Distributed Simulation of Dense Crowds

Sergei Gorlatch, Christoph Hemker, and Dominique Meilaender
University of Muenster, Germany
Email: {gorlatch,hemkerc,d.meil}@uni-muenster.de

***Abstract*—By extending previous approaches, we develop an agent-based model for the simulation of large, dense groups (crowds) of individuals. The model reflects behavioral complexity by assigning psychological and physiological attributes to the agents. In order to cope with the computational complexity, we design and implement a distributed, multi-server simulation framework in which the user can flexibly change both the simulation environment and parameters at runtime. We implement the simulation system using our Real-Time Framework (RTF) and demonstrate its scalability and speedup over multiple servers.**

***Index Terms*—Crowd simulation, Agent-based simulation, Parallel and distributed simulation, Real-Time Framework (RTF).**

I. Introduction and State of the Art

The simulation of the behavior of large and dense human crowds is a socially important and technologically challenging task. To represent the behavior of a crowd, three different kinds of models have been proposed in the literature: flow-based, entity-based, and agent-based models.

This paper develops an agent-based model for computer-based simulations that reproduces the motion of a crowd of individuals by a combination of psychological and geometrical rules with social and physical forces. We also design and implement a new approach to parallelize the simulation across several servers, using the Real-Time Framework (RTF) [2], developed at the University of Muenster.

In developing our model for crowds, we start with the HiDAC (High-Density Autonomous Crowds) model [5] based on the previous work [3] and improving the models suggested in [1], [4]. The resulting agent-based model has no central controlling unit; each agent corresponds to a simulated person with its own individual behavior.

We aim at a model for the challenging case with dense crowds, complex indoor scenarios with many rooms and large, unobstructed areas, with a possibility of panic situations. Our agents are designed to react dynamically to changes in their environment (e.g., if a door is interactively closed during simulation) and can select alternative routes. Agents pursue a global goal, e.g., leaving the building by following a sequence of waypoints at those doors that lead to the exit.

The high density of crowds and the complicated scenarios lead to highly intensive calculations. Therefore, we develop a distributed implementation of the simulation system that can run on multiple servers. We address the critical problem of scalability, which should allow for significantly higher numbers of agents than the sequential version, and we demonstrate the achieved speedup of simulation by conducting experiments with real-world scenarios.

II. The Model for Crowds

We implement several extensions and optimizations to HiDAC [5], in particular: a) the opportunity for dividing big rooms in several smaller rooms in order to balance the computations; b) adaptable creation of rooms, with nearly arbitrary number and arrangement of walls; and c) the introduction of local waypoints which are used for avoiding collisions and for moving around walls standing in the way.

Virtual World Representation. A map of the virtual world, e.g., a building, consists of several rooms which are augmented with walls and obstacles. We construct maps under the condition that walls and doors geometrically form polygons. This quite realistic assumption allows us to apply Jordan's curve theorem and to use the so-called "Ray casting algorithm" for deciding about the viewfield of the agents.

Collision recognition. Figure 1 shows how an agent recognizes a wall standing in its way (for that, test calculations are performed which we omit here because of lack of space) and runs around the wall. The values used for the corresponding

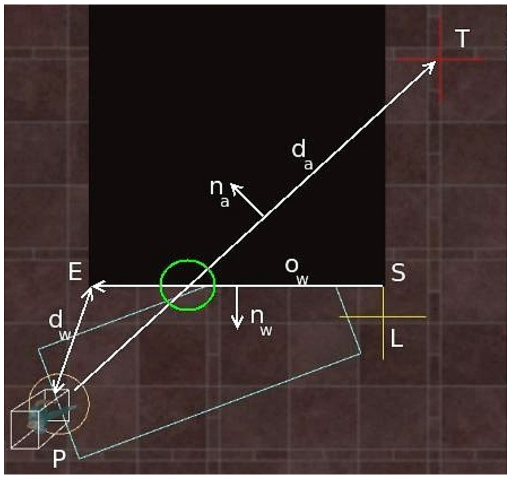

Fig. 1: Recognizing a wall standing in the way

calculations are as follows: the normalized vector n_w , the orientation vector o_w, the startpoint S and enpoint E, the current (global or local) goal T of the agent, the position P and its shortest distance to the wall d_w, and L is a potential local waypoint on one of its ends.

