Vis Comput (2010) 26: 467–476
DOI 10.1007/s00371-010-0457-7

ORIGINAL ARTICLE

Live path: adaptive agent navigation in the interactive virtual world

Shin-Jin Kang · YongO Kim · Chang-Hun Kim

Published online: 28 April 2010

Abstract We present a novel approach to adaptive navigation in the interactive virtual world by using data from the user. Our method constructs automatically a navigation mesh that provides new paths for agents by referencing the user movements. To acquire accurate data samples from all the user data in the interactive world, we use the following techniques: an *agent of interest* (AOI), a *region of interest* (ROI) map, and a *discretized path graph* (DPG). Our method enables adaptive changes to the virtual world over time and provides user-preferred path weights for smart-agent path planning. We have tested the usefulness of our algorithm with several example scenarios from interactive worlds such as video games. In practice, our framework can be applied easily to any type of navigation in an interactive world. In addition, it may prove useful for solving previous pathfinding problems in static navigation planning.

Keywords Agent navigation · Virtual world · Pathfinding · Navigation mesh

S.-J. Kang
School of Games, Hongik University, Yongi-gun, Chungnam, Korea
e-mail: directx@hongik.ac.kr

C.-H. Kim (✉)
Department of Computer Science, Korea University, Anam-dong Seongbuk-gu, Seoul, Korea
e-mail: chkim@korea.ac.kr

Y. Kim
NCSoft, Seoul, Korea
e-mail: koyanghi@korea.com

1 Introduction

In the real world, unexplored areas do not have roads. Over time, however, a very small number of people start exploring these areas and marking out the beginnings of paths. Others naturally follow these paths laid out for them by the pioneers, and the paths gradually expand and become more defined. Inspired by this natural process in the real world, we conceived a new navigation approach for interactive virtual worlds. Our system samples characteristic user movements in the interactive world, and then generates new paths that previously did not exist. Newly generated paths can be used to expand the previous static world or to compensate for the inadequacies of conventional agent pathfinding.

In recent years, with advances in graphics hardware, the number of interactive applications involving multi agents has increased. Consequently, users have more opportunities to interact with a large number of agents in video games, and they experience more complex interaction in online environments such as massively multiplayer online games and virtual worlds. Real-time pathfinding is a process that enables the real-time selection of optimal paths for agents within a defined environment. Traditionally, it has been studied within the domains of computer graphics and robotics.

Many real-time pathfinding approaches have focused on global navigation with roadmap-based or graph-based methods to reduce the spatial complexity. Kavraki [8] presents a graph-based pathfinding technique by using a probabilistic roadmap in a static world. In a preprocessing phase, this method builds a roadmap of possible motions of the robot through the environment. When a particular path planning query must be solved, a path is retrieved from this roadmap using fast graph search. These methods have been successfully applied to computer animation problems such as the animation of human-like characters [4, 7]. The probabilistic roadmap approach is suited for high dimensional environments. But the roadmap produced by the method can be low quality paths consisting of straight line segments requiring a smoothing process, which is not appropriate for real-time processes.

To satisfy the performance and quality of path generation at the same time, the two-level path planning method was proposed in [9, 20]. In this method, the first level deals with global path planning towards a goal, and the second level addresses local collision avoidance and navigation. The global path is computed using a roadmap graph that represents static objects in the scene without the presence of agents. A classic search algorithm is then used to compute distances from a goal position to any graph node.

As the size and complexity of the virtual world is increasing, an appropriate solution for real-time navigation in a very large virtual environment is required. Pettr'e et al. [12] addressed this issue for crowds by using a space structuring technique that automatically decomposes multilayered or uneven terrains into navigation corridors, giving rise to a navigation graph. In [13], Pettr'e et al. extended their autonomous navigation method by considering interactions between pedestrians, using a predictive approach. Their method supports a scalable simulation loop, which allows crowd situation update while distributing the available computational resources in space and time. Therefore, pathfinding can be implemented efficiently in a large virtual world.

In recent years, the virtual world is changing to a dynamic environment. The adaptive concept was presented as a complementary approach for agent navigation in dynamic environments. Sud et al. [15] present a novel approach to real-time path planning by multiple virtual agents in complex dynamic scenes. They used a Multi-agent Navigation Graph (MaNG), which is constructed using Voronoi diagrams. In [16], they also suggested Adaptive Elastic Roadmaps (AERO), which are reactive roadmap graphs using particle-based dynamics simulators.

Previous roadmap- or navigation graph-based pathfinding techniques have focused on agent navigation in the virtual world by using data from the environment itself. The environmental complexity of the virtual world has increased exponentially in just a few years, aiming to satisfy users' demands for "reality." In that world, a user can control a specific agent and interact with other agents with few restrictions. However, there are some limitations on handling these types of environments using previous pathfinding techniques alone, because of the irregularity and complexity of the interactive world. Our method has been devised to address these types of pathfinding problems. In this paper, we propose a new adaptive navigation method based on user interaction data. To our knowledge, this is the first adaptive navigation method that uses user data to evolve the static navigation data structure over time in an interactive virtual world. Figure 1 shows our system overview. The remainder of this paper is structured as follows. Section 2 reviews problems with previous pathfinding methods when applied to interactive worlds. Section 3 describes our proposed system for adaptive navigation, which comprises four components in addition to the metadata acquisition. Section 4 presents our results, and Sect. 5 contains our conclusions.

