Symposium on Computer Animation (2009)
N. N. and N. N. (Editors)

SteerBug: An Interactive Framework for Specifying and Detecting Steering Behaviors

Mubbasir Kapadia[†1] Shawn Singh[‡1] Brian Allen[§1] Glenn Reinman[¶1] Petros Faloutsos[‖1,2]

[1]University of California, Los Angeles, [2]Rhythm and Hues Studios

Abstract

The size of crowds that modern computer games and urban simulations are capable of handling has given rise to the challenging problem of debugging and testing massive simulations of autonomous agents. In this paper, we propose SteerBug: an interactive framework for specifying and detecting steering behaviors. Our framework computes a set of time-varying metrics for agents and their environment, which characterize steering behaviors. We identify behaviors of interest by applying conditions (rules) or user defined sketches on the associated metrics. The behaviors we can specify and detect include unnatural steering, plainly incorrect results, or application-specific behaviors of interest. Our framework is extensible and independent of the specifics of any steering approach. To our knowledge, this is the first work that aims to provide a computational framework for specifying and detecting crowd behaviors in animation.

Categories and Subject Descriptors (according to ACM CCS): I.6.8 [Simulation and Modeling]: Types of SimulationAnimation; I.2.11 [Artificial Intelligence]: Distributed Artificial IntelligenceIntelligent Agents

1. Introduction

Research in real-time pedestrian simulation has grown dramatically in recent years, and its importance is increasing in education, entertainment, urban simulation, and more. For the past several years, autonomous agents in virtual worlds were so novel that their existence alone was immersive and fulfilling. However, now that the initial novelty is wearing off, the believability and intelligence of these agents is becoming increasingly important. In some sense, the research field is quite mature, with a large variety of approaches that have been proposed, and a number of games and virtual worlds using autonomous steering agents in practice. However, there is a clear disparity between the variety of research approaches and the quality of steering agents in real applications. The ultimate goal is still a difficult challenge: to create believable, human-like steering behaviors in complex, dynamic environments for hundreds, even thousands, of agents in real-time.

A key challenge towards this goal is testing and debugging steering behaviors. To illustrate this challenge, consider a large-scale virtual world where developers want to find examples of an agent overtaking another agent – this could be for the purposes of debugging a specific part of their steering algorithm, verifying that particular code-paths are reached, or simply to find an interesting scenario to meet the needs of their specific application. Such a behavior is sufficiently rare that it is difficult to find, yet common enough to be important to the end-user, and too complex to be trivially detected. Generally, validating steering behaviors by visual inspection is impractical for large worlds. Smaller-scale simulations cannot rigorously cover all situations that agents will encounter in rich, unconstrained environments. Moreover, agents can vary dramatically in their physical characteristics, attitudes (i.e. aggressive or passive), and goals, which complicates testing/debugging even further. Mainly for these reasons, research contributions in steering tend to focus on only a subset of desired human-like behaviors, and industry

† mubbasir@cs.ucla.edu
‡ shawnsin@cs.ucla.edu
§ vector@cs.ucla.edu
¶ reinman@cs.ucla.edu
‖ pfal@cs.ucla.edu

applications tend to be conservative, only supporting behaviors and algorithms that are analytically predictable.

A common practice in the industry is to *stress test* and *soak test* an application. These are long-term *automated* tests designed to make sure that an application meets certain standards of quality for production. Stress and soak testing are traditionally used to identify if the application crashes, or if performance dips below some threshold. We propose to apply automated stress and soak testing to agent steering behaviors as well: not only for testing performance and application stability, but also to help identify situations where agents achieve some desired behavior, do something unexpected and interesting, or may need to be debugged. Such an automatic system needs to solve two challenging problems: first, it needs a flexible and intuitive way to specify steering behaviors of interest, and second, it needs a way of detecting steering behaviors during simulation. These are the issues we address in this paper.

Contributions. We propose an interactive framework for specifying and detecting steering behaviors. The framework computes a set of *time varying metrics* (TVMs) by applying a set of pre- and user-defined operators on the position and orientation traces of every object and agent in a simulation. Users can then specify steering behaviors by applying simple or complex conditions on these TVMs, which we call *rules*. Users can also input sketches to match to a particular TVM, such as drawing a spatial trajectory or a timeline of the agent's speed, and they can extend the framework with additional TVMs, operators, or modes of input as necessary. We demonstrate the framework's ability to detect simple behaviors as well as more elaborate behaviors with intuitive specifications.

