Navigating Virtual Agents in Online Virtual Worlds

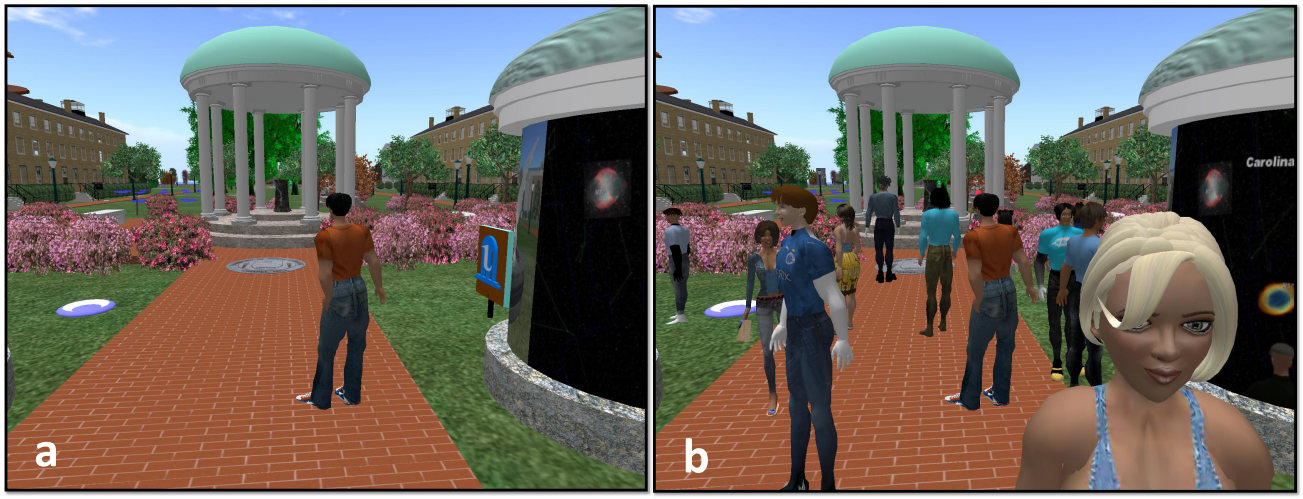

Figure 1: College campus: *(a) Many areas in online virtual worlds, such as this college campus in Second Life®, are sparsely inhabited. (b) We present techniques to add virtual agents and perform collision-free navigation. In this scene, the virtual agents autonomously navigate walkways, lead groups, or act as a member of a group. A snapshot from a simulation with* 18 *virtual agents (wearing shirts with any shade of blue) that automatically navigate among human controlled agents (wearing orange shirts) and improve the realism in the simulation.*

Abstract

We present an approach for navigating autonomous virtual agents in online virtual worlds that are based on a centralized server network topology. Each agent's motion is controlled through local and global navigation. Our local navigation model is based on artificial social forces that tends to compute collision-free paths between real and virtual agents and simulates repulsive and attractive behaviors. We perform global navigation for each virtual agent based on cell decomposition and compute high level paths. The overall computation is balanced by performing local navigation on client machines and global navigation on the server. We have implemented our navigation algorithm into the Second Life virtual world and highlight our results by simulating up to 18 virtual agents over multiple different client computers.

Keywords: crowd simulation, virtual worlds, avatar behaviors

1 Introduction

Large, online 3D virtual worlds have been growing rapidly in popularity. Millions of users are registered and are actively participating in worlds such as Second Life® (SL™), World of Warcraft™ (WoW), or OLIVE™ by Forterra Systems, Inc. In addition to these online worlds, Microsoft's Virtual Earth™ or Google Earth™ represent large and detailed navigable environments, although they do not currently support avatars. Many of these virtual worlds have been used for gaming applications and recently they have been shown useful for remote collaboration, economic planning, social simulations, and educational activities [Bainbridge 2007]. These online virtual worlds also provide an environment to study complex global behaviors such as traffic flows or evacuations.

The popularity and usefulness of these online platforms stems from their depth of immersion, generality and a large number of users. For example, Second Life's environment is largely programmable and the content is created almost entirely by its users. Moreover, users can interact with other avatars and with a large number of objects in the virtual world. With large amounts of virtual land available, users can create different situations or experiences, or visit the virtual worlds which have already been created.

Given the large growth and increase of land in virtual worlds, most areas of these online virtual worlds appear to have a low population density. At any time, thousands of users are logged on, but they are typically distributed over different parts of the virtual world. As a result, most portions of these worlds are sparsely inhabited. This low density can affect the realism or sense of immersion in these virtual worlds. In simulations or experiences where interaction with other avatars is essential, this poses a problem and can detract from the overall experience. For example, crowded areas like malls or busy streets or college campuses may not appear very realistic. One way to overcome this problem is to add virtual avatars to populate these worlds and improve the overall immersive experience. In this context, we define a virtual agent or avatar to be a member of the online world, whose motion is controlled entirely by a simulation, whereas a real or human agent is controlled by an actual user.

