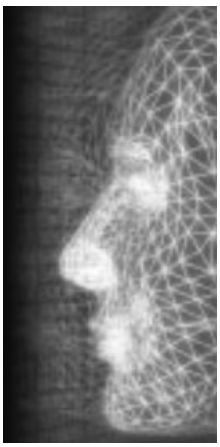
COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2009; **20**: 533–548
Published online 23 February 2009 in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/cav.277

SteerBench: a benchmark suite for evaluating steering behaviors



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

Steering is a challenging task, required by nearly all agents in virtual worlds. There is a large and growing number of approaches for steering, and it is becoming increasingly important to ask a fundamental question: how can we objectively compare steering algorithms? To our knowledge, there is no standard way of evaluating or comparing the quality of steering solutions. This paper presents SteerBench: a benchmark framework for objectively evaluating steering behaviors for virtual agents. We propose a diverse set of test cases, metrics of evaluation, and a scoring method that can be used to compare different steering algorithms. Our framework can be easily customized by a user to evaluate specific behaviors and new test cases. We demonstrate our benchmark process on two example steering algorithms, showing the insight gained from our metrics. We hope that this framework can grow into a standard for steering evaluation. Copyright © 2009 John Wiley & Sons, Ltd.

Received: 17 November 2008; Accepted: 28 November 2008

KEY WORDS: steering; pedestrians; crowd behaviors; benchmarking

Introduction

Immersive virtual worlds are quickly growing in importance for education, collaboration, and entertainment. Most virtual worlds are populated with autonomous, intelligent agents that require efficient, robust algorithms to navigate in a large-scale, complex environment. Even when an agent has cognitively decided its destination, the problem of maneuvering around the various obstacles and other agents in the environment is extremely complex, leaving the agent with an arbitrarily large number of possible steering choices.

This rich set of steering choices reflects a large number of techniques that can be used to implement steering in virtual worlds. Current algorithms are usually focused on a subset of the problem's challenges, and often papers have only enough space to showcase their novel features. Anyone who wants to implement a steering algorithm will see an overwhelming number of existing approaches and not know their strengths and limitations.

Given the importance of steering in modern applications and the growing number of steering algorithms, it is important and timely to ask the question, *how can we compare different steering approaches?* To our knowledge this paper makes the first attempt to answer this fundamental question.

A steering benchmark that characterizes and compares algorithms is also needed to help researchers focus on developing the *right* kinds of steering behaviors—for example, an algorithm should avoid collisions by making intelligent human-like steering decisions instead of relying on collision resolution to prevent interpenetration. Similarly, we want a framework to test and expose the predictive elements of the human steering process.

The main challenge in designing steering benchmarks is how to evaluate a steering algorithm *objectively*. The quality of results of a steering algorithm depends on many factors, many of which are situation-specific and/or depend on cognitive decisions by the agents. For example, an agent may decide to push through a crowd or politely go around the crowd, depending on the agent's situation and personality. To remain objective even with seemingly ad hoc constraints, we propose that steering evaluation should satisfy the three criteria: (1) the test cases should be representative of a broad range of steering situations that are common in real-world scenarios, (2) the evaluation should be blind to the

*Correspondence to: S. Singh, UCLA Depatment of Computer Science, Boelter Hall 4531-F, Los Angeles, CA 90095-1596, USA. E-mail: shawnsin@cs.ucla.edu

specifics of the steering algorithm, and (3) the evaluation should be customizable to allow a user to test for certain expected behaviors over others.

In this paper, we present *SteerBench*, a novel framework for comparing steering behaviors. SteerBench contains two major components:

- *A diverse suite of steering benchmarks*: We propose a forward-looking set of scenarios that capture the broad range of situations a steering algorithm may encounter in practical applications.
- *A method and metrics of evaluation*: We propose a set of metrics that can be used to evaluate the results of steering algorithms. We also propose a method of scoring the results, so that two steering algorithms can be compared.

Our test cases are carefully chosen to test several fundamental and challenging steering tasks. We separate the concepts of providing test cases and evaluating steering results. This allows users to specialize test cases to their needs, and our benchmark evaluation would still apply to the custom test cases. Our evaluation uses metrics computed only from the position, direction, and the goal that the agent is trying to reach. These metrics do not require any knowledge of the algorithm that produced the steering result, yet at the same time, the metrics are insightful for debugging and developing better steering behaviors.

Related Work

Benchmarking is a crucial process in many fields—ranging from business management to software performance. Benchmark suites have been developed for a variety of purposes related for graphics and multimedia, for example, 3DMark graphics hardware benchmark, RealStorm global illumination benchmark, animation for ray tracing,[1] SPEC architecture benchmark, and many more. In these fields, benchmarks have clear metrics for comparison: performance in seconds, signal-to-noise ratio, power consumption, area, monetary price, etc. Steering behaviors and other aspects of artificial intelligence do not have a clear objective metric—instead, much of the evaluation is inherently subjective.

Metrics of Evaluation

Pelechano *et al*.[2] use an interesting metric, *presence*, to evaluate crowd behaviors, however it requires a user's in-depth participation. Some steering papers have used number of collisions as a metric to minimize when developing the algorithm, but not as an explicit metric for evaluation (e.g., Reference [3]). Other works (e.g., Reference [4]) use specialized metrics, such as "rate of people exiting a room." Some papers on pedestrian simulation simply rely on subjective evaluation of their results. SteerBench proposes a rich set of metrics that can capture these and other properties of any algorithm automatically.

It is well accepted that efficient behaviors are natural. Efficiency metrics have been used to create more believable or natural motion for human animation, e.g., References [5–7] and in general it is common to use efficiency as an optimization criteria, for example when animating birds,[8] humans,[9] and fish.[10] Our work applies this same principle to the evaluation of steering behaviors, while keeping the user in control of the evaluation process.

Approaches to Steering

Most steering behaviors can be classified into three major categories: dynamics-based, agent-based, or data-driven. Dynamics-based models represent the environment with potential fields or flow maps, often treating each agent as a particle (e.g., References [11–13]). Rule-based models use heavy branching to determine the exact scenario being described, and perform an action associated with that scenario (e.g., References [3,14–18]). Recently, even hybrid approaches are being actively explored, capable of high density crowds (e.g., References [19–21]). Data-driven methods use existing video data or motion capture to derive steering choices that are then used in virtual worlds (e.g., References [22,23]).

Overview of SteerBench

SteerBench consists of two major parts: (1) a diverse set of test cases, and (2) a benchmark evaluation approach that computes several metrics and scores a steering algorithm.

The current suite contains 38 scenarios, ranging from fundamental sanity checks to very challenging deadlock and crowd scenarios. Several of the scenarios have many variations, resulting in a total of 56 test cases. The test cases can be roughly classified in five conceptual categories: (1) simple validation scenarios, (2) basic one-on-one interactions, (3) agent interactions including

DOI: 10.1002/cav

obstacles, (4) group interactions, and (5) large-scale scenarios. All scenarios are detailed in Tables 1–5.