2. Related Work

Research in steering has matured considerably over the last few years with a large number of approaches being tried and tested. These include, but are not limited to the following:

- **Dynamics based Approaches:** Dynamics based models represent the environment with potential fields or flow maps, where each agent is treated as a particle (e.g., [BH97, G*01, TCP06, HFV00]).
- **Rule-based Approaches:** Rule-based models rely on a predefined list of conditions and actions to perform steering. (e.g., [LD04, LMM03, PPD07, Rey99, RMH05, ST05, PAB07, Bou08, MH04]). Recently, even hybrid approaches are being actively explored, capable of high density crowds (e.g., [PAB07, SGA*, vdBPS*08]).
- **Data-driven Approaches:** 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]).

These approaches 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. However, this technique is used to evaluate the efficiency of approaches offline and is not intended to serve as an aid for *testing and development* in real-time.

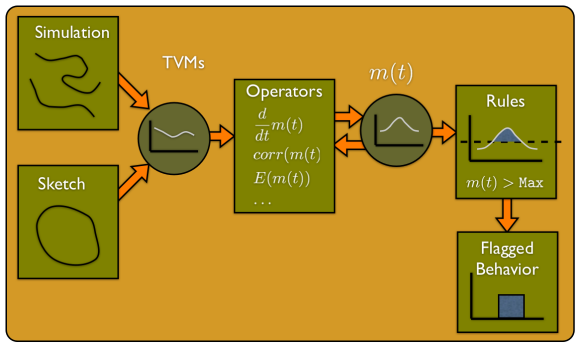


Figure 1: *Overview of the framework.*

The most mature and well studied industry that uses crowd simulations is the gaming industry. Games are complex softwares where several interdependent modules, often built independently, are integrated together in a complete system. Hence, the development risks in game development are particularly high. For this reason, *testing, debugging and verification* [SHHS02] form an integral part of a game's software development life-cycle.

There exists a clear disparity between the advancements in steering approaches and the techniques used to test them. Surveys among game developers [LS02, McF06] show that even though the importance of automated testing is recognized, only a fraction of current game projects employ automated testing. This fact is further emphasized for pedestrian simulations, where there is no clear way of avoiding manual testing. Manual testing does not scale well with complexity of environments and number of agents. For these reasons, the state of the art in the gaming industry, e.g. [Gam08, Ubi08], simulates only a small number of slow moving agents with predictable conditions that can be manually tested.

3. Our Framework

An overview of our framework is shown in Figure 1. The simulator of a virtual world, at every point in time, provides the (a) position and (b) orientation of every agent, and the (c) bounding geometry of each obstacle. Our framework interprets simulation by computing a large number of *time varying metrics* (TVMs): quantities defined at every point in time that measure some property of a single agent, a group of agents, or a region in the environment. TVMs are computed in three main ways: (a) by the simulation itself, (b) by ap-

plying operators to existing TVMs (Section 3.1), and (c) by using a sketch specified by the user (Section 3.2). TVMs capture key aspects of the motion of the agents and their interactions, and they are used to intuitively characterize the behaviors of agents.

We can then specify agent behaviors by defining conditions over the TVMs, which we call *rules* (Section 3.3). Each rule applies thresholds and performs logic operations on TVMs. This allows the framework to detect simple and composite sequences of behaviors, identify properties of agents in the simulation, and measure the similarity between behavior or sketches.

3.1. Computing Time Varying Metrics

The position and orientation TVMs computed by a simulation for each agent have most of the information we are interested in observing. However, it is not straightforward to specify complex behaviors with these TVMs alone. To allow users to easily specify complex behaviors, it is necessary to provide derived TVMs that have additional and more intuitive interpretations, such as velocity, and distance to nearest neighbor. The most important TVMs that we use in our framework are listed in Table 1. Most of our TVMs are computed by a set of *operators*, such as a derivative, or a correlation measure.