There are several challenges in adding realistic virtual agents to online virtual worlds, including;

- **Autonomous Behaviors**: It is desirable for virtual agents to act and behave as a real agent or human might in response to an situation. For navigation, this includes tendencies for moving either towards or away from other objects or agents, as well as standard formations.

- **Automatic Navigation**: Navigation refers to the ability of an agents to move in and around an environment. Plausible navigation usually requires a combination of local collision avoidance, path adjustments based on desired behavior, and global path computation. It is important that the motions should be plausible regardless of whether other agents are virtual or human controlled.
- **Networking issues** : The limited networking bandwidth and client-server latency add uncertainties into navigation computations and the overall behavior of the agents. Insufficient bandwidth can result in missing or incomplete information about an environment or the agents in that environment. The latency between a server and clients can result in agents making navigation decisions based on old or invalid information. This could result in visual artifacts such as collisions between the agents.
- **Performance**: A key criteria of online virtual worlds is real-time performance and interaction. One of the issues in the context of simulating virtual agents is the amount of computation necessary to simulate each virtual agent and how the resulting simulation scales with the number of agents. Generally, the addition of virtual agents should not greatly impact server performance any more than real or human users. Moreover, we will like to balance the computational load between the server and clients.

Main results: We present an approach for adding autonomous virtual agents into online virtual worlds with a centralized server network topology. In order to allow the virtual agents to have complete range of capabilities, each one is simulated as a full client in the virtual world. Therefore, no changes need to be made on the server to support the agents and the agents can be distributed among several client computers. We make no assumptions about the geographic or network locality of any agents, real or virtual. As is the case with current online virtual worlds, the server is responsible for sending and receiving updates for each of its clients.

We present techniques to control the motion of each agent based on local and global navigation. Local navigation determines how agents coordinate with nearby agents and obstacles, local path to their current goal, and the kind of behaviors that effect the motion of each virtual agent. We formulate basic repulsive and attractive behaviors between pairs of agents or agents and the obstacles, from which a wide variety of other behaviors can be created. We make simple assumptions about the linearity of motion within a short time frame to reduce the impact of latency or address the limited bandwidth between clients and the server and take them into account in the local navigation model. The global navigation algorithm determines how agents get to their goals within the environment. Therefore, each agent can act with a specific, global goal rather than just roaming around a predefined region. Furthermore, since the global planning will be the same for each agent, an algorithm which runs on the server can be used to balance the computational load between client and server.

We have implemented our approach into the Second Life virtual world. Clients are simulated through a motion controller based on Libsecondlife (LibSL), an open source project to reverse engineer the Second Life network protocol. We highlight our results by simulating up to 18 agents over two different client computers of varying computation power, in different geographic locations and using different internet bandwidths. In our experiments, each client computer can typically support about 10 to 15 agents. On a single client machine, the approach scales quadratically with the number of agents. Although, in our experiments the number of agents is relatively low. Otherwise, bandwidth becomes the primary bottleneck since each agent sends and receives a great deal of information.

Organization: This paper is organized as follows. Section 2 describes prior work in agent or crowd simulation and navigation. Our virtual agent model along with local and global navigation algorithms are described in Section 3. In Section 4 we describe implementation details and demonstrate the performance of our approach on two scenarios.

2 Related Work

In this section, we give a brief overview on prior work on motion of multiple agents; focusing on prior work in multi-agent simulation. For theory and applications of general motion planning for multi-robot systems, we refer the readers to [LaValle 2006]. And, for details on design and implementation of networked virtual worlds, we refer the readers to [Singhal and Zyda 1999].

There has been a great deal of work on modeling the motion of individual agents as well as those in small groups and large crowds [Ashida et al. 2001; **?**; **?**; Shao and Terzopoulos 2005; Thalmann et al. 2006; **?**]. Several different approaches have been proposed for generating motion of crowds; including agent-based methods, cellular automata, and methods based on discretized and continuous flows.

Agent-based methods typically include methods and rules for determining a heading based on information local to an individual agent. One of the earliest works by Reynolds [1987] described simple local rules for efficient and plausible flocking and herding behaviors. This model has been extended in many ways to include other factors including psychological [Pelechano et al. 2005] and sociological [Musse and Thalmann 1997] preferences. And, motion rules can be adapted to included velocity obstacles for improved avoidance [van den Berg et al. 2008].

Cellular automata methods model the evolution of agent locations by solving cellular automata. Different techniques for generating rules have been proposed, based on static and dynamic fields [Hoogendoorn et al. 2000], grid-based rules [**?**], and behavioral models [Tu and Terzopoulos 1994]. While these methods have been shown to capture a variety of globally emergent behavior, it should be noted that they are not physically-based.

