Visibility Path-finding in relation to Hybrid Strategy-based Models in Distributed Interactive Applications

Dermot Madden, Declan Delaney
Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare.
decland@cs.may.ie

Séamus McLoone, Tomás Ward
Department of Electronic Engineering, National University of Ireland, Maynooth, Co. Kildare.
{tomas.ward, seamus.mcloone}@eeng.may.ie

Abstract

The hybrid strategy-based modeling approach is a method for reducing the number of network packets that need to be transmitted to maintain global consistency in Distributed Interactive Applications. It combines a short-term model such as dead reckoning with a long-term strategy model. A key aspect of this approach is to determine strategies that users adopt in navigating the simulated environment to satisfy some objective or goal. Computer-generated artificial entities called BOTS, navigate by employing an Artificial Intelligence technique called path finding. This paper proposes using the A path finding algorithm to automatically compute strategies that human users might take through the simulated environment. Since the A* algorithm operates on a graph representation of the environment and because of the real-time constraints imposed on Distributed Interactive Applications, the paper also carries out a comparative analysis of two extreme graph representations of the environment – a standard regular grid and a minimal grid representation. The comparison shows that the minimal grid leads to an order of magnitude reduction in real-time computation compared to the regular grid. In addition the paths computed using the minimal grid and the A* algorithm are used to determine strategy models as part of the hybrid strategy-based modeling approach. It is shown that this reduces the network traffic required to maintain global consistency of entity dynamics in two simulated environments.*

1. Introduction

Networked computer programs that allow users to interact in real-time in a simulated environment are known as Distributed Interactive Applications (DIAs). One of the major challenges facing the development and deployment of DIAs is the maintenance of a consistent world view for all participants in the face of network latency. Several techniques for combating latency have been documented and implemented. Among these, the hybrid strategy-based modeling technique has shown a reduction in the number of packets that needs to be transmitted across the network to maintain global consistency [1]. This technique operates by identifying long-term strategies that users may adopt in the pursuit of goals and only communicating the current user strategy to other participants. It is a hybrid approach in that it combines a short-term dead reckoning model with one of several possible long-term strategy models. One key difficulty with this approach is the automatic determination of possible strategies users may adopt to achieve a given goal. In existing DIAs such as computer games, computer-generated artificial entities called BOTS, face the same difficulty – given an initial location and an objective, what is the best method to adopt to reach the objective while minimizing some cost function? This problem is referred to as path finding [2].

Path finding is an essential component of the artificial intelligence (AI) that makes BOTs appear more like human users. In DIAs such as networked games, up to 30% of computational time is spent on AI computations, predominantly path finding [3]. Path-finding algorithms consist of two parts:

1. the generation of a search graph that represents the underlining environment and
2. an algorithm that searches the graph representation to find a path connecting given initial and target locations.

The graph representation of a simulated environment represents the environment by a series of nodes, with paths through the environment being constrained to pass from one node to another. A minimal set of nodes can be achieved by using a points of visibility graph or minimal grid. Other possible representations of the environment exist. Probabilistic roadmaps randomly place nodes on a map, connecting newly placed nodes to nearby nodes [4]. Navigational Meshes split the simulated environment into

regions that are free from obstacles, only allowing navigation between these regions [5]. Potential Field methods give the target location an attractive force and obstacles repulsive forces; entities then follow the potential field to the target [6]. Often nodes are manually placed within the environment or the computed nodes are tweaked manually to improve the performance of BOTs, as in Unreal Tournament [7].

Path finding algorithms compute the path through the environment under some constraint using the node representation of the environment. The most widely used and arguably the best path finding algorithm used in DIAs is the A* (A star) heuristic search [8], although others exist such as Breadth-first search, Depth-first search and Dijkstra's algorithm [9].

The graph representation of the environment used in path finding can take many forms. Examples include a regular grid, a visibility grid or a navigational mesh. However there is a dearth of comparative results between various grid representations and path finding algorithms. The first part of this paper provides an initial contribution to this area by comparing an implementation of the A* path finding algorithm on both a visibility graph and regular graph representation of a simulated environment. It is shown that the visibility graph provides an order of magnitude saving in computational time compared to the same environment represented by a regular grid. The paper also proposes a novel technique based on the visibility graph and A* algorithm to automatically determine strategies users may adopt in navigating a simulated environment. Two sample environments are used to illustrate how strategies can be automatically computed and data will be presented to show that these strategies lead to a reduction in network packets compared to pure dead reckoning when used as part of the hybrid strategy-based modeling technique.