Name	Symbol	Units
Position	$\vec{x}(t)$	-
Orientation	$\vec{o}(t)$	-
Total path length	$l(t)$	m
Displacement over window	$\|\|x(t)-x(t-w)\|\|$	m
Speed	$s(t)$	m/s
Velocity	$\vec{v}(t)$	m/s
Acceleration	$\vec{a}(t)$	m/s^2
Kinetic Energy	$E(t)$	J=N.m
Degrees turned	$\theta(t)$	deg
Angular Speed	$\omega(t)$	deg/s
Angular Acceleration	$\alpha(t)$	deg/s^2
Number of collisions	$n_c(t)$	-
Collision penetration	$d_p(t)$	m
Time spent in collision	$t_c(t)$	s
Distance from target	$d_{\text{target}}(t)$	m
Dist. from nearest object	$d_{\text{obj}}(t)$	m
Dist. from nearest agent	$d_{\text{agent}}(t)$	m
Number of agents in region	$n_{\text{agents}}(t,\vec{x})$	-
Density of agents in region	$\rho(t,\vec{x})$	agents/m^2
Avg. $\vec{v}(t)$of agents in region	$\vec{\mu}_v(t,\vec{x})$	m/s

Table 1: *Some of the time varying metrics (TVMs) in our framework. Many more metrics are computed than shown here. All metrics, including total path length and window averages, vary over time.*

Our general notation for an operator is:

$$\text{operator}\{m_1(t), m_2(t), \ldots\}, \tag{1}$$

where $m_i(t)$ are input TVMs. Implicitly, all TVMs have a window interval $[t-w,t]$ that specifies the time window that the operator should use. The size, w, of this window can vary from one frame all the way to the beginning of simulation, i.e., $[0,t]$. Note that operations on TVMs produce TVMs that can also be operated upon. In fact, some of the TVMs that we use are computed by a sequence of operations. Our operators are listed in Table 2 and described as follows:

- **Basic operators:** These operators include the max and min of any TVM over a window, as well as an indicator function for custom events. In combination with the sum operator, indicator functions can also be used as counters. For example, to detect oscillations, we create a TVM that counts how many times the angular velocity flips direction (changes sign) over a window of 3 seconds.
- **Derivatives and integrals:** These operators are used to compute many more dynamics quantities from position and orientation, including velocity, acceleration, and more. Derivatives are very meaningful, because they characterize the rate of change of any TVM.
- **Statistical moments:** Statistical moments are yet more ways that a TVM can be interpreted. For example, we compute mean and variance of many TVMs over a 3-second window. The mean over a window allows us to interpret an instantaneous metric without the high frequency components, while variance over a window characterizes how noisy the TVM might be. In future work, when we consider integrating machine-learning techniques as a third way to specify behaviors, it may also be useful to compute skewness, kurtosis, and higher order moments.
- **Similarity measures:** It is often useful to compare how similar two TVMs are to each other. For example, we may want to identify when an agent's trajectory matches a user's input sketch, or we may want to identify when the distance between two agents is linearly decreasing. To measure the similarity of two TVMs, we measure their normalized correlation. In some cases we also need to measure the similarity of two spatial vectors. In this case, we measure their colinearity – which is 1 if the vectors face the same direction, -1 if they face opposite directions. In their simplest form, correlation and colinearity are both dot products, the major distinction is that correlation shifts the TVMs to the mean, while colinearity does not. We have experimented with the distance measure proposed in [ACH*91] and the Hausdorff distance. However, in our current framework, the correlation operator has provided sufficient results.
- **Geometric properties:** Many TVMs are inherently vector quantities, and so it is useful to compute lengths of vectors, Euclidean distance, computing collisions (i.e. thresholds over distances), and other related geometric properties.

Operator	Symbol	Formula
Indicator function	I{condition/event}	$I(t) = 1$ if condition/event occurred, 0 otherwise
Min	$\min\{m(t)\}$	$\min\{m(s)\|s \in [t-w,t]\}$
Max	$\max\{m(t)\}$	$\max\{m(s)\|s \in [t-w,t]\}$
Derivative	$\frac{d}{dt}\{m(t)\}$	$\frac{d}{dt}\{m(t)\}$
Sum over window	$\text{sum}\{m(t)\}$	$\sum_{s=t-w}^{t} m(s)$
Mean (average)	$\text{mean}\{m(t)\}$	$\mu(t) = \frac{1}{w}\sum_{s=t-w}^{t} m(s)$
Variance	$\text{var}\{m(t)\}$	$\sigma(t) = \frac{1}{w}\sum_{s=t-w}^{t}(m(s)-\mu(t))^2$
Correlation	$\text{corr}\{m_1(t), m_2(t)\}$	$\frac{\sum_{s=t-w}^{t}(m_1(s)-\mu_1(t))(m_2(s)-\mu_2(s))}{(w-1)\sqrt{\sigma_1(t)\sigma_2(t)}}$
Colinearity	$\text{col}\{\vec{m}_1(t), \vec{m}_2(t)\}$	$\vec{m}_1(t) \cdot \vec{m}_2(t)$