Flow-based approaches treat the motion of multiple agents like that of physical flows. In discretized approaches, agents are treated like particles in a 2D dynamic simulation [**?**; Helbing et al. 2003; Lokoba et al. 2005]. These particles are essentially advected along potential fields, providing the resulting motion. Our motion formulation is most closely related to that of Helbing et. al [2003]. Other flow-based approaches consider the motion like a continuous granular flow [**?**] or based on continuum dynamics [Treuille et al. 2006].

While most of these approaches consider the local motion of a model, several approaches have been proposed to include global navigation for groups of human agents. Voronoi graphs have been efficiently used to compute and update navigation graphs [Sud et al. 2007a]. Other statically generated navigation graphs allow agents to navigate and also include local behavior [**?**; Pettre et al. 2007]. Dynamically updating graphs have also been used to combine global navigation with local behavior [Sud et al. 2007b] in dynamic environments. In the current formulation, our global navigation is based on a simple cell decomposition, but any of the more sophisticated methods would also work in this framework.

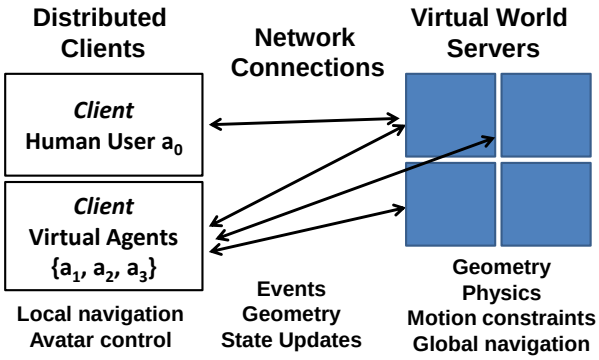

Figure 2: Online Virtual World Architecture: *The virtual world servers are responsible for maintaining and correcting the state of all avatars and objects in the world. We assume that all clients communicate with the individual servers in a centralized manner, rather than with each other. In this way, real and virtual agents can interact and be included, regardless of their physical location.*

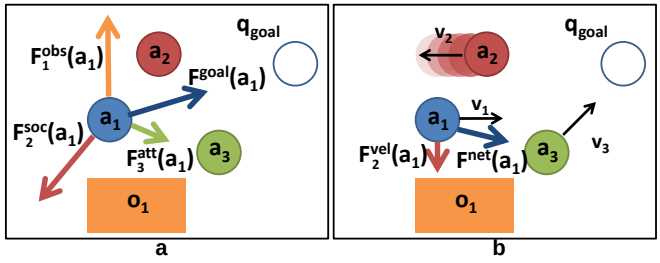

Figure 3: Local Navigation: *The local navigation model is based on the concept of social forces, where artificial forces are created to give the impression of pedestrian motion and desires. (a) Agent a_1 is acted upon by social repulsive force $\mathbf{F}_2^{soc}(a_1)$, attractive force $\mathbf{F}_3^{att}(a_1)$, obstacle force $\mathbf{F}_1^{obs}(a_1)$, and goal force $\mathbf{F}^{goal}(a_1)$. As the names imply, repulsive and attractive forces encourage the agent to move toward or away from agents and obstacles, respectively. The colors of the arrows correspond to the specific object that is used to generate the force. (b) An additional velocity bias force $\mathbf{F}_2^{vel}(a_1)$ is computed to account for agent a_2's velocity its likely resulting motion. By assuming a linear trajectory over a short period of time, we can additionally help to reduce the impact of network latency. Note that no velocity vias force is computed for a_3 since it is heading in the same direction as a_1. The final net force $\mathbf{F}^{net}(a_i)$ reflects the sum of all the forces and serve as the heading in the next step.*

3 Autonomous Virtual Agents

In this section, we introduce the notation used throughout the paper. The motion of each agent is governed by two levels of control: local and global navigation. These two navigation algorithms are combined and give the agents the ability to make decisions about nearby, or local agents and obstacles and at the same time provide a guiding path towards their destination. Furthermore, this separation leads to a natural way to distribute the navigation computation between multiple clients and the server.

3.1 Notation and Definitions

We consider an agent $a_i \in \mathcal{A} = \{a_0, \ldots, a_n\}$ and is represented by a cylinder of finite radius r_i. The state $\mathbf{q}_i$ of a_i at time t is given by $\mathbf{q}_i(t) = \{\mathbf{x}_i(t), \mathbf{v}_i(t)\}$ for position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$. Note that we make no distinction between whether other agents are controlled by human users (i.e. real agents) or are virtual agents. This allows our virtual agent to treat all other agents in a uniform manner.

We assume that environment E is composed of polygonal obstacles $\mathcal{O} = \{o_0, \ldots, o_m\}$. As before, the number of obstacles in $\mathcal{O}$ can vary for each agent. Typically, for each agent we only consider the obstacles that are within some fixed radius of the agent. Furthermore, we make no assumptions about whether or not obstacles are static or dynamic, or represented as rigid or articulated models. Additionally, we assume there is a centralized network topology, such that each agent connects to a server, or group of servers with a shared database. These servers are responsible for sending the agent's state information to all other agents connected to that particular server (See Fig. 2). In this way, virtual agents and real agents can seemlessly interact, and from any geographic location.