Pathfinding. Agent's movement is simulated by first assigning the agent a start room and a target room. Pathfinding reduces to the problem of finding the shortest path along the

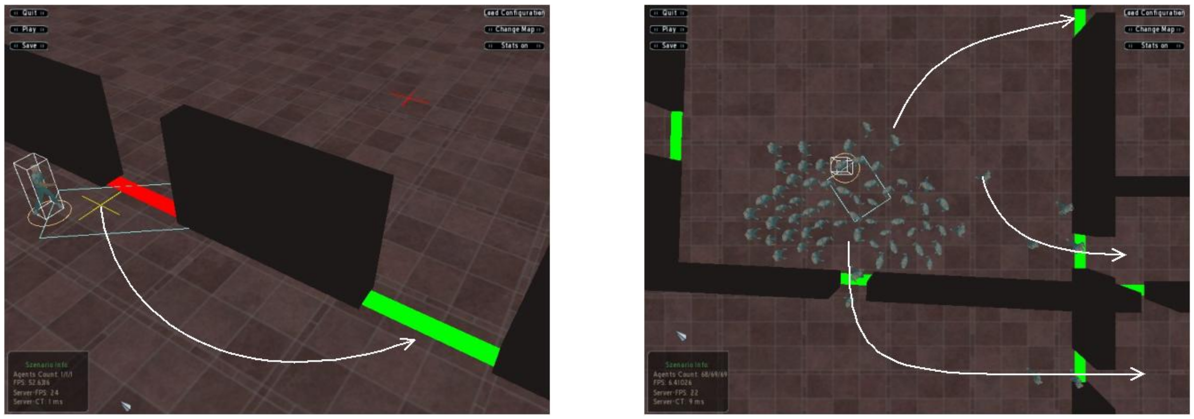

Fig. 2: Alternative routes for closed (left) and open (right) doors

edges of the graph that represents the simulated rooms. The weight of the graph edges plays an important role: the more agents stand before a door, the greater is door's weight: every single agent contributes to it by its diameter. The agent is excluded from the door weighting, as soon as it crosses the door or chooses another door. By using Dijkstra's algorithm for finding the shortest path, we determine the list of rooms which the agent should cross. During simulation, every agent selects the way on which it would bump into as few other agents as possible. Using door weights, the agent learns about the blocked and free passageways beyond its current room.

Alternative routes. An agent decides from one path to another in one of two cases: either a door on agent's way turns out to be closed or this door is blocked by other agents.

Closed doors which are located in the direct viewfield of an agent (see Figure 2, left) trigger a re-calculation of the shortest path; doors which are known to be closed/blocked are removed from this calculation by removing the corresponding edges from the graph of the building. For the doors which are blocked but not closed (Figure 2, right), the patience attribute of the agent is used for deciding either to wait or to aim at another door. The patience attribute is implemented as a counter: our model avoids too long detours by comparing the advantage of an alternative route with its overhead.

Agent's perception. Agents are designed to be similar to people in their perception of the environment. Every agent has its viewfield, which we also call its influence rectangle.

Within its viewfield, the agent tries to avoid collisions, see Figure 3. A collision is interpreted geometrically as an overlapping of an agent with another agent, an obstacle or a wall. The recognition and the consequent handling of the collisions is based on recognizing this overlappings and restoring an overlapping-free situation.

