ABSTRACT
In this paper, we present a generic and flexible framework for multi-agent based, real-time crowd simulation. Due to the modular architecture and the simple structure, all components can easily be modified or replaced. In contrast to many other crowd simulation systems available, our system is able to handle very different scenarios and behaviors with minimal adaptation effort. Furthermore, the presented system is able to simulate and visualize all scenarios in real-time on standard PCs without precalculation steps. Optimization techniques like multi-threading and cell space partitioning are discussed to further improve the simulation’s runtime performance. Finally, different scenarios with different behavior models are discussed to show the flexibility of our framework.

KEY WORDS
Multi-agent simulation, interactive simulation, multi-threading, software design

1 Introduction
In general, crowd simulation deals with modeling the movement of a large number of single individual characters. Furthermore, the individuals can have individual behaviors to model the effects of group dynamics and crowd psychology close to reality. Crowd simulation has many different uses, for example in safety modeling, entertainment software, virtual training, traffic planning, and urban model simulations. The wide range of applications for crowd simulation results in a complex behavior model that no current system can handle. Thus, a specific area of interest has to be defined before implementing a crowd simulation. Human behavior is highly complex due to many different influences like stress effects, emotions, or decision making. Up to now, there is no existing representation that has been known to be reliable. Hence there is no general human crowd simulation available. Most human crowd simulations only simulate a few simple behavior models to fit into a specific scenario. However sometimes a designer desires to apply different behavior models to test and validate the accuracy of a behavior model for a given scenario. The results of those simulations can be very realistic, if applied to the desired fields, but often lack the possibility to produce believable results in other scenarios. To get a specialized simulation running with another scenario, often huge redesign efforts are necessary - if it is possible at all. Therefore, the design of an innovative and flexible solution that is able to incorporate a broad variety of different behavioral models would be preferable. Furthermore, the possibility to design a wide range of behaviors, even other than human - like animals or autonomous vehicles - with minimal effort could increase productivity and minimize development time. These requirements lead to a modular design of the crowd simulation and of the agents as presented in this paper. Except for collision avoidance and the consideration of crowd density, many current crowd simulations do not support agents’ communication. Hence, the occupant’s movements and decision-making processes are independent of each other, which can lead to a lack of accuracy. Therefore our simulation framework has a messaging and an event handling module to allow modelling of social behavior. Furthermore our system is designed to be fast enough to do the whole simulation and visualisation with a suitable count of agents in real-time. Offline simulation is feasible for simulating static structures like escape way analysis, but it is not sufficient when trying to utilize crowd simulation with interactivity like in virtual training areas. Due to the fact that most of the current applications focus on analyzing simulation results through a post-processor hardly any evacuation application seems to support simulating more than 1,000 pedestrians and visualizing them in real-time.

The main emphasis of this paper is on the development of a flexible crowd simulation framework and a prototypical implementation of an efficient and flexible simulation model. The simulation is optimized to utilize modern hardware like multi-core processors to obtain optimal runtime results. Many previous works in the field of crowd simulation has shown that multi-agent simulation leads to plausible results and provides intuitive modeling of a human crowd from a microscopic perspective. Thus, a human crowd is modeled by using a multi-agent based approach. This paper is organized as follows: Section 2 gives an overview of related work in the field of crowd simulation. In section 3 a flexible module-based agent design is introduced. The agent’s modules are a finite state machine administering the agent’s different states and in turn
it’s cognitive behavior, a path-planner module, a messaging module to allow for inter-agent communication, and a layer defining the agent’s physical features. The module based agent design is necessary to keep the framework flexible as each module can be modified or replaced independently with minimal effort. This allows for the definition of behavior in a way that goes far beyond an exclusive investigation of evacuation scenarios. Thus, our implementation exhibits a greater flexibility than most of the existing evacuation simulation systems. Optimization techniques are presented in section 4. Binary space partitioning and multi-threading are used in our simulation. These have only been taken into account by a few applications so far. The obtained results are shown and discussed in section 5. The use of a representative simulation setup shows that it is possible to simulate and visualize up to 2,000 agents in real-time using our framework’s prototypically implementation.

