An Open Framework for Developing, Evaluating, and Sharing Steering Algorithms

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

University of California, Los Angeles

Abstract. There are very few software frameworks for steering behaviors that are publicly available for developing, evaluating, and sharing steering algorithms. Furthermore, there is no widely accepted methodology for how to evaluate results of agent steering simulations. This situation makes it difficult to identify the real underlying challenges in agent simulations and future research directions to advance the state of the art. With the hope of encouraging community participation to address these issues, we have released *SteerSuite*, a flexible but easy-to-use set of tools, libraries, and test cases for steering behaviors. The software includes enhanced test cases, an improved version of SteerBench, a modular simulation engine, a novel steering algorithm, and more. Care has been taken to make SteerSuite practical and easy-to-use, yet flexible and forward-looking, to challenge researchers and developers to advance the state of the art in steering.

1 Introduction

Steering is an important aspect of behavioral animation that allows autonomous agents to navigate through an environment, and this topic has generated a large amount of research in the fields of robotics, graphics, artificial intelligence, and even sociology and psychology. One of the most time-consuming tasks required for anyone who wants to experiment with steering behaviors is developing the infrastructure surrounding the actual steering algorithm. This includes developing a simulation framework, designing scenarios to test the steering algorithm, deciding the method of evaluating the results, and devising a way to present results

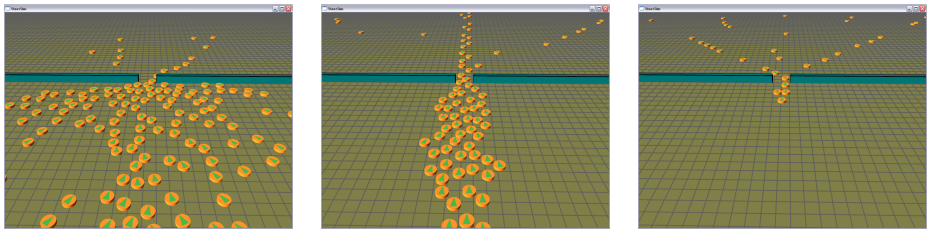

Fig. 1. Agents using the PPR algorithm to steer through the *bottleneck-evacuation* test case, shown here using SteerSim without the user-interface.

to others. Even in our own lines of steering research, we have come across significant practical hurdles, such as how to run tens of hundreds of simulations in a batch script, how to post-analyze them automatically, how to evaluate whether a steering algorithm is versatile and robust, and other related challenges. These tasks take weeks, even months of effort to do properly.

Over the past year, our research has accumulated into one such infrastructure. We proposed SteerBench [1], which explored the possibility of scoring the agents steering through a variety of challenging test cases. Recognizing that benchmark scores and detailed numerical metrics are not always enough, we also recently developed SteerBug [2], which uses pattern recognition techniques for recognizing user-specified behaviors of interest. Additionally, we have experimented with our own novel steering techniques: egocentric affordance fields [3] and the PPR algorithm (presented in this paper), and during the research process we developed a flexible simulation tool that provides common functionality to both algorithms.

We call the resulting framework *SteerSuite*. The source code and content are publicly available for download [4]. SteerSuite includes:

- Many diverse and challenging test cases with an open specification for creating more test cases,
- SteerBench, including several improvements,
- SteerSim, a simulation engine,
- The PPR steering algorithm,
- SteerLib, which includes functionality to make it easy to read the test case files, to record/replay agent simulations, and much more.

We chose to release this software for the following reasons:

- To make the implementation of our research works available for scrutiny and for use by others
- To propose a set of test cases as a starting point for the community to eventually create a standard suite of tests for steering algorithms
- To make it easy for users to start developing and testing their own steering experiments
- To make it easy to share results, in the form of benchmark/metrics reports and also in the form of recordings of the simulations.