3.2 Collision Free Navigation

Our goal is to automatically compute a path for each virtual agent that is collision-free with respect to other agents and the objects in the scene. Due to uncertainties and latencies caused by the network, the position of other agents and the obstacles is not precisely known. As a result, it is difficult to compute absolutely collision-free motion.

Our approach breaks down agent's navigation into two portions; local and global navigation. Local navigation allows the agents to make adjustments to their current path based on nearby agents and obstacles. Global navigation provides a sequence of subgoals for an agent so that it can reach its final goal or destination.

3.2.1 Local Navigation

The local motion model for each virtual agents is based on the concept of *social forces*, i.e. non-physical forces which mimic decisions and behavioral responses [Helbing et al. 2003]. These forces are used to guide an agent along a path towards its intermediate or final goal. The general idea is that each agent generates a force field around itself. At each discrete time instance, the agent a_i samples the force field at its current location. The resulting force is applied to a_i, resulting in a motion trajectory (See Fig. 3).

Our formulation of social forces is based on the model proposed by Helbing et al. [2003] and later extended by Lakoba et al. [2005]. In practice, this model is able to capture emergent behavior of crowds with varying agent per area densities. For agent a_i, our base local motion model is composed of several components. A social repulsion and attraction force ($\mathbf{F}_j^{soc}(a_i)$ and $\mathbf{F}_j^{att}(a_i)$, respectively) determines how much agent a_i wants to avoid or move towards agent a_j. Social repulsion mimics an individual's personal space as well as desire to move independently while social attraction causes a virtual agent to be in a group with certain other agents, such as friends walking together or a tour group exploring. Similar repulsive and attractive forces are defined for obstacles ($\mathbf{F}_j^{obs}(a_i)$ and $\mathbf{F}_j^{obsat}(a_i)$, respectively) which guide the agents toward or away from obstacle o_j. Moreover, a goal force, $\mathbf{F}^{goal}(a_i)$, encourages an agent to move towards its current goal (See Fig. 3(a)).

In order to compute total force applied to an agent, we aggregate these forces as follows:

$$\mathbf{F}^{net}(a_i) = \mathbf{F}^{goal}(a_i) + \mathbf{F}^{agents}(a_i) + \mathbf{F}^{obstacles}(a_i), \quad (1)$$

where

$$\mathbf{F}^{agents}(a_i) = \sum_{a_j \in \mathcal{A}, i \neq j} (\mathbf{F}_j^{soc}(a_i) + \mathbf{F}_j^{att}(a_i)),$$

and

$$\mathbf{F}^{obstacles}(a_i) = \sum_{o_k \in \mathcal{O}} (\mathbf{F}_k^{obs}(a_i) + \mathbf{F}_k^{obsat}(a_i)).$$

The primary component forces are defined as:

$$\mathbf{F}^{goal}(a_i) = \frac{\mathbf{v}_d \mathbf{e} - \mathbf{v}_i}{\tau}$$
$$\mathbf{F}_j^{soc}(a_i) = \alpha e^{(r_{ij} - d_{ij})/\beta} \mathbf{n}_{ij}$$
$$\mathbf{F}_k^{obs}(a_i) = A e^{(r_i - d_{ik})/B} \mathbf{n}_{ik}$$

where $\mathbf{v}_d$ is the agent's desired speed, $\mathbf{e}$ is the direction to the agent's current goal, τ is its reaction time, $r_{ij} = r_i + r_j$, $d_{ij} = \mathbf{x}_i - \mathbf{x}_j$, α is a social scaling constant, β is the agent's personal space dropoff constant, $\mathbf{n}_{ij}$ is the normal direction between a_i and a_j, A is an obstacle scaling constant, B is the obstacle distance dropoff constant, and $\mathbf{n}_{ik}$ is the vector from a_i to the nearest point on o_k to a_i. The attractive forces, $\mathbf{F}_j^{att}(a_i)$ and $\mathbf{F}_j^{obsat}(a_i)$, take the same form as their repulsive counterparts except that they have different values for the constants and an opposite sign for the scaling term. Since these forces are largely based on the social force model of Helbing et al [2003], we refer the reader to this for more details.

The motion behind a social force model requires the ability to numerically integrate and explicitly set an agent's position and velocity, much like that the simulation of multiple point masses (Cite: Siggraph course notes). However, since virtual worlds were built to be controlled by humans, a different type human interface is typically used which does not allow explicit setting of position and velocity. Instead, agents can only proceed in its current trajectory for a specified period of time. Therefore, our navigation model must be able to specify a heading rather than a new state. In order to determine a new heading, we perform numerical integration on the agent's velocity after applying the forces. The new velocity is used to project the current position for a specified amount of time, t_p. Note that it is common for the trajectory to change before the agent reaches the projected position.