Each test case specifies a number of agents and static objects. Static objects are specified by a bounding box. Agents are specified by their size, initial position, initial direction, desired speed, and target location(s). Additionally, each agent has a set of weights that configure the evaluation process. Finally, each test

Scenario	Description
Simple	One agent steering towards a target located to the left, right, or behind.
Simple-obstacle	One agent steering towards a target located behind an obstacle.
Simple-wall	Agents steering around a wall to reach a target.
Local-minimum	One agent steering around a U-shaped wall to reach a target outside the U-shape. The agent should not get stuck inside the U-shape.
Curves	One agent steering through an S-curve to reach the target. Note that the walls of the corridor have intentionally jagged outlines to reveal the resolution and/or smoothing abilities of steering algorithms.

Table 1. Simple validation scenarios. These scenarios are designed to test very basic, fundamental abilities of a steering agent that every algorithm should be able to handle

Scenario	Description
Oncoming	Two agents traveling in opposite directions towards a head-on collision.
Crossing	Two agents crossing paths at various angles.
Oncoming-trick	Two oncoming agents that will not collide because their targets are before their point of intersection. The agents should recognize that they do not pose a threat to each other and maintain their course.
Crossing-trick	Two crossing agents that will not collide because their targets are before their point of intersection. The agents should recognize that they do not pose a threat to each other and maintain their course.
Similar-direction	Two agents with slightly different goals traveling in a similar direction. The agents should not unnecessarily predict collisions.

Table 2. Basic one-on-one interactions. These scenarios test the ability of two agents to steer around each other in the absence of static obstacles

Scenario	Description
Oncoming-obstacle	Two oncoming agents, with an additional obstacle in the way.
Crossing-obstacle	Two crossing agents, with an additional obstacle in the way.
Surprise	Two agents that do not see each other until the last minute because of large obstacles. A few collisions are acceptable, but the agents should still proceed efficiently.
Squeeze	Two oncoming agents walking through a narrow hallway. Agents should understand that both of them can fit without having to slow down.
Doorway-one-way	Two agents enter a doorway from the same side. There is room for only one agent at a time through the doorway.
*Doorway-two-way	Two oncoming agents walk through a doorway from opposite sides. One agent must step aside to allow the other to go first.
Overtake	One agent is expected to overtake the other agent in a hallway.
*Overtake-obstacle	One agent is expected to overtake the other agent, with obstacles in the hallway.

Table 3. Agent-agent interactions including obstacles. These scenarios test the ability of an agent to navigate around static objects while interacting with other agents

***Indicates cases that we predict will be very difficult**

DOI: 10.1002/cav

Scenario	Description
Fan-in	A small group of agents aiming for the same target. This tests how agents cooperate while contending for the same space.
Fan-out	A small group of agents aiming for slightly separated targets. This tests whether agents will unnaturally stick to the crowd when their goal is in a different direction.
Cut-across	One agent cutting across a small group.
3-way-confusion	Three agents traveling in different directions, meeting at nearly the same time.
4-way-confusion	Four agents traveling in four opposing directions, meeting at nearly the same time.
*4-way-obstacle	Four agents crossing paths with a static object in the way.
Frogger	One agent encounters many perpendicular crossing agents.
Lone-vs.-group	One agent encounters a group of agents traveling in the opposite direction. Three test cases involve a group of 10 people with tight, medium, and large spacing, respectively.
*Oncoming-groups	A small group of agents encounters another small group of agents traveling in opposite direction. Three test cases use varying densities of the groups.
3-squeeze	Two agents facing the same direction encounter an oncoming agent in a narrow hallway.
*double-squeeze	Two agents facing the same direction encounter two oncoming agents in a narrow hallway.
*wall-squeeze	Two agents facing the same direction encounter an oncoming agent in a narrow hallway with an obstacle.

Table 4. Group interactions. Group interactions are composed of several agents and static objects, intended to test an algorithm's ability to handle a variety of common situations
***Indicates cases that we predict will be very difficult**

Scenario	Description
Hallway-one-way	Many agents traveling in the same direction through a hallway.
Hallway-two-way	Many agents traveling in either direction through a hallway. Agents should form lanes.
*Bottleneck-squeeze	All agents begin on one side of the arena, and must enter and traverse a hallway to reach the target. Note that hard corners at the bottleneck are much more challenging than rounded corners.
*Evacuation	All agents must exit a crowded room through a narrow door, and fan-out from the exit.
*Free-tickets	All agents are aiming for the same target in the middle of the arena, and have a random secondary goal once the middle target is reached. This scenario is particularly difficult because agents that reach the middle goal must then turn to face a dense oncoming crowd.
Random	Each agent is placed randomly in the arena and has an individual random target. Here, stress is placed on handling a large number of agents. Our default test case specifies 5000 agents for this scenario.
Forest	Each agent is placed randomly in an arena filled with small obstacles. Our default test case specifies 500 agents for this scenario.
Urban	Each agent is placed randomly in an arena filled with building-sized obstacles. Our default test case specifies 500 agents for this scenario.

Table 5. Large scale scenarios. These scenarios are designed to stress-test the ability of an algorithm to handle macroscopic situations, and to scale to a large number of agents
***Indicates cases that we predict will be very difficult**