To our knowledge, the only other openly available steering framework is OpenSteer [5] by Craig Reynolds. Our simulation engine, SteerSim, is inspired by the OpenSteerDemo component of Reynolds' software, however, beyond this similarity, both softwares are complementary to each other. OpenSteer provides a library of functions used for steering decisions of agents, including path following, seek, flee, and boids behaviors, and helper functions to determine the agent's state, while SteerSuite provides functionality related to the testing, evaluating, recording, and infrastructure surrounding a steering algorithm.

This paper discusses the novel aspects of SteerSuite. We first discuss the improvements made since the original SteerBench: improvements to the set of

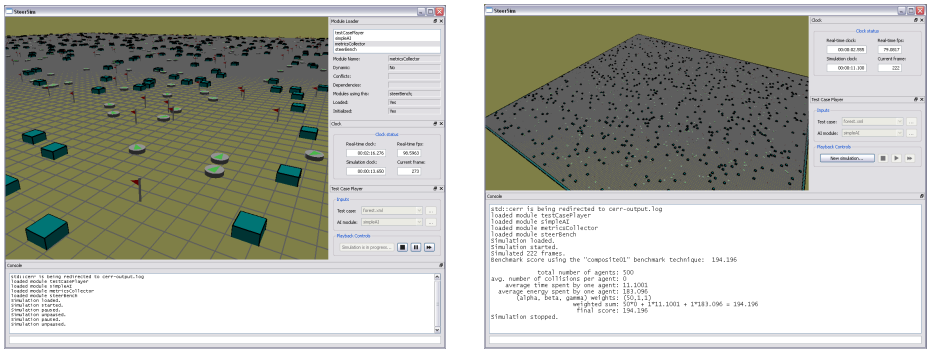

Fig. 2. Screenshots of the SteerSim user interface, benchmarking a simple steering algorithm on the *forest* test case.

test cases are discussed in Section 2, and the generalizations made to benchmarking are discussed in Section 3. Then, Section 4 describes the example steering algorithm provided with SteerSuite, called PPR (Plan, Predict, and React). The development and testing of this algorithm illustrates the various features of SteerSuite. We discuss debugging and evaluation in Section 5, and Section 6 concludes.

2 Improvements to Test Case Specifications

In the context of SteerSuite and SteerBench, a *test case* is a description of the initial conditions of agents and objects in an environment. The original set of test cases is described in [1], which focuses on testing normal everyday pedestrian behaviors. We hope to encourage the community to contribute more test cases for different application domains, which can eventually evolve into a widely accepted set of tests that steering algorithms are expected to use.

To this end, the test case format is designed to be flexible but easy to use. For portability, we migrated the test case format into XML. SteerSuite provides an XML Schema that describes the specifics of the format, so that users can easily create and validate their own test cases. We added the ability for test cases to specify "regions of agents" and "regions of obstacles," where the agents or obstacles are randomly placed; this feature was previously hard-coded into only a few test cases. SteerSuite also provides a library that can read these test cases and automatically set up all initial conditions, deterministically resolving all random regions and random targets before giving the initial conditions to the user.

We also added more elaborate goal specifications in the test cases. An agent's goal can be one or a combination of the following types:

– **Seek static location:** the agent should navigate towards a fixed location in space.

- **Flee static location:** the agent should move away from a fixed location in space. For example, agents should flee from a stationary vehicle that is on fire.
- **Seek dynamic target:** the agent should steer towards a moving target. The two common examples of this are (1) pursuit, where one agent chases another agent, and (2) meeting, where two friends want to steer towards each other.
- **Flee dynamic target:** the agent should flee a moving target, for example, when being chased.
- **Flow in a fixed direction:** the agent should progress in a particular direction, for example, when going down a hallway with no need to follow an exact planned path.
- **Flow in a dynamic direction:** the agent should follow a dynamically changing direction, which can be used for agents to advect along a potential field or velocity field, or to wander with a randomly changing direction.

