Comparing Interest Management Algorithms for Massively Multiplayer Games

Jean-Sébastien Boulanger
School of Computer Science
McGill University
Montréal, Canada
jboula2@cs.mcgill.ca

Jörg Kienzle
School of Computer Science
McGill University
Montréal, Canada
joerg.kienzle@mcgill.ca

Clark Verbrugge
School of Computer Science
McGill University
Montréal, Canada
clump@cs.mcgill.ca

ABSTRACT

Broadcasting all state changes to every player of a massively multiplayer game is not a viable solution. To successfully overcome the challenge of scale, massively multiplayer games have to employ sophisticated interest management techniques that only send *relevant* state changes to each player. This paper compares the performance of different interest management algorithms based on measurements obtained in a real massively multiplayer game using human and computer-generated player actions. We show that interest management algorithms that take into account obstacles in the world reduce the number of update messages between players by up to a factor of 6, and that some computationally inexpensive tile-based interest management algorithms can approximate ideal visibility-based interest management at very low cost. The experiments also show that measurements obtained with computer-controlled players performing random actions can approximate measurements of games played by real humans, provided that the starting positions of the random players are chosen adequately. As the size of the world and the number of players of massively multiplayer games increases, adaptive interest management techniques such as the ones studied in this paper will become increasingly important.

Categories and Subject Descriptors

C.2.4 [**Computer-communication networks**]: distributed systems—*distributed applications*; D.1.3 [**Programming techniques**]: Concurrent Programming—*distributed programming*

General Terms

Performance, experimentation, algorithms, measurement

Keywords

Computer games, interest management, distributed games

Netgames'06, October 30–31, 2006, Singapore.

1. INTRODUCTION

Since 1997 with the creation of Ultima Online, a new genre of online game has emerged, the massively multiplayer online (role-playing) game, short MMOG or MMORPG. Compared to a traditional multiplayer game in which usually up to 16 players play a relatively short-lived game, MMOGs offer the possibility for thousands of players to play together in a persistent world [14]. In a typical game, each client sees a graphical representation of the world and controls a player – an avatar – which can perform actions. Basic building blocks of such actions are, e.g., moving the avatar, picking up objects, or communicating with other players. MMOG implementations face huge scalability problems since they have to handle a massive amount of connected players, presenting them with a consistent view of the world, and still providing good performance and hence, an enjoying experience.

In order to provide a shared sense of space among players, each player must maintain a copy of the (relevant) game state on his computer. When one player performs an action that affects the world, the game state of all other players affected by that action must be updated. The simplest approach is for each player to maintain a full copy of the game state and that all players broadcast updates to all other players. The problem with this approach is that it does not scale: as the number of players increases, the messages sent over the network and to be processed by each client increase exponentially.

One of the most effective strategies to address this problem is to send to a player's computer only the messages that are relevant to its avatar (e.g., only the update message of objects it can see, or that are near). The world space of MMOGs contains a lot of information and a single player needs only to know about a subset of that information. *Interest management* (IM) is the process of determining which information is relevant to each player [16].

The information relevant to a player usually corresponds to the perception of its avatar. The perception, or expression of interest [16] of an avatar in interest management scheme is often based on proximity, modeled as a sphere around the avatar. However, the most common type of perception in MMOG is what an avatar can see, which does not always correspond to proximity. In particular, game worlds usually contain static obstacles that occlude regions of the game

space. An object that is close to an avatar, but behind a wall, is not relevant to that player.

Many interest management techniques [16] have been proposed and implemented in distributed simulation, networked virtual environments and games (see Section 2). However, very few experiments have been performed to evaluate and compare interest management techniques, especially in the context of MMOGs. Furthermore, most evaluation that has been done used simulation with randomly generated data. It is not clear beforehand that results from random data will hold in a real world environment.

In this paper we compare and evaluate eight interest management algorithms in the context of MMOGs using real player data. Three of the algorithms we evaluate simply consider the radius around the player as the region of interest. The five others algorithms also take into account obstacles in the world and attempt to leverage the fact that a player does not need to be updated about objects that are occluded. In particular, we introduce scalable visibility-based and reachability-based interest management algorithms that use a triangulation of the world space as its base structure. Overall our custom "Tile Path Distance" algorithm shows good, general performance under a variety of workloads.

The remainder of the paper is structured as follows: Section 2 presents background on interest management, and overviews the related work in that area. Sections 3 and 4 illustrate the eight evaluated interest management algorithms and describe our experimental environment. Section 5 discusses the results, and the last section draws some conclusions and presents future work.

2. BACKGROUND AND RELATED WORK

The general goal of interest management is to reduce the cost of data communication in a distributed game. This cost depends of course on the underlying communication architecture, and also the particular IM scheme in use. Below we review relevant concepts and game work in these areas.

2.1 Communication Architecture

Data communication in larger multiplayer games can be performed using a variety of packet delivery methods. This includes basic unicast as the most popular current choice, but also broadcast and multicast approaches as well.

Standard unicast approaches in games focus on the difference between TCP and UDP protocols. UDP is a simple best-effort protocol that offers no reliability and no packet ordering guarantee. It has very little overhead, making it appropriate for highly interactive games (e.g., first-person shooter, car racing) where speed of packet delivery is paramount. TCP guarantees ordered delivery of packets; this simplifies application programming, at a cost of noticeable overhead. TCP also has the advantage of working more transparently across firewalls, and has ended up being the protocol of choice for many commercial MMOGs (e.g., EVE Online, Lineage II, and World of Warcraft).

In more restrictive settings actual broadcast can be used. Local area networks (LANs) can be configured to allow a single packet to be sent to all hosts simultaneously, in a manner similar to UDP. This can make transmission of state data in MMOGs extremely efficient and simple. Unfortunately, broadcast is not typically allowed across router boundaries, and internet-based MMOGs are not able to take practical advantage of this efficiency.

Multicast systems provide unreliable, group-based packet delivery; a host can subscribe to one or many multicast addresses and receive all messages sent to those addresses. Transmission is not quite as efficient as basic broadcast, but is usually much more efficient than multiple unicast operations. Interest management systems in games often specify multicasting as a means of efficiently implementing interest groups and associated network communication [20]. Unfortunately, not all ISPs provide access to the multicasting internet layers; access, firewall, and resource concerns mean multicasting is still not a general choice for MMOGs.

2.2 Interest Management

Interest management can be abstracted using a publish-subscribe model [17]. *Publishers* are objects that produce events, *subscribers* are objects that consume events, and an object can be both a publisher and subscriber (e.g. a player's avatar). In this model, interest management consists of computing a function that determines when a subscriber discovers a publisher, subscribes and unsubscribes to/from its updates.

Interest management schemes can usually be separated into two broad categories: space-based and class-based, or extrinsic and intrinsic respectively [16]. Space-based interest management is determined based on the relative position of objects in the virtual environment, while class-based is determined from an object's attributes (e.g., type of object).