DOI: 10.1002/cav

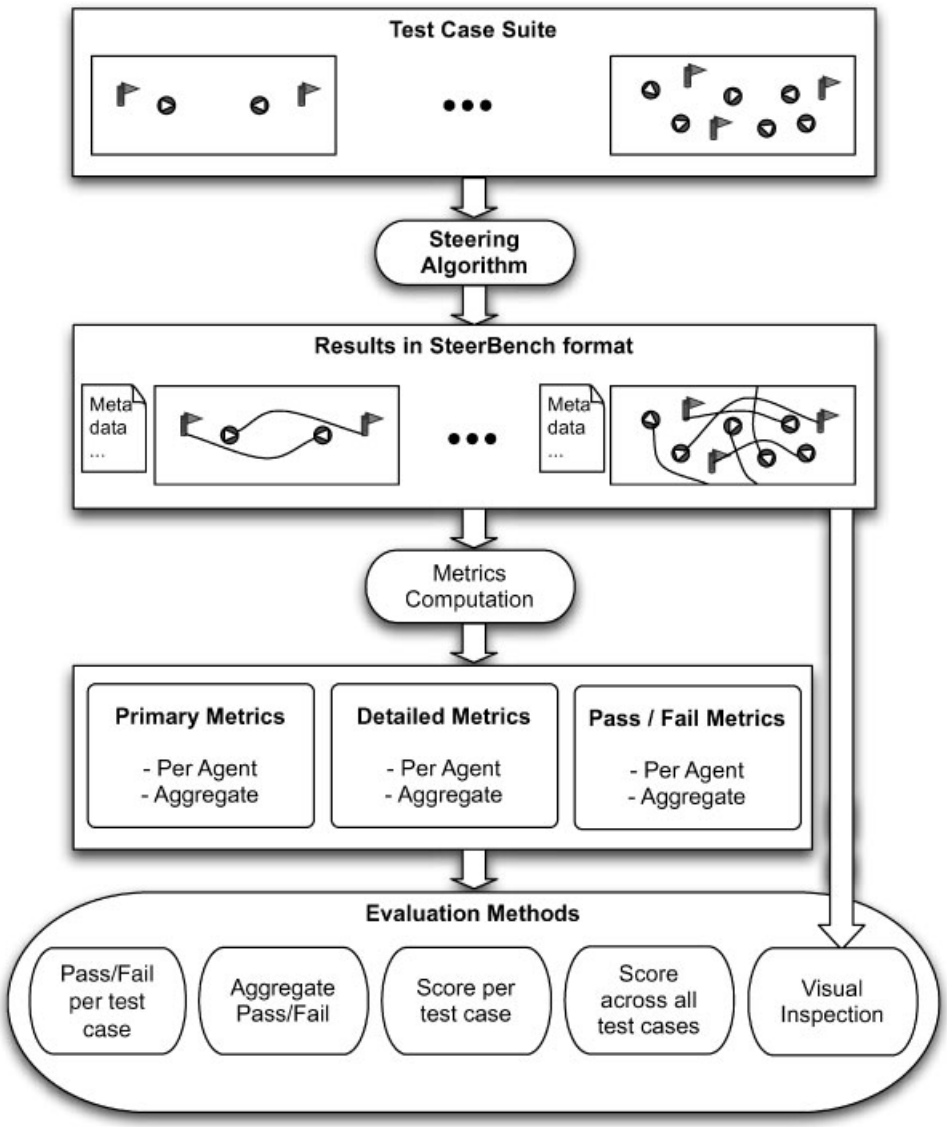


Figure 1. SteerBench workflow. A user records the performance of an algorithm on our test cases. Our framework then computes metrics and scores which can be used to evaluate the algorithm in a variety of ways.

case also specifies conditions that define "passing" or "failing" the test case.

The workflow of our framework is shown in Figure 1. First, the user runs his algorithm on our test cases, and records the position, direction, and goal target of every agent for every frame. SteerBench provides a C++ library that reads and writes a specific file format, for easy integration with existing steering algorithms. Second, our evaluation tool reads the recorded information and computes detailed statistics, primary metrics, an aggregate score, and whether the algorithm passed the test case.

Given the results of an algorithm on our test cases, our metrics, and our scoring method, the user has several options to evaluate steering algorithms. (1) The user can rank algorithms based on an overall score, computed across all test cases. Such a score is primarily meaningful for *comparing* two or more algorithms. (2) The user can rank algorithms based on one test case, or one agent's behavior within a test case. (3) The user can evaluate

DOI: 10.1002/cav

whether an algorithm "passes" or "fails" a test case, or count how many test cases it passes. (4) The user can examine detailed metrics of the performance of an agent, for debugging or detailed evaluation. (5) Finally, the user can visually inspect results of an algorithm using our test cases.

Benchmark Suite

Tables 1–5 describe the scenarios included in the current version of our benchmark suite. These scenarios are visually depicted in Figure 2. Many of these test cases are very challenging and forward-looking, and we expect that initially very few algorithms, if any, will be able to successfully handle all scenarios gracefully.

Test Case Design Choices

The main challenge when designing the benchmark suite is to cover the range of possible scenarios without having an inordinate number of test cases. With this in mind, we choose our test cases to be common, frequently appearing scenarios, but with challenging, worst-case parameters. For example, most of the static obstacles have sharp corners which are generally more difficult to handle than smooth ones.

In our experience, an agent in a typical real-world environment faces the following situations:

1. *Walking alone*: Real-world agents can steer directly to their goals, navigate around fixed obstacles, and maintain their desired speed when there are no other constraints. Such behaviors are fundamental, and must be included in a benchmark suite.
2. *Local interactions*: Most of the time, an agent interacts with only 2–4 nearby agents and a few obstacles at a time. SteerBench tests how an algorithm behaves with two, three, and four agents approaching each other in various directions, with and without static objects constraining the possible solutions.
3. *Walking as part of a small group*: Often an agent moves in the same direction with 2–6 other agents and encounters other groups and obstacles. Real-world agents generally have empathetic understanding of how the group should behave and willingly cooperate.
4. *Walking as part of a crowd*: Groups of hundreds or thousands of agents tend to form less frequently, and only in specific situations, for example when a large number of agents exit or enter a building at the same time. SteerBench includes test cases to test for crowd queuing, lane formation, and navigating against oncoming crowds.
5. *Deadlock avoidance*: Real-world agents usually understand when a situation will result in all agents getting stuck (i.e., deadlock) and how to avoid these situations by stepping out of the way or backtracking. Backtracking is particularly challenging for most algorithms because an agent may need to move away from its goal in order to resolve the deadlock.

Situations involving many agents can often be viewed as a sequence of local interactions. For example, a lone agent that tries to go through a dense oncoming crowd of hundreds of agents may only consider 4–5 people at a time. Based on this observation if a steering algorithm is capable of these local interactions, it will very likely also be capable of handling larger scenarios.

It is fair to ask the question, what is the effect of the specific number of agents in the large scale examples? We have selected default values for these parameters based on what we think are average cases in the real world. We have no reason to believe that the specific number of agents in the large scale examples is crucial. If an algorithm can handle a bottleneck scenario with exactly 500 agents, it should be able to handle around 500 as well. In any case, our goal is to provide an estimate of an algorithms performance, not a proof of its robustness or correctness.

Customizing the Suite

It is clearly not possible to cover every conceivable situation in our test cases. Therefore, we have designed our benchmarking approach to be flexible and customizable, so that users can quickly focus on the details of interest to their algorithm.

The user can easily create custom scenarios to use with our existing evaluation tools. This allows our benchmark evaluation process to be useful even for applications that cannot use our provided test cases. For example, behaviors found in a sports game will have unique steering scenarios that should be evaluated with unique criteria. A user can easily use our file-format to describe such scenarios and how they should be evaluated.