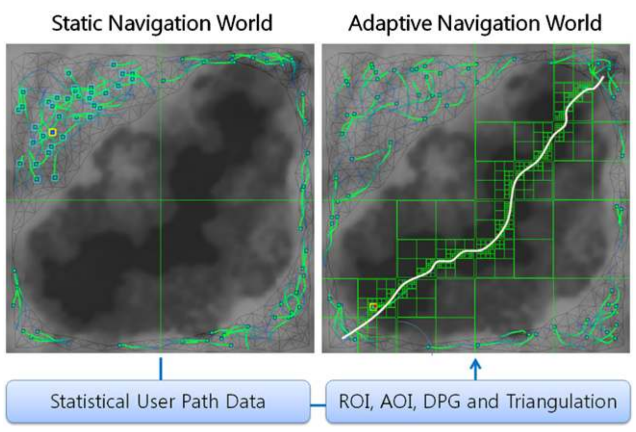

Static Navigation World
Adaptive Navigation World
Statistical User Path Data
ROI, AOI, DPG and Triangulation

Fig. 1 System overview

2 Pathfinding problems in the interactive world

The A^* algorithm is the most widely used search algorithm for real-time pathfinding in the interactive world [6]. The dynamic A^* algorithm (D^*) [17] can be another solution for unknown, partially known, and changing environments. The A^*-based algorithm requires a static quantized search space which depends on the nature of the world. Stout et al. [18] suggested various ways to quantize the world including rectangular grids, quad trees, convex polygons, and navigation meshes, etc. A navigation mesh is an abstract data structure used in artificial intelligence (AI) applications to aid agents in pathfinding through large spaces. Meshes are typically implemented as graphs, which enables their manipulation by the large number of algorithms defined for these structures. One of the most common uses of navigation meshes is in video games [3, 19] and commercial AI pathfinding middlewares [5, 11]. In this section, we present a brief description of the development process for pathfinding that mostly relies on navigation meshes and addresses common pathfinding issues.

2.1 Pathfinding based on navigation meshes

Pathfinding processes based on navigation meshes usually have two stages, a preprocessing stage and a run-time stage. The pathfinding approach varies according to the type of agents involved.

2.1.1 The preprocessing stage

(1) *Terrain data loading*: Loads the art resources that contain the world data.
(2) *Navigation mesh generation*: Using the loaded terrain data, areas over which the actual agent movement is allowed are processed into a navigation mesh. In identifying these mobility areas, one refers to the terrain gradient and any manual inputs from the developer. At this stage, the navigation graph can be used depending on the pathfinding requirement.

2.1.2 The run-time stage

(3) *Destination assignment*: The destination for the agent movement is assigned in accordance with event triggers or user interaction.
(4) *Shortest path search*: The A^* search algorithm is applied to the navigation mesh or other derived data structures to calculate the shortest path to the destination points inside of the navigation meshes.
(5) *Movement*: The agent moves along the derived shortest paths and avoids other moving agents by local deviations inside of the generated navigation mesh. If the navigation graph is used, the agent moves along the graph edges.

2.1.3 PC and NPC

In a user-participating virtual world, agents are classified as either playable characters (PCs) or non-playable characters (NPCs). NPC movement is relatively robust and simple compared with that of PCs. NPCs move only along the shortest path obtained in stage 4 by the system. If there is an area that is not covered by the navigation mesh acquired in stage 2, that area becomes an inaccessible area for NPCs.

Movement processes for PCs are slightly different from those of NPCs in the interactive world. PC movement is controlled by point designation and/or by directional control. Here, point designation refers to the user directly designating a destination point for the PC, and is usually achieved via a mouse click. Conversely, directional control refers to user input of the direction of the PC's movement from the current location and is usually achieved via directional keys. PCs whose destination has been designated by pointing find their path using the navigation mesh in the same manner as NPCs. PCs whose destination has been defined using directional keys, on the other hand, do not reference the navigation mesh but move in each designated direction by a preset distance instead.

2.2 Limitations

Under the processes described above, pathfinding produces the following issues. These types of issues still remain as practical problems, even in some million-unit-selling commercial interactive titles [2, 10].

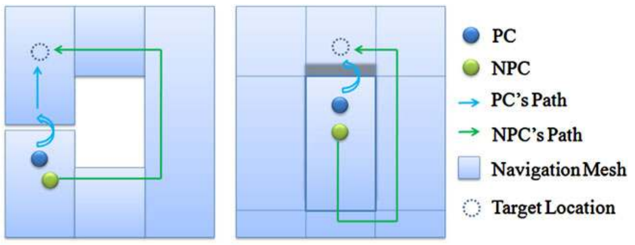


Fig. 2 Previous pathfinding issues—NPC detouring cases. PCs can jump over the small gap (*left*) and jump down from the hill (*right*), but NPCs cannot do this

2.2.1 Updating difficulty for the navigation mesh

Because of the huge data size of the terrain of virtual worlds, updating the navigation mesh in stages 1 and 2, still remains a time-consuming task. Consequently, it limits the dynamic update of the entire virtual world, forcing the world to be static or only partially dynamic within a limited region.