In the following section we describe the regular-grid and visibility graph representations of the underlying environment. Section 3 presents a comparative analysis of the A* algorithm using a regular-graph and a points of visibility or minimal graph. The application of the A* algorithm and the minimal grid to the problem of computing strategy models in a given simulated environment is described in section 4 and the results of simulating network traffic using such a strategy model is given in section 5. The paper then ends with some concluding remarks and an indication of future work in section 6.

2. Graph Representations of the Environment

In this section two possible representations of the simulated environment will be described: a regular search graph [10] and a minimal grid or visibility graph [11]. In general these representations are at opposite ends of the node spectrum, in that the minimal grid represents the environment by associating nodes with each obstacle, whereas the regular grid covers all areas of the environment that are free from obstacles with nodes. Therefore, the actual number of nodes in each case is a function of the number and size of obstacles within the environment.

2.1 Regular Graph

At the simplest level simulated environments consist of obstacles that must be avoided and obstacle-free or ground areas that can be freely navigated. Figure 1a shows an environment with a single rectangular obstacle. A regular search graph is formed by overlaying a regular grid of nodes on the ground area. Each node is connected to adjacent nodes and an entity can move from the node it currently finds itself to any node connected to that node. Such a regular grid is illustrated in Figure 1b. The absolute grid spacing or grid spatial resolution is dictated by the units of measure within the simulated environment, each node on the grid being one unit from other nodes in both a horizontal and vertical direction [10, 12]. Given such a grid together with both start and target locations, we can then search for a set of possible paths connecting the start and target nodes. Figure 1c shows the connected grid, while Figure 1d shows the shortest path solution linking the start and target nodes.

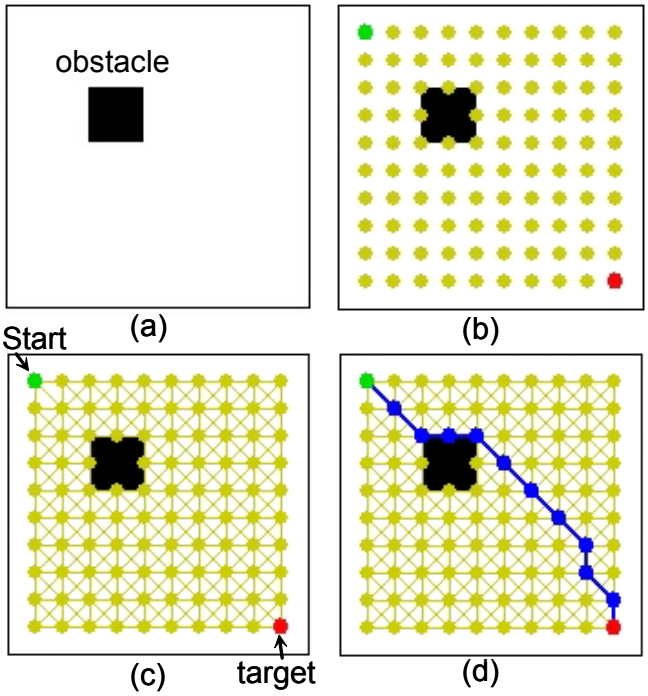


Figure 1: (a) Ground area with obstacle; (b) Regular grid; (c) Connected regular grid with initial and target nodes; (d) Connected grid with path solution.

2.2 Visibility Graph

In contrast to a regular graph, a visibility graph locates the nodes at the vertices of obstacles, or at locations that will accurately delimit the obstacle. Each node is then connected to all other nodes that are visible to it. This requires checking whether nodes are in line of sight of each other. Figure 2a shows the environment with a square obstacle. In Figure 2b nodes representing the vertices of the obstacle together with both start and target nodes are shown. Figure 2c demonstrates some of the paths that might be taken by an entity moving from the start to target node without going through the obstacle. The shortest path is indicated in Figure 2d.