In addition to creating custom scenarios, users can (and are encouraged to) change evaluation criteria of existing cases to suit their needs. While these modified criteria would not be part of the standard test case suite, they can be useful for personality-dependent steering or different types of agents, e.g., automobiles. In its

DOI: 10.1002/cav

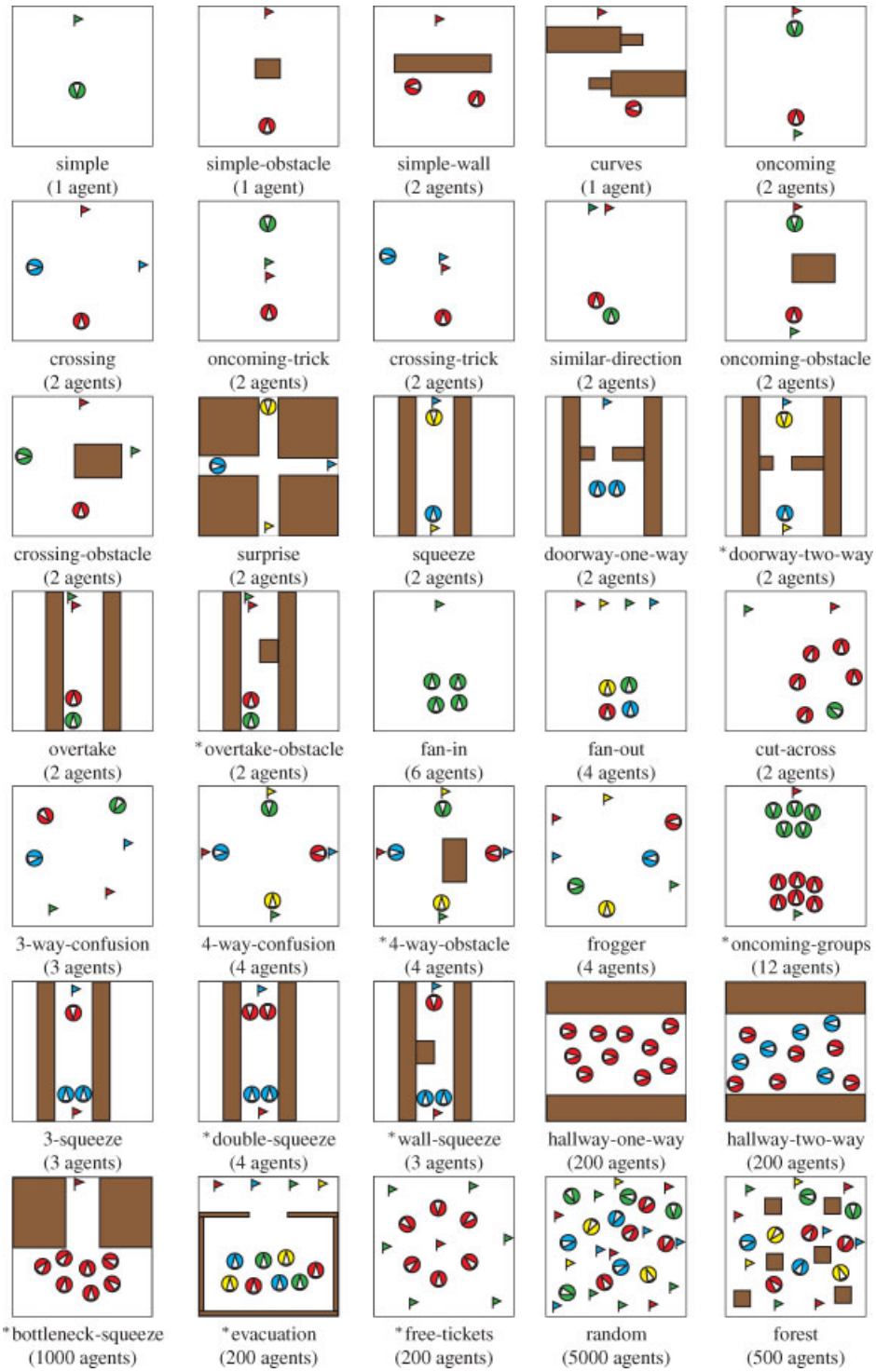


Figure 2. Visual depiction of the scenarios in the current version of our benchmark suite. The number in parentheses indicates how many agents are specified in the scenario. Scenarios annotated with an asterisk (*) are in our opinion more difficult scenarios. Some scenarios are not shown here.

Comp. Anim. Virtual Worlds 2009; **20**: 533–548
DOI: 10.1002/cav

current form, our test cases are intended to roughly approximate typical humans: agents have a diameter of 1 m (roughly the distance from elbow to elbow of an average human) and an average walking speed of 1.3 m/second.[24] The user can use the same (or slightly modified) initial conditions to simulate cars, bicyclists, or other steering agents, by customizing the evaluation criteria. Our evaluation tool will compute the statistics for the user to interpret appropriately. Examples of this are shown in results below.

Metrics of Evaluation

Given the suite of benchmarks, the next question is how to evaluate the result of a steering algorithm. First, at minimum, the algorithm must meet specific constraints defined by the test case—for example, in the *Overtake* scenario, the overtaking agent must reach the target before the slow agent. Second, assuming that an algorithm meets the minimum constraints for the test case, we can evaluate how efficiently the agents performed the task. To compute the success conditions and efficiency of a test run, SteerBench computes several metrics over the test run. This section describes these metrics.

The metrics are listed in Table 6. The three primary metrics are: (1) number of collisions, (2) time efficiency, and (3) effort efficiency. These metrics can be measured directly from the output of a steering algorithm, without any knowledge about the algorithm itself. We choose these primary metrics based on the widely accepted postulate (see Related Work above) that natural behaviors are usually very efficient, as long as they meet constraints appropriate to the task (e.g., no significant collisions with other agents). Optimizing time efficiency and effort efficiency are usually conflicting goals, and each test case defines the balance between these metrics.

Primary Metric 1: Number of Collisions

The first primary metric is the number of collisions that occur for a given agent. In most cases, fewer collisions indicate more realistic steering behavior. One notable exception is the *surprise* scenario, where it would be natural for two agents to collide because they do not see each other soon enough. It is worth noting that a user may choose to ignore incidental collisions, for example two agents brushing shoulders. Our detailed metrics described below allow such subtle customizations.

Primary Metric 2: Time Efficiency

Time efficiency measures how quickly the agent is able to reach its goal destinations. Of course, the quicker the agent reaches its goal, the more time efficient the agent is. Our evaluation tool measures time efficiency as the total time (in seconds) that an agent spends to reach its goal.

Primary Metric 3: Effort Efficiency

Effort efficiency measures how much effort an agent spends to reach its goal destinations. The less effort an agent spends, the more effort efficient the agent is. Our evaluation tool measures effort efficiency as total kinetic energy that an agent used to reach its goal.

Interpreting Efficiency Metrics

In most cases, optimizing time and effort efficiency are conflicting goals. In this way the combined interpretation of time and effort efficiency provides insight into a spectrum of behaviors. Some agents may desire to reach their destinations quickly, willing to spend more effort. Other agents may desire to save effort and slowly, politely progress towards their goals.