These forces provide a foundation for computing the localized motion for each agent. Based on these forces and reasoning about other avatars, additional agent behaviors are possible. For example, aggressiveness or urgency can be described by increased scaling constants for goal and repulsive forces. This would result in the agent being more biased towards the goal and other agents to be more likely to avoid this agent. Moreover, queuing behavior, such as when agents want to form a line or follow each other, can be added by attractive forces generated only by agents at the end of a queue.

Contact Handling: While the repulsive force generally allows agents to avoid contacts, they do not give guarantee of a collision-free motion. Social forces with a large magnitude can cause the agents to move directly towards each other. Moreover, network communication latencies can result in incomplete information about the location of other agents, objects or dynamic obstacles in the scene. In these situations, contacts must be resolved. We assume that the central server is ultimately in charge of resolving intersections between both agents and obstacles. Since latency is a factor even in the absence of virtual agents, it enforces hard constraints on agent positions to prevent intersection and returns that information to the client.

In order to reduce the impact of future collisions with the currently colliding obstacle or agent, additional forces are applied to each agent in collision. A pushing force, $\mathbf{F}_j^{push}(a_i)$, acts to force a separation between agents and a frictional force, $\mathbf{F}_j^{fric}(a_i)$ simulates the act of slowing down due to a collision. Unlike the associated repulsive force, these forces are only applied when an intersection has taken place.

For agent a_i and an intersecting agent a_j, the following forces are added to Eq 1:

$$\mathbf{F}_j^{push}(a_i) = \kappa(r_{ij} - d_{ij})\mathbf{n}_{ij}$$
$$\mathbf{F}_j^{fric}(a_i) = \lambda|\mathbf{F}_j^{push}(a_i)|\mathbf{t}_{ij}$$

where κ is a pushing spring constant, λ is a sliding friction constant, and $\mathbf{t}_{ij}$ is the tangent vector to $\mathbf{n}_{ij}$. Contacts with obstacles are treated in a similar fashion.

Network Modeling: Our virtual agent model is computed based on the information on a client database. This introduces certain amount of uncertainty in the position and velocity of other agents and also in other moving obstacles in two ways. First, some agents or obstacles may not have yet been sensed, i.e. their base information has not yet been received by the client from the server. Second, due to low network bandwidth or high network latency, the updated positions and velocities not received in time by both the server and client.

In order to account for these issues, we augment Eq. 1 with a velocity bias force, $\mathbf{F}_j^{vel}(a_i)$, to help reduce the impact of these events. Let $V_j = SSV(a_j, \mathbf{v}_j, t_{bias})$ by the spherical swept volume as agent a_j travels along heading $\mathbf{v}_j/||\mathbf{v}_j||$ for time t_{bias}. Then,

$$\mathbf{F}_j^{vel}(a_i) = \gamma e^{(r_i - d(V_j, \mathbf{x}_i)/\epsilon)} \mathbf{n}_{(i,V_j)},$$

where γ is a velocity bias scaling factor, ϵ is a bias dropoff distance, and $d(.,.)$ and $n(.,.)$ are the distance and normal direction between the a_i and volume V_j. Intuitively, the force naively assumes that agent a_j will proceed in its current direction for a fixed period of time and generates a force field around the volume swept of a_j along that heading (See Fig. 3(b)). As a result, other agents will tend to move away from their current direction of motion. Combined with $\mathbf{F}_j^{soc}(a_i)$, the resulting force is strongest closest to a_j and reduces further from the swept volume. The net effect of this force is that it gives virtual agents a way to estimate where other agents will go based on whatever information is available, and thereby reduces the impact of latency or bandwidth limitations. Behaviorally, this force has additional benefits in that the agents will tend to slow down and avoid each other as they move toward their goals.

Goal Selection: The last portion of our model for local motion involves selecting intermediate goals. The selection of goals can result in a variety of behaviors (HOW?). In our formulation, we use two different methods for selecting the goals. First, the goals are randomly selected within some radius of the agent. This gives the appearance of wandering or exploring within that radius. Second, scripted goals can be provided for other effects. For instance, in a panic situation the agents would be given goals away from the cause or location of the panic. Or, a tour group could be simulated by having a goal attached to a tour guide avatar.

3.2.2 Global Navigation

The local navigation model described above is used respond to various situations or perform local collision avoidance computations. However, the local model cannot be used to navigate through a

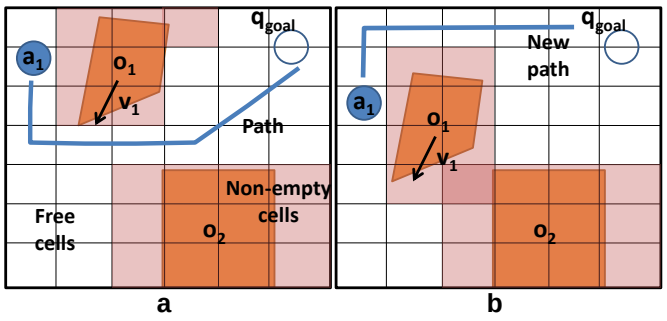

Figure 4: Global Navigation: *(a) Our cell decomposition approach first uniformly samples a grid to determine whether cells are free, mixed, or full, and a path is extracted from the grid for each agent. (b) As o_1 moves, the path may become invalid. When possible, a new path is extracted and the agents are notified to verify and update their path.*

complex environment. This is largely because there may be no straight-line route to the current goal everal static obstacles in the way. Moreover, in the presence of various agents and obstacles, local navigation could result in a local minima and the virtual agents may get stuck.