2.2.2 Incomplete navigation mesh coverage

At stage 2, issues with the navigation mesh generation algorithm and/or mistakes by the world resource developer can lead to areas where the navigation mesh does not exist. In this case, unexpected inaccessible areas exist in the world, which subsequently causes the following problems.

– NPC detouring: At stage 5, an NPC may travel an unnecessarily long route, irrespective of the terrain. This inefficient detouring of NPCs lowers the realism of NPC actions in the virtual world and may enable a user to exploit this glitch, resulting in various forms of NPC interaction abuse. In particular, exploitable vulnerabilities such as this in a combat situation severely threaten the balance of the virtual world. Figure 2 shows the NPC detouring cases in two discontinuous areas.
– PC detouring: Controlling a PC via point designation may result in the PC traveling an unnecessarily long route. This forces the user to limit the designation of destination points to nearby and linear points from the PC's current position, consequently necessitating an excessive number of point designations.

2.2.3 Movement range discrepancy between PC and NPC

There is a difference in the degree of controllability freedom between PCs and NPCs. Compared with the NPC, which refers to the navigation mesh only, PC can explore more areas and choose a better path by experimenting in various circumstances. This discrepancy produces adaptation limitations related to agent pathfinding in the interactive world.

3 System

This paper proposes a new method for agent pathfinding that applies user movement data to address the issues with conventional pathfinding approaches in the interactive world. This idea was conceived with the assumption that users can choose better paths instinctively if they control their characters. In the early stages of investigating an interactive world, users do not usually select the best routes when controlling their characters, because of the lack of world information. But as time goes on, users become accustomed to the world, and they start to discover better paths than the system can generate. For example, they can jump onto the roof of a building, or hide in places that other agents cannot reach, using unexpected skills. Until now, the system has not been able to handle this type of situation without manual intervention. However, if a system were capable of learning the user's experimentally optimized paths, and integrating these findings into its virtual world, it would give more realism to the user and avoid any vulnerabilities of the virtual world in advance. This process can be called *adaptive path expansion*, and it is not based on any specific algorithm but based on statistical data from the real world.

In the interactive world, users can accumulate their play experiences, but NPCs usually cannot. Therefore, as time goes on, PCs become smart, whereas NPCs retain their initial static behavior. This sometimes causes the user to become bored or leads to the vulnerabilities of the agent being exploited. Evolutionary computation can be a solution to this problem. Much research in the artificial intelligence domain has used neural networks and genetic algorithms. But to evolve the NPC correctly, an appropriate objective function based on various parameters is required. Users' path data can be part of the necessary *metadata* used to evolve the NPC. The trained paths generated by the system can be utilized as the key metadata for agent evolution. Knowing the users' previous paths, the NPC may be able to make smarter decisions. To achieve these two capabilities, namely *adaptive path expansion* and *collecting metadata for agent evolution*, we use a combination of processes, as follows.

3.1 Adaptive path expansion

Referencing user movement paths in an interactive world requires the gathering of relevant data. The data must be sampled from areas that have pathfinding issues and must contain information needed for updating the path. However, it is difficult to anticipate when and how this type of data might be manifested and particularly difficult to make such forecasts in massive virtual worlds. Furthermore, monitoring the paths of the thousands of agents who simultaneously connect to and play in a virtual online world via a server is almost impossible. In addressing this issue with a systemically efficient and automated approach, we use three automatic sampling techniques, as follows.

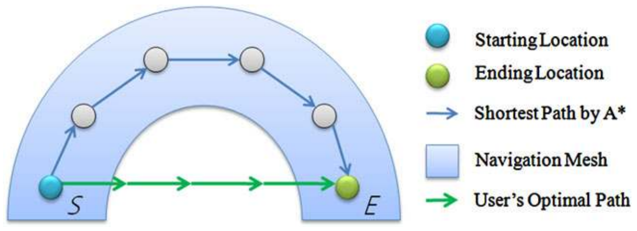


Fig. 3 Checking the shortest path update by PC

3.1.1 Region-based sampling: the ROI map

Virtual worlds are usually enormous in scale, and pathfinding issues within such environments are difficult to identify because they usually occur at underexplored areas within the world. Automated data sampling necessitates the identification of problematic areas within a limited time frame. Therefore, to solve this problem, we propose a *region of interest* (ROI) map. Using such a map was motivated by Pettr'e's idea [12] of utilizing regions of interest in adjusting the level of detailed processing of movements. First, we construct an ROI map for the entire virtual world in terms of scalable two-dimensional grids called monitoring blocks (MBs). An MB has an ROI level that is proportional to the possibility of finding an optimal path within the area. The levels range from level 0, where no pathfinding issues are anticipated, to level N, where pathfinding issues are likely. By using spatial partitioning techniques such as quad tree [1], each MB can be divided into four smaller MBs recursively until one reaches a level N.