2 Related Work

Currently several crowd simulation systems have been developed. Due to the wide application area and different simulation and visualization complexities of those systems it is hard to compare them. Typically systems mostly differ in the complexity of agent behavior, the graphical level of detail of each agent, and the complexity of the environment. Therefore a feature-based categorization of each model is used for comparison. Aspects like the modeling method, the purpose, the space representation, the methods for simulating movement and behavior, the output or the use of visualization and CAD drawings are used for this categorization [5]. Usually the systems use a mix of different approaches in the different categories. A detailed review of some of these models can be found in [2, 6, 7]. Alternatively a slightly different categorization of egress models is introduced in [3, 8]. In this context, egress simulation means the act of escaping emergency situations. Most of the existing applications often have a limited variety of scenarios which they can be used for. Only several specific cases like evacuation scenarios, which the systems are designed for, can be simulated to yield realistic results. They are specialized to incorporate features to simulate egress situations (occupants responding to smoke conditions, toxicity, or heat), but lack the capability for being used for different applications i.e. computer graphic effects in films or city planning simulation of behavior other than escape-panic. Consequently, these models can produce quite good results at the desired fields. In contrast, they often need to be completely redesigned to simulate other cases. For instance, the software Legion 3-D from Legion Studios gives good results for pedestrian simulation However, it cannot simulate more than pedestrian behavior.

Another categorization aspect is how the application represents results to the user. Many applications also do not support visualization in 3-D. Either the simulation results are only represented as text or a simple 2-D visualization or they are recorded and used by a virtual reality software for visualization afterwards. However, a real-time visualization of the current simulation state at framerates suited for interactivity is only supported by a few applications allowing the simulation of less than 1,000 agents. In the following, some closely related models for crowd simulation are introduced.

High-Density Autonomous Crowds (HiDAC) can be used to simulate different types of crowds. The application area is ranging from extreme panic situations (escaping fire) to high-density crowds under normal conditions (leaving a movie theatre) [19]. A maximum number of 500 simulated agents is indicated by the developers of HiDAC. This comparatively small number can be explained by the relatively complex behavior engine of the agents. Another simulation system is Multi-Agent Communication for Evacuation Simulation (MACES). This system uses the Human Performance Moderator Function Server (PMFserv) as a mature, psychological model for human behavior [9]. It was first released in 2006 and it is said to be able to simulate up to 1,000 agents in real-time. Both of the described crowd simulation systems are not yet commercially available to the public, as they are still under development.

Finally one of the most common crowd simulation software is Massive Prime software. It focuses on simulating large crowds for generating computer graphic effects in movies and has been successfully utilized in movies, simulating up to 100,000 virtual characters in one scene. The Massive Prime software gives quite realistic results but due to the fact that the required calculations for such large crowds are highly complex, the simulation is an offline process. Rendering a large scene can take up to several days. Thus the Massive Prime software is not used for egress simulation. In the field of rational agent research, a wide range of methods, often formal and grounded logics to support agent reasoning [12], inter-agent communication [13], and autonomous planning and learning [14] has been developed. These methods enable unembodied agents to sense and respond to their virtual environments. However extensive computing resources are required to support these abilities and therefore the sophistication of an agent’s model is highly limited by the number of agents that need to be simulated in real-time. Real-time crowd simulation systems can only be based on a few parts or basic ideas of these models, which are important for the desired simulation scenario. In the following sections the agent model used in our simulation framework will be introduced and optimization techniques that enable the simulation to run at real-time framerates on standard PCs are presented. Afterwards, some different simulation scenarios are shown and discussed. They show the performance and flexibility of our crowd simulation system.
3 The Crowd Simulation Framework

In this section the crowd simulation’s framework is presented. First the modular Agent Model is introduced and then the single modules are described in detail.

Figure 1 shows the framework’s overall agent architecture. The three major structures of the agent are the Behavior Engine, the Path Planner and the Messaging System, which are introduced in the following chapters.

3.1 Behavior Engine

An agent’s behavior is described by the agent function that maps any given perception sequence to an action. Implementing the agent function results in the agent’s behavior. The term “behavior” can have many meanings. To get a human-like behavior, human behavior representations (HBRs) are used to model people’s behavior to various degrees. They either represent parts of individuals like legs, aggregates of individuals or individuals themselves. Cognitive functions like planning or reasoning, performance restrictions like sensing bandwidth or decision latencies and the influence of factors like stress, fatigue or discomfort are usually mapped by using an HBR [15]. To get a minimal configuration behavior engine three functions need to be present. It needs to accept input from the environment and update its internal state accordingly. It also has to choose additional information from its knowledge base to expand its internal representation of the world state. Finally responses, that can lead to achieving its goals based on gathered information and its internal state, have to be generated. The behavior engine determines actions by applying information from the knowledge base to the internal state representation.