Detailed Metrics

The rest of the metrics are shown in Table 6. The sliding window in our current implementation is an interval of 3 seconds. We compute a new window for each frame, and finally store the max/min of these values. The collision metrics, max time spent in collision and max penetration can indicate whether the collision events were serious or not—for example, agents that brush past each other will have a very small penetration.

These detailed metrics are interesting to examine when a user knows the expected behavior of a scenario. A user will usually be able to identify one or two metrics that indicate unnatural behavior in the scenario. For example, if a character is expected to go straight towards its goal with very little turning, then the "total degrees turned," should be close to zero. Similarly, if an agent is expected to have exactly one abrupt turn in the scenario, then "max turning over a window interval" should be somewhat large, while the "average angular speed" should be small,

DOI: 10.1002/cav

	Metric	Interpretation
Primary metrics	# Unique collision events	If too large, can indicate poor behaviors
	Total time agent spent to reach goal (seconds)	Time efficiency (lower is better)
	Total kinetic energy spent (Kg (m/second)^2)	Effort efficiency (lower is better)
Collision metrics	Maximum time spent in a collision	If small, then collision events may be forgivable
	Maximum penetration of collisions (m)	
Turning metrics	Total degrees turned (deg)	Amount of turning effort
	Average angular speed (deg/second)	Time-independent amount of turning effort
	Maximum instantaneous angular speed (deg/second)	If small, indicates the agent turned very smoothly
	Maximum turning over window (deg)	Indicates whether the agent had to turn quickly at least once
	# Times angular speed changed sign	If too large, might indicate steering oscillations
Distance and Speed metrics	Total distance traveled (m)	Path length
	Average Speed (m/second)	Can compare to desired speed
	Maximum instantaneous speed (m/second)	If too large, can indicate unrealistic or cheating behaviors
	Maximum distance traveled over window (m)	Might indicate if the agent had to speed up temporarily
	Minimum distance traveled over window (m)	If small, indicates the agent had to stop or slow down significantly
Speed-change metrics	Total speed change (m/second)	Amount of movement effort, $\|\text{velocity}_1\| - \|\text{velocity}_2\|$
	Average speed change (m/second^2)	Time-independent amount of movement effort
	Maximum instantaneous speed change (m/second^2)	If small, indicates the agent moved very smoothly
	Maximum speed change over window (m/second)	If too large, might indicate unrealistic locomotion
	# Times speed change flipped sign	If too large, might indicate unrealistic oscillations in speed
Acceleration metrics	Total acceleration (m/second)	Second-order effort efficiency (lower is better)
	Average acceleration (m/second^2)	Time-independent effort
	Maximum instantaneous acceleration (m/second^2)	If small, indicates the agent moved very smoothly
	Maximum acceleration over window (m/second)	If large, might indicate unrealistic behavior

Table 6. SteerBench metrics and their intuitive interpretation. Metrics can certainly have more interpretations than given here

DOI: 10.1002/cav

and "max turning over a window interval" should be roughly equal to "total degrees turned."

Evaluating Constraints for a Test Case

As mentioned previously, each test case specifies constraints that an algorithm must satisfy. These constraints are not "success" criteria—it is not possible to define a complete set of criteria that define the success of an algorithm. Instead, the conditions are meant to identify obvious misbehaviors that an algorithm should not have.

So far we have found that the constraints required by a test case can be defined in terms of the above metrics. The first constraint is that the agent reaches its goal. Most of the smaller test cases require zero collisions, and most test cases also require that agents reach their goals within a certain amount of time. We can also detect other forms of misbehavior in this way, such as when an algorithm uses a huge instantaneous velocity, greatly exceeds the desired velocity, or has too many oscillations over a window interval.

Benchmark Scoring

After computing the metrics described in the previous section, the final task is to compute a meaningful score that represents the quality of the steering behavior. A score can be computed for (1) a single agent in a test case, (2) all agents in a test case, or (3) across all test cases. In this section we describe our method of computing such scores.

A Note About Benchmark Scores

It is important to note that scoring is not intended to be a proof of an algorithm's effectiveness. Even in widely accepted benchmarks such as SPEC CPU benchmarks, the scoring method does not attempt to avoid bias or exploits that can skew scores. Instead, the purpose of scoring is to create *a simple number* that allows for a quick, intuitive estimate of evaluation, especially when comparing two approaches. To do a more rigorous analysis of the pros and cons of an algorithm, users need to manually examine the detailed metrics, and perhaps even tailor their own test cases.

Scoring One Agent in One Test Case

An agent's score is computed by combining the three primary metrics described above: number of collisions, time efficiency, and effort efficiency. Each test case describes their relative importance as numerical weights, and each agent in the test case has its own unique set of weights. Note that even though these weights can be customized, each test case has specific default weights, and users are not required to tweak or tune any paremeters in SteerBench.

The score for a single agent is a weighted sum of the primary metrics: $S_i = w_{c,i}(\alpha C) + w_{t,i}(\beta T) + w_{e,i}(\gamma E) = C' + T' + E'$, where S_i is the score of the ith agent, $w_{c,i}$, $w_{t,i}$, and $w_{e,i}$ are the weights of the ith agent, C is the number of collisions, T is time efficiency, and E is effort efficiency. The constants α, β, and γ are used to transform C, T, and E into normal form—that is, these constants are used so that the three metrics are measured on the same scale. Determining appropriate values of α, β, and γ is a challenging problem that we leave for future work. For now, these constants are all 1.0, and we absorb this complexity in the manually chosen weights.

Scoring all Agents in One Test Case, and Across all Test Cases

To evaluate all agents in test case A, we compute the averages of the primary metrics over n agents, and then compute a weighted sum with an additional set of weights associated with the test case: $S_A = \frac{w_{c,A}}{n}\sum_{i=1}^{n} C' + \frac{w_{t,A}}{n}\sum_{i=1}^{n} T' + \frac{w_{e,A}}{n}\sum_{i=1}^{n} E'$. Because each agent can have its own set of weights, a user can easily control the relative importance of agents for the scoring process in their own test cases. By default in our test cases, all agents have equal priority. Finally, to score an algorithm across m test cases, we can compute a sum total of the scores from each test case, $S_m = \sum_A S_A$.

Example Evaluations

In this section we demonstrate the features of SteerBench by showing several examples of benchmark evaluation. Features include: (1) benchmark scores that concur with user opinions and correctly indicate when one test-run is better than another, (2) the ability of a user to tune test cases to favor certain types of behavior over others during scoring, (3) the ability of our framework

DOI: 10.1002/cav

to evaluate cars or other steering agents, not just pedestrians, (4) the usefulness of detailed metrics for examining an algorithm, and (5) the method to combine the scores across agents and across test cases.