Space-based interest management is usually based on proximity, and can be understood in terms of an *aura-nimbus* information model [5]. The *aura* is the area that bounds the presence of an object in space, while the nimbus or *area-of-interest* is the space in which an object can perceive other objects. In its simplest model both the aura and nimbus can be represented by fixed-size circles around the object. An object x is then aware of another object y when the nimbus of x intersects the aura of y. The pure aura-nimbus model has been implemented in many systems, such as MASSIVE-1 [9], Morgan et al.'s approach based on standard message-passing middleware [15], and commercial middleware such as through Quazal's *Duplication Spaces* technology [17]. In the latter case the implementation of the game-specific interest management function is left to the game developer, allowing for multiple, programmer-controlled publish-subscribe domains.

The advantage of a pure aura-nimbus implementation is that it allows fine-grained interest management in which only the relevant messages are sent to subscribers. It is especially suitable when there is a connection for each client with the server (e.g., TCP connection). The drawback of a pure aura-nimbus model is that it does not scale well because of the cost of computing the intersection between the area-of-interest and the auras of objects [19]. The computation of intersections between subscriber and publishers has a complexity of $O(MN)$ where M is the number of sub-

scribers and N the number of publishers in the world. This computation can become a bottleneck in systems without broadcast or multicast capabilities.

To mitigate the limitations of a pure aura-nimbus model, region-based interest management is used by many systems as an approximation [3, 7, 8, 12]. In region-based interest management the world space is first partitioned into static regions. The interest management determines the regions that intersects the area of interest of the subscriber and forms the *area-of-subscription* from the union of the intersected regions. The area-of-subscription represents an approximation of the true area-of-interest; this approximation, however, is often cheaper to compute than a pure aura-nimbus model. The quality of an interest region approximation is highly dependent on the shape and size of regions. Regular square tilings are quite popular and straightforward; studies have shown, however, that hexagons can better approximate the aura-nimbus model [7]. Other systems such as Spline allow the designer to create regions of any shape or size [4].

Similar to basic aura/nimbus designs, region-based interest management maps nicely onto multicasting. In NPSNET, for example, the space is divided into hexagons, and a multicast group is assigned to each hexagon [12]. A publisher sends events to the multicast group of the hexagon it occupies, and subscribers subscribe to multicast groups within their area-of-interest. If hexagon sizes are carefully chosen to be large enough, the number of subscribed groups of an object can be bounded, limiting the number of subscription and unsubscription requests. Region-based IM works best when objects are evenly distributed among the regions, and load balancing is important in situations where many objects gather in the same large region. The "three-tiered" IM partially addresses the load balancing problem by providing a dynamic subdivision of regions using an octree structure [3].

Interest management approaches discussed so far mostly consider an area-of-interest for the subscriber that is independent of the geography of the environment. Visibility-based interest management considers the vision of player instead of a fixed radius. *RING,* for instance, implements visibility-based interest management by dividing the environment into rectangular regions and precomputing visibility between regions [8]. At run-time a player will receive updates about objects that are within regions that are visible from his or her current region. Hosseini et al. [11] developed another visibility-based IM in which the visibility information is taken from each client's existing visibility culling performed in the course of graphic rendering. The advantage of this technique is that the visibility of objects is determined precisely and at no more cost since the information is already computed for rendering. Of course clients must first receive information about the position of all objects that may be visible in the world to be able to compute the visibility. Thus if position is your main source of messages, the technique is not advantageous.

Visibility requirements can be generalized, and audible range, radio contact, etc. are also bases for distributing game information. In general multiple forms of IM can be required in a game, each with different transmission and reception properties. Quazal's Duplication Spaces allows relatively arbitrary functionality to be used for regulating multiple, co-existing information domains [17]. *e-Agora* also uses a system of multiple, independent domains; e.g., both chat and navigation data [13].

Interest management need not always be a binary decision, and it has also been investigated with respect to scaling information quality. Han et al. build on the observation that some information (e.g., closer objects) is more relevant than others and make a distinction between high and low fidelity data. Users interested in a common area create groups for which a representative is elected and responsible to send low fidelity data to other peers. This allows for observation of distant areas, but at reduced scale, and thus reduced bandwidth requirements.

2.3 Comparing Interest Management

Our study here attempts to evaluate multiple IM schemes under multiple workloads. Others have also looked into the relative performance of different approaches, mostly using simulation or artificially-generated movement data. Han et al., for instance, compare their interest-based group approach with aura-based approach using simulation [10]. More detailed simulation results are given by Zou et al.; they evaluate two grouping techniques: cell-based and entity-based grouping for interest management with multicast. Extensive simulation study is then done to evaluate the trade-offs of group formation versus message dissemination [20]. Fiedler et al. compare the use of hexagonal versus rectangle tiling using randomly generated player movement. Their results show that a smaller number of channels are subscribed too when using hexagons. They also show that the use of smaller tiles result in a smaller percentage of the world being subscribed to and a smaller number of events received [7]. Funkhouser compares the RING visibility-based approach with full message broadcast using randomly generated player movement [8]. Morgan et al. evaluate the scalability of their middleware based interest management approach with randomly generated movement that attempt to reproduce realistic gathering of players by randomly positioning common targets within the world [15].

Using real user data from a military simulation, Rak and Van Hook [18] evaluate region-based IM under multicast. They find that while the smaller the region size the better the IM filtering, it is nevertheless expensive to subscribe and unsubscribe to multicast group: optimality represents a trade-off between the region size and the number of multicast groups.

3. INTEREST MANAGEMENT

We ran our interest management experiments inside *Mammoth*, a massively multiplayer online game development framework developed at McGill University [2]. Mammoth provides an implementation of a 2D game world in which each player has an avatar that can move around, pickup and drop items and talk to other players. Players and items are the only mutable objects in the world, all other objects are static (e.g. buildings, walls, trees, rivers, roads, etc.). Figure 1 shows a screenshot of the main Mammoth map.

Figure 1: Mammoth World Map with dimensions of 30x30.

For our experiments we implemented eight different interest management algorithms. They are described in the following subsections. The first three algorithms simply consider a circular area-of-interest around the player. The other five go a step further and try to leverage on the occlusion created by obstacles in the world.

3.1 Euclidean Distance Algorithm

The Euclidean distance algorithm (see Figure 3) is a simple implementation of the aura-nimbus model. The area-of-interest is a circle around the position of the player with a radius that covers the maximum distance a player can see. The aura is the position of the object. If the distance between an object and a player is smaller than the radius of the area-of-interest, the player subscribes to the object's updates.

The main advantages of this algorithm is that it is easy to implement, and computing the Euclidean distance between two points is inexpensive. The disadvantage is that the algorithm must compute the distance between all pairs of subscribers and publishers in the space. As the number of objects increases in the game, the algorithm does not scale well.

3.2 Square Tile Algorithm

The square tile algorithm (see Figure 4) is a region-based interest management that divides the world into equal-sized squares. The size of squares is set according to the radius of interest of players. At any location, the subscriber is interested in at most nine tiles, the subscriber's current tile and the eight (or less) neighboring tiles. Whenever a player performs an action, the action is broadcast to all players subscribed to the square in which the action has taken place.