The agent architecture defines a structural model of the components that constitute an agent, as well as the interconnections of these components together with a computational model, that implements the basic capabilities of the agent. Based on the introduced HBR, a simple model comprising a few states, sensors, and actuators is able to design and implement a prototypical human agent, sufficiently realistic enough to yield characteristical crowd phenomena. Due to the fact that an agent’s behavior can be modeled easier if the behavior is separated into different layers, a multi layered architecture is used for our behavior engine. As proposed in Reynolds’ article ”Steering behaviors of autonomous characters” [16], movement of an autonomous agent can be broken down into a hierarchy of three levels. A similar three layer hierarchy is also described by Blumberg and Galyean [1]. This approach is adopted and incorporated within the introduced HBR, as its three layers are planning, steering, and locomotion.

3.1.1 Planning Layer

The aim of a scalable crowd simulation is to obtain a behavior model that is simple enough to allow real-time simulation of a large number of agents, yet sufficiently complex to provide realistic, believable results. Due to these requirements, the usage of a finite state machine (FSM) seems to be convenient. A FSM decomposes a model of behavior into a finite number of states and transitions between them. In Figure 2 you can see a simple FSM with a few defined states and the related transitions. Each state can perform actions, if it is entered in active state or left. Thus a behavior can easily and intuitively be modeled with a FSM. Furthermore it is highly flexible. Beside these advantages, there are a couple of good reasons from the software engineering perspective to use a FSM. They are simple and quick to code, they enable easy debugging and have little computational overhead, granting a very good performance. Therefore a FSM was designed and implemented for controlling the agents’ behaviors in our framework. As our behavior engine was designed to be adaptable to different scenarios with minimal effort, this is a very important point.

3.1.2 Steering Layer

The steering layer is responsible for calculating the desired trajectories required to reach the goals and plans set by the planning layer. For autonomous embodied agents, steering behaviors define the microscale movement. They produce a steering force, that describes where and how fast an agent should move to get there. In contrast to the planning behavior, which is cognitive and can be modeled by using FSMs, the steering behavior is located on a reflex-based level. For
the scenarios presented in section 5, a few of Reynolds’ proposed steering behaviors that are fundamental for crowd simulation have been incorporated into the steering behavior module of our framework. They are based on the design that is described in [17]. For most cases, a combination of steering behaviors is used to reach the desired overall steering behavior. If an agent needs to follow a path, it also has to avoid other agents at the same time. A simple approach used in our framework is to multiply each steering behavior with a weight, to sum all of the weights, and to clip the resulting steering vector to the maximum steering force the agent is allowed to apply. New behaviors can be developed and integrated to the framework if desired.

3.1.3 Locomotion Layer

The locomotion layer is the most basic layer of the agent’s behavior engine. It represents the mechanical and physical aspects of an agent’s movement. The locomotion layer translates the steering behavior’s movement recommendations into movements that are subject to physically imposed constraints. The separation of a steering layer and a locomotion layer enables the framework to utilize the same steering behavior for completely different types of locomotion. This additionally minimizes the modification effort. The locomotion model used in our framework is a simple idealized locomotion model. At each simulation step, the desired steering force (clipped to the maximum force) acts on the agent’s point mass. This results in an acceleration equal to the steering force divided by the agent’s mass. The acceleration and the current velocity are accumulated and limited to the maximum speed. Finally the resulting velocity vector is added to the agent’s current position.

3.2 Pathfinding

A virtual environment for a crowd simulation can be quite complex due to a complex layout or geometry. However the agent needs a capability to navigate through the environment. Therefore paths have to be determined before traveling from one point to another. To navigate, an agent needs an internal topological representation of the environment that considers places as well as relations between them. For this representation, graphs are used by the most commonly used topological navigation strategies. It represents the network of connections an agent may use to move around its environment. An agent’s movement is not restricted to move along the graph’s edges from node to node. It can use the navigation graph to plan paths between them. Due to the fact that there usually is more than one path to reach a specific point, the agent also needs the possibility to find the shortest path. Therefore, the edges’ costs must be taken into account. Our pathfinding module uses Dijkstra’s algorithm [4] in the presented scenarios. It is very efficient and a number of implementations of this algorithm already exists. In certain configurations, the target node can be within the line-of-sight of the current agent’s position. The path planner executes a very fast line-of-sight test before executing any other operation. Thus, path planning can be skipped for this situation. Special situations that have to be handled arise in environments which are inhabited by many agents, that are likely to interfere with each other. Especially in panic simulations, this results in the appearance of congestion at bottleneck places. Typical bottleneck places are narrow doorways or tight passages. Also counterflow can force the agent to deviate from its path and get stuck. To avoid being trapped in such a situation, the agent needs the capability to plan a new path to its target. This is typically done by calculating the estimated time of arrival (ETA) for each node and replan if the available time exceeds the ETA. To get better results, more than one ETA can be calculated along a path to allow the agent for a fast replanning even if it gets stuck at the beginning of a path.