Algorithms used for evaluation: We emphasize that the focus of our work is the *benchmark evaluation process*, not the resulting behaviors of the algorithms. Therefore, for the rest of this section we refer to algorithm A and algorithm B as the two steering algorithms that we compare. Algorithm A is a straightforward rule-based approach that integrates A-star planning, dynamic threat-prediction, basic obstacle avoidance, and space-time planning. We use this algorithm because it can be tweaked to perform intelligently, naively, or simply ignoring other dynamic agents. Algorithm B is loosely based on an implementation of potential-fields. Algorithm B has some of the classic pitfalls of potential-fields,[25] but is capable of some interesting behavioral functions that the rule-based approach cannot perform, such as the ability to overtake or cluster into groups.

Benchmark Scoring Example

Table 7 shows algorithms A and B being compared for 24 of our test cases. Algorithm A passed 20/24 test cases, failing the *Overtake*, *Oncoming-groups*, and both *Hallway* scenarios. Algorithm B passed 21/24 test cases, failing *Wall-squeeze* and both *Hallway* scenarios. It is still possible to examine primary and detailed metrics on the failed test cases. To compute scores, we used only two sets of weights, one set designed to weigh time efficiency more importantly, and another set designed to weigh effort efficiency more importantly. In most cases, we weighed it more important to minimize effort, but in some cases it is more favorable to finish sooner.

Test case	Score for A	Score for B	% users favored A (%)	% users favored B(%)
Simple	**273**	303	**100**	0
Simple-obstacle	**264**	268	**95**	5
Curves	**2527**	2687	**95**	5
Oncoming	**269**	283	**100**	0
Crossing	**266**	276	**84**	16
Oncoming-trick	132	**130**	42	**52**
Crossing-trick	117	**116**	11	**89**
Oncoming-obstacle	294	**285**	32	**68**
Crossing-obstacle	**254**	260	**84**	16
Surprise	**412**	420	**89**	11
Squeeze	**336**	341	**95**	5
Doorway-one-way	**2304**	2470	**89**	11
Doorway-two-way	**355**	375	**100**	0
Overtake	2741	**2423**	5	**95**
3-way-confusion	**291**	312	**79**	21
4-way-confusion	**272**	284	**95**	5
4-way-obstacle	**272**	328	**95**	5
Frogger	**242**	254	**74**	26
Oncoming-groups	640	**589**	16	**84**
3-squeeze	326	**319**	11	**89**
Double-squeeze	328	**309**	16	**84**
Wall-squeeze	**2596**	2935	**95**	5
Hallway-one-way	**9225**	10049	**89**	11
Hallway-two-way	**2660**	3813	**100**	0

Table 7. Comparison between two algorithms using SteerBench's default scoring process. Lower scores are better. Note that these scores are not meant to be interpreted alone, but only to be compared between algorithms for the same test case. SteerBench's scores concurred with users' opinions

DOI: 10.1002/cav

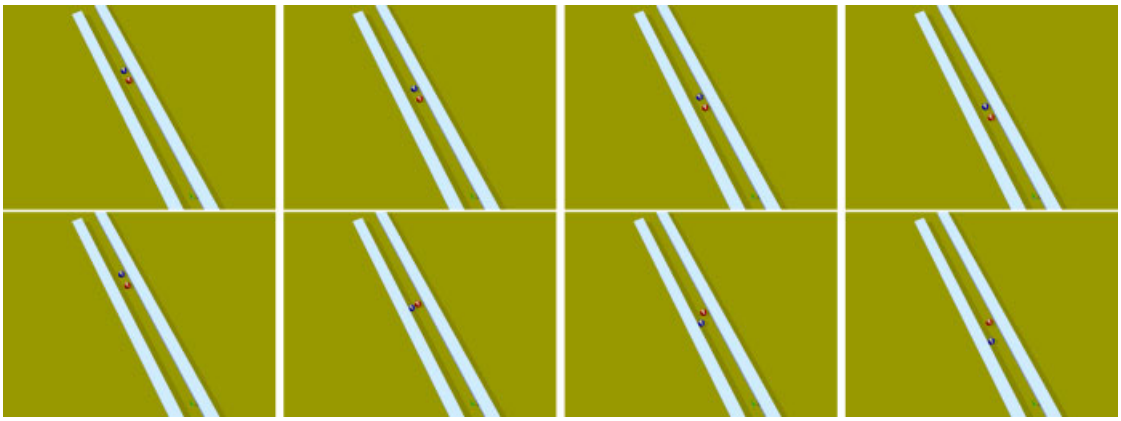


Figure 3. Snapshots of the Overtake scenario performed by algorithm A (top) and algorithm B (bottom). Algorithm A does not succeed in overtaking.

We conducted a brief user study of 19 subjects, also shown in Table 7. They were asked to give their opinion about which algorithm was more intelligent in each test case. For each test case, the subjects of the study did not know which algorithm was shown first or second, and they were not given any criteria related to our metrics for how to evaluate the agents.

To compare algorithms A and B, we can (1) count the number of pass/fails or (2) count the number of better scores. If we count only the number of pass/fails for each algorithm, it appears that algorithm B wins by a small margin. However, note that the constraints for "passing" a test case are meant only to capture obvious errors and cheats. On the other hand, if we count the number of better scores, algorithm A wins by a much larger margin, and concurs with user opinions. This is because the benchmark score can capture more information about which algorithm is more believable.

Customizating the Evaluation

Example 1: Favoring politeness versus aggressiveness: Figure 3 shows the two algorithms running the *Overtake* scenario. Algorithm A is "polite" and does not overtake, while algorithm B successfully overtakes. Table 8 shows the primary metrics and total score for the agent that is expected to overtake. The final scores are dependent on the choice of weights w_c, w_t, and w_e. For the primary metrics and final score, lower is better. This table shows that the scoring process is capable of favoring "polite" behavior or "aggressive" behavior, depending on the weights a user defines for a test case. Using the polite set of weights, algorithm A scores better, and using the aggressive set of weights, algorithm B scores better. By default, the *Overtake* scenario uses aggressive weights.

	Algorithm A	Algorithm B
# Collisions	0	1
Time efficiency	25.7	16.95
Effort efficiency	254.7	400.3
$w_{c,\text{polite}}$	50	50
$w_{t,\text{polite}}$	1	1
$w_{e,\text{polite}}$	1	1
Polite score	280	467
$w_{c,\text{aggressive}}$	50	50
$w_{t,\text{aggressive}}$	100	100
$w_{e,\text{aggressive}}$	1	1
Aggressive score	2829	2145

Table 8. Primary metrics, weights, and final weighted-sum scores of the *Overtake* scenario, computed with SteerBench. Lower scores are better; with polite weights algorithm A scores better, and with aggressive weights, algorithm B scores better. Note that polite and aggressive scores cannot be compared because they use different weights

Example 2: Ability to evaluate non-pedestrian situations: Table 9 shows our evaluation results for two variations of the *Frogger* scenario. In the first variation, all agents are pedestrians, and all pedestrians are expected to be able to avoid collisions without unnecessarily slowing down. In the second variation, one agent is a car, and the pedestrians must avoid crossing the car's path.