The square tile algorithm scales well as the complexity of the computation to determine the zone of interest is constant. However, it is a rather bad approximation of the radius of interest of the player.

3.3 Hexagonal Tile Algorithm

The hexagonal tile algorithm (see Figure 5) divides the world into equal-sized, regular hexagons. A player subscribes to objects in the tiles that intersects its area-of-interest. Hexagonal tiles are known to be a better approximation of a player's circular area-of-interest.

3.4 Ray Visibility Algorithm

When using ray visibility, the only objects of interest are those that a player sees (see Figure 6). To determine if an object is visible to a player, we trace a line from the position of the player to the position of the object, up to a maximum length. If the line does not intersect with any obstacle in the world, the two objects are visible to each other.

Ray visibility is in a sense the perfect interest management algorithm, since it accurately calculates the exact area of interest of a player at a given point in time. It therefore provides the lower bound on the number of messages that must be exchanged between players, and an indication of the cost of a relatively expensive interest calculation. Note, however, that optimal visibility is not always desired: to prevent slow gameplay caused by network latency, it is often recommended to *pre-fetch* information about objects that are "soon to be discovered," if perhaps not actually visible yet. The inflexibility of ray visibility in this respect means that in practice it is more subject to problems such as the missed interaction problem [15] or late discovery of objects.

3.5 Triangulation of the World Space

The four next algorithms are original algorithms we designed to use triangular tiles. The advantage of triangular tiles is that they can accommodate arbitrary polygonal obstacles; i.e. exclude them from the partitioning of the world space. In this section we explain how we partitioned the world space into triangles, and then we introduce four algorithms that use the triangular tiles for interest management.

Converting to world space into a polygon with holes

Triangulation of a polygon is a well-studied problem in computational geometry, so the first step to partition the world into triangles is to transform the world space into a polygon with holes. The contour of the polygon is formed from the limits of the world and obstacles are represented by holes. The conversion is relatively straightforward for world spaces that are planar (or can be mapped to a plane). In a triangulation being used for interest management, however, the existence of long, thin triangles is undesirable; the distance between two points in relatively flattened triangle can be much larger than for regular triangles, and so they are less appropriate for approximating a player's area-of-interest. Our choice of triangulation algorithm reflects this requirement, and we further perform two additional preprocessing steps in order to get a suitable triangulation for interest management.

The first step is to remove obstacles smaller than a parameterizable threshold. This is motivated by the fact that small obstacles do not occlude a significant part of the world. A

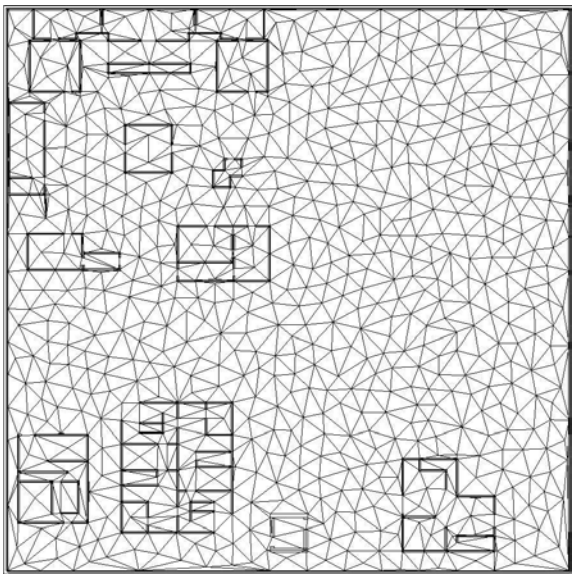

Figure 2: Delaunay triangulation of the Mammoth world map with a maximum area constraint of 1.0.

telephone post for example would occlude a very small space, furthermore, as soon as an avatar moves slightly to the right or left it will see any object hidden behind the pole. Hence, from an interest management point of view, small obstacles are not really significant. Furthermore triangulation with small obstacles results in some inconveniently small or thin triangles (since one edge has to follow the obstacle).

Arbitrarily wide, but quite thin obstacles also reduce triangle quality. Thin rectangular obstacles are thus converted to lines as a second preprocessing step. This allows us to avoid the generation of thin triangles that would have an edge on the shortest edge of a thin rectangle.

Delaunay triangulation with constrained area

The Delaunay triangulation [6] has the property of maximizing the minimum angle in every triangle in the triangulation; heuristically this will help in avoiding thin triangles. We also put a constraint on the maximum area of a triangle output by the triangulation. Extra *Steiner points* are added to ensure a maximum triangle size, and this allows us to explore fine or course-grained partitionings. Figure 2 shows the triangulation of the Mammoth world map (Figure 1) with an area constraint of 1. The map contains 155 rectangles that represent obstacles (i.e., walls), and the triangulation created 1678 triangles.

In the context of this work we assume a stationary environment in which obstacles do not change. However, efficient algorithms exists to re-triangulate a subset of a triangulation and could be used to dynamically re-create tiles for a part of the world space in which obstacles changed.

The triangulated world space can be stored as a graph in which each tile is a vertex and two vertices are connected by an edge if the corresponding tiles are neighbors (i.e., have a point in common that is not on an obstacle). Using a graph to store the partitioned space has the main advantage that

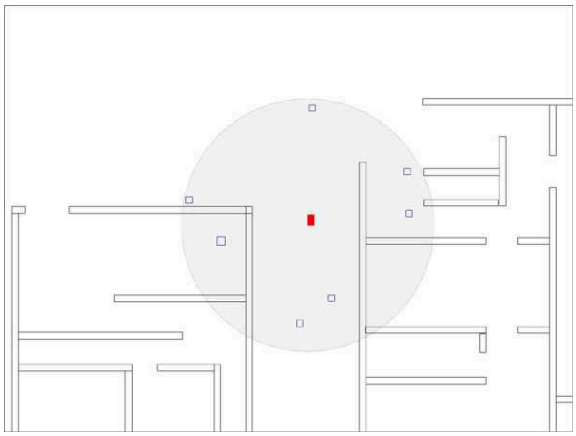

Figure 3: Euclidean distance algorithm with interest radius of 2.0.

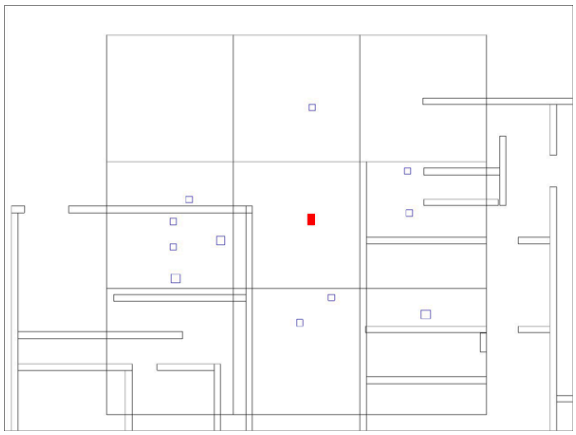

Figure 4: Square tiles algorithm with tiles of size 2.0 and one neighbor.

the problem of determining an area of interest can be done using a simple graph search. When using a breadth-first search rooted at the current tile of a player, only a localized subset of tiles that is proportional to the size of the player's area-of-interest is visited. Furthermore the graph also encodes information about occlusion of space: two regions that are close in Euclidean distance but separated by an obstacle are equally separated in the graph.

