estimate, for the first time, the coverage of a steering algorithm. We also propose a new set of performance metrics and a robust statistical method for automatically analyzing the effectiveness of steering algorithms.

3. Scenario Space

Like real people, virtual agents make their steering decisions by considering their surrounding environment and their goals. The environment usually consists of static obstacles and other agents. In this section we describe how we represent all the elements of a steering problem, which we refer to as a *scenario*.

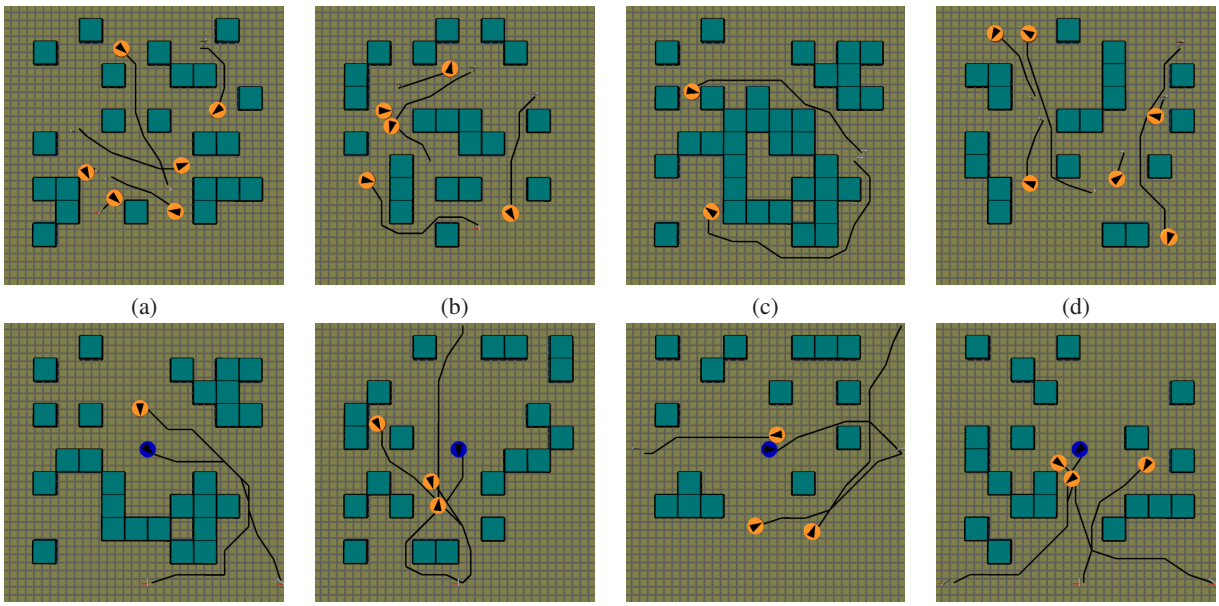
We define a *scenario* as one possible configuration of obstacles and agents in the environment. The configuration of an obstacle is its position in the environment along with the information of its bounding box (we assume rectangular obstacles). The configuration of an agent includes its initial position, target location, and desired speed. The configuration of agents and obstacles can be extended or modified to meet the needs of any application. The *scenario space* is defined as the space of all possible scenarios that an agent can encounter while steering in dynamic environments. The ratio of the subspace of scenarios that a steering algorithm can successfully handle is defined as the *coverage* of the algorithm. An ideal steering algorithm would be able to successfully handle all the scenarios in this extremely high dimensional space, thus having a coverage of 1. In order to be able to determine the coverage of a steering algorithm, we need the ability to sample the scenario space in a representative fashion and to objectively determine the performance of an algorithm for a particular scenario.

Section 3.1 describes a set of user-defined parameters used to define a space of scenarios. In Section 3.2, we describe the results of our experiment to determine coverage of three steering algorithms in the complete space of scenarios. We observe that the value of coverage for each of these algorithms does not converge for even up to 10,000 sample points. Section 3.3 describes a set of constraints that are imposed on the complete scenario space to define the space of representative scenarios. We observe rapid convergence of coverage of steering algorithms in the representative scenario space.