Table 2: *Some of the operators used to compute TVMs. All metrics except derivative and colinearity can be defined over an arbitrary window size w, ranging from instantaneous ($w = 0$) to the entire simulation history.*

3.2. Sketch-Based Interface

Sketches can be used to easily describe interesting behaviors that cannot be intuitively specified using rules. The user may sketch the behavior of any TVM. However, sketches are especially well suited to represent spatial TVMs such as the trajectories of agents. Given a sketch, we compute its similarity to the related TVM over a sliding window. When the similarity measure crosses a threshold, the agent is said to exhibit the behavior described by the curve. The user also has the freedom to draw multiple curves on the sketch-pad. Each curve represents the desired behavior of a separate agent. This multi-agent behavior is detected when each of the individual behaviors are simultaneously detected in different agents when they have the same spatial relationship as the sketch. Using this interface, multiple agent interactions can be easily specified that may not be intuitively described using rules, as shown in Figure 8. Currently, we use a correlation operator to compute the similarity measure between curves and provides translation and scale invariance. Rotation invariance can be incorporated into our system by detecting patterns over derivatives of spatial TVMs.

We refer the reader to the vast body of existing literature in pattern classification [DHS01, KLV00] and shape matching [VV01] to identify advanced techniques such as the Frechet Distance [Rot07] and methods based on shape descriptors [NF07] which could be incorporated into our framework in future work.

3.3. Using Rules to Specify Behaviors

Rules are conditions on one or more TVMs of one or more agents that represent an interesting behavior in a simulation. An atomic rule is a single condition on one TVM for one agent. When the condition is true, it represents that the agent exhibited a basic behavior or property at that time. For example, when the instantaneous speed of an agent, $s(t)$, crosses a desired threshold (2.0 m/s for walking), it indicates the agent walked unrealistically fast. This can be represented by the following atomic rule: $a = s(t) > \texttt{Desired}$. Atomic rules serve as the building blocks for complex rules, which represent sequential or parallel occurrences of basic behaviors, even for multiple agents over different TVMs. We employ the following syntax for complex rules:

- $a_1 \vee a_2 \wedge a_3$: A behavior in which atomic behaviors, a_1 or a_2 and a_3 are simultaneously exhibited.
- $[c_1] \wedge [c_2]$: A behavior in which complex behaviors c_1 and c_2 are seen in different agents.
- $c_1 \diamond c_2$: A behavior in which complex behaviors c_1 and c_2 are seen in the same agent one after another.
- $[c_1] \diamond [c_2]$: A behavior in which complex behaviors c_1 and c_2 are seen in different agents one after another.
- $[c_1]_a \diamond [c_2]_b$: A behavior in which complex behaviors c_1 and c_2 are seen in agents a and b respectively, one after another.

Using this syntax-based interface, we have devised a set of rules that describe common anomalies and interesting steering behaviors. These rules are discussed in more detail in Table 3. In addition, custom rules and metrics can be defined to detect other behaviors to meet the needs of any specific application. As long as the necessary information exists in the input simulation, any behavior that a user may wish to specify can be defined using the right combination of metrics and rules.

4. Using the Framework

This section discusses the use of our framework in: (1) detecting anomalies that arise in most steering simulations (Section 4.1), (2) specifying and detecting user-defined behaviors of interest (Section 4.2), (3) matching user-defined sketch patterns over the trajectories of one or more agents (Section 4.3), and (4) facilitating stress and soak testing of agent steering (Section 4.4).