3.6 Tile Distance

The tile distance algorithm (see Figure 7) is based on the Euclidean distance between a player and a triangular tile. The set of tiles of interest for a player is computed as the set of tiles connected to the current tile of the player that intersect the player's area of interest. The algorithm implements a breadth-first search from the player's current tile. The tile distance algorithm is an approximation of the player's area of interest, but also has the property that tiles that are not reachable within the player's area-of-interest are ignored. We can see this property of the algorithm in Figure 7: the tiles inside the building on the right are visible because they are connected to the player's current tile. On the other hand

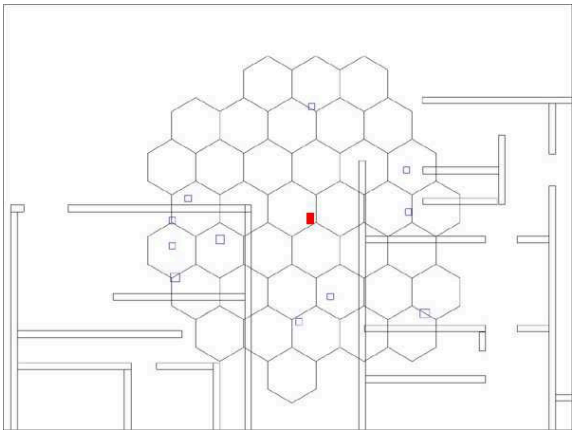

Figure 5: Hexagon tiles of area size 1.0 with interest radius of 2.0.

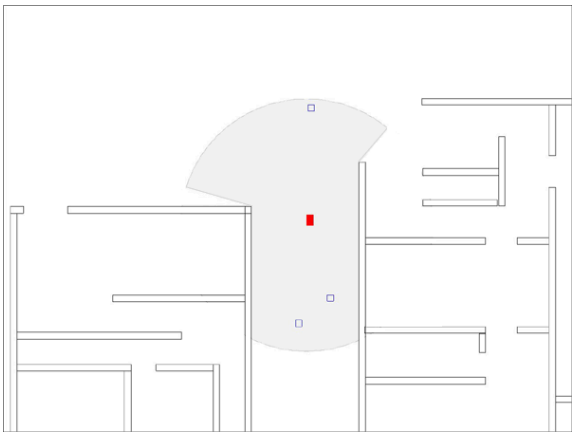

Figure 6: Ray visibility algorithm with interest radius of 2.0.

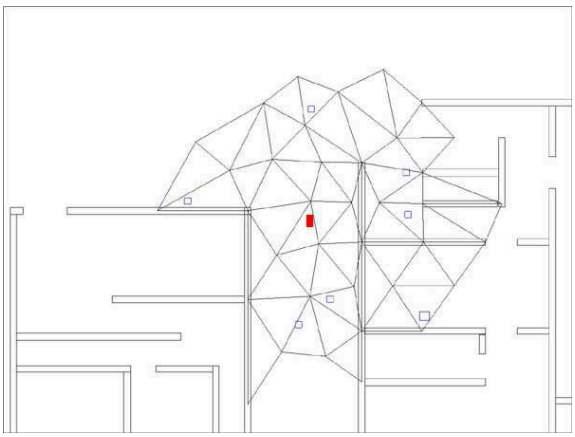

Figure 7: Tile Distance algorithm with interest radius of 2.0.

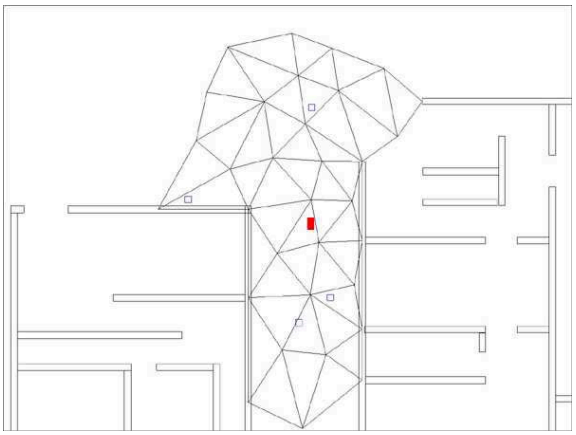

Figure 8: Tile visibility algorithm with interest radius of 2.0.

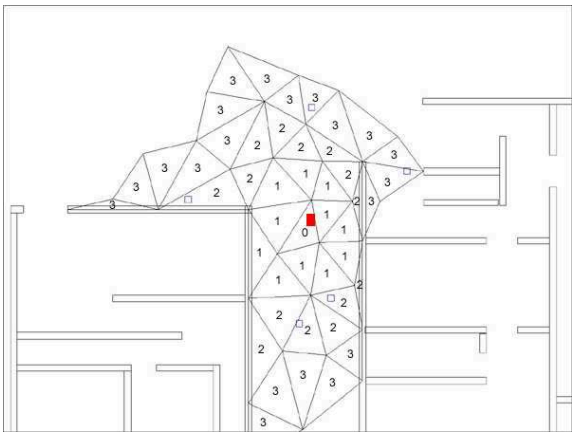

Figure 9: Tile neighbor algorithm with a depth of 3 neighbors. The numbers indicate the depth from the player's tile.

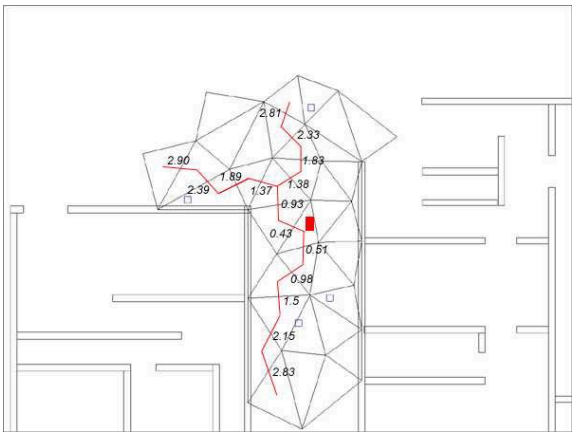

Figure 10: Path distance algorithm with a path length $\leq$ 3.0. The numbers indicate the path length from the player's tile.

the tiles inside the building on the left are not chosen because there is no path within the interest radius connecting them to the player.

3.7 Tile Visibility

The tile visibility algorithm (see Figure 8) is based on the visibility between tiles. For efficiency reasons, the algorithm involves a precomputing step during which the visibility between each tile is computed. A tile is considered visible from another tile if there exist a point in each of the two tiles that can be connected by a line segment that does not intersect an obstacle. The algorithm takes advantage of the fact that the visibility of tiles is static, as opposed to the visibility of players that changes with their position. The algorithm approximates the visibility of a player by selecting the tiles that are visible from the player's current tile.