In the model, we express also pushing behaviour which means that an agent is actively pushing onto other agents, in order to reach its goal faster, in spite of possibly bringing the others to falling down. An opposite behavior to pushing is the agent waiting patiently: every agent owns a circular area

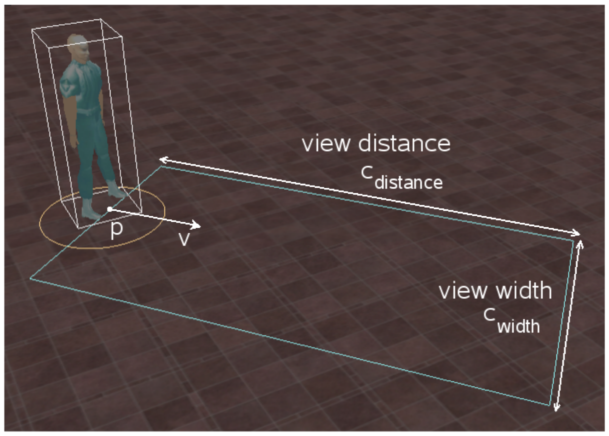

Fig. 3: The viewfield of an agent

within its viewfield, so-called circle of influence, such that if another agent appears within this circle then the agent waits till the other leaves. In order to avoid the situation that two agents block each other, each agent has a timeout value, which limits its waiting and which depends on its level of patience. Similarly, we model the effects of hectic running forwards and backwards in very dense scenarios.

Intelligent collision avoidance. In the model, we must avoid the situation that if after the collision of an agent with an object, the priorities and the external influences on the object do not change, then the agent will collide with the same object again and again. Therefore, we implement two additional features: a) intelligent collision avoidance changes the priorities of the agent for a short period of time after the collision; b) modified collision avoidance changes the external influence onto the agent after the collision, such that the same collision becomes very improbable.

Agents' falling down. An agent can be brought to falling down when other agents are pushing it too hard. In order to ensure that the simulation is near to reality, the other agents try to avoid the fallen agents considering them as obstacles;

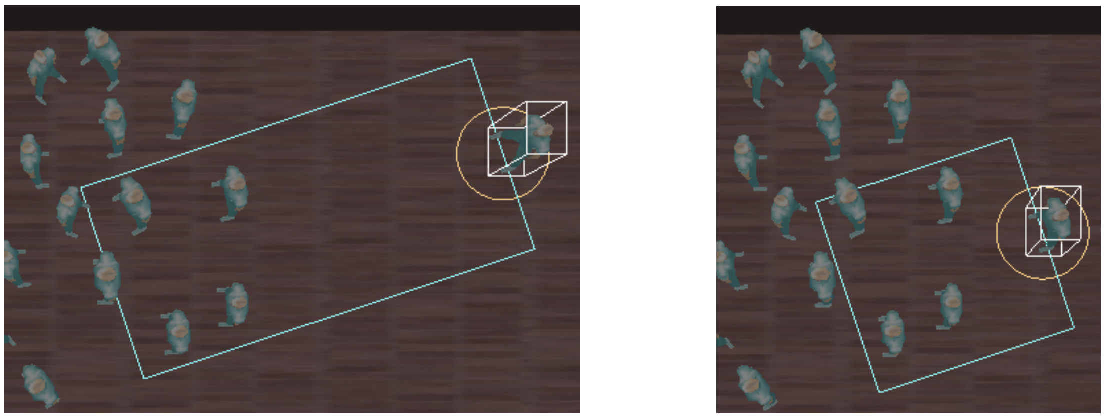

Fig. 4: Dynamic change of viewfield

however, if the pushing is too hard then it may happen that the agents are running directly on the fallen agents. This is done by assigning parameter values in the equations which compute agents' movements.

Panic propagation. For modeling panic behaviour, we modify physiological and psychological parameters of an agent. The panic factor of an agent, if increased, allows for a higher maximum speed and acceleration rate. In addition, panicing agents do not wait for others, and their personal circle is decreased, thus making such agents more active at pushing. Panic propagation is modeled by means of the panic level parameter: this level gets increased by each contact with a panicing agent, and as soon as it is greater than the patience factor, the agent itself becomes panicing.

Right move preference. In order to realistically model the collision prevention among two agents, we provide agents with a preference to move rather to the right than to the left.

Dynamic sight adaptation. We adapt the viewfield of agents depending on the density of the crowd: in a very dense environment, the agent takes care of its near neighborhood and ignores the agents which are far from it, see Figure 4.