Rule	Interpretation
$d_p(t) > \texttt{Max} \wedge t_c(t) > \texttt{Max}$	Thresholded collision event.
$s(t) > \texttt{Max}$	Agent violates physical constraints on speed.
$\vec{a}(t) > \texttt{Max}$	Agent violates physical constraints on acceleration.
$\omega(t) > \texttt{Max}$	Agent violates physical constraints on angular speed.
$d_{obj}(t) < \texttt{Min} \wedge$ $corr\{d_{obj}(t), 1-t\} > \texttt{0.9}$	Agent is steering into an obstacle.
$d_{agent}(t) < \texttt{Min} \wedge$ $corr\{d_{agent}(t), 1-t\} > \texttt{0.9}$	Agent is steering into another agent.
$mean\{d_{obj}(t)\} > \texttt{Max}$	Wall-hugging behavior.
$s(t) = \texttt{Zero}$	Agent momentarily stops.
$s(t) > \texttt{Desired}$	Agent momentarily speeds up.
$mean\{s(t)\} = \texttt{Zero}$	Agent stationary over window.
$mean\{s(t)\} > \texttt{Desired}$	Agent exceeds desirable speed consistently.
$sum\{\theta(t)\} > \texttt{Desired}$	Noticeable turn over a short period.
$mean\{\omega(t)\} > \texttt{Desired}$	Agent consistently exceeds desired angular speed.
$\vec{a}(t) > \texttt{Desired}$	Sudden acceleration.
$s(t) = \texttt{Zero} \wedge d_{obj}(t) > \texttt{Max} \wedge$ $d_{agent}(t) > \texttt{Max}$	Agent stationary in absence of obstacles and threats.
$sum\{l(t)\} > \texttt{Max} \wedge$ $\|x(t) - x(t-w)\| < \texttt{Min}$	Agent moved without progress (backtracking).
$sum\{I\{\omega(t) \text{ flips sign}\}\} > \texttt{Max}$	Agent has oscillations.
$sum\{\theta(t)\} > \texttt{Max} \wedge$ $sum\{I\{\omega(t) \text{ flips sign}\}\} < \texttt{Min}$	Agent steered in a circle.
$corr\{d_{target}(t), t\} > \texttt{0.9}$	Agent traveling away from the target.

Table 3: *Some of the rules used to identify behaviors of interest.*

4.1. Detecting Common Bugs in Steering

There are several common anomalous behaviors that are prevalent in steering simulations today. These include unnatural oscillations, steering into walls, circular motion, wall-hugging, unnecessary stopping, and violation of physical constraints. We provide the user with a list of pre-defined behaviors, defined using rules, which represent these common bugs. A list of these rules is provided in Table 3. We demonstrate the use of these pre-defined rules to validate a simulation of 500 agents in an environment with randomly positioned obstacles. We discuss some of these rules here and refer the reader to the supplementary video for a demonstration of the rules being flagged.

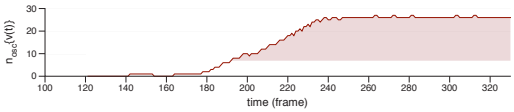


Figure 2: *Demonstration of oscillations being flagged because the number of oscillations in angular speed over a 3-second window,* $sum\{I\{\omega(t) \text{ flips sign}\}\}$*, crosses a threshold.*

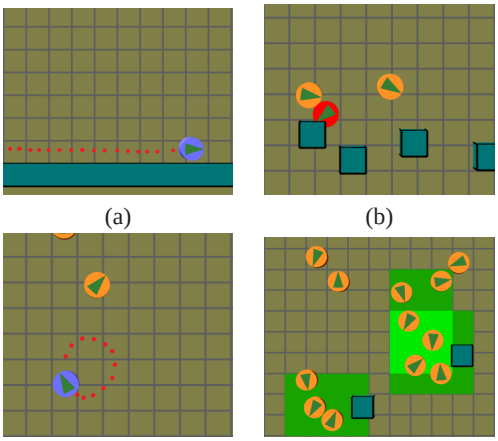


Figure 3: *Common bugs in steering: (a) Wall Hugging. (b) Thresholded collisions. (c) Circular motion. (d) Dense environments.*

Oscillations: Dynamics-based approaches are often susceptible to oscillations in the motions of the simulated agents. Using our framework, we describe a simple rule that can be used to detect oscillations:

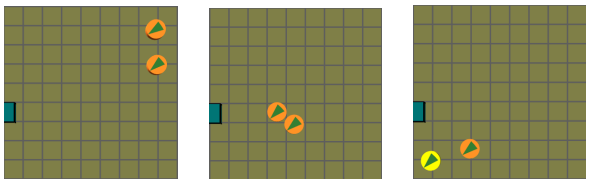
$$sum\{I\{\omega(t) \text{ flips sign}\}\} > \texttt{Max}.$$

Figure 2 illustrates the rule being flagged as the number of oscillations over a window in the angular speed of the agent crosses a threshold.