3.8 Tile Neighbor

The tile neighbor algorithm (see Figure 9) determines tiles of interest for a player based on neighbor relationships between tiles. The algorithm performs a breadth-first search from the current tile of the player, and collects all the tiles until it reaches a given depth. For example, for a depth of one, the algorithm will only collect the immediate neighbors of the current tile. Figure 9 shows an example with a depth of three.

The main advantage of this algorithm is that it is very simple to compute. The disadvantage is that since the shape and size of triangles is not uniform, it is difficult to predict if a given depth will fully cover the area of interest of a player.

3.9 Tile Path Distance

The path distance algorithm (see Figure 10) is somewhat similar to the neighbor algorithm, but instead of taking the graph depth as a limiter for its search, it takes the shortest-path distance. We define the path distance between two tiles as the sum of the distances between the centers of the triangles connecting the two tiles. The intuition behind the algorithm is that the subscriber is interested in the tiles within a certain "reachable" distance from his current position. The algorithm is an approximation of that distance. The algorithm has the same problem as the Neighbor algorithm, i.e. it cannot guarantee to fully cover the area of interest of a player. However, since it uses a criteria based on the distance, it tolerates cases of abnormal triangles. This property can be observed by comparing Figure 9 and 10; the path distance algorithm has a more "roundish" shape, for instance it cuts out the thin triangle on the left side along the wall.

The path-distance algorithm uses static information (such as the distance between neighboring triangle centers) that can be pre-computed or cached. It can also easily adjust to different interest radii.

4. EXPERIMENTAL SETTING

In this section we describe the environment and methodology we used to perform experiments with interest managements. The goal of the experiments was threefold:

- Evaluate and compare the use of different interest management algorithms in an MMOG-like setting.
- Evaluate the feasibility of using triangulation-based tiling to perform obstacle-aware interest management.
- Compare results obtained from real-player traces with results from randomly generated traces.

The eight algorithms described in Section 3 were implemented within the Mammoth framework. Each experiment consisted of a replay of trace data collected from either the movements of real-players playing a non-trivial multiplayer game (*Orbius,* described below), or by using artificial, randomly generated data.

4.1 Mammoth Implementation Details

The Mammoth framework has a client-server communication architecture, each client is connected to the server through a TCP/IP connection and exchanges messages with the server only. On the server-side virtual channels are implemented on top of Java NIO, and clients can subscribe to an unlimited number of channels. In Mammoth, subscription and unsubscription to channels is a relatively fast server-side only operation. A message sent on a channel is unicast to each client subscribed to that channel.

To implement interest management within the framework, we assign one virtual channel to each publisher. Publishers publish events on their channel only. Subscribers (i.e., players) subscribe to the channels of objects within their area-of-interest. The interest management is computed on the server at a set rate. When a publisher match the interest of a subscriber that does not already have a subscription to that object, the server sends a copy of the object within a *content message* and subscribes the subscriber to the channel of the publisher so it will receive future updates (i.e., *update messages*). The server also unsubscribes a subscriber if it detects that it is no longer interested in a particular object.

4.2 The Orbius Game Trace

The real-player trace that was used for experiments was collected in a gaming event involving 28 participants playing a custom Mammoth game implementation, *Orbius.* The Orbius game was designed to reflect the general characteristics of larger multiplayer games. Players had to explore the world, collaborate as teams, and interact with opposing teams in order to win the game. The game was played in the 30x30 world shown in Figure 1, and a single player has size 0.1.

The Orbius game trace was replayed for the experiment using "bot" game clients that can read the trace and replay the actions of a player. For the experiments below, 28 bots (one per player trace) were run on seven computers (four bots per computer) connected to the server through a local area network.

Note that due to the distributed nature of the system, there are multiple nondeterministic factors that makes it impossible to guarantee an identical replay of the same game. We computed a variation by replaying the same experiment five

times and found that the largest variation in number of messages between experiments was of 0.4%.

4.3 Random Traces

Two random player movement traces were generated to compare with Orbius data. Both random traces were designed to send the same number of movements over the same period of time as the Orbius trace. Each of 28 clients sends one random move of length 0.5 in a randomly-chosen direction, at a constant rate of one move every 740 milliseconds; these parameters were derived from the average number of message sent by players divided by the experiment length. The main difference between the two random traces is the starting position of players. In the first trace players were all initialized in the space outside of buildings, similar to the way Orbius players were initialized; in the second trace 4 players were initialized outside and 24 inside of buildings—14 were initialized within the same building.

There are two main reasons for considering these variations in random traces. From our observations of random movements players initialized outside buildings are unlikely to end up inside of buildings, and symmetrically players initialized inside a building are unlikely to exit from that building. This has the potential to make a significant difference in interest management performance—players inside a building have many opportunities for occluded sight, and thus may be able to take better advantage of visibility-based algorithms. Relative distribution is another important property with respect to region-based IM; the second trace thus also allows us to examine a scenario in which half of the players are located in one place rather than being more uniformly distributed.

4.4 Measurements

To evaluate our implementations we considered two main types of messages between the server and clients: *content messages* and *update messages*. Content messages are used when an object is discovered by a player that has no copy of that object; a full copy of the state of the target object is sent to the player—this represents a replication cost. Update messages are sent to existing subscribers to inform them of a change in the state of an object that is already known. In our experiments these are primarily player position updates. Content messages are generally more expensive because they contain the full state of an object while update messages contain only a partial state. In the Orbius trace, we calculated that on average a content message was 1.5 times the size of an update message.

For each experiment we measured the number of each type of message received at each client and computed an average over the 28 clients. We also measured the CPU consumption of the dedicated machine running the server using JSysmon [1]. CPU utilization was polled every second, averaged over the total length of the experiment (12 minutes), and was gathered on a dual-core 3GHz Pentium D with 2GB of memory.

5. RESULTS

In this section we present results from experiments we conducted. First, based on the Orbius (real player) movement data, we discuss the trade-offs provided by choice of tile size in terms of message filtering capability versus CPU overhead. We then compare the effectiveness at relevance filtering of the eight algorithms, followed by results on the scalability of our different IM approaches. Finally, we compare results obtained from real-player traces with results obtained from randomly generated traces. This is intended to give some indication of the kinds and magnitude of factors in workload data that can influence or obscure results.

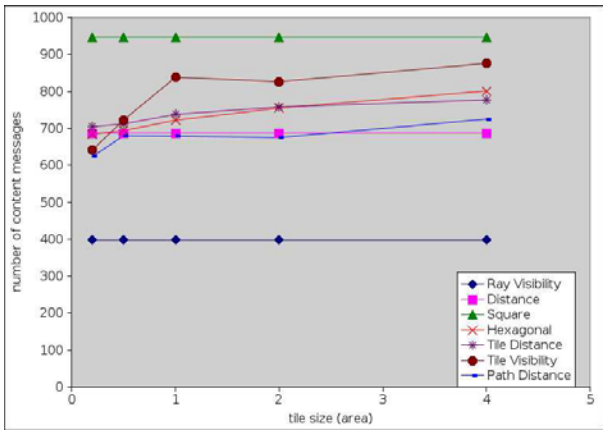

Figure 11: Average number of content messages received by a client for varying tile areas.

