A Novel Network Architecture for Crowded Online Environments

J.R. Parker
Nathan Sorenson
Digital Media Laboratory
Faculty of Fine Arts/Drama

Abstract - A scheme for autonomous player avatar behaviour is suggested, in which player patterns are identified and then used to navigate through subdued parts of a first person multiplayer game. Thus, instead of sending a complex sequence of key presses, a single packet would be transmitted giving start and end coordinates, and the avatar would be controlled by an AI system in between. We present and evaluate a novel architecture for crowded online virtual environments that employs a multi-layered state description to maintain relaxed state synchronization among clients. The proposed technique significantly reduces total bandwidth costs and is suitable for especially crowded virtual environments.

Keywords: multiplayer games, networked games, latency.
Subject Descriptors: D.1 PROGRAMMING TECHNIQUES; C.2.5 Local and Wide-Area Networks

1. Introduction

Online games and virtual environments allow hundreds of individuals to collaborate in an interactive setting, facilitating activities such as commerce, entertainment, and social networking. These systems use network bandwidth to send and receive the positions of the players and objects in a game. For example, Second Life, a popular online world, boasts over ten million registered users [Linden Research 2008]. This popularity, while evidence of the usefulness of online virtual worlds, is also the source of considerable network congestion, which impairs the usability of the environment by introducing latency.

The basic algorithm used by a typical multiplayer online game uses a client-server communication scheme where the server(s) know the true positions, orientations, velocities, and all other attributes of all objects, players, lights, and sound sources within the game. This architecture is designed to ensure that a single, synchronized state is accurately reflected on all the networked machines. This is a difficult task since the complexity of this state grows as the number of interacting users increases, and the bandwidth required to transmit this state to every client machine becomes prohibitive [Funkhouser 1995]. A key aspect of this paradigm is that every motion of the player’s avatar is instigated by the player, and corresponds to a key press, mouse motion, or controller button push.

Latency in such a game can be described as the time delay between a key press (action request) and the time when that action is embodied on the player’s screen. It is caused by computational delays in servers and client machines, bandwidth limitations on the Internet, and natural light speed transmission delays and bounces from one intermediate network machine to another - it is rare for a player to link directly to a game server. Instead, there will be a set of computers that read the player’s data packet and send it on to the next computer on the route to the server. Each such stopover creates a slight delay, and these sum to a significant figure. The delay affects the way that players interact with the game and how enjoyable the experience is [Claypool and Claypool 2006; Dick et al. 2005; Pantel and Wolf 2002].

Some online games (and games in general) recognize the concept of a macro, and allow the player to collect a sequence of button or key presses into a single key. Players of fighting games especially find this useful; the macro corresponds to a sequence of moves, or a combination in fighting parlance, and good players have a number of these that can be deadly against beginning players. This can be useful in reducing latency if fewer commands are sent to the server as a result.

Because of latency considerations, online virtual environments, as they are currently designed, must impose a limit on the number of users that can interact with each other at any given moment. The M-COVE architecture overcomes this limitation by abandoning the requirement that a single, synchronized state be duplicated perfectly on every client machine. It does not focus on eliminating the inconsistencies introduced among clients but rather capitalizes on this lack of synchronicity to realize significant bandwidth savings; once the need to keep states strictly synchronized is abandoned, state descriptions can employ far less precision and therefore consume much less bandwidth. M-COVE achieves this by regarding all objects on a client not as direct embodiments of other users, but rather as autonomous agents equipped with behaviour models that aim to realistically mimic user behaviour. As well, server state is expressed both in terms of individual object behaviour, where necessary, and in terms of large-scale crowd dynamics, which ultimately requires less bandwidth. This multi-layered approach results in a novel virtual environment architecture that incorporates both loose synchronicity and crowd animation techniques in order to facilitate highly populated virtual environments.