III. Distributed simulation system

In order to organize an efficient simulation process for our crowd model, we address two issues: 1) we reduce the amount of information which is updated in each simulation step by means of the intelligent interest management, and 2) we parallelize/distribute the simulation computations by employing multiple servers and thus accelerating the computations.

Interest management. Interest management stands generally for the differentiation between important and not important information. With it, each client receives only those information updates which are relevant of its simulation state. The AoI (Area of Interest) management deals with the updates of not only agents but also other entities, e.g. obstacles. Figure 5 shows the effect of applying the AoI management.

Distribution of simulation. For distributing the simulation among several servers, we use the Real-Time Framework

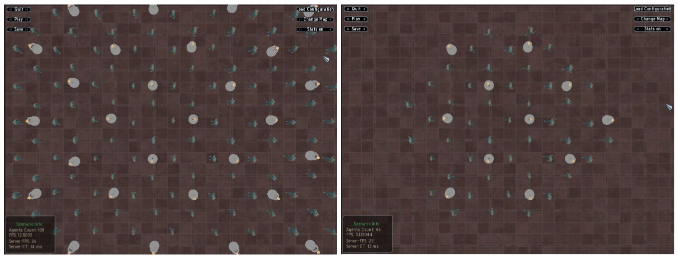

Fig. 5: Area of Interest: off (left) and on (right)

(RTF). The intuitive technique traditionally used in many distributed applications is 'zoning': the environment is split into disjoint zones, in which computations are handled by different servers. For crowd simulations, the 'zoning' approach has several drawbacks. First, agent interaction over zone borders is prevented, since information is exclusively available only to one responsible server. Thus, an agent cannot make a decision based on observing other remote agents, which is often necessary in practical scenarios. Moreover, when simulating dense crowds, we cannot distribute the computational workload where it is especially needed: zone borders can only be placed in sparsely populated areas, thus eventually leaving the simulation of a very densely populated area to one server. Finally, strict separation of data among servers requires the client, responsible for visualization, to communicate frequently with every single server in order to render a complete picture of the simulation state.

The novelty of our system is the use of 'replication' rather than 'zoning' for computation distribution. Replication means that each server holds the complete simulation data, see Fig. 6. Each server is computing updates only for its so-called active agents; all other agents are called shadowed on this server, and their updates are computed by other servers (every agent is active on exactly one server) and received from them. This allows us to distribute the workload evenly between servers, even in densely crowded scenarios, without hindering agent interaction as with 'zoning'. Additionally, a client now only

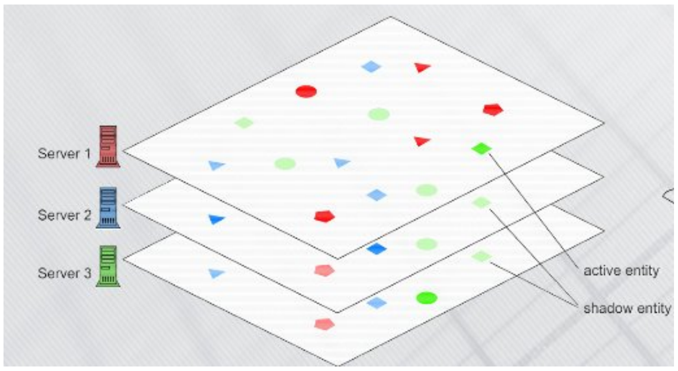

Fig. 6: Replication: active and shadow agents

needs to connect to one server to receive a complete picture of the simulation state for visualization. Replication in our system is implemented using RTF which supports both replication and zoning and advanced combinations of both. The simulation environment is described on a high level of abstraction in an RTF-specific 'map' which determines the distribution of geometrical space on available servers. Our current system employs a single area replicated over the network: each server comes with its own HiDAC unit. Using mechanisms offered by RTF, agents can be added to a unit, removed from it, and migrated to a different unit at runtime.

IV. Event Management

An event in our simulation system is a short-lived, serializable object which is used within the simulation process to send messages between the hosts (clients and servers). The event management subsystem is usually initiated by the client, e.g., to add or remove a particular obstacle. Another kind of interactions happen among servers in order to reflect changes of the simulation state.