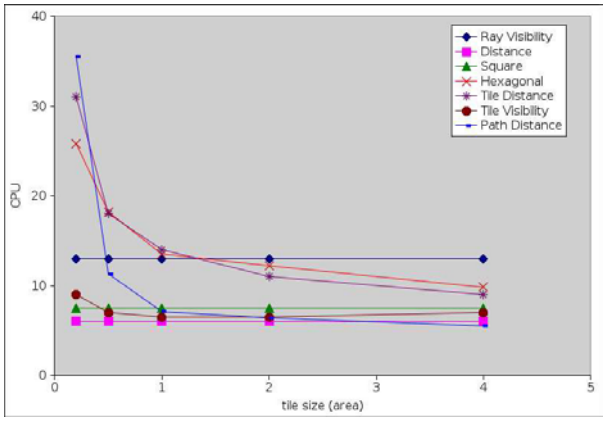

Figure 12: CPU consumption of the server for different tile areas.

5.1 Tile Size

We investigated the effect of changing the tile size (area). Our hypothesis is that smaller tiles would better approximate the player's area-of-interest and filter more irrelevant messages; however, it would also have a higher computation overhead.

Figure 11 shows the average number of content messages sent from the server to a client for different tile sizes. The size of tiles is expressed in terms of average area; in the case of triangular tiles we use the maximum area since our triangulation algorithm does not guarantee an equal area for

each triangle. The players' interest radius is fixed to five, a reasonable value based on player experiences in the Orbius game.

In most cases a smaller tile size results in slightly less content messages, with the Tile Visibility algorithm gaining the most from a smaller tile size. The effect is surprisingly subtle; even with 28 players Mammoth is not a densely populated world, and tile size does not have an overall large impact. Unsurprisingly Ray Visibility and (Euclidean) Distance do not change; these IM schemes are based on absolute distance, and tiling has no direct impact. Square also does not change since its tiling does not change for a fixed interest radius.

Figure 12 shows the average CPU consumption at the server for each tile size. Here the impact of tile size is more obvious. Algorithms such as Hexagonal, Tile Distance, Neighbor, and Path Distance make use of breadth-first search, and naturally as tile size decreases the number of tiles that must be visited increases, and so does CPU cost. Tile Visibility displays the smallest increase due to reduced tile size; this is likely due to using precomputed information rather than dynamic searches in order to determine tile visibility.

The results of Figure 11 and 12 suggest two things. First, the gains from the use of smaller IM regions are not necessarily large, and may depend on the game environment. Secondly, while there is a definite and large tradeoff between quality of the approximation and the overhead cost of smaller tiles, particularly at very small tile sizes, preprocessing of the environment can greatly mitigate the costs. For our subsequent experiments we used a tile size of 1.0, as a reasonable point where improvements in the number of content messages are mostly realized while CPU cost is still not excessive.

5.2 Message Filtering

Message filtering is the primary role of IM; in order to determine the effectiveness of the different IM algorithms we measured the number of update and content messages received by each client from the server. Figures 13 and 14 show the average number of messages per client for the eight interest management algorithms under different interest radii. The Ray Visibility algorithm represents a theoretically perfect filtering of messages and can be used as a reference to evaluate the efficacy of other algorithms.

The first observation to be made is that there is a considerable difference in number of messages between Ray Visibility and Distance. This suggests that there is considerable potential to reduce the number of messages by taking advantage of obstacles—IM approaches that consider visibility should perform significantly better at filtering.

By far the worst filtering is provided by the square tiling. For the smallest radius of interest the number of messages sent (update or content) is more than twice the number of Ray Visibility. This trend only gets worse as the radius of interest increases: our square tiling algorithm guarantees a fixed region of nine tiles in all cases, and so as the tile size increases to accommodate a larger visibility radius the squares provide a worse and worse approximation of the real

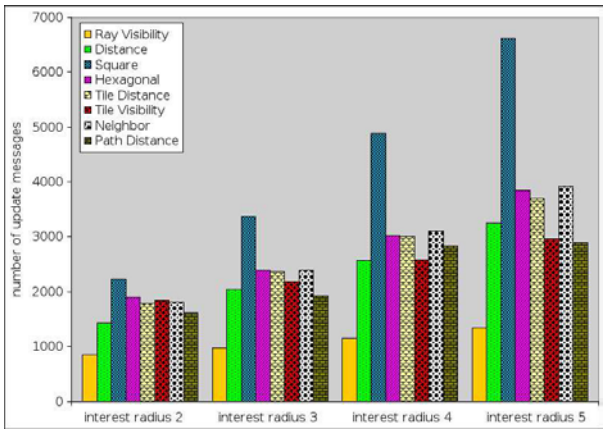

Figure 13: Average number of update messages per player with various interest radius sizes.

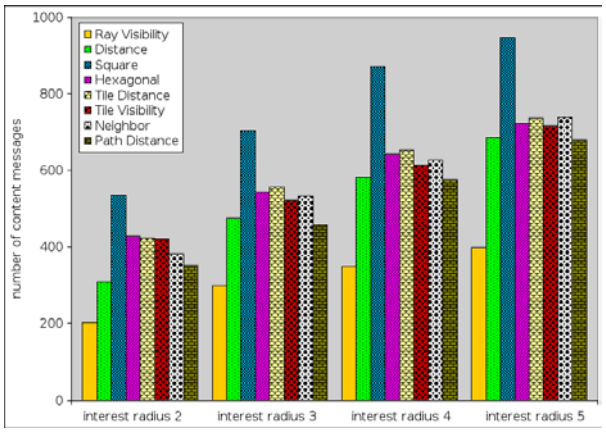

Figure 14: Average number of content messages per player with various interest radius sizes.

player interest region. Regular tilings do not take advantage of actual visibility, and this is also exacerbated by a larger interest radius, which heuristically includes more obstacles.

Hexagonal tiles represent a significant improvement over square tiles, an observation also made by others [7]. If we compare the number of messages for the hexagonal tiling with our pure (Euclidean) Distance in Figure 13 and 14 we find there are only 17% more update messages and 14% more content messages. This improvement over square tiles is mainly due to the accuracy of approximating the real region of interest. Neither our square nor hexagonal tile based approaches take advantage of actual visibility, but as the interest radius increases in proportion to the tile size hexagonal tilings better approximate a circular region of interest.

Tile Distance is an obstacle-aware algorithm, and thus should show improved results over Hexagonal. Here, however, it performs only marginally better in terms of update messages, and marginally worse with respect to content messages. Obstacles in Mammoth are many, but mainly concentrated in a few areas, and Orbius players tended to stay largely outside. Moreover, obstacles in Mammoth do not tend to result in complex, maze-like environments; tiles with-

in a given area-of-interest tend to be connected and thus included in the tile distance. Overall Tile Distance reduces only about 6% of update messages and 1% of content message over an algorithm that considers all triangles intersecting the area-of-interest irrespective of connectivity.