2. Latency Amelioration

Many systems have been designed to alleviate network congestion in crowded online environments. Early efforts to limit latency looked at end-to-end patterns [Gautier et al. 1999] and algorithms inspired by operating systems [Palazzi et al. 2006], and the limits of these approaches may well have been reached. Other work done on latency reduction has mainly examined the packet level [Bangun and Dutkiewicz 2001] and used statistical means and simulations [Widmer et al. 2002] to predict overall traffic levels and delay times. Games cannot be played using large amounts of data from clients for security reasons [Cronin et al. 2003], so we must avoid looking at giving clients more critical data.

Reducing the effects of latency has sometimes been a matter of prediction of players future positions, thus relaxing the requirement for strict synchronicity. Systems such as Deva3 [Pettifer and West 1999] and Torque [Garage Games 2008] can realistically
reproduce server state on client machines using less bandwidth. Client objects can be updated less frequently as client-side prediction techniques such as dead-reckoning [Singhal and Cheriton 1995] ensure their motion is smoothly interpolated. However, updates still need to be sent to every object several times a second, causing the amount of bandwidth needed to update a particular client to increase linearly with respect to the number of objects on the server. These techniques alone are, therefore, insufficient for facilitating large crowds.

If the positions are predicted incorrectly, the situation jumps into a new state (warps) when the server detects the problem, and it may be several hundred milliseconds or more before that occurs. The basic strategy is that of ‘dead reckoning’ and ‘time warp’ even today. According to [Bernier 2001]

“For instance, if the client is running at 50 frames per second (fps) and has 100 milliseconds of latency (round-trip), then the client will have stored up five user commands ahead of the last one acknowledged by the server. These five user commands are simulated on the client as a part of client-side prediction.”

Back up five steps is very disruptive in game play [Mauve 2000].

There are many methods based on limiting bandwidth usage by restricting the number of individuals able to simultaneously interact. They do not provide environments capable of sustaining large amounts of users; rather, they essentially partition large worlds into collections of smaller ones. Spline [Barrus et al. 1996] partitions virtual worlds into “Locales,” and maintains synchronization only between objects in the same Locale. Similarly, NPS-NET [Macedonia et al. 1994] and DIVE [Frécon and Stenius 1998] divide the world into hexagonal cells and multicast groups called “lightweight groups,” respectively, to achieve similar results. SCORE [Léty et al. 2004] seeks to address the problems encountered when too many users occupy a single world partition by providing the ability to dynamically subdivide the partitions into sub-regions called “cells.” Similarly, Steed and Angus [Steed and Angus 2005] overcome the limitations of static world subdivision by employing “frontier sets” to dynamically determine which clients may potentially interact, based on the architectural layout of the environment.

Bendford et. al. [1997] present a technique for the aggregation of crowds into single entities called “Third Party Objects,” allowing for more simultaneous users than is possible using object-by-object updates. However, these Third Party Objects must be explicitly defined by the system author through the provision of classifying processes that determine whether a given object is “inside” or “outside” a crowd. For example, the authors suggest that any client who enters one side of a virtual stadium is considered to be a part of the “red supporters” crowd and otherwise part of the “blue supporters” crowd. Once a client enters one of these statically defined crowd areas, it loses its individually unique avatar representation and becomes subsumed by an iconographic representation of the crowd as a whole in the author’s example: a single “giant sized person.”

It would be preferable if crowds were not arbitrarily represented as single abstract entities, but rather as large collections of individuals with the high-level aim of reproducing the crowd’s appearance. As Blumberg and Gaylen have shown [Blumberg and Galyean 1995], objects can indeed be provided with relatively simple behavior models that enable them autonomously to interact with one another at the high level of interaction as a group. Ken Perlin’s Improv system [Perlin and Goldberg 1996] employs this approach in a networked environment, allowing client agents to asynchronously interpret high-level animation directions issued from the server. In the context of an online virtual environment, then, one can consider a client agent’s “goal” to be the reproduction of the server’s current state.