(a) (b)

Figure 1: *The success rate of the four algorithms in the complete (a) and representative (b) scenario space vs the number of samples (size) of the test set.*



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 2: *Figures (a)-(d) Scenarios randomly generated in the complete scenario space. A black line indicate an agent's optimal path to the goal. Figures (e)-(h) Scenarios randomly generated in the representative scenario space. Our sampling process ensures that all agents interact with the reference agent (in blue) which is always placed in the center of the environment.*

3.1. Parametrization of Scenario Space

The space of all scenarios is determined by the number of obstacles and agents, the size of the environment, and the size of obstacles. A user may wish to test his steering algorithm on local interactions between agents in small environments with 2 or 3 agents. Alternatively, a user may wish to stress test his or her algorithm on large environments with a large distribution of agents and obstacles. We expose these parameters to the user to allow him to define the space of scenarios to meet the need of his application.

The set of parameters, P is defined as follows:

- **Environment size**. The size of the environment is defined as the radial distance, *r*, from the egocentric agent that is positioned at the center of the environment.
- **Obstacle Discretization**. Obstacles are represented by a grid of rectangular blocks that are either on or off. The

size of these blocks is determined by two parameters: resolution in X d_x and resolution in Y d_y. These values specify how many cells exist within the width and height of the environment as determined by the radial distance r defined above.
- **Number of agents**. The number of agents in a scenario is governed by two user-defined parameters: the minimum and maximum number of agents (n_{min}, n_{max}).
- **Target speed of agents**. Some steering algorithms can specify a target speed for an agent. The range of possible values is determined by a minimum and maximum speed parameter (s_{min}, s_{max}).

Given a specific set of parameter values P that define a space of scenarios, we can procedurally or randomly sample scenarios with initial configurations of obstacles and agents that lie in that scenario space.

3.2. The Complete Scenario Space

The complete scenario space, $\mathbb{S}(\mathrm{P})$ represents all the possible scenarios that can be generated for a particular set of user-defined parameters P. In order to prevent sampling of *invalid* scenarios that have no solution, we place certain validity constraints on the scenario space.

- **Collision-Free**. The initial configurations of obstacles and agents must not be in a state of collision.
- **Solvable**. There must exist a valid path taking an agent from its initial position to its target location.

The space $\mathbb{S}(\mathrm{P})$ is infinite and cannot be sampled exhaustively. Instead, we aim to find a representative set of samples that describes this space sufficiently. To determine whether we can generate such a set, we first perform a random sampling experiment in $\mathbb{S}(\mathrm{P})$ where $\mathrm{P} = \{r = 7, d_x = d_y = 10, n_{min} = 3, n_{max} = 6, s_{min} = 1, s_{max} = 2.7\}$.

A scenario is randomly generated as follows: First, we generate the obstacles by randomly turning on or off cells in our obstacle grid. Next, we select a number of agents to simulate by randomly sampling the range defined above. For each agent, we choose a random obstacle-free position and orientation. We also choose a random obstacle-free position for each agent's goal. All positions are chosen within the radius r and all orientations are sampled uniformly within $[0, 2\pi)$.

The performance of an algorithm for a scenario is evaluated as a boolean measure of whether or not it could complete the scenario. A scenario is said to be successfully completed if all agents reach the goal within a time threshold without any collisions. The coverage of an algorithm is the ratio of all scenarios that it could successfully complete.

In this experiment we iteratively increase the number of sample points from $N = 100$ to $10,000$. The results are illustrated in Figure 1(a). We observe that the coverage of an algorithm fluctuates between 0.9 and 0.95 and does not converge within reasonable bounds. Also, the minimum coverage of the three reference algorithms is quite high (> 0.9). Similarly, even the baseline reactive algorithm seems to perform well with a coverage of approximately 0.89. These observations suggest that the experiments contain many trivial or easy scenarios that greatly skew the computed measure of coverage, and affect its convergence. To get a better picture of the areas in the scenario space that algorithms may have trouble succeeding, we propose the *Representative Scenario Space*, and an egocentric evaluation method, which are described below.