The Neighbor algorithm filters slightly less update messages than Hexagonal in most cases, and filters only slightly more content messages. The problem with Neighbor arises from the difficulty in determining the depth that matches a given interest radius, primarily due to the fact that tiles have irregular sizes. For our experiments we determined the depth qualitatively by choosing a value that would fully cover the area-of-interest in a surface where there is no obstacle.

Tile Visibility can filter more update messages and slightly less content messages than Hexagonal tiles. The advantage of Tile Visibility seems to grow with the increase of the interest radius. For an interest radius of two it filters 3% more update messages than Hexagonal, and for an interest radius of five, 23% more. We explain this increase in effectiveness by the fact that obstacles occlude a larger proportion of the area-of-interest when the surface is greater.

Path Distance performs better than the similar Neighbor algorithm; it filters about 16% more update messages and 10% more content messages. This can be mostly attributed to the fact that Path Distance is easier to adjust to a given interest radius. Path distance also performs slightly better than Tile Visibility in most cases (5% less update messages and 10% less content messages), but is subject to approximation errors—it is possible that the tile area does not fully cover the interest radius of the player. Path Distance does, however, have a potential advantage in allowing for more "incremental" discovery of the world. Under Tile Visibility large groups of tiles can be added to the area-of-interest by player moving only a small distance, peeking past a corner for instance. Path Distance will have already included many of the newly visible triangles. Overall Path Distance is the algorithm that has the closest results to the Ray Visibility lower bound, generating about twice the number of messages.

The results suggest that both visibility and path distance seems to be reasonable algorithms to perform interest management in a world with obstacles. The Path Distance algorithm, however, has a few practical advantages over the Tile Visibility algorithm. Most importantly it does not require a complex preprocessing step, and thus is much more accommodating of changes in parameters, such as the interest radius.

5.3 Scalability

To evaluate the scalability of the algorithms in densely populated areas we increased the number of objects in the world and measured the number of content message received by each player, as well as the CPU consumption of the server. The first run of the experiment had 186 objects which was the initial number of objects put by the designer in the map; here we experiment with 1000, 2000, and 3000 objects.

Figure 15 shows the increase in average number of content messages received by each client, and Figure 16 shows the

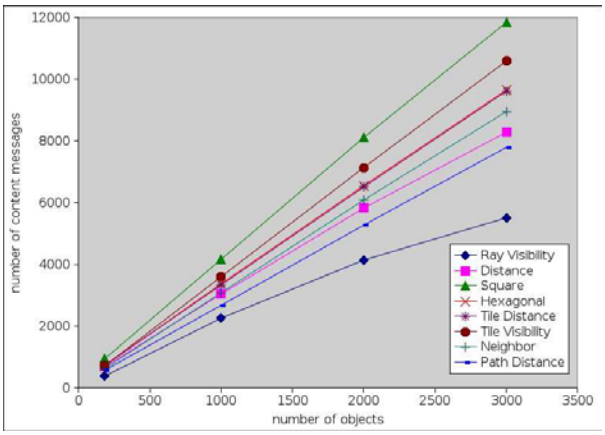

Figure 15: Average number of content messages per player with an increasing number of objects in the world.

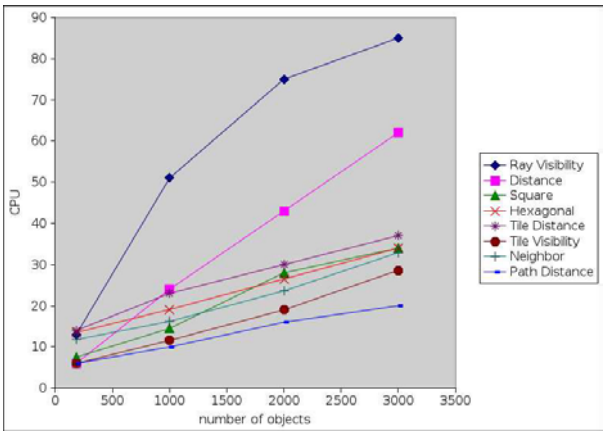

Figure 16: CPU consumption with an increasing number of objects in the world.

server CPU consumption corresponding with the increase in object density. The former shows a linear relation for most of our algorithms; content messages themselves are not a major source of scalability concern. CPU consumption shows much more separation. Here it is clear that algorithms based on tiles have a great scaling advantage over algorithms such as Ray Visibility and Distance. A subscriber in a tile-based situation is localized to tiles that are close to the subscriber while for the other two algorithms the interest computation is done with every publisher in the world. This makes tiling much more appealing at all but the lowest density of game objects.

5.4 Real-Player versus Random Movements

Many interest management analyses make use of randomized data. Real player behaviour, however, has potential to be quite different from any simple randomized model. To determine if experiments using real player traces and experiments using randomly generated traces would give similar results, we compared the results from our Orbius trace with the results from the two randomly generated traces (see Section 4).

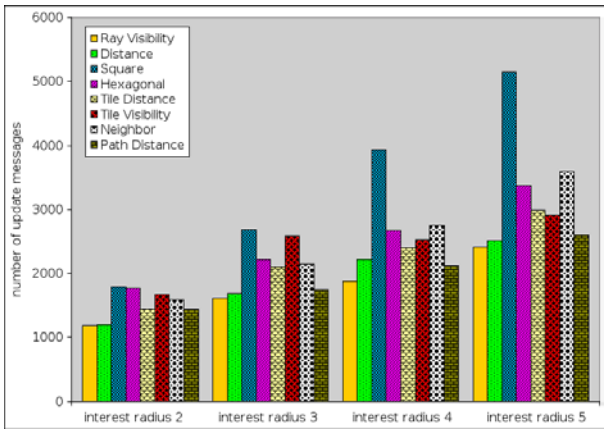

Figure 17: Average number of update messages per player with various interest radius sizes for random data starting outside buildings.

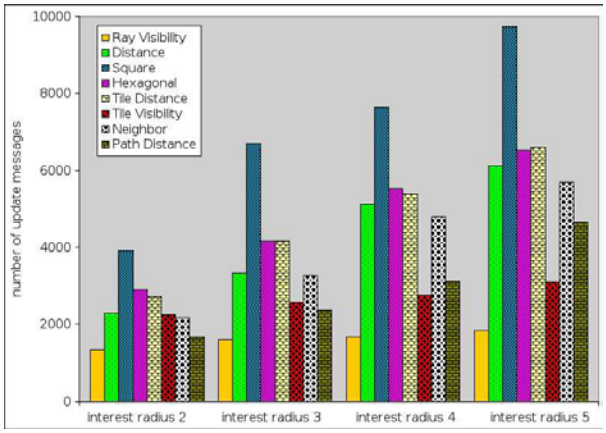

Figure 18: Average number of update messages per player with various interest radius sizes for random data starting mostly inside buildings.

Figure 13, 17, and 18 shows the average number of update messages received per player for the three sets of traces (i.e., Orbius, random outside, and random inside). The three traces give similar relative results between algorithms. For instance, in most cases Tile Visibility and Path Distance algorithms perform better than Square and Hexagonal tiles. In absolute terms, however, there is considerable difference in the number of messages received. For example, Tile Visibility filters 40% more messages over Hexagonal with the random inside trace, only 6% more with the random outside trace, and 12% with the Orbius trace. The Distance algorithm nearly matches Ray Visibility in random outside, but is much worse in random inside.