Nearer to the principles of the work to be presented here, crowd animation techniques extend this notion further by directing large groups of entities as a whole, instead of individually. For example, Craig Reynolds’ flocking agents can pursue a common target while maintaining a herd formation using a very simple reactive behavior model [Reynolds 1987]. Recent work has continued to improve crowd direction, such as Trueille et al’s Continuum Crowds approach [2006]. Continuum Crowds are inspired by Hughes’ method of modeling crowds as a set of continuous differential equations [Hughes 2002] (as opposed to individual interacting agents,) and are controlled by a two-dimensional gradient field that globally optimizes the crowd’s path to a goal with respect to factors such as travel distance and opposing flow. This approach demonstrates interesting effects such as the formation of traffic lanes and vortices, yet requires an expensive re-computation of the gradient field at every time-step for every unique goal, and as such is only suitable for situations with relatively few competing objectives. It is therefore not applicable to networked online environments where every agent’s goal (i.e. a server-specified location) is unique.

3. M-COVE Architecture

M-COVE aims to combine elements of the discussed techniques into a unified architecture that fluidly describes world state in terms of both individual objects and large-scale crowd dynamics, offering, where appropriate, the simplicity of synchronization of the former perspective or the low bandwidth requirements of the latter. The server maintains a description of each individual object’s behavior as well as a two dimensional representation of the environment’s overall flow of crowd traffic, and transmits a subset of this data to each client. The client, then, enacts this state through local agents that correspond to the server objects. These agents are equipped with a behavior model that seeks to maintain reliable synchronization when directly in view of the client where synchronization is important, while “blending into the crowd” and following the large-scale traffic flow otherwise.

It is clear enough that a complex behavior in a multiplayer game can be represented as a recognizable series of key presses, but it can also be encoded in other simple ways. For example, in a first person shooter simulation if a player is close to hitting you with a bullet, but misses, it should not always be necessary to send the exact bullet and time locations to the client so that the near-miss could be calculated precisely by the client machine. Nor would they need the keypress sequence the opponent used to nearly miss them. The event could be classified as a “near miss” and a “near miss” code (integer ID) would be transmitted to the client which would interpret it, based on it’s own local state configuration, in a visually interesting way.

3.1. Server Updates

As it is never possible to predict exactly how the client will traverse the virtual environment, the agents must always be ready to gracefully assume a synchronized state. This is achieved by ensuring all agents get occasional updates from the server regarding their server objects’ “actual” location. It is then the agents’ responsibility to stay within a reasonable distance from this goal.

The updates issued from the server to the autonomous agents on the client can take one of two forms. “Detailed orders” contain 16 bytes of precise location and velocity information and tend to be sent at a high rate (around ten times a second). This form of update is used for agents in the immediate vicinity of the client, where accurate representation of server state is essential. In this case, the behaviour model reduces to dead-reckoning interpolation, achieving synchronicity in a similar manner to typical online virtual environments. “General orders” however, are sent to the vast
majority of agents at a much slower rate (less than once a second), and consist of the object’s current server position compressed into two bytes. This information represents the overarching goal that a client agent is to achieve at its own discretion. Updates are sent as “detailed orders” when

\[ d_{\text{client}} < 2 - d_{\text{near}} \]

where \( d_{\text{near}} \) is a constant denoting the range under which agents will be updated every update cycle and \( d_{\text{client}} \) is the distance of the agent to the client. The factor of 2 ensures agents are provided with some space to smoothly transition from a loosely synchronized to a tightly synchronized behaviour.

Whether or not an update will be given to a particular agent on an update cycle is determined by a probabilistic scheduling algorithm, which gives priority to agents nearest the client without completely starving distant agents:

\[ P\left[ \text{update} \right] = \left( \frac{d_{\text{near}}}{d_{\text{client}} + \varepsilon} \right)^\alpha \]

where \( P\left[ \text{update} \right] \) is the probability that an update is sent (clamped to \( [0, 1] \)), and \( \alpha \) is an attenuation exponent which can be increased dynamically to reduce the total number of orders sent to distant agents This can be done, for example, every time a specified bandwidth threshold has been exceeded, theoretically allowing for any number of evenly distributed agents to be supported while maintaining constant bandwidth usage.