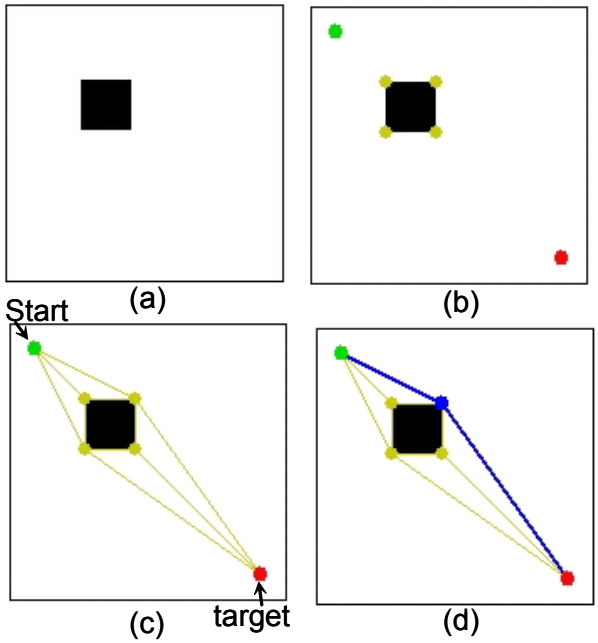


Figure 2: (a) Ground area and obstacle; (b) Visibility graph with start and target nodes; (c) Connected visibility graph; (d) Connected graph with path solution.

In this example the initial and target nodes are connected to all other visible nodes. Other connection criteria can be used; the initial and target nodes may be connected to the nearest visibility graph node, or to the k nearest nodes, or to all visible nodes within a set distance. Obviously the possibility of a direct connection between the initial and target nodes should be tested for. Visibility graphs themselves are generated offline and it has been shown that visibility graphs can be generated in O*(nlogn + E)* time [13], where *n* is the number of nodes and *E* is the number of edges in the graphs.

In the next section the regular and visibility graph representation of the environment will be used by the A* path finding algorithm to discover the shortest path through an environment. This will provide a comparative analysis and greater understanding of the two representations.

3. An Analysis of A* using Two grid Types

A test platform was developed to compare the performance of the A* algorithm using two graph representations. This phase of the work aimed to understand the A* algorithm and the visibility graph representation of the environment in more detail and provide experimental evidence of the cost saving achieved by using the visibility graph with the A* algorithm. This provided motivation for considering path finding as a means for automatically determining possible strategies within a Distributed Interactive Application.

The A* search algorithm evaluates each node using the sum of two cost functions. The first one calculates the cost of the path from the initial location to the location being evaluated. The second function provides a heuristic estimate of the remaining cost to reach the target location. The sum of these provides an estimate of the total path cost through the evaluated node. During each iteration of the search, A* evaluates the nodes connected to the node with the lowest estimated path cost, expanding the best node first [8].

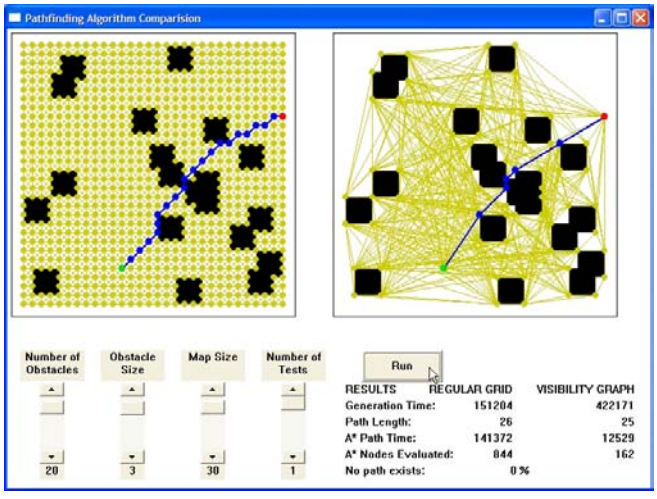


Figure 3: A sample map with randomly generated obstacles and the path computed using A* for a regular graph (left) and a visibility graph (right).

The developed test platform generates a random map and creates a visibility graph and regular grid; the A*

algorithm is applied to these grid representations and the results are displayed together for ease of comparison. In the experiments performed here environments were randomly generated using varying numbers of obstacles; each obstacle measured 5 by 5 units. An example of such an environment is given in Figure 3. A regular grid and visibility graph were then created for each environment and the tests were carried out.