With an ROI map deployed in the environment, the system should be able to detect abnormal shortest routes selected by the user. The quickest and the most direct approach is to look for instances where the route traveled by the user between two specific points is shorter than the distance measured between those points as calculated by the A^* algorithm used in the system. Consider the map of Fig. 3, where we need to check the shortest path update by the user in a certain period. Let S be a starting location at time t_0 and E be an ending location at time $t_0 + \Delta t_E$. Then P_i is the actual PC's location at $t_0 + \Delta t_i$ and A_i is the simulated PC's location at $t_0 + \Delta t_i$, which is calculated by the A^* algorithm on the existing navigation mesh. If $\sum_{i=0}^{N-1} d(P_i, P_{i+1}) < \sum_{i=0}^{N-1} d(A_i, A_{i+1})$, where $d(.,.)$ is the Euclidean distance between two points, then the path $\sum_{i=0}^{N-1} d(P_i, P_{i+1})$ is the new updated shortest path by the user between S and E at time $t_0 + \Delta t_E$.

The discovery of such an instance is evidence that an area might have a pathfinding issue. If all user distances were to be calculated, pathfinding issues could be identified accurately. However, monitoring region constraints make this impossible. The ROI map enables approximation of this detection process. The system selects one of the MBs in real time and assigns one agent within the selected MB to act as the sample agent. The system then measures the movements of the sample agent within that MB for a fixed duration and checks for the emergence of new shortest routes. If no new shortest route is found after multiple attempts, the system selects another MB and repeats the process. If the sample agent does travel by a new shortest route, the ROI map assumes the presence of a pathfinding issue area within the selected MB and subsequently raises the ROI level for the relevant MB. With these new level assignments, relevant MBs are divided to form half-sized quad sub-MBs, and the process is repeated to elevate the problematic area to the highest detection level. These detailed MBs are then used as the minimum extraction areas during path sampling (see Sect. 3.1.3). In this way, the proposed system utilizes an ROI map to monitor levels, aiming to sample data effectively from areas where new shortest routes have emerged.

3.1.2 *Agent-based sampling: the AOI*

In the monitoring stage, if we select the sample agent without having a plan, this can reduce the detection performance. To reduce the search space and increase the performance of correct-hit, we have adopted an agent level monitoring technique. A user who has discovered a new shortest route in a specific area of a virtual world is far more likely than other users to repeat the same pattern next time in the area. To reflect this probability, we have defined and implemented the profiling of the *agent of interest* (AOI). The concept involves identifying and monitoring PC agents that have a high likelihood of identifying new shortest routes. A PC that has achieved or will achieve a new shortest route becomes an AOI. These AOIs are frequently identified for monitoring in the ROI map. There are many possible cases whereby PCs will make new paths. We estimate these cases by considering the following three general-purpose properties.

- Violation of region entrance: A PC may have a high possibility of making new paths in uncovered areas of the navigation mesh. If the world has geometric data about inaccessible areas in addition to the navigation mesh, the system can easily detect the presence of the PC inside it. Usually because of memory limitations, this type of information is not maintained at the run-time stage. However, by adopting a region approximation scheme, the system can maintain the data via a minimal data structure. We generated an approximated nonnavigation mesh area by using a convex hull of ungrouped points in the map. If a PC enters inside the convex hull, we designate it as an AOI. If p_i is the location of selected agent i in an MB, the evaluation function $h(p_i)$ for this can be described as follows. Here, $D(.,.)$ is the collision detection function for the convex hull and V_c is the outline vertices of the convex hull

$$h(p_i) = \begin{cases} 1 & \text{if } (D(p_i, \text{All } V_c) = \text{true}), \\ 0 & \text{else.} \end{cases} \quad (1)$$

- Level of isolation: A new path may not be placed in an easily accessible area for every PC. That area may be located in some unreachable area. From this consideration, we evaluate an isolation level, which is calculated by counting the number of adjacent agents in radius r around the selected agent i

$$s(p_i) = \begin{cases} 1 & \text{if } (\text{num}(p_i, r) = 0), \\ 0 & \text{else.} \end{cases} \quad (2)$$

- Possibility of propagation: During online virtual world play, there is a chance that a normal PC will witness another PC achieving a new shortest route and follow suit. These PCs come to carry AOI status. In other words, PCs who are present in the vicinity of an AOI during its discovery of a new shortest route are also assigned as AOIs.

By considering the preceding properties, we designed the following OR operation function for an AOI evaluation. Here, g_i is the Boolean value which is updated when agent i has found a new shortest path, and j_i is another Boolean value which is updated when a propagation event occurs. $h(p_i)$ is used to reflect the violation of a region entrance, and $s(p_i)$ is used for checking the level of isolation. While g_i is a deterministic term, j_i, $h(p_i)$, and $s(p_i)$ are probabilistic terms for AOI assignment. These three terms can be combinational depending on the content of the virtual world.

$$\text{AOI}_i = \left\{ p_i | g_i = 1 \vee j_i = 1 \vee h(p_i) = 1 \vee s(p_i) = 1 \right\} \quad (3)$$

Furthermore, the total number of AOIs in each MB is also referenced in the selection of the next MB (3.1.1). This increases the accuracy of sampling for target regions. The number of AOIs increases in proportion to the total number of ROI levels in the MBs. However, to cap the monitoring load, there is a maximum maintaining time as an AOI and a maximum number of allowed AOIs. Via this process, our system designates and monitors AOIs to identify those agents who are most capable of determining new shortest routes. Table 1 shows the profile table for AOIs.