Algorithm	$\mathbb{S}(\mathrm{P})$	$\mathbb{R}(\mathrm{P})$	SteerBench
PPR	0.919	0.583	0.86(36/42)
Egocentric	0.915	0.568	0.86(36/42)
RVO	0.931	0.591	0.86(36/42)
Reactive	0.887	0.459	0.83(35/42)

Figure 3: *The estimated coverage of the steering algorithms in the complete space* $\mathbb{S}(\mathrm{P})$*, the representative space* $\mathbb{R}(\mathrm{P})$*, and the 42 cases of SteerBench [SKN*09].*

3.3. The Representative Scenario Space

We eliminate trivial scenarios by applying the following constraints on the complete scenario space and the associated sampling method:

- **Reference Agent**. The first agent is always placed at the origin of the environment and is known as the reference agent. The scenario is evaluated with respect to the reference agent.
- **Goals and Orientations**. The goal of an agent is restricted to one of 8 choices that are located at the boundary of the scenario. The agent's initial orientation is always pointing towards the agent's goal.
- **Agent Spatial Positions**. Instead of uniformly sampling the space for agent positions, we model the probability of a location in the environment $\vec{x}$ being sampled using a normal distribution $\mathbb{N}(\vec{x}, \vec{\mu} = \vec{O}, \sigma^2 = 0.4)$. This implies that agents are more likely to be placed closer to the origin, i.e. closer to the reference agent, which increases the likelihood of interaction between agents.
- **Agent Interactions**. We place a constraint on the configuration of an agent placed in the scenario to ensure that it interacts with the reference agent. We compute an optimal path (using A*) for the agent from its start position to its goal. If the planned path of the agent intersects with the planned path of the reference agent in space and time (we assume constant speed of motion along the optimal path) then the agent is considered relevant and is placed in the scenario.
- **Agent Speeds**. Instead of varying the desired speed of agents, we keep it a constant (1.7 m/s) as we observe that desired speed variations do not have a large impact on the resulting behavior of most steering approaches.

The resulting space of scenarios that meet these constraints is the representative scenario space, denoted by $\mathbb{R}(\mathrm{P})$.

We change our evaluation method of a scenario to be with respect to the reference agent alone. Hence, an algorithm is successful on a scenario if the reference agent reaches its goal and there are no collisions with other agents.

We run the same sampling experiment described above in the representative scenario space (Figure 1(b)). We observe convergence of coverage between $N = 5,000$ to $10,000$. We also observe that the coverage of the algorithms is much lower. The three reference algorithms can only complete approximately half of the scenarios sampled. We also see a much larger difference in the coverage of the baseline reactive algorithm in comparison to the three reference algorithms, as one would expect. Figure 3 compares the coverage of algorithms in $\mathbb{S}(\mathrm{P})$, $\mathbb{R}(\mathrm{P})$, and using the test cases provided in SteerBench [SKN*09]. The algorithms have very high coverage in both $\mathbb{S}(\mathrm{P})$ and SteerBench. The reactive algorithm fails in only one more scenario than the other three reference steering techniques in the 42 test cases that SteerBench provides. In contrast, the scenarios generated are much more challenging in $\mathbb{R}(\mathrm{P})$ which is reflected in low coverage values and a much larger difference between the baseline reactive technique and the three more sophisticated ones.

In conclusion, we can make two important observations. First, the representative space sampled with our constrained sampling technique can produce test sets that expose the difficulties of steering algorithms. Second, approximately 10,000 samples seem to be enough for analyzing an algorithm, as indicated by the convergence of the coverage of the four algorithms.

4. Evaluation Criteria

We evaluate a scenario by computing 3 primary metrics that quantify the success of the egocentric agent in completing the scenario. These metrics characterize whether or not the egocentric agent successfully reached its goal, the total time it took to reach its goal, and the total distance traveled in reaching the goal. By defining the metrics as a ratio to its optimal value, we can compare and evaluate these metrics on an absolute scale.

- **Scenario Completion**. For an algorithm a and a scenario s, if the reference agent reaches its goal within the time limit without colliding with any agents or obstacles, the scenario is said to have successfully completed. In this case, $m_{\mathrm{c}}(s,a) = 1$ else $m_{\mathrm{c}}(s,a) = 0$.
- **Path Length**. The path length $m_{\mathrm{l}}(s,a)$ is the total distance traveled by the egocentric agent to reach its goal.
- **Total Time**. The total time $m_{\mathrm{t}}(s,a)$ is the time taken by the egocentric agent in reaching the goal.