The performance of the A* algorithm using each grid was computed by exploiting the high resolution hardware counter, which is supported by Microsoft Visual C++ 2003. This counter had a frequency of 3,579,545 counts per second on the test platform employed (AMD Athlon XP 2600+ with 512MB RAM). The actual counts were not transformed into absolute time values as again we are only interested in a relative comparison.

A series of experiments was performed to determine the computation time using both graphs as the number of obstacles in the environment was increased from 0 to 140 on a map size of 50 by 50 units. In each case, 1000 random maps with random obstacles and random start/target positions were generated and the average time count for A* to finish its search was measured.

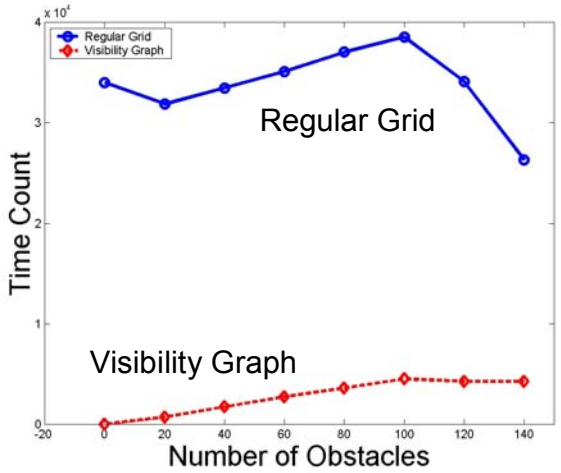


Figure 4: The time spent by A* to find a path using the visibility graph and the regular grid, as the number of obstacles was increased on 50 by 50 maps with random obstacle positions.

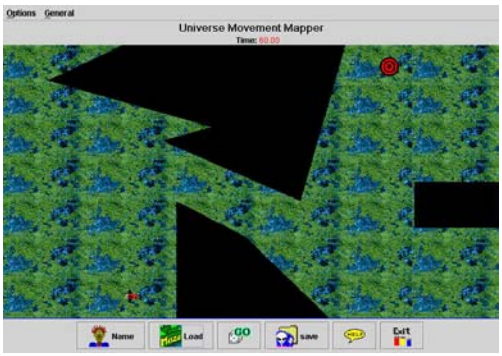
Figure 4 illustrates the computation time as a function of the number of obstacles. When there are 100 obstacles in the environment a search using the visibility grid is calculated in 10% of the time required by the regular grid, demonstrating that the visibility graph is faster by 90%. However as the number of obstacles increases two effects are noticed: (1) there is an increase in the search times for both representations and (2) the visibility grid gets progressively slower in comparison to the regular grid. These two effects may be explained as follows. For the regular grid, an increase in the number of obstacles means a reduction in the number of nodes and connections as more ground space is taken up with obstacles. However, there is also more time wasted in computing dead end paths. For the visibility graph, an increase in obstacles results in an increase in the number of nodes and connections with the number of connections being $O(m^2)$, where m is the number of nodes. Despite this, even with 140 obstacles, the visibility graph is 80% faster.

The experiments performed indicate that the A* path finding algorithm performs significantly better using a visibility graph than a regular graph. Currently path finding is only used by computer-generated characters navigating in the environment. We use path finding to automatically generate strategies that human users may adopt in navigating the environment. These strategies can be used in the hybrid strategy-based model technique. These issues are developed in the following section.

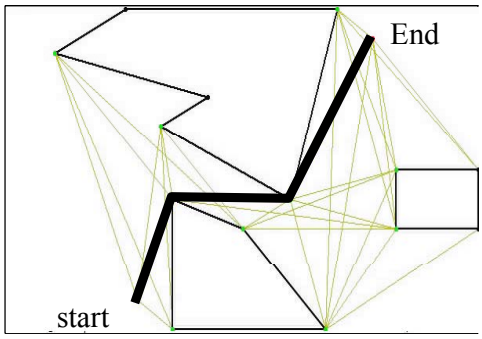
4. Strategy Models using Visibility graphs

Motivated by the efficiency of the path finding algorithm using a visibility graph representation and the fact that path finding is used by computer generated characters, it was decided to use path finding to compute possible strategies that human users might choose when navigating an environment.