Table 1 Profiling table of AOIs

Num	AgentID	ZoneID	g_i	j_i	$h(p_i)$	$s(p_i)$	Remaining time (min)
1	5032	31	1	0	1	0	75
2	1012	17	0	0	0	1	64
3	15	9	0	0	1	1	57
…	…	…	…	…	…	…	…
30	965	24	0	1	0	0	12

3.1.3 Path sampling: the DPG

After the regions and agents for monitoring have been narrowed down, the system is ready to sample actual paths to obtain the positional data required to generate a navigation mesh. A trajectory clustering technique [14] which uses an artificial neural network can be a solution for this stage. However, the computational cost for the large data set in a massive world should be considered. We achieve this via two objectives. Firstly, we try to minimize the path sampling period, which is a costly process. We use actual path sampling only for an MB that has reached the topmost ROI level. In other words, the MBs in the ROI map that continued to raise its ROI levels through AOI tracking in previous stages did not track the actual paths of individual agents. This selective process can streamline path extraction by minimizing the temporal sampling load. Secondly, we try to minimize the storage for extracted path data. Virtual worlds usually provide a massive environment to mimic the scale of the real world. Storing all the path data as positions on the map of such a vast virtual environment will invariably lead to an excessive memory load. To address this issue, we have used a *discretized path graph* (DPG). A DPG is a graph generated by discretizing actual movement paths to a predefined resolution. An edge of a DPG is an approximation to a sampled path, and each vertex is a crossing point of the paths. Such a graph reduces the amount of storage space needed for terrain position data. Using the DPG, the system needs to calculate optimal position data for each new navigation mesh. We address this by accumulating intersection strength values in the vertices for path intersections within the DPG. Figure 4 shows user's paths and generated DPG. Specifically, points along a path with many intersections carry a high intersection strength value. Intersections having a certain strength value or higher are categorized as *feature path points* (FPP) sets, and are used as input data for navigation mesh generation. Our path sampling approach can be classified as time-based sampling assuming a constant speed by the PC. It can be integrated with ease into the following navigation mesh generation technique, and is capable of functioning efficiently.

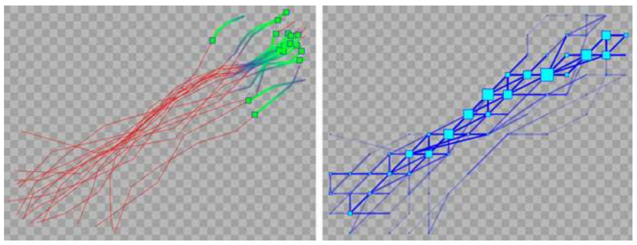


Fig. 4 User's paths (*left*) and generated discretized path graph (*right*)

3.1.4 Navigation mesh generation

The data obtained from the automatic monitoring process described in Sect. 3.1.3 are FPP sets, which are the discrete values of the most frequent intersections along the accumulated user paths. These unconstructed point sets are distributed over areas where the navigation mesh is rarely present, and the system must be able to utilize these points to generate a navigation mesh that can seamlessly connect with the existing navigation mesh. To achieve this, the proposed method relies on Delaunay triangulation [1]. Because of the uniform spacing of FPPs, Delaunay triangulation can generate stable triangles. It maximizes the minimum angle of all the angles of the triangles in the triangulation and thereby tends to avoid skinny triangles. The navigation mesh's format affects pathfinding using local coordinates. If the navigation mesh is not evenly distributed, movement along actual paths may seem jerky. The navigation mesh obtained by applying Delaunay triangulation to areas within a fixed radius of points identified via the monitoring process enables each of the triangles to obtain stable areas, and thereby enables natural movement.

If the sample points are located far away from the existing navigation mesh, the generated navigation mesh can become isolated from the existing mesh. However, the method presented in Sect. 3.1.3 usually performs continuous sampling within the grids, giving spatial continuity to the sampling points and thereby limiting the possibility of isolated navigation mesh generation. Delaunay triangulation usually generates stable triangles, but sometimes it can produces asymmetric skinny triangles from distantly separated points. To eliminate these triangles, we added an edge length constraint to the Delaunay triangulation. It can prevent unexpected connect between newly generated meshes and existing meshes. Figure 5 illustrates multiple input paths and generated navigation meshes with our system. Our system uses the four techniques described above. The appropriate combination of these techniques depends on the properties

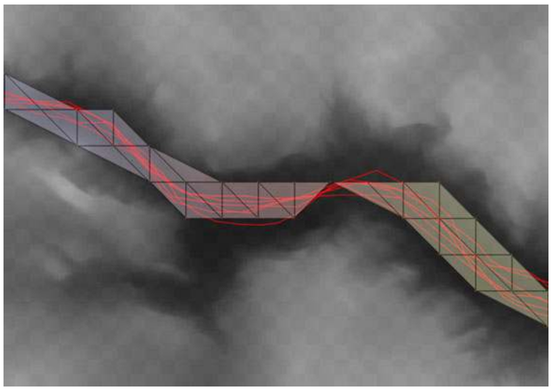


Fig. 5 Automatic navigation mesh generation. *Red lines* are input paths from users