In these situations, a global planner is necessary. The global planner provides agents with a sequence of subgoals which will eventually lead to the final goal. There are a wide variety of options available for route planning or roadmap computation [LaValle 2006]. For simplicity we use a cell decomposition based approach, extended for use with unknown and changing environments, as well as partial or incomplete information.

Our approach initially samples all navigable surfaces, such as floors in the buildings or walkways and streets in open areas. A variety of sampling schemes could be used, such as a one based on a quadtree for each surface primitive. Instead, our current formulation uses a uniform sampling of the surfaces. The sampling resolution can be computed based on the size and some information about the environment. For each sample, we classify it as either free sample, enclosed by an obstacle, or mixed. Then, we connect neighboring free samples in order to generate a connectivity graph. When an agent requests a path from the global planner, an A* search is performed on this graph at runtime, and a sequence of subgoals is computed and stored. The free or colliding state of cells are updated as changes occur in the environment. These changes in turn notify the virtual agents that they probably need to recompute a path (See Fig. 4).

Server-side planning: Within a localized region, there is little need to duplicate the global navigation system since it will be essentially the same for each virtual agent. One simple optimization would be to make a single module which would handle planning for each agent. By placing this module on the server, the cost of planning and replanning is reduced. Furthermore, it acts as a means to balance some of the overall computation by placing some work on the server while the client does the rest.

4 Discussion and Results

In this section we describe our implementation, show results of our virtual agent model, and discuss possible issues with the approach.

4.1 Implementation

A preliminary implementation of his approach has been developed for Second Life, based on the LibSecondlife (LibSL) framework. LibSL is a project to reverse engineer the SL network protocol. With this, it is possible to create a SL client session without using the official SL viewer resulting in avatars that appear as full clients to the server. Furthermore, it includes data structures for avatars and objects as well as events for when data has been updated. Support for simultaneously controlling multiple avatars also exists. While the number of avatars per computer is limited, additional computers can be used to host additional clients. In this way, the algorithm can scale easily to multiple computers as long as there exists a single host server.

The avatar motion control model along with the local agent dynamics have been implemented on top of this framework. A forward Euler integrator is used to estimate a heading and the velocity is locally computed. For increased interactivity, the ALICE chatterbot engine has also been integrated into virtual agents. This allows them to have small conversations with users and to define goals based on the location of agents which a virtual agent is chatting with.

For testing, one PC ran only virtual agents while a second PC ran both virtual agents and a viewer. These computers were placed on different networks, a university department connection and an at home cable modem respectively. Furthermore, the computers were physically located in different states.

4.2 Results

We have tested our approach in a two scenarios:

- **Populating a city block**: When exploring the numerous cities in Second Life, it does not take long to realize that many of these cities see very few avatars at any given time. In contrast, the equivalent city block may be teaming with individuals. This city block includes 2 streets, a fountain which agents cannot directly cross, and a patio of a building which the agents can walk through. Our virtual agents are added to the city block to improve realism (Fig. 5). Each agent randomly selects a goal and travels to that location, after which it selects another goal. All agent and obstacle interactions are with repulsive forces. Basic avoidance as well as a little bit of lane formation is apparent. The server managed 18 virtual agents and 3 real agents. The 17 virtual agents were distributed between two PCs.
- **Campus tour guide**: To show a wider variety of behaviors, we added agents to a university campus; an area which would see a great deal of activity in the real world. The environment is composed of a walking path between a building and a campus landmark. This scene included three classes of virtual agents; wandering students, a tour group, and a tour guide. The students were modeled with only repulsive forces and selected goals at random. The tour guide also selected goals at random, but had a small attractive force with its tour group. And, the tour group had a higher attractive with each other and the tour guide, but a repulsive force with any other agent. Grouping along with following is easily observable in this scene (See Fig. 1 and Fig. 6). There are a total of 20 agents, 2 real and 18 virtual agents, in the scene. As before the virtual agents were simulated on two client PCs, at different locations.