Circular Motion: An agent traveling in a circle can be described as the total degrees turned over a window being approximately 360 degrees while the agent does not perform oscillations in turning. Using the TVMs described above, the rule for a circular motion behavior is:

$$sum\{\theta(t)\} > \texttt{Max} \wedge sum\{I\{\omega(t) \text{ flips sign}\}\} < \texttt{Min}.$$

Dense regions: Detecting clusters of agents grouped in a region of the environment is often indicative of bottlenecks or incorrect steering behaviors. The environment density TVM serves as a powerful indicator of where other bugs may be occurring. The rule used to flag dense regions is: $\rho(t, \vec{x}) > \texttt{Max}$.



(a) (b) (c)

Figure 4: *Flagging of overtake behavior. These snapshots correspond to the TVMs shown in Figure 5.*

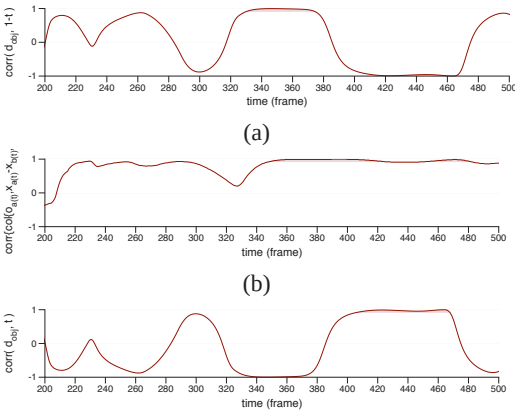


Figure 5: *TVMs of an overtaking behavior: (a) Distance between agents decreases (distance correlates with $1-t$). (b) The transition from being behind the agent to in front of the agent (colinearity correlates with $1-t$). (c) Distance between agents increases.*

4.2. Detecting More Elaborate Behaviors

Our framework can easily be extended by writing custom rules that flag interesting behaviors that are specific to the steering application. We demonstrate the expressive power of our rule-based syntax by specifying and detecting two complex behaviors: *overtaking* and *pick-pocketing*.

Overtaking: Overtaking is a high-level behavior that represents a sequence of interactions between two agents (Figures 4 and 5). We specify agent A overtaking agent B with the following sequence of behaviors: (1) The distance between agent A and B initially decreases, (2) agent A reaches, and passes Agent B (the act of overtaking) and, (3) agent A goes further ahead and the distance between the two agents increases. The rule for overtaking is:

$$\begin{aligned} &\mathrm{corr}\{d_{\mathrm{target}}(t), 1-t\} > \texttt{Max} \\ \diamond\ &\mathrm{corr}\{\mathrm{col}\{\vec{o}(t), \vec{x}(t) - \vec{x}_{\mathrm{target}}(t)\}, 1-t\} > \texttt{Max} \\ \diamond\ &\mathrm{corr}\{d_{\mathrm{target}}(t), t\} > \texttt{Max} \end{aligned}$$

Pick-Pocketing: Imagine a scenario where two thieves cooperate to steal someone's wallet. The first thief is the distractor, grabbing the attention of the victim so that the second thief can steal the wallet, unnoticed. This composite behavior can be described using the following sequence of atomic behaviors: (1) The distractor nears the victim (distance from target monotonically decreases), (2) the distractor distracts the victim (victim faces in the direction of the distractor), (c) the thief closes in on victim (distance from target monotonically decreases) and, (d) the thief makes a run for it (distance from target monotonically increases) (Figure 6). Using our rule-based syntax, we describe this behavior as follows:

$$\begin{aligned} &[\mathrm{corr}\{d_{\mathrm{target}}(t), 1-t\} > \texttt{Max} \wedge d_{\mathrm{target}}(t) < \texttt{Min}]_{\mathrm{distractor}} \\ \diamond\ &[\mathrm{col}\{\vec{o}(t), \vec{x}(t) - \vec{x}_{\mathrm{distractor}}(t)\} > \texttt{Max}]_{\mathrm{victim}} \\ \diamond\ &[\mathrm{corr}\{d_{\mathrm{target}}(t), 1-t\} > \texttt{Max} \wedge d_{\mathrm{target}}(t) < \texttt{Min}]_{\mathrm{thief}} \\ \diamond\ &[\mathrm{corr}\{d_{\mathrm{target}}(t), t\} > \texttt{Max}]_{\mathrm{thief}} \end{aligned}$$

4.3. Use of Sketch-Based Interface

The sketch-based interface serves as an easy way for a user to specify behaviors that may be non-intuitive using the rule system. Using sketches, our framework can identify behaviors such as a complicated trajectory (Figure 8a and 8b), and multi-agent interactions (Figure 8c and 8d). A demo of this interface is shown in the supplementary video.