Table 2 Pseudocode for the adaptive navigation expansion process

```
Algorithm

1. At Preprocessing
    Parameter_initialize()
    Construct_Event_Block (non_navigation_mesh_area)

2. At Run time4
Callback Function:
    AOI Update (EventID)
Main:
    MB tMB = Select_MB_in ROI( )
    AGENT tAgent = Select_Target_Agent (tMB)
    If(tMB.ROI_level > MAX_MONITORING_LEVEL)
        PATH p = Sample_Path (tAgent)
        POSITION sPosition = Descretize (p)
        DPG.Push_list (sPosition)
    Else
        If(Check_Path_update (tAgent.path)= true)
            AssignAOI(tAgent)
            AssignAOI(tAgent, PROPAGATION_RADIUS)
            tMB.IncreaseROI_level ( )
            tMB.DivideMB()
        End if
    End if

3. At Postprocessing or At Run time
    FPP sets = Construct_FPP(DPG.list)
    MESH tMesh = Triangulation(sets)
    Update_world_Navigation_Mesh(tMesh)
```

of the interactive world. Table 2 shows the pseudocode for our adaptive navigation expansion.

3.2 Collecting metadata for NPC evolution

In addition to automatic path expansion, statistical user paths can be referenced by the metadata for smart-agent path planning. As described above, PCs usually develop optimal paths in various environments. Sometimes, their unexpected path planning makes the system impossible to deal with. For example, if agents have been planned to move via the specific paths to guard critical point, PCs can become accustomed to these static paths over time and find a detour, thereby evading the agents. In other cases, if an NPC knows the PC's statistically most likely meeting spot, they can expose more community related contents to the target PC by placing a related NPC there. To make this possible, an amount of metadata has to be collected from the system. In the commercial interactive world, this type of work is usually performed manually by monitoring personnel. Our system can become an automatic framework for this work, with a minor system modification. If we design and implement a specific evaluation function such as (3) for an AOI by satisfying the monitoring cases, we can have target AOIs in the ROI map. Then, by applying the ROI map and DPG, FPP sets are acquired. This FPP set can be exported to as log data, and then automatically utilized for various purposes in the interactive world. We can estimate the level of the user's traffic at a certain point by referencing the accumulating intersection strength values of FPPs. By using these data, we construct a *user influence map*. The value in this map is used as the weight value $w(n)$ for the heuristic term of the A^* search algorithm. The weight value decreases at the high user traffic node n and increases at the low traffic node as a penalty cost. Here, in (4), $f(n)$ is the total cost for the A^* search algorithm. $g(n)$ is the path-cost function, which is the cost from the starting node to the current node n, and $h(n)$ is the estimated distance from the current node to the goal. α is the penalty ratio for the user influence map between [0,1]. This weight term does not make the total cost equal to zero, which can cause pathfinding confusion. If the α is zero, there is no influence from the user's path data.

$$f(n) = g(n) + h(n) + \alpha w(n) \tag{4}$$

Our FPP set has spatial continuity because of the continuous path sampling described in Sect. 3.1.3. But the intersection strength value in FPP sets may not be continuous. This may cause a few local zigzag paths with the A^* search algorithm. To minimize this case, we applied a 1D 1×3 average filter to intersection strength values along the FPPs before converting to the value of the influence map. With this simple user influence map, the modified A^* search algorithm gives a more content-oriented weighted path to the NPC. By using this path, the NPC can show more adaptive movements compared with the previous fixed movement patterns. Experimentally, using our system, this data collection process can be performed at run time, but periodic monitoring can also be made available.

4 Implementation

The proposed system's framework was constructed using Windows XP, DirectX 9.0, and the Ogre3D Rendering Engine. Experiments with the system were performed on a computer equipped with an Intel 1.83 GHz processor, 2 GB of memory, and an NVIDIA 8400 GT graphics card. Experiment 1 was designed to validate the proposed system's process for sampling multi-agent paths and generating new navigation meshes. The map modeled a mountain that had two hidden narrow paths. We deployed 140 agents in the map. The agents roamed around the map with their designated goals, and some agents tried to use the two hidden paths. In the early stages, our ROI map monitored the agents randomly, but as an agent approached an inaccessible area, the system designated the agent as an AOI via (3). Then it started to check its path update at short time intervals. If an agent updated the newest path, the system increased the ROI level of the MB and refined the MB into sub-quad MBs. With the aid of our adaptive sampling techniques, with few missing MB selections, and within 550 iterations, our system discovered two hidden path areas. It also successfully generated adaptive navigation meshes around them by using DPG in real time. For this experiment, an 80×80 resolution DPG was used and 112 FPPs were generated in 26 ms. 181 MBs were used for monitoring. Figure 6 shows the sampling process and experiment result.