The overall run-time performance in both scenarios was about the same. Virtual agents were distributed over 2 client computers, in

Figure 5: City block: *A sequence of still images following the evolution of virtual agents in a small city block. Several virtual citizens (circled in red and wearing shirts in various shades of blue) move around the streets and occasionally stop by the fountain. Virtual agent behaviors include exploration, attraction to the fountain for a period of time, and avoidance. Agents* a_1 *and* a_2 *are identified to demonstrate their avoidance of each other as they cross near the fountain. A human controlled agent (orange) acts as an obstacle and must be avoided. The scene has* 18 *virtual agents distributed over* 2 *PCs.*

two different states. The load on each computer had to be tailored based on the PC's processing power, graphics hardware, and internet connection. After tuning the computers, interactive performance was not a problem. So, it is likely that this approach will scale with additional client computers.

While the simulation was relatively cheap, the biggest limiting factor was the available bandwidth. Each agent, even if they're hosted on the same client computer, sends and receives a extensive amount of information. In our experiments, each PC could support about 8 to 15 agents without noticeable decay in performance. In most cases, the servers seemed capable of handling well over 20 agents, even with the large amount of traffic generated by moving agents.

It should be noted that our limit of 18 agents is unrelated to the performance. Instead, it was the number of accounts on Second Life we were able to borrow.

5 Conclusion and Future Work

We have proposed a virtual agent navigation model for online worlds. In our implementation, each agent is viewed as a full client to the server allowing for a wide range of capabilities. Agents navigate through a local motion model which determines coordination between agents as well as agent behaviors. Global navigation is used to help an agent reach goals beyond the scope of the local motion model. Our preliminary results are promising and the algorithm can control several virtual agents from a single computer. This work works as a first step for more complete autonomy for virtual agents.

While the approach works well, there are a couple of limitations. First, the approach generally requires more bandwidth that server-based solutions. Virtual agents whose locomotion is computed entirely on the server only need to send data to the clients as opposed to the virtual agent needing to both send and receive information. However, server-based solutions will probably require modifications on the server and is not as easily deployable as the approach described here.