The transparent distribution of simulation is supported by event management: interactions concerning the whole simulation are implemented as atomic multicasts; interactions concerning different servers are forwarded automatically; interactions must not bring the simulation into a non-consistent state.

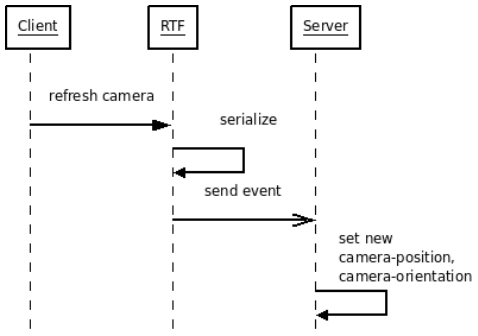

Fig. 7: Event ClientRefreshCameraPosition

As an example, we consider the situation when a client changes its camera focus (viewfield) during the simulation process. In this case, the new position and the orientation of the camera are read from the data of the virtual camera. As soon as these values' changes are greater than a predefined threshold, the client sends an event with the actual data to the server to which this client is connected. The server updates the viewfield of the client and uses it for the AoI management. The event is implemented as a message which is sent only to the connected server, see Figure 7.

Another example is the `ClientToggleDoorState` event for opening/closing a particular door. As a rule, doors are managed by a single server, such that a client which is not connected to this server should forward the event to it, see Figure 8.

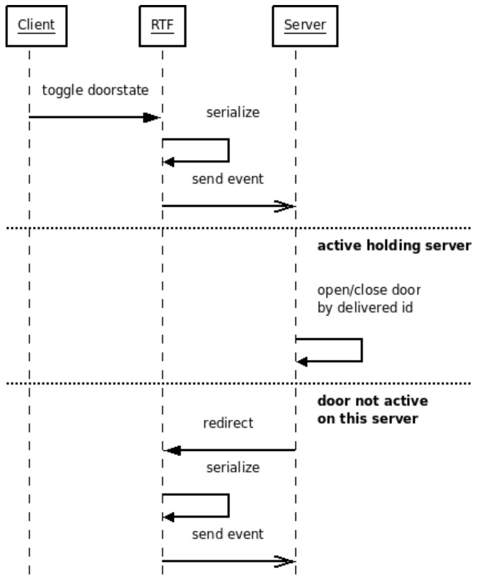

Fig. 8: Event ClientToggleDoorState

A special case is the event `ClientAddAgent`, with a possibility for a server to forward the event to another server. This happens if this other server manages fewer agents than the server to which the event was initially sent. Forwarding the message allows to balance the load between servers.

The only exception in the event management is the event `ServerSendDoorWeights` which is not sent from a client but rather from each replicated server to the activating server. This server manages a global object which contains the actual door weights of the simulation. The event sends the difference vector which reflects the change during the previous simulation step on the server. This vector is then used for computation on the server which manages the global object.

V. Experimental Results

In order to assess the performance of our simulation system, we conducted a series of tests in a high-load setup which emphasized those elements of a simulation scenario that