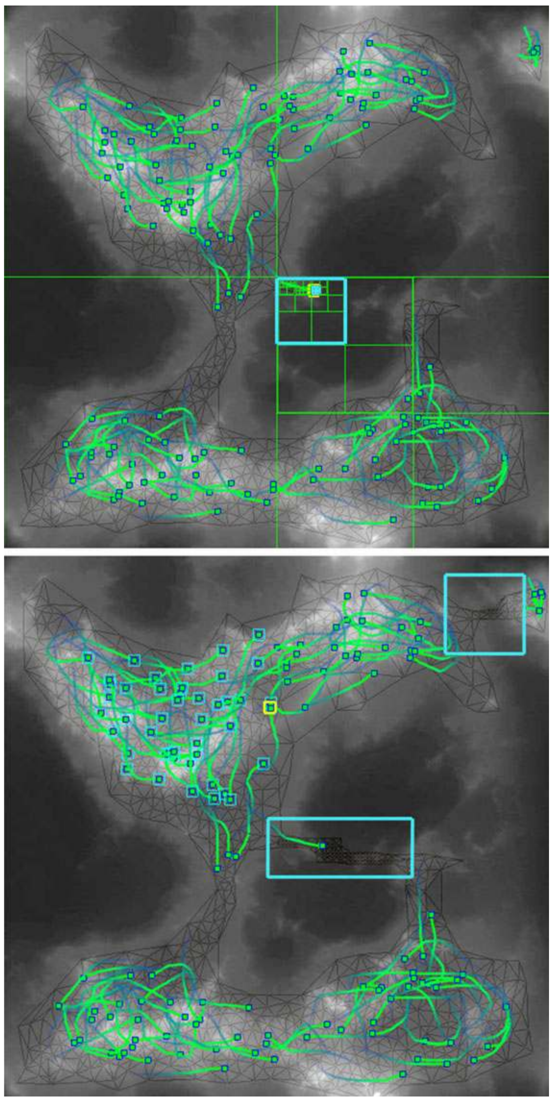


Fig. 6 The sampling process (*above*) and the simulation result (*below*) in experiment 1

Experiment 2 was designed to validate the utility of the proposed method in a video game environment. The map comprised 15 NPCs and 1 PC moving about within a town level. There was one narrow path in the center areas where the navigation mesh did not generate during the development process. Because of that, an NPC detouring case could occur around this area. But a PC, as per their directional control inputs, could cross over into the other side via the narrow path. The system detected this agent as an AOI, confirmed the new optimized path, raised the corresponding area's ROI level, and then formed a DPG using the path data obtained. Intersection points within the DPG having at least a specific level of intersection strength were chosen as the final sample points, and the system used these for processing automatic triangulation to generate a new navigation mesh. With this new navigation mesh, NPCs that previously had no path included the new mesh into their pathfinding process to reach the formerly inaccessible area. The experiment showed that the proposed system automatically monitored the movement routes of PCs to generate new paths and to enable NPCs to share the newly discovered paths. Figure 7 shows the results of experiment 2.

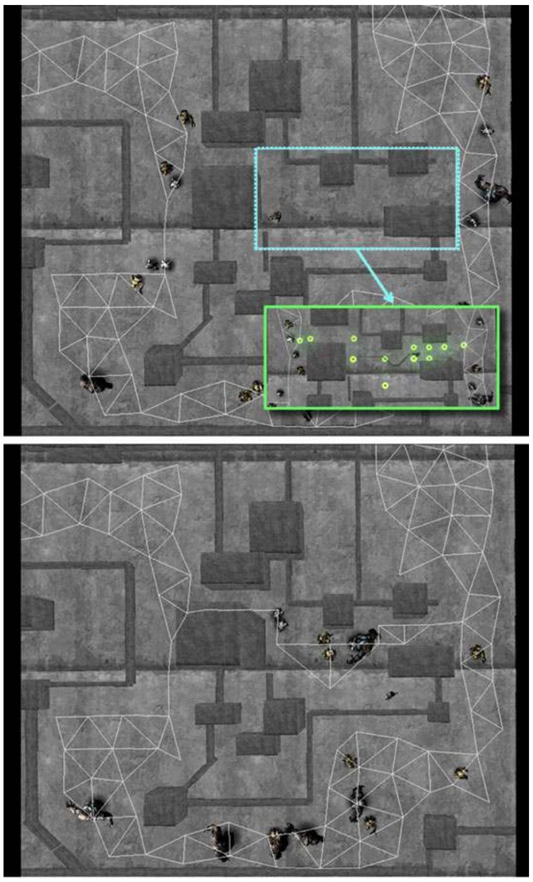


Fig. 7 Original navigation meshes (*above*) and adaptive agent navigation on the newly generated meshes (*below*) in experiment 2