DOI: 10.1002/cav

Metric	Frogger-ped	Frogger-car
w_c	10	10
w_t	200	200
w_e	1	1
# Unique collision events	0	0
Time efficiency	11.40	12.05
Effort efficiency	31.62	58.42
Benchmark score	2311.6	2468.4
Maximum time spent in a collision	0	0
Maximum penetration of collisions (m)	0	0
Total degrees turned (deg)	40.81	4.14
Average angular speed (deg/second)	2.41	0.24
Maximum instantaneous angular speed (deg/second)	62.96	18.33
Maximum turning over window (deg)	25.30	4.12
# Times angular speed changed sign	1	0
Total distance traveled (m)	13.69	13.56
Average speed (m/second)	0.810	0.782
Maximum instantaneous speed (m/second)	1.30	1.30
maximum distance traveled over window (m)	3.90	3.90
minimum distance traveled over window (m)	3.00	1.92
Total speed change (m/second)	21.32	57.55
Average speed change (m/second^2)	1.26	3.32
Maximum instantaneous speed change (m/second^2)	3.74	9.60
Maximum speed change over window (m/second)	9.36	48.19
# Times speed change flipped sign	2	2
Total acceleration (m/second)	31.62	58.42
Average acceleration (m/second^2)	1.87	3.37
Maximum instantaneous acceleration (m/second^2)	3.76	9.60
Maximum acceleration over window (m/second)	14.36	48.95

Table 9. Detailed metrics collected for two variations of the *Frogger* scenario. They are effectively two different scenarios, so they cannot be directly compared, but examining the detailed metrics is insightful

Specifically, the pedestrians are expected to stop for the car without turning too much. These metrics are computed for the pedestrian immediately next to the car in the snapshot depicted in Figure 4. The reader should be aware that the overall scores between the two variations of *Frogger* are not meant to be compared. They are effectively two different scenarios—one using only pedestrians, and one using a car.

A tremendous amount of insight can be gained by examining the detailed metrics in Table 9. For example, note that for *Frogger-car*, the four statistics of (scalar) speed-change are nearly identical to the four statistics of (vector) acceleration. This implies that the agent's effort is almost entirely due to changes in speed, and not due to turning. It is also interesting to compare the total, average, instantaneous, and window forms of one metric. For example, in the *Frogger-car* scenario, notice that the maximum acceleration over a 3-second window (48.95 m/second) is close to the total acceleration (58.42 m/second)—this means there was one major event that caused most of the effort to be spent in less than 3 seconds.

Example 3: Evaluation of all agents in a test case: Figure 5 shows the *Oncoming-groups* scenario, using four different algorithms. Table 10 shows the corresponding scores, where all agents are averaged to form a score for the entire test case.

This example shows the ability of our evaluation to score an entire test case. As seen in Figure 5, algorithm B has a more intelligent way of handling the scenario, resulting in only one collision where two agents "brush shoulders". The intentionally dumb algorithm is almost

DOI: 10.1002/cav

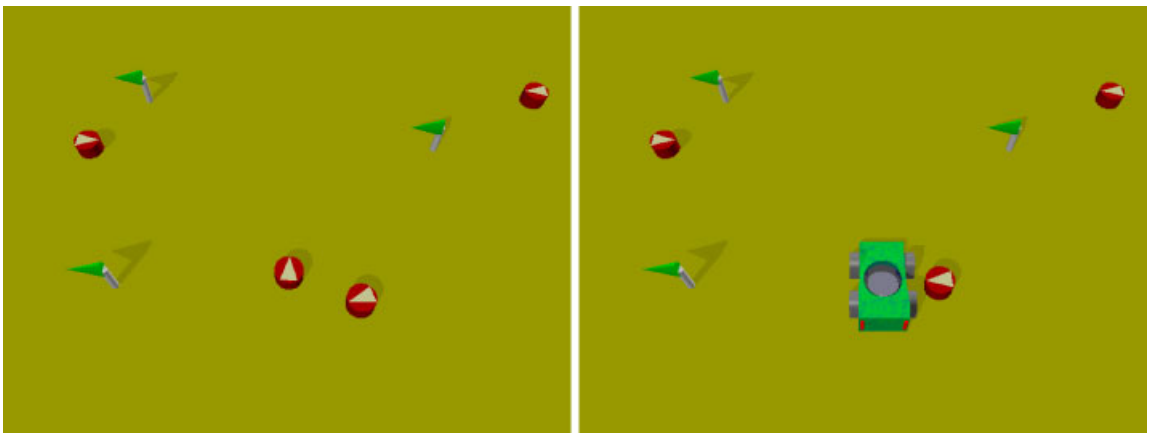


Figure 4. Variations of the Frogger scenario, showing the use of SteerBench for other types of agents. Left: the original scenario using only pedestrians. Pedestrians are expected to slow down only a little, and turn to avoid each other. Right: Frogger scenario converted into a car. Here, the pedestrians are expected to stop and wait for the car to pass, without turning. Car model by Christian Perle.

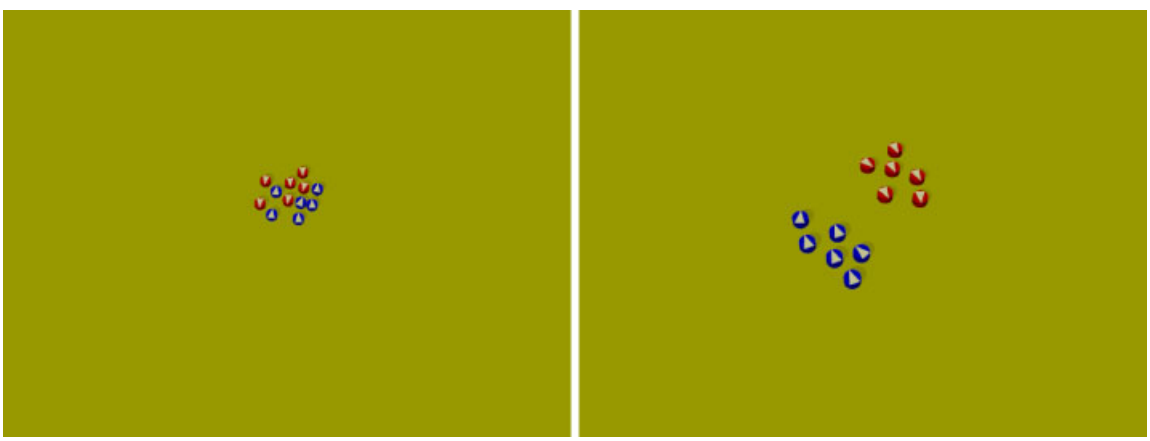


Figure 5. Two instances of the Oncoming-groups scenario. Left: algorithm A. The agents get jumbled, but still work efficiently. Right: algorithm B. The agents cluster into groups nicely.