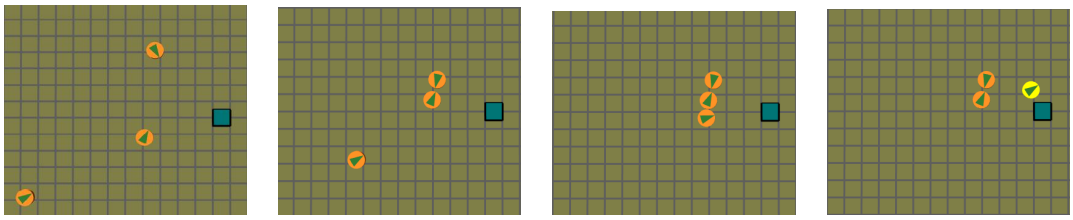
4.4. Facilitating Soak and Stress Testing

Our framework can be used as a powerful tool to provide additional insight while performing stress and soak tests on a steering simulation. We demonstrate this by running an evacuation simulation of 2000 agents over 10000 frames and check for physically invalid steering behaviors. We observe that the number of bugs increases exponentially with increase in time as illustrated in Figure 9. A more careful analysis reveals that a bottleneck is created at the exit point and the agents exhibit aggressive behavior by squeezing through, resulting in collisions.

Performance: We evaluated the performance of our framework by comparing the processing times of the simulation with and without SteerBug. We observe a 22% increase in computational costs when all the rules in SteerBug are activated. This overhead can be reduced due to the selective computation of TVMs based on the rules that are activated.

5. Conclusion

We have proposed an interactive framework for specifying and detecting steering behaviors. At the core of our framework lies a set of time-varying metrics and associated op-



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

Figure 6: *Pick-pocketing behavior being flagged. These snapshots correspond to the TVMs shown in Figure 7.*

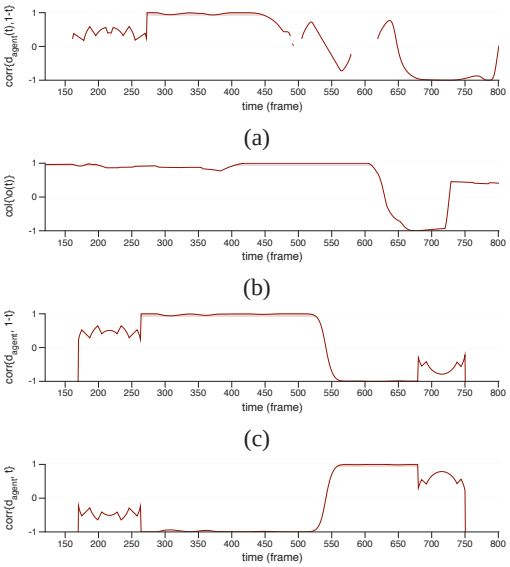
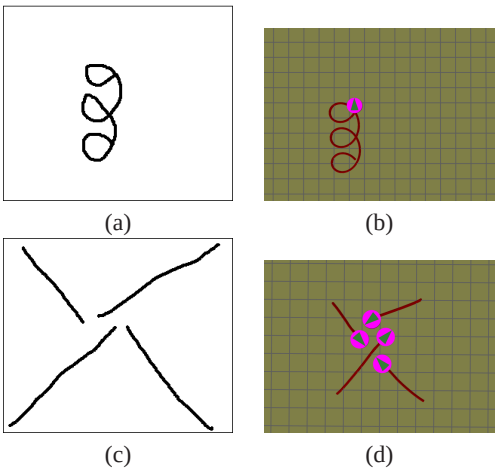


Figure 7: *TVMs used for pick-pocketing: (a) The distractor nears the victim (distance from victim monotonically decreases). (b) The distractor distracts the victim (victim turns to face the distractor). (c) The thief closes in on the victim (distance from target monotonically decreases). (d) The thief runs away (distance from victim monotonically decreases).*



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

Figure 8: *Sketch-based specification of a 3 loop behavior (a) and a four-way crossing behavior (c), and corresponding instances being flagged at (b) and (d) respectively.*