3.2. Patch Grid

The M-COVE architecture expresses the large-scale dynamics of crowd movement by imposing a coarse two-dimensional grid over the virtual environment. Each element in the grid is referred to as a “patch,” and contains statistics regarding observed crowd activity. When a server object moves through a patch, it marks its entry and exit points. The vector difference is quantized into one of 8 directions and stored. The average density of objects on the patch, as well as the average time spent on a patch, is also recorded. This information can be collected by the server over time and needs to be sent to each client only once, as this information represents long-term usage tendencies that are unlikely to change from moment to moment. By taking this information into account, the behaviour model reproduces traffic patterns typical of the given environment and therefore exhibits collision avoidance behaviour and simple path-planning. Agents, for example, will assume intelligent paths through and around environmental obstacles simply because they tend to follow the paths of actual users—there is no need for expensive path searching or cost optimization.

3.3. Behaviour Model

The M-COVE architecture makes use of a reactive behavioral model that takes into account its current update received from the server, its current position relative to the client viewpoint, as well as the properties of the patch it is currently standing on and chooses one of four possible behaviors. If an agent currently has a “detailed order” consisting of precise location and velocity information which is also “fresh,” (meaning it has not yet arrived at the specified location,) it employs classical dead-reckoning behaviour to smoothly interpolate its position to the specified location.

If an agent with fresh orders is out of view and currently farther than WARP_THRESH from its goal location, it simply warps to that location. If the desired location is within view of the client, the agent warps to the closest off-screen point and proceeds from there. This behaviour ensures agents do not become trapped in local minima present in the patch grid when trying to work their way towards their destination. Agents that are some distance from the client’s perspective are most likely to warp, as it is these agents who are most likely to have a great amount of distance to cover between sporadically received orders. Since they are generally far away, this behaviour is rarely noticeable.
If an agent has no fresh orders the agent will stop and wait for a moment with a probability (clamped to [0, 1]):

\[
P[\text{wait}] = \frac{t_{\text{avg}} - t_{\text{cross}}}{t_{\text{client}} - t_{\text{cur}}}
\]

where \(t_{\text{avg}}\) is the average time an agent spends on this patch, \(t_{\text{cross}}\) is the time it would take to cross the patch at normal movement speeds, and \(t_{\text{cur}}\) is the time the current agent has spent on the patch.

Finally, if none of the previous cases hold, the agent picks a direction to move such that its progress towards its goal is maximized. This goal is not simply moving towards its destination as quickly as possible, but is rather a system of competing goals. For example, an agent must reproduce the crowd dynamics represented by the underlying patch, but it also would like to maintain a relatively straight path of movement to reduce the amount of unnatural and erratic motion.

The chosen direction can be represented as the result of the following sum of twelve unit vectors:

\[
\hat{v} = \alpha_{\text{goal}} \hat{v}_{\text{goal}} + \alpha_{\text{dense}} \hat{v}_{\text{dense}} + \alpha_{\text{dir}} \hat{v}_{\text{dir}} + \alpha_{\text{avoid}} \hat{v}_{\text{avoid}} + \sum \alpha_{qi} \hat{v}_{qi}
\]

where \(\hat{v}_{\text{goal}}\) is the direction towards the agent’s server-specified location and

\[
\alpha_{\text{goal}} = f \left( \frac{c_1}{d_{\text{client}} + \varepsilon} + c_2 \cdot d_{\text{goal}} \right)
\]

where \(f = 1\) if the current order is “fresh” and 0 otherwise. \(c_i\)'s are adjustable weighting coefficients and \(d_{\text{goal}}\) is the distance to the agents server-specified location.

\(\hat{v}_{\text{dense}}\) is the quantized direction pointing to the adjacent patch that experiencing the greatest “density debt,” that is, the greatest difference between the number of objects that are usually on the patch and the number of agents currently on the patch. Hence, \(\alpha_{\text{dense}} = c_3 \cdot \text{debt}\)

\(\hat{v}_{\text{dir}}\) is the current direction of the agent and \(\alpha_{\text{dir}} = c_4\). This vector encourages agents to maintain smooth forward movement.