In addition, we compute optimal values of path length and total time to serve as an absolute reference that can be used to normalize the values of $m_{\mathrm{l}}(s,a)$ and $m_{\mathrm{t}}(s,a)$. The optimal path length, $m_{\mathrm{l}}^{\mathrm{opt}}(s,a)$, and optimal time, $m_{\mathrm{t}}^{\mathrm{opt}}(s,a)$, are the path length and time taken to travel along an optimal path to the goal by an algorithm a for a particular scenario s, ignoring neighboring agents. Using the optimal values, we can compute the ratio for a particular metric $m(s,a)$ as follows:

$$m^{\mathrm{r}}(s,a) = \frac{m^{\mathrm{opt}}(s,a)}{m(s,a)} \times m_{\mathrm{c}}(s,a). \tag{1}$$

The value of $m^{\mathrm{r}}(s,a)$ is equal to 1 when the value of the metric is equal to its optimal value and is close to 0 when the value is far away from its optimal value. Also, $m^{\mathrm{r}}(s,a)$ is only computed when the scenario has successfully completed. Using Equation 1, we can compute $m_{\mathrm{l}}^{\mathrm{r}}(s,a)$ and $m_{\mathrm{t}}^{\mathrm{r}}(s,a)$ to effectively quantify the performance of a steering algorithm for a particular scenario which can be compared across algorithms and scenarios.

5. Coverage, Average Quality and Failure Set

In this section, we show how we use our representative scenario space and evaluation criteria to derive a set of well-defined, statistical metrics that characterize key aspects of a steering algorithm.

Scenario Set. The scenario set $S_{\mathrm{m}}^{\mathrm{a}}(T_1,T_1)$ for an algorithm a on a metric m is defined as the subset of all scenarios within the representative space of scenarios for which the value of $m(s,a)$ is in the range $[T_1,T_2)$.

$$S_{\mathrm{m}}^{\mathrm{a}}(T_1,T_2) = \{s | s \in \mathbb{R}(\mathrm{P}) \wedge T_1 <= m(s,a) < T_2\}. \tag{2}$$

Using only T_1 we can find the success set of an algorithm as the set of the scenarios for which the metric was greater than a threshold. Similarly, using only T_2 allows us to define a failure set of an algorithm.

The common failure set $S_{\mathrm{m}}(0,T_{min})$ for all algorithms $a \in A$ is the intersection of the failure sets $S_{\mathrm{m}}^{\mathrm{a}}(0,T_{min})$ of all evaluated steering algorithms:

$$S_{\mathrm{m}}(0,T_{min}) = \bigcap_{a \in A} S_{\mathrm{m}}^{\mathrm{a}}(0,T_{min}). \tag{3}$$

The common failure set can be used to identify particularly difficult scenarios.

Coverage. The coverage $c_{\mathrm{m}}^{\mathrm{a}}$ of a steering algorithm a can be computed as the ratio of the subset of scenarios in the scenario space that a steering algorithm can successively handle with respect to a particular metric, $m(s,a)$.

$$c_{\mathrm{m}}^{\mathrm{a}} = \frac{|S_{\mathrm{m}}^{\mathrm{a}}(T_{max},1)|}{|\mathbb{R}(\mathrm{P})|}, \tag{4}$$

where $|S|$ denotes the cardinality of the set S.

Average Quality. The average quality of a steering algorithm for a particular method of evaluation can similarly be

computed as the average value of $m(s,a)$ for all sampled scenarios.

$$q_{\mathrm{m}}^{\mathrm{a}} = \frac{\sum_{s \in S_{\mathrm{m}}^{\mathrm{a}}(T_{max},1)} m(s,a)}{|\mathbb{R}(\mathrm{P})|}. \quad (5)$$

Using Equations 4 and 5, we can compute coverage and average quality for $m_{\mathrm{s}}(s,a)$, $m_{\mathrm{l}}^{\mathrm{r}}(s,a)$ and $m_{\mathrm{t}}^{\mathrm{r}}(s,a)$. Note that the coverage and average quality for $m_{\mathrm{s}}(s,a)$ will be the same since it is a boolean value.

The three concepts defined in this section provide a rigorous and objective statistical view of a steering algorithm. They can be intuitively used to evaluate the effectiveness of a single algorithm or to compare different approaches.

6. Results

Using the concepts and evaluation method proposed in previous sections, we can now analyze and compare our four steering algorithms. All algorithms are tested on the same set of 10,000 scenarios randomly selected from the representative scenario space, $\mathbb{R}(\mathrm{P})$, with user defined parameters $\mathrm{P} = \{r = 7, d_x = d_y = 10, n_{min} = 3, n_{max} = 6, s = 2.7\}$. In Section 3.3 we showed that the success rate of the four algorithms converges for test sets with 5,000-10,000 samples in the representative scenario space. This is a good indication that a test set of size 10,000 should be sufficient for our analysis. It takes our system a few minutes to run 10,000 samples (depending on the performance of the steering algorithm).