Each type of goal takes additional data, such as the point location, named target, or direction vector. This additional data can optionally be declared as "random". In future work we may add support for more controlled randomness, allowing the user to specify sampling distributions and regions.

This goal specification deserves mention because it addresses an important practical consideration: a steering algorithm is only one of many components of an autonomous virtual character, and eventually it will be necessary to interface the steering algorithm with a high-level intelligent controller (i.e. the artificial intelligence of the agent). The described goal specification is our first attempt to characterize all possible ways that artificial intelligence may want to interface with steering. If the goal specification receives positive feedback, we will develop more test cases that use these new goal types, with an emphasis on normal everyday steering tasks where real human pedestrians may think in terms of these goal types instead of steering to a static target.

3 Improvements to Benchmarking

The original SteerBench benchmark process was a monolithic process that collected numerous metrics of an agent simulation and then computed a weighted sum of three primary metrics for the final benchmark score. For more information about these metrics, refer to the original SteerBench work [1]. In SteerSuite, we have generalized this process by separating the concepts of metrics collection and benchmark techniques. This separation allows users complete flexibility; users can easily experiment with their own benchmark techniques regardless of what metrics are used, or they can focus on using metrics to debug or analyze a simulation.

The metrics are updated once per frame, and can be accessed and examined by the user for any agent at any frame of the simulation. The desired benchmark technique is also updated once per frame, given access to the metrics of all agents and of the environment. The benchmark technique then provides functionality

to (1) get a simple score or (2) to output details of how the score was computed, (a) for all agents in a test case or (b) for an individual agent. Benchmarking can be done on-line while running the steering algorithm, on-line with a recording of the simulation being replayed, or off-line with a command-line tool.

SteerSuite provides several basic benchmark techniques. The original SteerBench work, which used a weighted sum of three primary metrics, is called the `composite01` benchmark technique. We also developed a `composite02` technique which uses four primary metrics: the first three metrics are the original three metrics from `composite01`: (1) number of collisions, (2) time efficiency measured as seconds to reach goal, and (3) sum total of kinetic energy samples along the path (which was called effort efficiency). For `composite02`, we add (4) a sum total of acceleration samples along the path, another measure of effort efficiency.

4 PPR: The SteerSuite Steering Algorithm

In this section we describe the PPR (Plan, Predict, and React) algorithm, which is currently the main example algorithm provided with SteerSuite. As the algorithm is described, it illustrates how various features of SteerSuite can be used.

The algorithm is implemented as a plugin to SteerSim, the simulation engine that is part of SteerSuite. SteerSim has a modular architecture, so that almost all useful functionality is provided in modules. Modules have access to most of the simulation engine's data, including a spatial database, clock, camera, access to other modules. Modules can even add components to the graphical user interface. When a simulation is started, the engine uses the PPR module to create and initialize each agent, providing each agent its initial conditions (including a sequence of goals) that came from a test case. As the simulation runs, the engine automatically updates every agent. Modules have the opportunity to perform preprocessing and postprocessing at every frame, which is useful for metrics collection or for a steering algorithm that requires a global processing stage, but the PPR steering algorithm does not use this feature.

The PPR algorithm is a novel rule-based pedestrian steering algorithm that combines three (potentially conflicting) aspects of human steering into a single steering decision. The three aspects are:

- Plan: The agent selects a local target that is used to smoothly steer along the planned path.
- Predict: The agent makes predictions about other agents and determines how to steer to avoid the most imminent predicted collision.
- React: The agent steers to avoid problems interacting with other agents in its immediate surroundings.

All three aspects produce a steering decision, and the challenge of this approach is how to combine these steering decisions, or at least how to choose which steering decision to use at any given time. We address this by using a state machine and a set of rules. The implementation is divided into six main phases, described below. (There are actually more phases and states of the agent that are part of

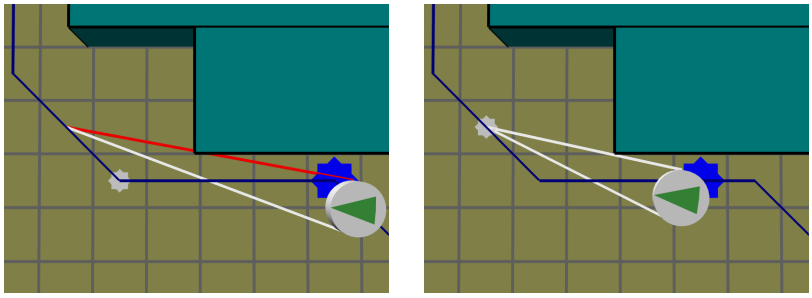

Fig. 3. Short-term planning. The local target (white star) is chosen as the furthest point such that all path nodes between the agent's closest path node (blue star) and the local target have line-of-sight to the agent.

future research.)

Long-term planning phase. Given a goal target, the agent plans a path to its goal using the standard A-star algorithm [6], only planning around static obstacles in the scenario. The graph used by A-star is a rectangular grid where each node is connected to its eight neighbors. This type of graph is chosen because the spatial database provided by SteerSuite is a grid, and it allows traversal costs to be associated with each grid cell. A grid-based graph can result in costly A-star searches, but this choice avoids the need for users to manually create A-star graphs, and the amortized cost of path planning remains very low.

Short-term planning phase. Given the planned path, the agent chooses a local target along the path to steer towards. This local target is chosen as the furthest point along the path such that all path nodes between the agent and the local target are visible to the agent (Figure 3). This criterion smooths the agent's path while enforcing the agent to follow the path correctly around obstacles. This, combined with path planning described above, is all the agent needs to steer correctly around fixed obstacles.

To implement the short-term plan requires visibility testing, which can be done with ray tracing. The spatial database in SteerSuite provides fast ray tracing support for this purpose; the grid data structure allows us to greatly reduce the number of objects that need to be tested for ray-intersection. Users can add support for arbitrary objects in the spatial database by implementing a few geometric helper functions, including a ray-intersection routine.

Perception phase. To perform natural predictions and reactions, it is important to model what the agent actually sees. We model an agent's visual field as a 10 meter hemisphere centered around the agent's forward facing direction. The SteerSuite spatial database makes it possible to perform range queries in the spatial database to collect a list of these objects. Furthermore, objects that

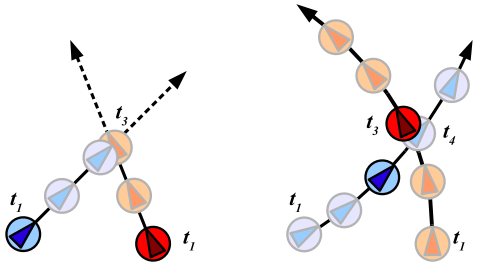

Fig. 4. Space-time prediction. Left: agents predict a collision, knowing their trajectories will overlap at the same time, t_3. Right: agents steer to avoid the collision. Note that space-time prediction correctly avoids a false prediction between the blue agent at t_4 and the red agent at t_3, because they reach that point at different times.

do not have line-of-sight are not added to the list of objects the agent sees.

Prediction phase. The agent predicts possible collisions, only with agents in its visual field, using a linear space-time predictor based on [7]. Given an agent's position, P, velocity V, and radius r, our linear predictor estimates the agent's position at time t as

$$\text{Agent's future position} = P + t \cdot V. \tag{1}$$

A collision between agent a and agent b would occur at time t if the distance between their predicted positions becomes less than the sum of their radii:

$$\|(P_a + t \cdot V_a) - (P_b + t \cdot V_b)\| < r_a + r_b. \tag{2}$$

Solving this expression for time t results in a quadratic equation. The agents collide only if there are two real roots, and these two roots represent the exact time interval of the expected collision.

If collisions are predicted, they are handled similar to [8]. Each predicted threat is classified as one of three possible types: oncoming, crossing, or similar direction collisions. For oncoming threats, agents will both choose to steer to the right, or to the left, depending on which side is closer. For crossing threats, the agents first determine who would reach the intersection first. The agent that will reach the collision first will decide to speed up and turn slightly outward, and the agent that will reach the collision later will slow down and turn slightly inward (Figure 4). This behavior is very subtle, but the difference between using these predictions or disabling the predictions is very clear.

The prediction phase also updates a state machine that decides how to steer. SteerSuite provides a useful state machine helper object for this purpose. The possible states and corresponding steering actions are described in Figure 5. In most cases, if an agent needs to react to something more immediate, it will override the predictive steering decision.

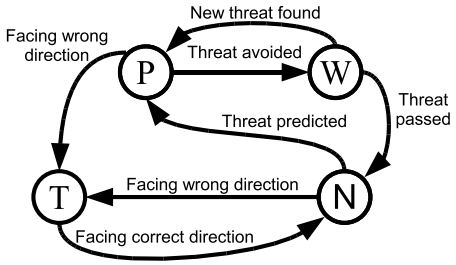

Fig. 5. State machine used to integrate plan, prediction, and reaction behaviors. The prediction updates the state machine, and the reaction phase uses the current state to help choose the final steering decision. The agent steers normally in state N, proactively avoids threats in state P, waits for avoided threats to pass in state W, and re-orients itself towards the short-term-planned target in state T. These behaviors may be overridden by reactions.

Reaction phase. The reaction phase implements both reactive steering and the rules to decide which steering decision to use, and outputs a steering command to the locomotion phase. The agent traces three forward-facing rays, 1 meter to the front of the agent, and 0.1 meters to the side. If these rays intersect anything, the agent may need to react, possibly overriding steering decisions made by prediction. When reacting, the agent takes into account the relative location and orientation of the new obstructions. This results in a very long list of rules that account for all possible configurations: there can be up to three perceived obstructions (one per forward-facing ray), each obstruction may be an agent or an obstacle, and any obstructing agents can be classified as oncoming, crossing, or facing the same direction. For efficiency, the rules are implemented using a hierarchy of conditions instead of checking each rule one after the next. This way, identifying any rule requires only a (informally) logarithmic number of conditional branches instead of linear. The top level of the decision hierarchy is illustrated in Figure 6. Once the exact rule has been identified, the agent outputs a "steering command" to the locomotion phase. As with the previous phases, this phase benefits from SteerSuite's ray tracing and geometry helper functionality.

Locomotion phase. The locomotion phase receives an abstract steering command that tells an agent to turn, accelerate, aim for target speed, and/or scoot left or right. This command is converted into a force and angular velocity that moves the agent's position and orientation using simple forward Euler integration. Note that, even though we use dynamics to move the agent, the locomotion phase constrains the output to model what pedestrians could realistically do. Most likely, in a future version of SteerSuite, we will generalize the locomotion

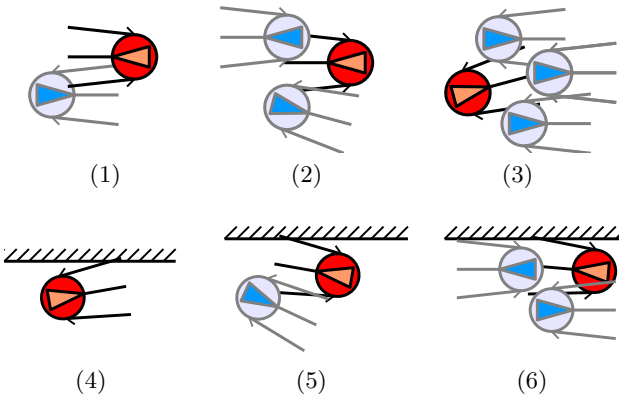

Fig. 6. Main cases of the reaction phase, determined by three short forward-facing rays traced for each agent: (1) one agent, (2) two agents, (3) three agents, (4) obstacles only, (5) one agent and obstacles, (6) two agents and an obstacle. Each case has many sub-cases depending on the position, orientation, and state of the other agents and obstacles.

phase into a few SteerSuite helper functions that will be available to any steering algorithm.

5 Debugging and Evaluating an Algorithm

The process of debugging and evaluating an algorithm is where SteerSuite features really become useful. Like OpenSteer, the simulation engine allows the user to visualize the simulation, draw annotations, and step through the simulation manually. This is already very useful, allowing users to annotate what the agents are thinking. Here we describe additional debugging and evaluation features of SteerSuite.

Test cases. The test cases have proven to be the most important aspect of debugging and evaluating, for us as well as other users of SteerSuite. With these test cases, it is possible to test the full spectrum of expected steering behaviors for a given application. The test cases are also crucial for presenting results; when it is impractical to demonstrate the wide range of results in a presentation or paper, instead it is possible to summarize the results of an algorithm based on existing test cases.

Simulation recordings. SteerSuite further provides the ability to record simulations, using SteerLib in the user's code or using SteerSim directly. We have found this feature to be invaluable. It is often easier to visualize recordings of large simulations in real-time, instead of watching a slow simulation while it is running. At the same time, recordings retain more flexibility than a pre-recorded

movie, allowing the user to move and zoom the camera while interactively visualizing the replay. In combination with command-line tools to perform batches of simulations and batches of benchmark analysis, we sometimes record hundreds of simulation experiments in mere minutes, examine the data, and then narrow down which simulations to inspect visually and to re-simulate. SteerSim can also run on an architecture simulator, which has only a command-line interface, and recordings allows us to later verify that the command-line simulation worked properly. Eventually we hope that these recordings can become a common way to report results, with two main benefits: (1) users easily can provide a large amount of results for others to see, and (2) users can benchmark other people's simulations with their own downloaded version of SteerSuite, so they can trust that the metrics and scores were not altered.

Benchmark scores. The ability to simplify an algorithm's performance into a single number has been useful when we try to compare a large number of simulations. While there is very little information in the single number, it still helps narrow down which simulations should receive closer attention – for example, while debugging PPR, we searched for the simulation with the "worst" benchmark score and then examined that simulation. One limitation is that some benchmark techniques cannot be used to compare scores across test cases. After becoming familiar with the typical range of scores for each test case, this limitation is not very significant, and other benchmark techniques do not have this limitation in the first place.

Detailed metrics. There are several positive experiences we had with detailed metrics (refer to the original SteerBench paper [1] to see a list of these metrics) while developing the PPR and egocentric affordance fields algorithms. For example:

- On several occasions during development, we caught instances where agents were oscillating unexpectedly, according to the "number of times the angular velocity changed sign" and "number of times acceleration flipped direction" metrics, but the oscillations were barely visible in the simulation. It turned out these oscillations were a result of the agent repeatedly switching between two steering decisions, and without the metrics we would not have caught these errors.
- At one point, in some scenarios such as bottleneck-squeeze, we saw a surprising number of collisions, which did not seem true from the visualization. Upon closer examination, it turned out that the agents were "scooting" (side-to-side motion) into the wall as they were walking along the wall. This was technically not an oscillation of turning or of velocity, so it was useful that we verified some obvious metrics to find this error.

Finally, we encourage interested readers to download the SteerSuite software and explore the benefits and limitations of the PPR algorithm by seeing how it performs on the test cases. The PPR algorithm is not intended to robustly solve all test cases, but rather to be a starting point for users to start using

SteerSuite. There is also a large and growing number of crowd simulation and agent steering approaches in existing literature (e.g [7–23] are just a few); we encourage developers to port or implement any existing steering techniques that they find interesting, reporting their experiences with these algorithms in the form of recordings and benchmark results of our test cases.

6 Conclusion and Future Work

In this paper we described the novel aspects of SteerSuite, a publicly available software framework for developing, testing, and sharing steering algorithms. The ideas in SteerSuite are just one proposed way to answer many interesting questions: how can we evaluate steering behaviors? What types of tests should we require every steering algorithm to pass? How should a steering algorithm be interfaced with higher-level artificial intelligence in a practical application? We hope that SteerSuite will generate interest in these questions within the community.

A major goal of SteerSuite is to make infrastructure tasks very easy for users. If such tools become widely accepted, whether it is SteerSuite or some other future software, the research community will be able to communicate and share results more easily, thus promoting more rigorous evaluation of steering algorithms, and ultimately helping to push forward the state of the art in steering.

We also discussed the PPR steering algorithm, which demonstrates a versatile set of behaviors, and along the way it also showcases how various parts of SteerSuite can be used. In the future we plan to integrate the egocentric affordance fields algorithm into SteerSuite as another example steering algorithm, and we plan to migrate the SteerBug framework [2] into SteerSuite as well.

Acknowledgements

This work was partially supported by the NSF grant CCF-0429983. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF. Parts of the user interface were contributed by Brian Hou and Tingyu Thomas Lin. We would like to thank Craig Reynolds and Ronan Boulic for interesting discussions and suggestions. We would also like to thank Intel Corp. and Microsoft Corp. for their generous support through equipment and software grants.

References

1. Singh, S., Kapadia, M., Naik, M., Reinman, G., Faloutsos, P.: SteerBench: A Steering Framework for Evaluating Steering Behaviors. Computer Animation and Virtual Worlds (2009)
2. Kapadia, M., Singh, S., Allen, B., Reinman, G., Faloutsos, P.: SteerBug: An Interactive Framework for Specifying and Detecting Steering Behaviors. In: ACM Siggraph/Eurographics Symposium on Computer Animation (SCA). (2009)

3. 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). (2009) 215–223
4. Singh, S., Kapadia, M., Reinman, G., Faloutsos, P.: Steersuite. http://www.magix.ucla.edu/steersuite/
5. Reynolds, C. http://opensteer.sourceforge.net/
6. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on **4**(2) (July 1968) 100–107
7. Paris, S., Pettre, J., Donikian, S.: Pedestrian reactive navigation for crowd simulation: a predictive approach. In: EUROGRAPHICS 2007. (2007) 665–674
8. Shao, W., Terzopoulos, D.: Autonomous pedestrians. In: SCA '05: Proc. of the 2005 ACM SIGGRAPH/Eurographics symp. on Computer animation. (2005) 19–28
9. Brogan, D.C., Hodgins, J.K.: Group behaviors for systems with significant dynamics. Auton. Robots **4**(1) (1997) 137–153
10. Goldenstein, S., et al.: Scalable nonlinear dynamical systems for agent steering and crowd simulation. Computers and Graphics **25**(6) (2001) 983–998
11. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. In: SIGGRAPH '06: ACM SIGGRAPH 2006 Papers. (2006) 1160–1168
12. Helbing, D., Farkas, I., Vicsek, T.: Simulating dynamical features of escape panic. Nature **407** (2000) 487
13. 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)
14. 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 (2003) 122
15. Reynolds, C.: Steering behaviors for autonomous characters. In: Game Developers Conference. (1999)
16. 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
17. 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) 99–108
18. Boulic, R.: Relaxed steering towards oriented region goals. Lecture Notes in Computer Science 5277, MIG 2008 (2008) 176–187
19. Metoyer, R.A., Hodgins, J.K.: Reactive pedestrian path following from examples. The Visual Computer **20**(10) (November 2004) 635–649
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 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, 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) 109–118
23. Lerner, A., Chrysanthou, Y., Lischinski, D.: Crowds by example. Computer Graphics Forum **26**(3) (September 2007) 655–664