\(\hat{v}_{\text{avoid}}\) is the direction that is must be taken to move out from the client’s viewpoint. This direction is useful if the client is approaching an agent that has out-of-date directions and therefore likely out-of-sync. It would be undesirable for the client to be able to directly interact with an agent that was “not really there.” Also, by moving off-screen this agent is now able to warp to its destination if necessary. The weight is determined by:

\[
\alpha_{\text{avoid}} = c_5 \cdot \text{age} \cdot \text{vis} \cdot \frac{d_{\text{client}} + \varepsilon}{d_{\text{client}} + \varepsilon}
\]

where \(\text{age}\) is the age of the latest order received beyond a specified threshold, \(\text{vis} = 1\) if the agent is in view of the client and 0 otherwise.

Finally \(\hat{v}_{qi} \ldots \hat{v}_{q8}\) are the eight quantized compass directions and \(\alpha_{qi} = c_6 \cdot t_i\) where \(t_i\) is the average number of objects on this patch that travel in direction \(\hat{v}_{qi}\).

The resulting vector is normalized and the agent checks the patch to ensure that a non-zero number of agents have traveled in that direction. If not, this likely signals a nearby obstruction. In this case, the agent chooses the nearest direction to the desired direction that does have a non-zero usage.

It should be evident that each of the elements comprising the behaviour model have a straightforward functional purpose and either a yes/no trigger condition or an associated linear weight. It follows that any desired behaviour can be similarly given a weight and inserted into the present model. Therefore, this framework is suitable for top-down modification, offering a flexibility that allows for the fulfilment of a variety of different domain requirements.

4. Results

For testing, a sample environment is populated by simulated clients that travel throughout the world in a non-uniform manner via specified waypoints. Using traditional client/server synchronization techniques representative of most current online virtual environments, the Torque engine is able to reasonably accommodate 60 of these clients when restricted to a 2,000 Bps bandwidth limit. Using the proposed probabilistic update algorithm, the M-COVE architecture is able to accommodate approximately 500 simulated clients within the same 2,000 Bps limit before unnatural movements become noticeable.

Furthermore, the client front-end displays the same traffic flow patterns and congestion areas that exist on the server.

It is probable that the factor that ultimately limits the maximum number of agents supported will be the processing power that is needed to evaluate the behaviour model, not the bandwidth used for updates. Though it is not likely to improve the bandwidth efficiency, it is hoped that further adjustment of the behaviour model will result in more natural individual movement and more realistic crowd flow dynamics. More rigorous testing will be conducted in the immediate future.

5. Conclusion

By combining classical synchronisation techniques with recent work in crowd animation, M-COVE is able to express world state in terms of both individual client behaviour and overall crowd dynamics. At the core of the architecture is a reactive behaviour model that is computationally inexpensive to evaluate and simple to extend. Furthermore, M-COVE has been implemented with an existing commercial game engine, suggesting that this architecture could be readily implemented in other typical virtual environment systems. M-COVE is therefore a novel, flexible architecture that exploits the lack of synchronicity in online virtual environments to support dramatically more simultaneous users than is possible with existing methods.

The M-COVE behaviour model currently operates under the implicit assumption that the only activity that needs modeling is client movement. Clearly, in most real-world virtual environments there are several activities in which clients can engage themselves, such as collaboratively building new spaces, as in Second Life or battling hostile creatures, as in World of Warcraft [Blizzard Entertainment 2008]. Clearly, the presented model would have to be expanded significantly to account for these unique domain requirements. The current behaviour model also assumes a world where overall traffic flow remains relatively stable over time. Volatile crowd characteristics would need to be considered for applications where this is not the case.

Since M-COVE operates in a way that does not require global synchronization, it is likely that this is a technique that would lend itself particularly well to distributed servers. It might even be possible to modify this system to operate in a pure P2P environment.
6. References


Figure 1: Torque implementation with 800 simulated users.