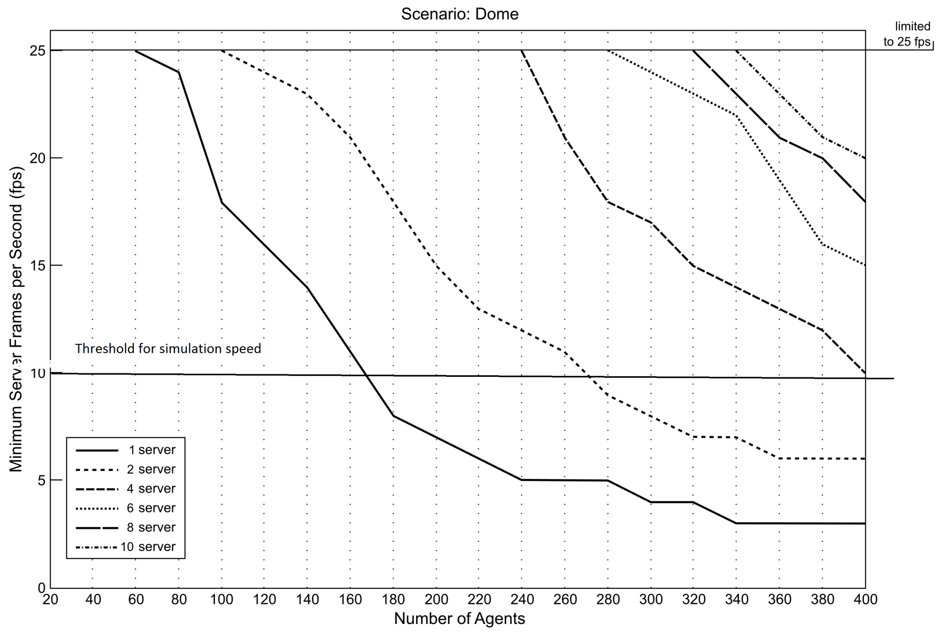

Fig. 9: Simulation speed on 1 to 12 servers.

lead to bottlenecks in the system's performance. We studied a complex indoor environment with many rooms and one large, unobstructed area, which is much more challenging than the simpler scenarios which are usually studied in the literature. While other agents hidden from agent's sight can be disregarded in many calculations, open space takes away this potential performance gain. Also, our testing setup ensures permanent agent movement because this induces additional computational workload. E.g., a scenario with 400 stationary agents may require less computing power than a scenario with 200 moving agents. Measurements were conducted on a local network of common desktop PCs (servers).

The measured value in the experiments is the rate of simulation in *frames per second* (fps) on the weakest system server. Measurements were done as follows: First, the server environment was prepared, comprising 1, 2, 4, 6, 8, 10, or 12 servers. Then, the test scenario was populated with 20 agents. After 1 minute runtime, the server simulation speed was measured. The simulation then again was populated with 40 to 400 agents, with a step of 20 agents, and measurements were taken again after 1 minute.

Our series of tests with the evacuation scenario for the St. Paulus Cathedral in Muenster (a medieval building of about 5000 sqm with a complex system of doors) produce the results shown in Figure 9. We observe that an increase in the number of servers allows for the simulation of more agents, or, at a fixed number of agents, increases the rate of simulation in *fps*. A value of 10 fps is an empirically found threshold to ensure correct calculations in our implementation: rate $\ll 10$ fps may lead to calculation errors, e.g., agents passing through walls. As shown in Figure 9, four servers already suffice to achieve this threshold for up to 395 simulated agents. Regarding scalability, one server can simulate 170 agents at 10 fps, whereas two servers manage 280 agents at the same frame rate (an increase of 64%), and four servers can increase this number further to 395 (132%).

VI. Conclusion

In this paper, we extended and modified the HiDAC approach to crowd simulation [5]. The advantages of our system include: flexibility (interactive changes at runtime), extensibility (accommodating new behavioral factors) and efficiency (real-time response) and, most importantly, the scalability over the number of servers used for the simulation of especially large, dense crowds. We also demonstrated that our Real-Time Framework (RTF) [2], originally created for applications like multi-player online games, supports high-performance distributed implementation of agent-based simulations at runtime and ensures their high scalability.

References

[1] Michael Batty. Polynucleated Urban Landscapes. *Urban Stud*, 38(4):635–655, 2001.

[2] Frank Glinka, Alexander Ploss, Sergei Gorlatch, and Jens Müller-Iden. High-level development of multiserver online games. *Int. Journal of Computer Games Technology*, 2008(5):1–16, 2008.

[3] D. Helbing, L. Buzna, A. Johansson, and T. Werner. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation Science*, 39(1):1–24, 2005.

[4] R. L. Hughes. The Flow of Human Crowds. *Ann. Rev. of Fluid Mechanics*, 35:169–182, 2003.

[5] Nuria Pelechano, Jan M. Allbeck, and Norman I. Badler. Virtual crowds: Methods, simulation, and control. *Synthesis Lectures on Computer Graphics and Animation*, 3(1):1–176, 2008.