3.3 Messaging

The presented crowd simulation framework also has a powerful message handling which is capable of managing more than just agent-to-agent communication. Events are broadcasted upon occurrence to all relevant objects in the simulation that in turn can take appropriate action. The support for sending, handling, and responding to events enables the design of complex behaviors. Panic propagation by screaming can be simulated either by line-of-sight or area message delivery. A message in our framework is an enumerated type that has a shared meaning among all entities in the simulation, thus providing a common interface for communicating events. A routing protocol was implemented for direct agent to agent communication. It entails the sender’s ID, the receiver’s ID and the message itself. Furthermore a data container is available in a telegram for extra information or data to keep the message design flexible. For message routing purposes, a message dispatcher is available. This way it is easy for an agent to send a message encapsulated in a telegram to any agent if it knows the agent’s ID. As mentioned before, additional sending modes like broadcast, line-of-sight, and area transmission are supported by the message dispatcher.

4 Optimization

An embodied multi-agent simulation is a very expensive computation task, especially if moving agents share the same environment and have to take other agents into account while updating themselves. Even though the presented implementation can handle hundreds of single agents in real-time, it is often desired or necessary to simulate several thousand agents to get realistic, believable results. This section introduces two different optimization methods: multi-threading and space partitioning. By combining these methods, it is possible to further increase the number of agents that can be calculated simultaneously in
4.1 Multithreading

Most new PCs are based on a multi-core architecture. Consequently it is reasonable to benefit from possible performance boosts by writing multi-threaded applications. The dynamics of the multi-agent system is based on the simulation of each agent. Hence the major work will be carried out by updating all agents in a simulation step. To get a high speed up, the program’s hotspots are parallelized here. However it is very important for multithreaded applications to take data dependencies into consideration. Data dependence exists if the order of statement execution affects the results of a program. This might occur when the same location of storage is used by different tasks (i.e. collision checks by different agents). In our simulation framework, the agents’ update takes place within a single for-loop. This for-loop can easily be parallelized using OpenMP, which is a open source, portable, scalable software, that gives programmers a simple and flexible interface for developing parallel applications [11]. OpenMP also supports automatic for-loop parallelization, which is used in our framework.

4.2 Space Partitioning

For a plausible crowd simulation, collision detection and collision avoidance for neighboring agents is a highly important behavior. Due to the dynamic nature and the embodiment of the agents, each agent has to be aware of all other agents in each step of its simulation update. Since every agent has to complete the same task, the computational load for doing so will increase exponentially as the number of agents grows linearly (all-pairs-test problem). Typically, the number of tests is \( O(nm) \), whereas \( n \) and \( m \) are the number of possibilities for each of the two parameters. In an agent simulation where \( n = m \) the computational cost is \( O(n^2) \), which should be avoided in any real-time simulation. This case can be prevented by using binary space partitioning. A grid is projected onto the simulation’s environment and each agent is associated to a cell it is positioned at. Due to the fact that only agents in the same or neighboring cells can potentially collide, all other agents can be ignored for the corresponding agent’s collision test. To determine neighboring cells that need to be taken into account, a bounding box around the agent is used. This way, a major computational speed up can be achieved. Corresponding results and benchmarks are shown and discussed in the following section.

5 Results

In this section a performance comparison is done, taking into account the presented optimization methods first. Afterwards some completely different simulation scenarios are presented to show the flexibility of our framework.