6.1. Coverage and Average Quality

The coverage and average quality for each algorithm for all three metrics are given in Table 5 and Table 6. Note that the values of $m_{\mathrm{l}}^{\mathrm{r}}(s,a)$ and $m_{\mathrm{t}}^{\mathrm{r}}(s,a)$ are only considered when the algorithm successfully completes the scenario, i.e. $m_{\mathrm{c}}(s,a) = 1$. To compute coverage for $m_{\mathrm{l}}^{\mathrm{r}}(s,a)$ and $m_{\mathrm{t}}^{\mathrm{r}}(s,a)$, we specify the thresholds equal to the mean of the average quality for each metric computed for the three algorithms (`Reactive` is not considered). Thus, the coverage gives us a measure of the ratio of the number of scenarios that are above the average quality measure for that metric.

Algorithm	$m_{\mathrm{l}}^{\mathrm{r}}(s,a)$	$m_{\mathrm{t}}^{\mathrm{r}}(s,a)$
`PPR`	0.789	0.683
`Egocentric`	0.723	0.63
`RVO`	0.743	0.731
`Reactive`	0.617	0.586

Figure 5: *The average quality $q_{\mathrm{m}}^{\mathrm{a}}$ of the steering algorithms for ratio to optimal path length, $m_{\mathrm{l}}^{\mathrm{r}}(s,a)$, and ratio to optimal total time, $m_{\mathrm{t}}^{\mathrm{r}}(s,a)$.*

Observations. We observe that the average quality of the algorithms for path length, $m_{\mathrm{l}}^{\mathrm{r}}(s,a)$, is approximately 0.75.

Algorithm	$m_{\mathrm{s}}(s,a)$	$m_{\mathrm{l}}^{\mathrm{r}}(s,a)$	$m_{\mathrm{t}}^{\mathrm{r}}(s,a)$
`PPR`	0.583	0.748	0.608
`Egocentric`	0.568	0.681	0.515
`RVO`	0.591	0.762	0.662
`Reactive`	0.459	0.212	0.178

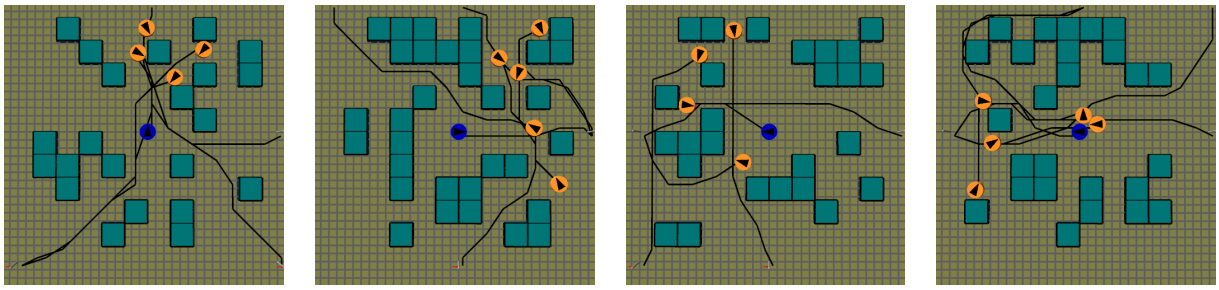
Figure 6: *The coverage $c_{\mathrm{m}}^{\mathrm{a}}$ of the steering algorithms for the three metrics.*

This implies that the three algorithms generally produce solutions with path lengths that are 75% of the optimal values. In contrast, the average quality of algorithms for total time, $m_{\mathrm{t}}^{\mathrm{r}}(s,a)$, is approximately 0.68 which is considerably lower. This is because steering algorithms generally prefer to slow down instead of deviating from their planned paths. When comparing `PPR` and `RVO`, we notice that `PPR` has a better quality measure for path length than time. This is because `PPR` has a greater proclivity for predictively avoiding dynamic threats by slowing down if it anticipates a collision. Due to the variable resolution nature of the perception fields modeled in `Egocentric`, the trajectories produced by this method are curved and produce less optimal results. The performance of `Reactive` is reflected in its measure of coverage. We observe that `Reactive` can only solve 45% of the scenarios (compared to nearly 60% for the other 3 algorithms), and that only 20% of its solutions are above the average quality measure.