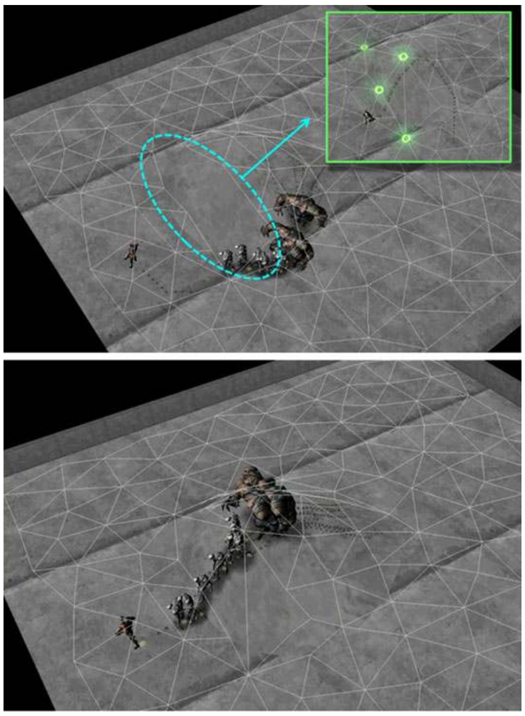


Fig. 8 An NPC routing case (*above*) and adaptive NPC navigation (*below*) in experiment 3

Experiment 3 addressed the issue of another NPC detour. In a terrain having at its center a hill with a steep slope on one side, an NPC could only travel to the top of the hill by going around the steep slope. With no navigation mesh around the steep side of the hill, because of differences in elevation, the NPCs were unable to include the area in their pathfinding process. But a PC could move over the hill with directional control, and they could evade hostile pursuing NPCs. Under these conditions, the proposed system identifies the new path and allows the NPCs to follow the same path as that taken by the PC. This effectively prevents user exploitation and enables more realistic pathfinding. Figure 8 shows the adaptive agent navigation for the NPC routing case.

Experiment 4 was performed to find out if the proposed system is capable of resolving the conventional PC detouring issue. In this experiment, there was an alternate path on a hill at the center of the area. This path was accessible to PCs via directional keys, but there was no navigation mesh for the path, because of differences of elevation in the terrain. With the conventional pathfinding process, point-clicking a destination behind the hill would have resulted in a long detour around the hill. With the proposed system, however, the system learned the alternative path as a PC navigated through it and thereby enabled other agents to use the newly discovered path. Figure 9 shows the adaptive PC navigation result.

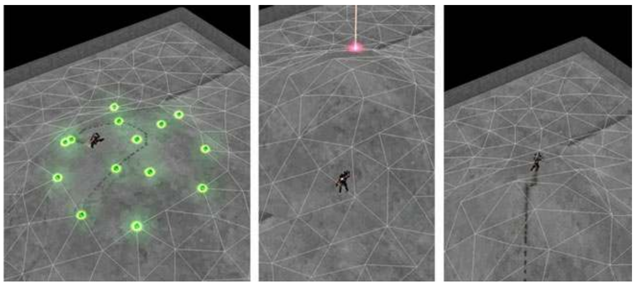


Fig. 9 A PC routing case (experiment 4). FPPs set around hill (*left*), point designation via mouse click (*middle*), and PC navigation on the newly generated navigation mesh (*right*)

5 Conclusions

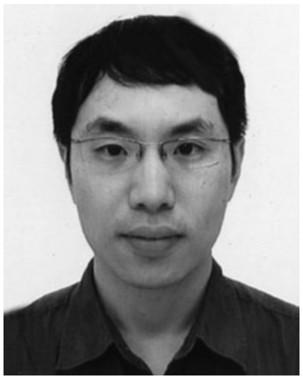
The system proposed in this paper has proven to be capable of referencing a user's path data to resolve the pathfinding issues caused by limitations in conventional static navigation mesh-based methods, The proposed method combines real-time sampling techniques to extract user location information from a vast amount of user data to generate new navigation meshes automatically. It enables robust adaptive changes of the virtual world in pathfinding. In addition, the same data can be used as the metadata for agent evolution. It provides user-preferred path weights for smart-agent path planning. Our method is advantageous in that it can be applied to any type of interactive virtual world directly and with ease. Furthermore, the proposed method has the potential to be extended to applications that utilize the collection of user data at run time.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments. This work was supported by the Hongik University new faculty research support fund.

References

1. Berg, M., Kreveld, M.V., Overmars, M., Schwarzkopf, O.: Computational Geometry, pp. 291–306. Springer, Berlin (2000)
2. Blizzard: World of Warcraft. http://www.worldofwarcraft.com/ (2009)
3. Bungie/Microsoft: Halo3. http://halo.xbox.com/en-us/games/halo3/ (2008)
4. Choi, M.G., Lee, J., Shin, S.Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Trans. Graph. **22**, 182–203 (2003)
5. Intel: Havok. http://www.havok.com/ (2009)
6. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. SIGART Newsl. **37**, 28–29 (1972)
7. Kamphuis, A., Overmars, M.H.: Finding paths for coherent groups using clearance. In: SCA04: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 19–28 (2004)
8. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high dimensional configuration spaces. IEEE Trans. Robot. Autom. **12**, 566–580 (1996)

9. Li, Y., Gupta, K.: Motion planning of multiple agents in virtual environments on parallel architectures. IEEE Int. Conf. Robot. Autom. **1**, 1009–1014 (2007)
10. NCSoft: Aion. http://na.aiononline.com/ (2009)
11. PathEngine: PathEngine. http://www.pathengine.com/ (2009)
12. Pettr'e, J., Ciechomski, P.H., Maim, J., Yersin, B., Laumond, J.P., Thalmann, D.: Real-time navigating crowds: scalable simulation and rendering. Comput. Animat. