TITLE: EGOCENTRIC AFFORDANCE FIELDS IN PEDESTRIAN STEERING

ABSTRACT:

Research in the area of pedestrian simulation has seen a dramatic rise in recent years. With the potential for this work being realized in a wide variety of areas, ranging from urban planning and training simulations to games, life-like steering motions for each individual have become critical for a truly immersive and realistic experience. In this paper we propose a general framework for local path-planning and steering that can be easily extended to perform high-level behaviors. Our framework is based on the concept of affordances: the possible ways an agent can interact with its environment.

EGOCENTRIC AFFORDANCE FIELDS:

An important feature of realistic steering is missing in traditional approaches: humans constantly compute a local short-term space-time plan to steer through their immediate environment, which includes dynamic objects and other agents. This short-term plan is essential for natural steering in crowded environments, as well as for resolving deadlock situations.

This paper presents a novel technique that bridges the gap between steering and planning by using information fields in an egocentric fashion, where the origin of the field is centered about the agent at all times. This allows us to implicitly account for time when planning, removes the storage and computation problems of global field-based approaches, and achieves linear scalability when parallelized.

Our approach is based on the concept of affordances -- the ways in which an agent can interact with its environment. We define fitness to be a measure of how appropriate the associated affordance (associated action) will be. An affordance field is a scalar field that has a fitness associated with every affordance in the space of all possible actions. A final decision is the affordance associated with the optimal fitness.

Discrete Egocentric Fields

We implement a discretization of the egocentric fields as a connectionist architecture that uses nodes arranged in concentric circles to maintain egocentric spatial
information. At all times, the central node (root) represents the current position of the agent. Each node perceives information corresponding to its “spatial awareness” in the environment. Our method allows us to take advantage of variable resolution discretization, where the accuracy of information is higher near the root and decreases further from the origin. The information storage per unit area is dense close to the origin, and density decreases further from the root. In combination with the egocentric property, variable resolution fields allow us to avoid costly computations where data is far away, both temporally and spatially. This is appropriate in the context of agent navigation, since the immediate surroundings are often more important for making navigation decisions. Figure 1 visualizes the structure of the discrete egocentric fields.

*Figure 1: Visualization of variable-resolution discrete egocentric fields*

*Figure 2: Data flow diagram for steering using egocentric fields*
Applying Discrete Egocentric Fields to Steering

Figure 2 describes an overview of our steering framework, and Figure 3 illustrates the working of the steering algorithm using egocentric fields. The first phase of our method is to gather and interpret sensory information of the environment, which is stored using egocentric perception fields. Our steering framework uses the following egocentric perception fields:

- **Static Field**: The static field, $P_{\text{static}}$, represents the configuration of the obstacles in the environment surrounding the agent.

- **Dynamic Field**: The dynamic field, $P_{\text{dynamic}}$, represents the configuration of all dynamic objects by providing the predicted positions of neighboring agents at various points of time.

- **Velocity Field**: The velocity field, $P_{\text{velocity}}$, is a vector field that provides the direction and speed magnitude of neighboring agents.

- **Traversability Field**: $P_{\text{traversability}}$ is a linear combination of $P_{\text{static}}$ and $P_{\text{dynamic}}$.

- **Local Dynamic Field**: The dynamic field is subject to a kernel function that considers regions in which the dynamic threats are most imminent. The resulting fields are known as local dynamic fields, denoted by $P_{\text{local-dynamic}}$.

- **Relative Velocity Field**: $P_{\text{relative-velocity}}$ provides the relative velocity of neighboring agents with respect to the agent’s velocity.
Figure 3: The Steering Algorithm. (a) The current state of the environment. (b) Static Perception Field indicating low traversability at position of obstacle. (c) Dynamic Perception Field for initial speed. (d) Dynamic Perception Field for new speed, which avoids the dynamic obstacle. (e) Resulting affordances indicating a path of high fitness to goal.

Implicit Space-Time Planning

Previous approaches implement space-time planning by representing time as an additional planning dimension in the state space. This incurs a considerable processing overhead which becomes intractable in large crowd simulations. Our approach naturally supports efficient space-time planning using egocentric fields. Due to the egocentric nature of our data representation, we can represent time implicitly as the distance from the origin to a point of interest. Hence, a mapping exists between time and a particular layer of an agent’s field. $P_{\text{dynamic}}$ and $P_{\text{velocity}}$ are computed by considering this time-layer associativity and the neighbors surrounding the agent. If the difference in the time taken by the agent and its neighbor to travel a particular distance is below a threshold, then a dynamic threat is predicted at that instance of space and time. $P_{\text{dynamic}}$ reflects the potential space-time threats for the agent, and $P_{\text{velocity}}$ stores the current velocity of
the potential threats. This facilitates the choice of steering decisions that predictively avoid collisions in the future, thus preventing deadlock situations.

Once the perception fields are populated, the affordance fields are computed to provide a fitness value to each possible speed and direction. The speed affordance fields \( A_{\text{speed}}(s) \) provide the relative fitness of each speed affordance \( s \), defined as the distance of the most imminent threat for that value of \( s \).

Direction Affordance Fields, \( A_{\text{direction}}(\theta) \), quantify the relative strengths of all possible directions in which an agent can steer. A pedestrian in a crowd bases its direction of travel on the presence of static objects in the environment as well as other pedestrians. In order to compute the direction affordance field, we first compute the spatial affordance field, which provides the fitness value for all points in the spatial domain. The fitness values of points immediately surrounding the agent are used as values of direction affordance.

Once the speed and direction affordance fields are computed, the final step is to select the speed and direction having optimal fitness. The target speed \( s_{\text{target}} \) maximizes \( A_{\text{speed}}(s) \), i.e., it maximizes the distance from all imminent threats. The target direction \( \theta_{\text{target}} \) is chosen to maximize \( A_{\text{direction}} \).

**Results**

We test our steering algorithm on a wide variety of scenarios ranging from simple sanity cases, oncoming agents, crossing agents, group interactions, and narrow passageways, to large scale cases involving hundreds and thousands of agents. We observe that agents exhibit natural local agent interactions and human-like behaviors.

The importance of space-time planning is demonstrated in test cases involving complicated interactions between three or more agents with a potential for a deadlock situation. These include doorway, overtake, confusion, and squeeze (narrow passage) scenarios. The natural, anticipatory behaviors where agents *predictively* move out of the way to allow other agents’ passage, thus preventing deadlocks, is a result of implicit space-time planning. In contrast, we observe that purely reactive approaches suffer from agents colliding with one another, or fail to arrive at a solution, resulting in deadlocks.

We also demonstrate several bottleneck and densely crowded scenarios where agents cooperatively wait or steer around each other. Note that many previous approaches steer unnaturally into each other and into obstacles, relying on collision resolution and “greedy” reactive steering decisions to progress the agents. Figures 4, 5, and, 6 demonstrate our algorithm on a wide variety of test cases. A video demonstration of our results can be found [here](#).
Figure 4: (a) Agents walking through a hallway. (b) Queue formation as agents enter narrow passageway. (c) Simulation of a forest-like scenario with a large number of agents and obstacles. (d) 5000-agent simulation with random positions and goals.

Figure 5: Two agents traveling in the same direction encounter two oncoming agents in a narrow passageway.

Figure 6: An agent overtakes another agent while traveling through a narrow passageway.

Parallelization

In this section, we provide the performance results of the parallelized version of our algorithm. The experimental setup used for the simulation experiments is described below:
Hardware Specification: Intel® Xeon® Processor 7030 processor-based platform (with 2 dual-core processors, each with Intel® Hyper-Threading Technology)
Operating System: Red Hat® Linux® 4.1.2
Programming Language: C++ and OpenMP®
Compiler Specification: GCC 4.1.2

Table 1 outlines the profiling results for task decomposition of our approach. We observe that the function block which populates the direction affordance field takes up approximately 71% of the total execution. The recursive nature of this procedure makes it an unlikely candidate for parallelization. Hence, we turn our focus to decomposing the problem based on agents.

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time (ms)</th>
<th>Execution profile</th>
<th>Need for Synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Grid Database</td>
<td>0.0136</td>
<td>4%</td>
<td>No</td>
</tr>
<tr>
<td>Populate dynamic perception field</td>
<td>0.0142</td>
<td>7%</td>
<td>Yes</td>
</tr>
<tr>
<td>Compute speed affordance field</td>
<td>0.0211</td>
<td>9%</td>
<td>No</td>
</tr>
<tr>
<td>Compute direction affordance field</td>
<td>0.1681</td>
<td>71%</td>
<td>No</td>
</tr>
<tr>
<td>Total</td>
<td>0.2337</td>
<td>100%</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1: Task Decomposition of Steering Algorithm

In our approach, agents read each other’s data in order to reach and predict around each other while steering. This inter-agent communication prevents simply splitting agents and farming them off to different threads. Hence, our algorithm requires synchronization at the agent level for parallelization. We implement synchronization between agents using a simple read-write locking scheme with multiple readers and a single writer. Each agent has a lock associated with it. Whenever an agent wishes to write to its internal data structures, it yields for any neighboring agents that are currently reading from it. Similarly, future readers yield to agents that are currently updating their state. We observe that a deferred-waiting strategy performs consistently better than busy-waiting with increase in number of threads. Also guided scheduling outperforms other scheduling strategies. This is because the guided scheduling strategy allocates spatially co-located agents on the same thread. Hence, two agents that read each other’s data (because they are spatially co-located) will have fewer synchronization problems because they are likely to be updated serially.
Figure 7: Performance comparison with increase in number of threads for Random test case, guided scheduling, busy-wait synchronization.
Figure 8: Performance evaluation of 100, 500, 1000, 2000, 3000, 4000 and, 5000 agents with increase in number of threads.

Figure 7 illustrates the performance comparison of guided scheduling and deferred-wait synchronization with increase in number of threads. Figure 8 illustrates the speedup achieved with increase in number of threads for different crowd sizes. The less-than-linear speedup with increase in number of threads is due to synchronization between agents as they acquire locks to read the data of neighboring agents. The egocentric nature of the fields ensures that agents will only request locks of other spatially co-located agents that fall within the boundary of their field.

AUTHOR:

Mubbasir Kapadia is a PhD candidate in the UCLA Computer Science Department, advised by Professor Petros Faloutsos. Mubbasir received a Bachelors in Computer Engineering from D.J Sanghvi College of Engineering, Mumbai, India in 2007. He
received a MS in Computer Science from University of California, Los Angeles in 2008. Mubbasir is a joint member of the Modeling, Animation, and Graphics Lab and the Game Design Lab at UCLA. His current research focuses on the application of automated planning and machine learning principles to simulate autonomous virtual humans. For more information, please visit www.cs.ucla.edu/~mubbasir

REFERENCES:


ADVISORS: PETROS FALOUTSOS & GLENN REINMAN

Petros Faloutsos is an assistant professor at the Department of Computer Science at the University of California at Los Angeles. He received his PhD degree (2002) and his MSc degree in Computer Science from the University of Toronto, Canada and his BEng degree in Electrical Engineering from the National Technical University of Athens, Greece. Professor Faloutsos is the founder and the director of the graphics lab at the Department of Computer Science at UCLA. He is an associate editor for the Journal of The Visual Computer and has served as a Program Co-Chair for the ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005. 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 the University of California, Los Angeles. He received his PhD and MS in Computer Science at the University of California, San Diego in 2001 and 1999 respectively. He received a BS in Computer Science and Engineering from the Massachusetts Institute of Technology in 1996. He is one of the thrust leaders at the Center for Domain Specific Computing established by NSF.
FURTHER RESOURCES:

UCLA MAGiX lab:  http://magix.ucla.edu/


Video demonstration of results:  www.cs.ucla.edu/~mubbasir/videos/egocentric.mp4