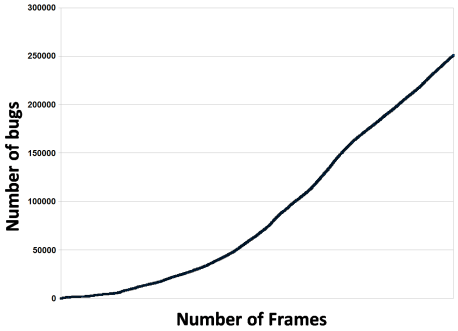


Figure 9: *Results of evacuation simulation of 2000 agents over 10000 frames: Number of bugs flagged vs. time.*

erators, that can be combined with the proposed rule-based scheme to define complex agent behaviors. We have demonstrated the effectiveness of our approach with a number of challenging examples. Our framework is extensible and independent of the specifics of any steering algorithm. To our knowledge, this papers takes a first step towards solving the challenging problem of testing and debugging large scale simulations that involve autonomous agents.

6. Acknowledgements

We wish to thank the anonymous reviewers for their comments. The work in this paper was partially supported by NSF grant No. CCF-0429983. We thank Intel Corp., Microsoft Corp., and AMD/ATI Corp. for their generous support through equipment and software grants. We also thank Gabriele Nataneli for helping to develop the sketch based interface.

References

[ACH*91] ARKIN E., CHEW L., HUTTENLOCHER D., KEDEM K., MITCHELL J.: An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 3 (1991), 209–216.

[BH97] BROGAN D. C., HODGINS J. K.: Group behaviors for systems with significant dynamics. *Auton. Robots 4*, 1 (1997), 137–153.

[Bou08] BOULIC R.: Relaxed steering towards oriented region goals. *Lecture Notes in Computer Science 5277, MIG 2008* (2008), 176–187.

[DHS01] DUDA R., HART P., STORK D.: *Pattern Classification*. Wiley, 2001.

[G*01] GOLDENSTEIN S., ET AL.: Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers and Graphics 25*, 6 (2001), 983–998.

[Gam08] GAMES R.: Grand Theft Auto iv, 2008.

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamical features of escape panic. *Nature 407* (2000), 487.

[KLV00] KULKARNI S. R., LUGOSI G., VENKATESH S. S.: Learning pattern classification— a survey (invited paper). 134–162.

[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* (2007), pp. 109–118.

[LCL07] LERNER A., CHRYSANTHOU Y., LISCHINSKI D.: Crowds by example. *Computer Graphics Forum 26*, 3 (September 2007), 655–664.

[LD04] LAMARCHE F., DONIKIAN S.: Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum 23* (2004), 509–518(10).

[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* (2003), IEEE Computer Society, p. 122.

[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.

[MH04] METOYER R. A., HODGINS J. K.: Reactive pedestrian path following from examples. *The Visual Computer 20*, 10 (November 2004), 635–649.

[NF07] NATANELI G., FALOUTSOS P.: Robust classification of strokes with SVM and grouping. In *ISVC '07* (2007), Springer-Verlag, pp. 76–87.

[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* (2007), pp. 99–108.

[PPD07] PARIS S., PETTRE J., DONIKIAN S.: Pedestrian reactive navigation for crowd simulation: a predictive approach. In *EUROGRAPHICS 2007* (2007), pp. 665–674.

[Rey99] REYNOLDS C.: Steering behaviors for autonomous characters. In *Game Developers Conference* (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.

[Rot07] ROTE G.: Computing the fréchet distance between piecewise smooth curves. *Comput. Geom. Theory Appl. 37*, 3 (2007), 162–174.

[SGA*] 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*, ACM, pp. 99–106.

[SHHS02] SANTHANAM P., HAILPERN B., HAILPERN B., SANTHANAM P.: Software debugging, testing, and verification. *IBM Systems Journal 41* (2002), 4–12.

[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.

[ST05] SHAO W., TERZOPOULOS D.: Autonomous pedestrians. In *SCA '05: Proc. of the 2005 ACM SIGGRAPH/Eurographics symp. on Computer animation* (2005), pp. 19–28.

[TCP06] TREUILLE A., COOPER S., POPOVIĆ Z.: Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (2006), pp. 1160–1168.

[Ubi08] UBISOFT: Assassins creed, 2008.

[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), pp. 139–147.

[VV01] VELTKAMP R. C., VELTKAMP R. C.: Shape matching: Similarity measures and algorithms. pp. 188–197.