The hybrid strategy-based modeling approach reduces the number of update packets that need to be communicated between participants of a DIA to maintain global consistency within a reasonable error threshold. The hybrid model consists of a short-term dead reckoning model and at least one long-term strategy model. To reduce the number of packets that need to be transmitted, the local user transmits information to inform other users of the model that best represents their current activity. Remote users maintain this model until the local user decides a change is needed based on some threshold criteria, in this case an error tolerance value. Because the long-term strategy model may represent a path of any form, it can be communicated using a single packet in contrast to a dead reckoning representation of the same path, which may require several packets. In previous work strategies were chosen by visually selecting the most representative user steady-state trajectory [1, 14]. Here the strategies are computed automatically using the A* algorithm and visibility graphs. Two test environments were constructed and strategies for the goal of navigating from a start to an end location in the shortest time possible were computed using A* and visibility graphs; environments 1 and 2 are shown in Figures 5 and 6 respectively.

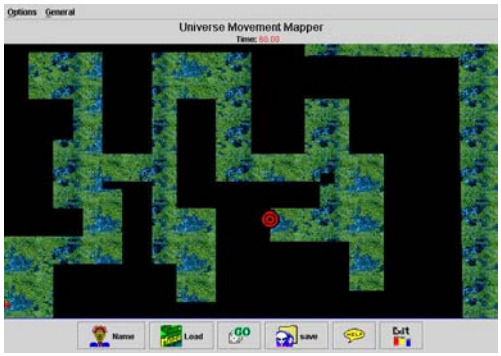


(a)

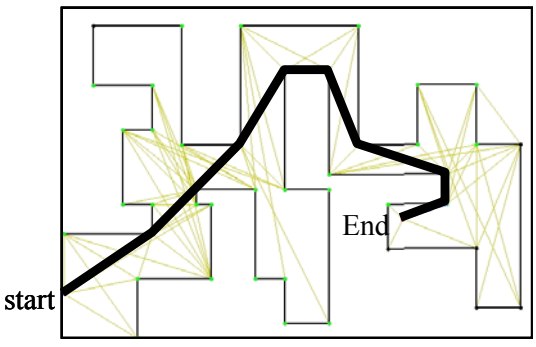


(b)

Figure 5: (a) User environment 1 (b) path generated automatically by A* algorithm.



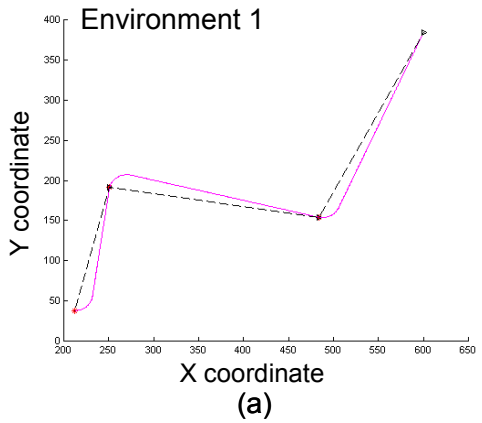
(a)



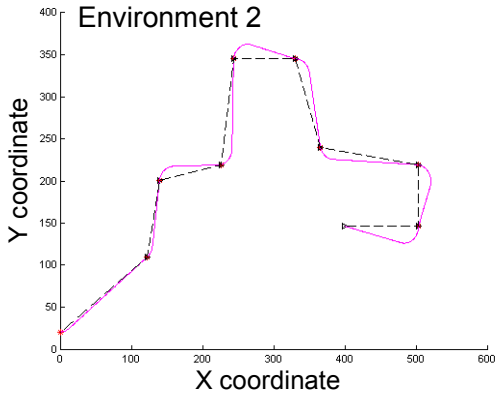
(b)

Figure 6: (a) User environment 2 (b) path generated automatically by A* algorithm.

By smoothing the paths the strategies can be made to match actual user movement more realistically. Figure 7 shows the smoothed paths for both environments after applying a standard smoothing algorithm that takes entity dynamics into account [2]. These smoothed paths became the strategies that were used to simulate the generation of network packets as part of the hybrid strategy model. This simulation is described in the following section.



(a)



(b)

Figure 7: A realistic turns routine was added to create more realistic strategies, indicated by the solid line. The dashed line shows the strategy found by path finding for (a) environment 1 and (b) environment 2.