Metric	Algorithm A	Algorithm B	Intentionally dumb	No collision avoidance
w_c	50	50	50	50
w_t	1	1	1	1
w_e	1	1	1	1
Average Collisions/agent	0.33	0.167	0.67	3.7
Time efficiency	39	44.6	40.6	36.3
Effort efficiency	601	545	600	611
Benchmark score	657	598	673	831

Table 10. Comparison of several algorithms executing on the *Oncoming-groups* test case. Here, the primary metrics and score are computed for the entire test case instead of an individual agent. Algorithm B wins, and algorithm A comes in second

DOI: 10.1002/cav

as fast as algorithm A, and therefore has a similar (but worse) score than algorithm A. Finally, without collision avoidance, agents are very fast, but SteerBench correctly penalizes the resulting number of collisions.

Discussion

The above examples illustrate the flexible relationship between time efficiency and effort efficiency. To improve time efficiency, an agent must put forth more effort to reach its goal sooner. To improve effort efficiency, an agent must try to reduce its total effort by steering the smoothest path possible. In most cases, an algorithm must compromise or prioritize between the two metrics, and it is up to the specific test case (or the user) to define which primary metrics take priority over others.

An algorithm cannot "cheat" in the default test cases by trying to maximize efficiency without actually accomplishing the goal, for the following reasons. First, SteerBench assigns infinity to time efficiency if the agent fails to reach its goal. Second, an agent cannot achieve a good score by ignoring all other agents and reaching its own goal regardless of collisions—such collisions will greatly penalize the agent's score, as shown in the example in Table 10. Finally, the pass/fail constraints defined by test cases ensure that the most obvious errors and cheats are avoided. Thus, the only way for an algorithm to outperform other algorithms using our benchmark is to behave better—quickly reaching its goal while minimizing effort and successfully avoiding all potential collisions.

Conclusion

This paper proposed SteerBench, a framework for evaluating steering behaviors. Our framework includes a diverse suite of test cases and an objective method of evaluation. Furthermore, it is blind to the specifics of the steering algorithm, and it is extensible and customizable. We hope that, with constructive feedback from the community, this framework can grow into a standard for evaluating steering algorithms.

ACKNOWLEDGEMENTS

The work in this paper was partially supported by NSF grant No. CCF-0429983. We thank Intel Corp., Microsoft Corp., ATI Corp., and AGEIA Corp. for their generous support.

References

1. Lext J, Assarsson U, Moller T. A benchmark for animated ray tracing. *IEEE Computer Graphics and Applications* 2001; **21**(2): 22–31.
2. Pelechano N, Stocker C, Allbeck J, Badler N. Being a part of the crowd: towards validating VR crowds using presence. In *Proceedings of Autonomous Agents and Multi-Agent Systems*, 2008; 136–142.
3. Shao W, Terzopoulos D. Autonomous pedestrians. In *SCA'05: Proceedings of the 2005 ACM SIGGRAPH/ Eurographics Symposium on Computer Animation*, 2005; 19–28.
4. Helbing D, Farkas I, Vicsek T. Simulating dynamical features of escape panic. *Nature* 2000; **407**: 487.
5. Reitsma PSA, Pollard NS. Evaluating motion graphs for character animation. *ACM Transactions on Graphics* 2007; **26**(4): 18.
6. Safonova A, Hodgins JK. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics* 2007; **26**(3): 106.
7. Witkin AP, Kass M. Spacetime constraints. In *SIGGRAPH'88: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, Beach RJ (ed.). ACM, New York, NY, 1988; 159–168.
8. Wu J-c, Popović Z. Realistic modeling of bird flight animations. *ACM Transactions on Graphics* 2003; **22**(3): 888–895.
9. Hodgins JK, Wooten WL, Brogan DC, O'Brien JF. Animating human athletics. In *SIGGRAPH'95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, 1995; 71–78.
10. Tu X, Terzopoulos D. Artificial fishes: physics, locomotion, perception, behavior. In *SIGGRAPH'94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, 1994; 43–50.
11. Brogan DC, Hodgins JK. Group behaviors for systems with significant dynamics. *Autonoumous Robots* 1997; **4**(1): 137–153.
12. Goldenstein S, Karavelas M, Metaxas D, Guibas L, Aaron E, Goswami AC. Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers and Graphics* 2001; **25**(6): 983–998.
13. Treuille A, Cooper S, Popović Z. Continuum crowds. In *SIGGRAPH'06: ACM SIGGRAPH 2006 Papers*, 2006; 1160–1168.
14. 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**(10): 509–518.
15. 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*, IEEE Computer Society, Los Alamitos, CA., 2003; 122.
16. Paris S, Pettre J, Donikian S. Pedestrian reactive navigation for crowd simulation: a predictive approach. In *EUROGRAPHICS 2007*, 2007; 665–674.
17. Reynolds C. Steering behaviors for autonomous characters. In *Game Developers Conference*, 1999.
18. Rudomín I, Millán E, Hernández B. Fragment shaders for agent animation using finite state machines. *Simulation Modelling Practice and Theory* 2005; **13**(8): 741–751.

DOI: 10.1002/cav

19. Pelechano N, Allbeck JM, Badler NI. Controlling individual agents in high-density crowd simulation. In *SCA'07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007; 99–108.
20. 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, New York, NY, 99–106.
21. 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; 139–147.
22. Lee KH, Choi MG, 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; 109–118.
23. Lerner A, Chrysanthou Y, Lischinski D. Crowds by example. *Computer Graphics Forum* 2007; **26**(3): 655–664.
24. Knoblauch RL, Pietrucha MT, Nitzburg M. Field studies of pedestrian walking speed and start-up time. *Transportation Research Record* 1996; **1538**: 27–38.
25. Koren Y, Borenstein J. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings, 1991 IEEE International Conference*, 1991; **2**: 1398–1404.

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.

Petros Faloutsos is an Assistant Professor at the Department of Computer Science at the University of California at Los Angeles. He received his Ph.D. (2002) and his M.Sc. degree in Computer Science from the University of Toronto, Canada and his B.E. degree in Electrical Engineering from the National Technical University of Athens, Greece.

Professor Faloutsos is the founder and the Director of the Graphics Lab in 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.

Glenn Reinman is an Associate Professor in the Department of Computer Science at UCLA. He received his B.S. from MIT in 1996 and his Ph.D. 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.

DOI: 10.1002/cav