6.2. Failure Set

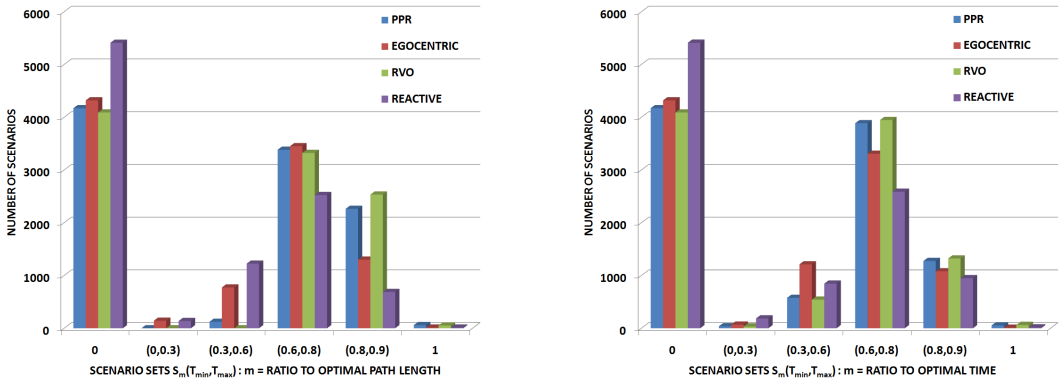
The coverage and average quality provide a good aggregate measure of the performance of an algorithm over a large sample of scenarios and serve as a good basis of comparison. However, it is particularly useful to be able to automatically generate *scenarios of interest* where an algorithm performs poorly. Our framework automatically computes a failure set for an algorithm as the set of all scenarios where a particular metric falls below a threshold. Figure 7(a) and (b) measures the number of scenarios for which $m_{\mathrm{l}}^{\mathrm{r}}(s,a)$ and $m_{\mathrm{t}}^{\mathrm{r}}(s,a)$ fall within a specified region. The set $S_{\mathrm{m}}^{\mathrm{a}}(0,0)$ clusters all scenarios for which the algorithm has failed to find a solution ($m_{\mathrm{s}}(s,a) = 0$). The set $S_{\mathrm{m}}^{\mathrm{a}}(1,0)$ measures the number of scenarios for which the algorithms produced optimal solutions for $m_{\mathrm{l}}^{\mathrm{r}}(s,a)$ or $m_{\mathrm{t}}^{\mathrm{r}}(s,a)$. A small number of samples in this cluster is indicative that scenarios produced in the representative space are challenging and require complex interactions between agents. The sets $S_{\mathrm{m}}^{\mathrm{a}}(0,0.3)$ and $S_{\mathrm{m}}^{\mathrm{a}}(0.3,0.6)$ represent scenarios for which a steering algorithm generated highly sub-optimal solutions.

We also find the common failure set $S_{\mathrm{m}}(0,0)$ of all four steering algorithms. This set represents the set of scenarios for which no steering algorithm could find a solution. In these cases, the agents either reach a deadlock situation and time out or reach their goals by colliding with other agents. Figure 4 highlights some particularly challenging scenarios



(a) (b) (c) (d)

Figure 4: *Challenging scenarios sampled in the representative space that resulted in collisions or no solution.*



(a) (b)

Figure 7: *Failure sets of each algorithm for ratio to optimal path length $m_l^r(s,a)$ and ratio to optimal time $m_t^r(s,a)$.*

that fall within the common failure set. Note, that the narrow passageways in the figure are traversable. For 10,000 sample points, the cardinality of the failure set is $|S_m(0,0)| = 1,710$. This means that 17% of the scenarios that were sampled could not be successfully handled by any steering approach. By visually inspecting these scenarios, we arrive at the following generalization for particularly challenging scenarios:

- **Series of sharp turns** Narrow passageways where agents had to make a sequence of sharp turns often resulted in soft collisions.
- **Complex Interactions** Scenarios where the reference agent was forced to interact with multiple crossing and oncoming threats in the presence of obstacles often resulted in failure.
- **Deadlocks** In certain situations, agents need communication and space-time planning to effectively cooperate on resolving a situation, such as one agent backing all the way up in a very narrow passage to allow another agent to pass first.