5. Simulation Results

User data was gathered from fourteen distinct users. Each of these users maneuvered an entity through a maze from a given start position to a given target location. Users had no prior knowledge of the maze and were restricted to viewing a circular area of the maze centered on their current location at any point in time. Users repeated the task of traveling from the start to the target node in the shortest time possible. With each attempt their knowledge of the maze increased until they converged on

a steady-state trajectory. To determine the number of packets that would have to be sent over the network to represent a user trajectory a simulation was performed which took the trajectory as input and then simulated the number of packets generated in three cases:

1. using a pure dead reckoning model only;
2. using a hybrid strategy model with the strategy chosen by visual analysis of the user steady-state trajectories;
3. using a hybrid strategy model with the strategy computed automatically using the A* path finding algorithm and the visibility graph as described earlier.

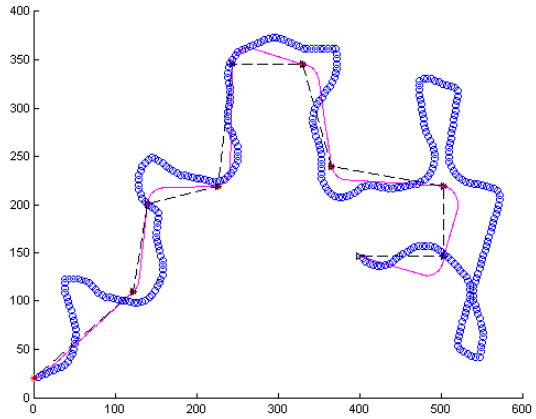
Table 1: Environment 1: packets transmitted for pure dead reckoning and the hybrid method; strategy model chosen by visual analysis and path finding. Trial 1 is the initial trial. Threshold value: 25.

	User 1			User 2		
Seq	DR	Hybrid		DR	Hybrid	
		Visual	Path finding		Visual	Path finding
1	31	32	33	16	13	17
2	26	27	27	22	16	19
3	18	15	11	28	28	29
4	8	3	4	15	1	5
5	10	7	5	18	9	17
6	14	4	7	13	8	8
7	13	13	9	13	5	12
8	8	4	7	10	4	1

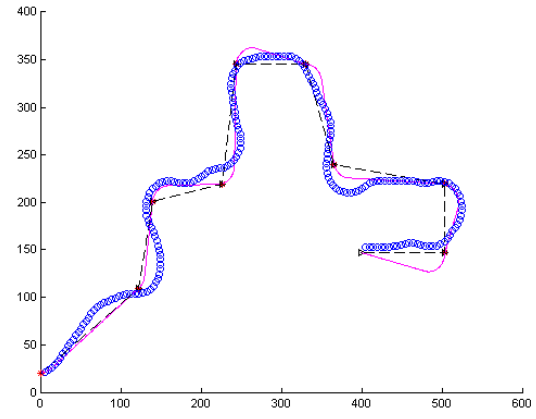
Table 2: Environment 2: packets transmitted for both pure dead reckoning and the hybrid method; strategy model chosen by visual analysis and path finding. Trial 1 is the initial trial. Threshold value: 25.

	User 1			User 2		
Seq	DR	Hybrid		DR	Hybrid	
		Visual	Path finding		Visual	Path finding
1	35	42	36	52	44	51
2	38	32	34	25	23	24
3	22	6	9	34	23	26
4	18	3	3	21	4	10
5	16	6	3	24	20	21
6	20	3	3	18	7	11

The results of the simulations are presented in Tables 1 and 2 for environment 1 and environment 2 respectively. Examination of these results shows that there is a reduction in the number of packets that need to be transmitted using the hybrid strategy-based model approach. In addition, there is very little discrepancy between the number of packets transmitted using a strategy constructed from visual analysis of recorded user trajectories and a strategy constructed automatically using path finding. Two sample trajectories are illustrated in Figure 8 for user 1 navigating through environment 2. As their experience within the maze increases their trajectory converges to the steady-state strategy identified using path finding. It must be noted that the results presented are for a threshold of 25 units. This choice was based on the variance of the recorded user steady-state trajectories. Results for a lower threshold value have been presented elsewhere [1, 2].



(a)



(b)

Figure 8: User 1 navigating in environment 2 – (a) first trajectory - exploratory; (b) final trajectory - steady-state. The strategy model (solid line), path finding model (dashed) line and user trajectory (wide line of circles) are shown.