5.1 Benchmark

In Figure 3, a comparison of the frame rates which are achieved by the different optimization techniques is illustrated. The horizontal axis indicates the number of simultaneously simulated agents. The vertical axis indicates the simulation’s average calculation time. To fit the requirements for being real-time, a minimum update rate of 25Hz has to be reached. This border is illustrated by the black, horizontal line. The maximum number of agents that can be simulated without optimization is approximately 730 agents. Using the presented multi-threading approach with OpenMP on a dual-core processor increases this number to approximately 1,000 agents. Further improvements are achieved by using binary space partitioning. The simulation is able to calculate approximately 1,250 agents in real-time using space partitioning only due to the fact that we avoid the all-pairs-test problem. Finally by combining both optimization techniques significant improvements are obtained. Using this optimization setup, the simulation is able to simulate approximately 2,000 agents in real-time, which is roughly three times higher than without optimization.

5.2 Scenarios

To show the simulation’s flexibility, a few scenarios with different behavior models have been implemented. Among these scenarios are pedestrian counter-flow studies in densely packed environments, theater filling, fire-escape panic scenarios, pedestrian simulation in cityscapes, and a simple infantry battle scenario. Figure 4 shows a fire-escape scenario in a small shopping mall. The black dot in part A represents an inactive fire agent. When activated, it broadcasts a fire message to all agents in the line-of-sight that switch to a panic state (turn red) and try to escape the mall. In part B and C you can see the agents escaping. This way the most used escape routes can be found. In addition it could be investigated how these routes depend on
the location of the fire source. This can be useful for architectural planning to find bottlenecks in escape routes. In part D the agent-to-agent communication was disabled, so no screaming is simulated. In contrast to the parts B and C, many grey non-panic agents still move around on the main escape routes, which is an unrealistic behaviour. The same mall layout was used in Figure 5 for a mall simulation under normal conditions. On the left side of the mall, the agents wander around shopping (grey colored). In this scenario an agent would respond to its desires (drink, eat, medical treatment, etc.) based on the behavior model. If they get hungry or thirsty, they try to get to the corresponding food stands. If they get injured, they become slow and try to reach the first aid area. The red arrow demonstrates a highly frequented route in the shopping mall. On the right side of the mall, a movie is shown at the cinema so that many agents try to get into the cinema through a small entry. To get the best seat the agents are acting very aggressively. Figure 6 shows a pedestrian simulation under normal conditions. On the right side more complex geometry and textures were used for visualization. Also some post-processing effects are enabled. This shows that our crowd simulation is also capable of creating high quality rendering results of the simulation in real-time. The battle scenario shown in Figure 7 illustrates a complete different behaviour than scenarios presented before. The agents are running into battle, frequently sending out hit messages in a close range. This simulates a sword swing. Enemy agents within rage are marked as hit (getting grey) and try to get back to their base for medical treatment. To protect them, friendly agents follow injured ones to the base. The presented results show that new behaviors can be modeled within our framework, which is not supported by most existing crowd simulation applications.

6 Conclusion

In this paper a flexible, modular and extensible framework for crowd simulation was introduced. Within this framework a finite state machine for modeling the agent’s cognitive behavior was designed and implemented. It is designed for a high degree of flexibility of the behavior representation. Some modules were introduced that are used by the human agent prototype. Beside the agent’s path planning module, a powerful event-handling system was integrated. In addition to dispatching general events, it also provides communication capability like line-of-sight or area broadcast for the agents. It has been shown that complex behavior can be created with the combination of a finite state machines and a powerful messaging system. It is also possible to see the simulation results in a real-time 3-D environment allowing for real-time interaction with the simulation. All modules are part of this prototypical implementation. It is possible to replace the implemented modules
by new ones for additional or different functionality. Finally the simulation was optimized using multi-threading and binary space partitioning to gain maximum scalability. The corresponding optimized results were compared to non-optimized simulation results. An increase in performance by approximately a factor of three can be observed. It was possible to simulate almost 2,000 agents in real-time on a standard PC. A few improvements considering the implemented prototype are certainly possible. Algorithms to automatically generate navigation graphs from existing geometry could be utilized. A possible implementation could incorporate the generation of a shortest-path roadmap as proposed in [18] to accelerate a simulation’s setup, as the graph’s points of visibility do not have to be placed manually anymore. Connecting edges would be identified on the fly. Furthermore incorporate partial topographical knowledge of an agent’s environment would be reasonable. The agents do not have access to the whole navigation graph at the beginning of the simulation. They need to explore the environment to obtain knowledge. Additionally agents could send knowledge to other agents. The messaging system’s telegrams integrated in our framework already contain a generic data container that can easily be used for this approach.

References