7. Conclusion and Future Work

In this paper, we address the fundamental challenge of evaluating and analyzing steering techniques for multi-agent simulations. We present a method of automatically generating and sampling the representative space of challenging scenarios that a steering agent is likely to encounter in dynamically changing environments with both static and dynamic threats. In addition, we propose a method of determining *coverage* and *quality* of a steering algorithm in this space.

We observe that the three agent-based steering approaches we examined are capable of successfully handling 60% of the scenarios that are in the representative scenario space. After examining their failure sets, we see that particularly challenging scenarios include combinations of oncoming and crossing threats in environments with limited room to maneuver, and situations where agents find themselves in deadlocks that require complex coordination between multiple agents. Steering approaches usually time out in these cases or allow collisions so that agents can push through the deadlocks.

The work in [TCP06, GCC*10] optimizes metrics such as

path length, time, and effort in order to generate collision-free trajectories in multi-agent simulations. It would be particularly interesting to see if steering methods that are based on optimality considerations have better coverage and quality using our method of evaluation. Another factor contributing to the low coverage of the evaluated methods is the non-holonomic control of the agents. Many nuanced locomotion capabilities of humans such as sidestepping and careful foot placement are not modeled by these approaches, which greatly limits their ability to handle challenging scenarios. Recent work in navigation [SKRF11] has addressed these limitations in an effort to better model the locomotion of virtual humans. However, modeling agents as discs is still common practice in interactive applications such as games. Our approach can be extended to handle different types of locomotion.

This paper analyzes steering algorithms based on a particular parameterization of the scenario space that focuses on interactions between a small number of proximate agents. Further investigation is needed in order to determine the sensitivity of the evaluation based on these parameters. In addition, applications may require different scenario spaces, for example situations involving large crowds in urban environments. It would be particularly beneficial to design a specification language whereby users can specify and generate benchmarks that meet their requirements.

Our current approach performs random sampling in this space in order to calculate the coverage of an algorithm. In the future, we would like to investigate adaptive sampling methods that use our evaluation criteria to identify and sample more densely areas of interest. Further analysis is also required to automatically cluster and generalize scenarios that are challenging for steering algorithms. Defining sub-spaces in this extremely high dimensional space that are of interest to the research community can prove valuable in the development of the next generation of steering techniques.

8. Acknowledgements

The work in this paper was partially supported by Intel through a Visual Computing grant, and the donation of a 32-core Emerald Ridge system with Xeon processors X7560. In particular we would like to thank Randi Rost, and Scott Buck from Intel for their support.

References

[BH97] BROGAN D. C., HODGINS J. K.: Group behaviors for systems with significant dynamics. *Auton. Robots 4*, 1 (1997), 137–153.

[BMOB03] BRAUN A., MUSSE S. R., OLIVEIRA L. P. L. D., BODMANN B. E. J.: Modeling individual behaviors in crowd simulation. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 143.

[Feu00] FEURTEY F.: *Simulating the Collision Avoidance Behavior of Pedestrians*. Master's thesis, The University of Tokyo, School of Engineering, 2000.

[GCC*10] GUY S. J., CHHUGANI J., CURTIS S., DUBEY P., LIN M., MANOCHA D.: Pledestrians: a least-effort approach to crowd simulation. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2010), SCA '10, Eurographics Association, pp. 119–128.

[HBJW05] HELBING D., BUZNA L., JOHANSSON A., WERNER T.: Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation Science 39*, 1 (2005), 1–24.

[Hen71] HENDERSON L. F.: The statistics of crowd fluids. *Nature 229*, 5284 (February 1971), 381–383.

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamical features of escape panic. *NATURE 407* (2000), 487.

[KSA*09] KAPADIA M., SINGH S., ALLEN B., REINMAN G., FALOUTSOS P.: Steerbug: an interactive framework for specifying and detecting steering behaviors. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), ACM, pp. 209–216.

[KSHF09] KAPADIA M., SINGH S., HEWLETT W., FALOUTSOS P.: Egocentric affordance fields in pedestrian steering. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), ACM, pp. 215–223.

[LCHL07] LEE K. H., CHOI M. G., HONG Q., LEE J.: Group behavior from video: a data-driven approach to crowd simulation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 109–118.

[LCL07] LERNER A., CHRYSANTHOU Y., LISCHINSKI D.: Crowds by example. *Computer Graphics Forum 26*, 3 (September 2007), 655–664.

[LCSCO10] LERNER A., CHRYSANTHOU Y., SHAMIR A., COHEN-OR D.: Context-dependent crowd evaluation. *Comput. Graph. Forum 29*, 7 (2010), 2197–2206.