If all trials of the two users in both environments are considered, dead reckoning generates 596 packets. In comparison the hybrid technique (employing a visual model) generates 402 packets and the hybrid model

(employing a path finding model) generates 438 packets. This corresponds to reductions of 33% for the visual heuristic Hybrid model, and 25% for the path finding Hybrid model. The results show that the reduction in the number of generated packets compared to dead reckoning is similar for strategies constructed either visually or using path finding. The key difference between the two is that the path finding strategy is generated automatically.

6. Conclusions and Future Work

It has been shown that visibility graphs reduce path finding computation time by between 80 and 90%, depending on the number and the density of obstacles in the simulated environment. The visibility graph is also insensitive to map size, unlike the regular grid, which makes it suitable for large virtual environments with few obstacles.

The A* path finding algorithm was implemented on a visibility graph and used to automatically generate strategy models. These were used in the hybrid strategy-based modeling approach and showed a reduction in the number of packets needed to maintain global DIA consistency. The reduction was commensurate with previous work using heuristic techniques for defining strategies. The advantage of automatic identification of strategies using A* and visibility graphs is twofold:

1. by reducing the number of packets that need to be transmitted network latency is reduced, as packets only need to be sent when the strategy changes; if communication is lost for a longer period of time, remote users can be rendered locally as continuing on the strategy they were on before network connection was lost;
2. in the case of dynamic goals, it is proposed that strategies to satisfy the goal can be recomputed in real time.

Future work will focus on using path finding techniques to determine strategies on the fly for dynamic goals. This will work by pre-computing the visibility grid for all users and then employing the A* algorithm to search in real time for a path between a current position and a dynamic target position. The path finding implementation will be modified to use Hershberger and Suri's methods, which provides a shortest path in O*(nlogn)* time [15].

Acknowledgement

This material is based upon works supported by Enterprise Ireland under grant no. SC/2002/129/.

References

[1] Delaney, D., T. Ward, and S. Mc Loone. *On Reducing Entity State Update Packets in Distributed Interactive Simulations using a Hybrid Model*. in *Proceeding of the 21st IASTED International Multi-conference on Applied Informatics, February 10-13*. 2003. Innsbruck, Austria.

[2] Pinter, M., *Toward more realistic Path finding.* Game Developer, 2001: p. 54-64.

[3] Woodcock, S., *Game AI: The State of the Industry 2000-2001: It's Not Just Art, It's Engineering.* Game Developer, August 2001.

[4] Kavraki, L.E., et al., *Probabilistic Roadmaps for Path finding in High-Dimensional Space.* IEEE Transactions on Robotics and Automation, 1996. **12**(4): p. 556-580.

[5] Tzour, P., *Building a Near-Optimal Navigation Mesh*, in *AI Game Programming Wisdom*. 2002, Charles River Media. p. 171-185.

[6] Hwang, Y.K. and N. Ahuja, *A Potential Field Approach to Path finding.* IEEE Transactions on Robotics and Automation, 1992. **8**(1): p. 23-32.

[7] Epic Games, *Unreal Tournament website: http://www.unrealtournament.com/*. 2004.

[8] Russell, S. and P. Norvig, *Artificial Intelligence - A modern Approach*. 1995: Prentice Hall.

[9] Stout, B., *Smart Moves: Intelligent Path finding.* Game Developer, 1996: p. 28-35.

[10] Yap, P., *Grid-based Path finding - Lecture notes in Artificial Intelligence*. 2002. pp. 44-55.

[11] Lozano-Perez, T. and M.A. Wesley, *An Algorithm for planning collision-free paths among polyhedral obstacles.* Communications of the ACM, 1979. **22**(10).

[12] Matthews, J., *Basic A* Path finding Made Simple*, in *AI Game Programming Wisdom*. 2002, Charles River Media. p. 105-113.

[13] Ghosh, S.K. and D.M. Mount, *An Output-sensitive algorithm for computing visibility graphs.* Society for Industrial and Applied Mathematics (SIAM) Journal of Computing, 1991. **20**(5): p. 888-910.

[14] Delaney, D., T. Ward, and S. Mc Loone. *Reducing Update Packets in Distributed Interactive Applications using a Hybrid Model*. in *16th International Conference on Parallel and Distributed Computing Systems, August 13-15*. 2003. Reno, USA.

[15] Hershberger, J. and S. Suri, *An optimal algorithm for Euclidean shortest paths in the plane.* Society for Industrial and Applied Mathematics (SIAM) Journal of Computing, 1997. **28**(6): p. 2215-2256.