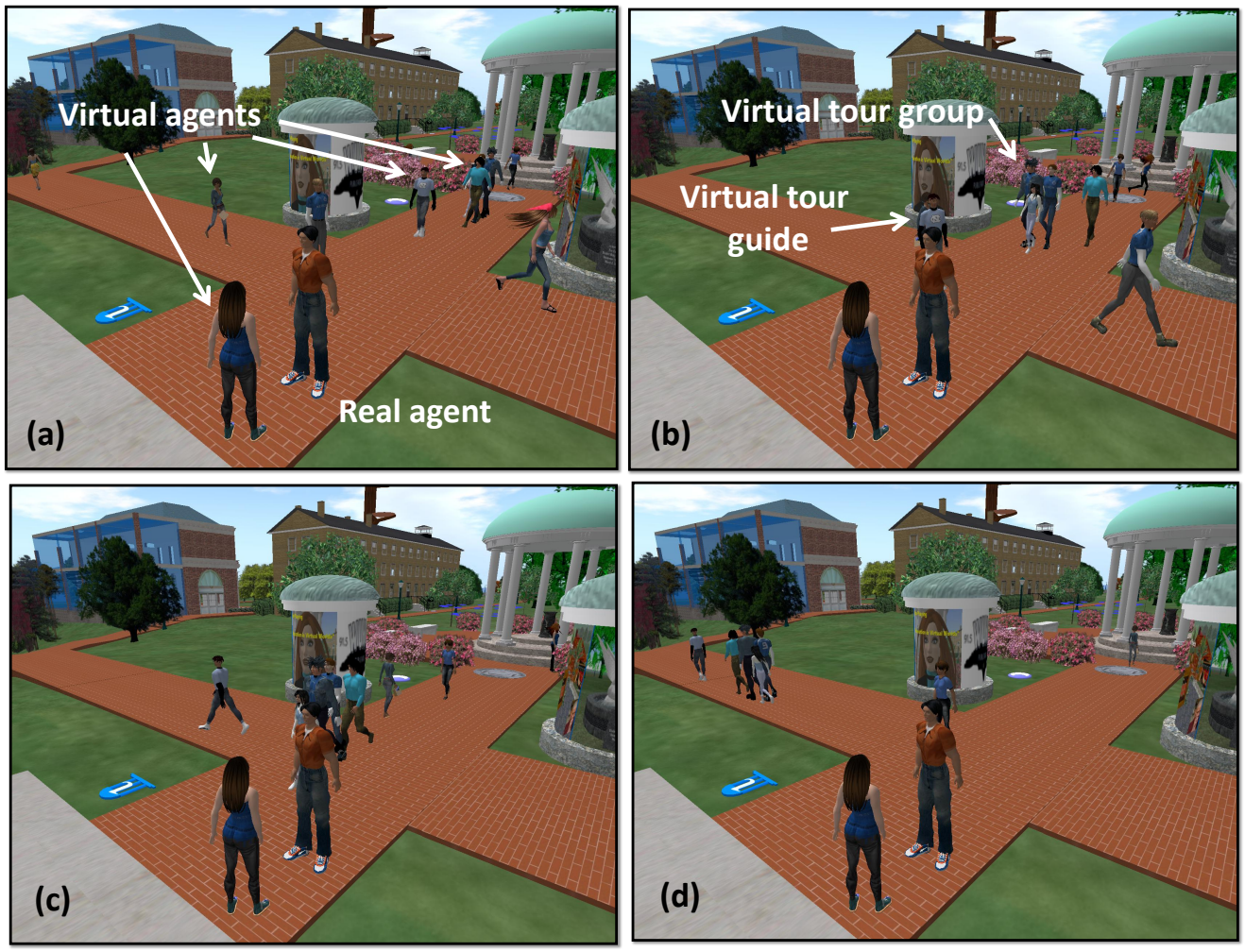

Figure 6: Campus tour guide: *A sequence of still images following the evolution of virtual agents in a campus scene; at the University of North Carolina's Second Life campus. To better simulate the real world, several virtual agents (wearing shirts with various shades of blue) have been added to the scene. Agent behaviors include leading a group, following, being part of a group, selecting random goals for exploration, and avoidance. This models a virtual tour guide leading a small tour group while also avoiding the human controlled agents. There are* 18 *virtual and* 2 *real agents in this scene.*

Since we only plan for the position of agents, motions beyond standard walking and running are not currently supported. For instance, the virtual agents would not be able to crawl, jump over items, or climb; they can only walk or run around or toward agents or obstacles.

Also, while the local navigation model effectively moves agents through the environment, there is no guarantee that they will actually reach their current goal. However, this has not be a problem in our experiments thus far. Furthermore, since the navigation model is subject to network latencies, some of the resulting motion does not always very natural in appearance.

There are several directions for future work, including addressing the issues above. For more realistic motion, a better route planner could be used along with a more sophisticated motion controller. For instance, it would be better if it took the virtual agent motion constraints into account. There are a huge number of applications which could benefit from these agents. More sophisticated agents could help to realize a wide variety of situations. And, we also hope to better address limitations on computational and network resources. For instance, in many cases, virtual agents far away from human agents likely do not need to be actively moving. This can help to reduce traffic and computational overhead of a large number of agents. And finally, we hope to scale the work to much larger crowds for virtual worlds.

References

ASHIDA, K., LEE, S. J., ALLBECK, J., SUN, H., BADLER, N., AND METAXAS, D. 2001. Pedestrians: Creating agent behaviors through statistical analysis of observation data. *Proc. Computer Animation*.

BAINBRIDGE, W. S. 2007. The scientific research potential of virtual worlds. *Science*.

HELBING, D., BUZNA, L., AND WERNER, T. 2003. Self-organized pedestrian crowd dynamics and design solutions. *Traffic Forum 12*.

HOOGENDOORN, S. P., LUDING, S., BOVY, P., SCHRECKENBERG, M., AND WOLF, D. 2000. *Traffic and Granular Flow*. Springer.

LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

LOKOBA, T. I., KAUP, D. J., AND FINKELSTEIN, N. M. 2005. Modifications of the helbing-molnr-farkas-vicsek social force model for pedestrian evolution. *Simulation*.

MUSSE, S. R., AND THALMANN, D. 1997. A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Computer Animation and Simulation*, 39–51.

PELECHANO, N., O'BRIEN, K., SILVERMAN, B., AND BADLER, N. 2005. Crowd simulation incorporating agent psychological models, roles and communication. *First International Workshop on Crowd Simulation*.

PETTRE, J., GRILLON, H., AND THALMANN, D. 2007. Crowds of moving objects: Navigation planning and simulation. In *ICRA*, 3062–3067.

REYNOLDS, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. *Comput. Graph. 21*, 4, 25–34. Proc. SIGGRAPH '87.

SHAO, W., AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 19–28.

SINGHAL, S., AND ZYDA, M. 1999. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley Professional.

SUD, A., ANDERSEN, E., CURTIS, S., LIN, M., AND MANOCHA, D. 2007. Real-time path planning for virtual agents in dynamic environments. *Proc. of IEEE VR*, 91–98.

SUD, A., GAYLE, R., ANDERSEN, E., GUY, S., LIN, M., AND MANOCHA, D. 2007. Real-time navigation of independent agents using adaptive roadmaps. *Proc. of ACM VRST*. to appear.

THALMANN, D., O'SULLIVAN, C., CIECHOMSKI, P., AND DOBBYN, S. 2006. *Populating Virtual Environments with Crowds*. Eurographics 2006 Tutorial Notes.

TREUILLE, A., COOPER, S., AND POPOVIC, Z. 2006. Continuum crowds. *Proc. of ACM SIGGRAPH*, 1160 – 1168.

TU, X., AND TERZOPOULOS, D. 1994. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of SIGGRAPH '94*, A. Glassner, Ed., 43–50.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. C. 2008. Interactive navigation of individual agents in crowded environments. *Proc. of ACM Symposium on Interactive 3D Graphics and Games*.