[LD04] LAMARCHE F., DONIKIAN S.: Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. In *Computer Graphics Forum 23.* (2004).

[LMM03] LOSCOS C., MARCHAL D., MEYER A.: Intuitive crowd behaviour in dense urban environments using local laws. In *TPCG '03: Proceedings of the Theory and Practice of Computer Graphics 2003* (Washington, DC, USA, 2003), IEEE Computer Society, p. 122.

[Lov94] LOVAS G.: Modeling and simulation of pedestrian traffic flow. In *Transportation Research Record* (1994), pp. 429–443.

[LS02] LLOPIS N., SHARP B.: By the Books: Solid Software Engineering for Games, 2002. Games Developers Conference, Round Table.

[McF06] MCFADDEN C.: Improving the QA Process, 2006. Games Developers Conference, Round Table.

[MRHA98] MILAZZO J., ROUPHAIL N., HUMMER J., ALLEN D.: The effect of pedestrians on the capacity of signalized intersections. In *Transportation Research Record* (1998), pp. 37–46.

[PAB07] PELECHANO N., ALLBECK J. M., BADLER N. I.: Controlling individual agents in high-density crowd simulation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 99–108.

[PAB08] PELECHANO N., ALLBECK J. M., BADLER N. I.: *Virtual Crowds: Methods, Simulation, and Control*. Synthesis Lectures on Computer Graphics and Animation. Morgan & Claypool Publishers, 2008.

[PPD07] PARIS S., PETTRÉ J., DONIKIAN S.: Pedestrian reactive navigation for crowd simulation: a predictive approach. In *EUROGRAPHICS 2007* (2007), vol. 26, pp. 665–674.

[PSAB08] PELECHANO N., STOCKER C., ALLBECK J., BADLER N.: Being a part of the crowd: towards validating vr crowds using presence. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1* (2008), AAMAS '08, pp. 136–142.

[Rey87] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM, pp. 25–34.

[Rey99] REYNOLDS C.: Steering behaviors for autonomous characters, 1999.

[RMH05] RUDOMÍN I., MILLÁN E., HERNÁNDEZ B.: Fragment shaders for agent animation using finite state machines. *Simulation Modelling Practice and Theory 13*, 8 (2005), 741–751.

[RP07] REITSMA P. S. A., POLLARD N. S.: Evaluating motion graphs for character animation. *ACM Trans. Graph. 26* (October 2007).

[SGA*07] SUD A., GAYLE R., ANDERSEN E., GUY S., LIN M., MANOCHA D.: Real-time navigation of independent agents using adaptive roadmaps. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology* (2007), ACM, pp. 99–106.

[SKFR09] SINGH S., KAPADIA M., FALOUTSOS P., REINMAN G.: An open framework for developing, evaluating, and sharing steering algorithms. In *Proceedings of the 2nd International Workshop on Motion in Games* (Berlin, Heidelberg, 2009), MIG '09, Springer-Verlag, pp. 158–169.

[SKHF11] SINGH S., KAPADIA M., HEWLETT W., FALOUTSOS P.: A modular framework for adaptive agent-based steering. In *Proceedings of the 2011 symposium on Interactive 3D graphics and games* (2011), I3D '11, ACM.

[SKN*09] SINGH S., KAPADIA M., NAIK M., REINMAN G., FALOUTSOS P.: SteerBench: A Steering Framework for Evaluating Steering Behaviors. *Computer Animation and Virtual Worlds* (2009). http://dx.doi.org/10.1002/cav.277.

[SKRF11] SINGH S., KAPADIA M., REINMAN G., FALOUTSOS P.: Footstep navigation for dynamic crowds. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 203–203.

[ST05] SHAO W., TERZOPOULOS D.: Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2005), ACM, pp. 19–28.

[TCP06] TREUILLE A., COOPER S., POPOVIĆ Z.: Continuum crowds. *ACM Trans. Graph. 25*, 3 (2006), 1160–1168.

[vdBLM08] VAN DEN BERG J., LIN M. C., MANOCHA D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of ICRA* (2008), IEEE, pp. 1928–1935.

[vdBPS*08] VAN DEN BERG J., PATIL S., SEWALL J., MANOCHA D., LIN M.: Interactive navigation of multiple agents in crowded environments. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games* (2008), ACM, pp. 139–147.