These differences correlate with the general properties of player behaviour in these games. In random inside players largely move in an environment dense with obstacles, and thus obstacle-aware algorithms do quite well. Players in random outside are far from obstacles, and obstacle-aware algorithms do not improve the performance nearly as much.

Algorithm	RO Avg	RO Max	RI Avg	RI Max
Ray Visibility	-37.9%	-44.3%	-33.4%	-38.8%
Distance	17.5%	22.8%	-43.2%	-49.7%
Square	20.4%	22.2%	-40.3%	-49.6%
Hexagonal	9.3%	12.1%	-41.0%	-45.4%
Tile Distance	17.4%	20.0%	-41.3%	-44.2%
Tile Visibility	0.5%	-15.4%	-11.0%	-18.6%
Neighbor	10.5%	12.0%	-27.4%	-35.3%
Path Distance	13.7%	25.1%	-17.4%	-37.8%

Table 1: Relative difference in number of update messages between Orbius data and the two Random data sets. Columns 2 and 3 show the change from Random Outside to Orbius, while columns 4 and 5 show the relation between Random Inside and Orbius. Negative values indicates fewer messages for Orbius, and max difference is in terms of absolute value.

This is also evident in Table 1. Orbius data has more of a mixture of inside and outside movements, and Ray Visibility and Tile Visibility thus perform better in Orbius than in outside random data, and show less improvement in Orbius with respect to the use of obstacle-dense, random inside data. Table 1 further shows the variance induced by the different workloads. Real game movements are particularly amenable to visibility based schemes, with Path Distance having overall good absolute and relative performance.

6. CONCLUSION AND FUTURE WORK

Good interest management designs are important to good network performance in massively multiplayer games, and we have compared a variety of algorithms incorporating various levels of visibility and map conformance. Experiments within the Mammoth framework have shown that taking obstacles into account reduces the number of update messages that have to be sent between players by up to a factor of 6. Not surprisingly the technique is especially effective when there are many obstacles, e.g. within buildings.

We have shown that it is possible to define regions of interest based on a partitioning of the world space into triangular tiles. Among the different tile-based interest management algorithms, our own "Tile Path Distance" (see Section 3.9) seems to exhibit the most interesting properties. The number of update messages that have to be sent between players is the closest to the ideal number (given by the Ray Visibility algorithm), but the computational effort required to run the algorithm is 3 to 6 times lower! In addition, the unnecessary update messages sent to a player are the ones concerning game state that is very likely to be of interest to the player in a near future, which can increase game responsiveness in case of network lag.

Finally, we have demonstrated that measurements taken during a game using computer-controlled players performing random movements can be used to predict measurements taken during a game with real human players. However, the starting position of the random players has to be chosen carefully; factors such as the ratio of players starting inside buildings or other closed spaces compared to those starting outside can have a significant impact, and need to reflect the situation in the real game.

Based on the observations of this paper we intend to investigate the performance of hybrid / adaptive interest management algorithms in the future. For instance, it makes sense to use a fast distance-based interest management algorithm such as "Euclidean Distance" when players are mainly outside (in an area with very few obstacles), and then switch to a reachability-based interest management algorithm such as "Path Distance" when players congregate inside buildings or other areas with many obstacles.

The evaluation of triangular partitioning was limited to a comparison with simple square and hexagonal partitioning, in the future we would like to compare triangular partitioning with more dynamic space partitioning structure such as BSP trees, R-trees, and octrees.

Our experiments were also limited to a stationary environment in which obstacles do not change position or shape. In the future we would like to use dynamic re-triangulation of the world space with our path distance algorithm in order to support changes to the environment. We also plan to port our current 2D visibility model to a 3D environment as soon as the Mammoth framework provides a 3D engine.

Partitioning the world into triangles is not only useful for interest management, but also for pathfinding and dynamic zoning. In the future we intend to work on integrating the algorithms dealing with these three concerns, and experiment with them in a distributed setting with multiple servers.

7. ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable comments. This research has been supported by the National Science and Engineering Research Council of Canada and the Canadian Foundation for Innovation.

8. REFERENCES

[1] JSysmon. `http://jsysmon.sourceforge.net`.

[2] Mammoth: The massively multiplayer prototype. `http://mammoth.cs.mcgill.ca`, 2006.

[3] H. Abrams, K. Watsen, and M. Zyda. Three-tiered interest management for large-scale virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 125–129, 1998.

[4] J. W. Barrus, R. C. Waters, and D. B. Anderson. Locales and beacons: efficent and precise support for large multi-user virtual environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 204–213, 1996.

[5] S. Benford and L. E. Fahlen. A spatial model of interaction in large virtual environments. In *Third European Conference on Computer Supported Cooperative Work*, pages 107–123, 1993.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications*. Springer, 1997.

[7] S. Fiedler, M. Wallner, and M. Weber. A communication architecture for massive multiplayer games. In *Proceedings of the 1st workshop on Network and system support for games*, pages 14–22, 2002.

[8] T. A. Funkhouser. RING: a client-server system for multi-user virtual environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 85–92, 1995.

[9] C. Greenhalgh. Awareness-based communication management in the MASSIVE systems. *Distributed Systems Engineering*, vol. 5, no. 3:129–137, 1998.

[10] S. Han, M. Lim, and D. Lee. Scalable interest management using interest group based filtering for large networked virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 103–108, 2000.

[11] M. Hosseini, S. Pettifer, and N. D. Georganas. Visibility-based interest management in collaborative virtual environments. In *Proceedings of the 4th international conference on Collaborative virtual environments*, pages 143–144, 2002.

[12] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham. Exploiting reality with multicast groups. *IEEE Comput. Graph. Appl.*, 15(5):38–45, 1995.

[13] M. Masa and J. Žára. Generalized interest management in virtual environments. In *Proceedings of the 4th international conference on Collaborative virtual environments*, pages 149–150, 2002.

[14] R. D. P. McFarlane. Network software architecture for real-time massively-multiplayer online games. Master's thesis, McGill University, 2005.

[15] G. Morgan, F. Lu, and K. Storey. Interest management middleware for networked games. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 57–64, 2005.

[16] K. L. Morse. Interest management in large-scale distributed simulations. Technical report, Department of Information & Computer Science, University of California, Irvine, 1996.

[17] Quazal. *Duplication Spaces^{TM} Quazal Multiplayer Connectivity White Paper*, January 2002. `http://www.quazal.com`.

[18] S. J. Rak and D. J. V. Hook. Evaluation of grid-based relevance filtering for multicast group assignment, 1996.

[19] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. SIGGRAPH Series. Addison-Wesley and ACM Press, New York, 1999.

[20] L. Zou, M. H. Ammar, and C. Diot. An evaluation of grouping techniques for state dissemination in networked multi-user games. In *Proceedings of the ninth Internation Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 33–40, 2001.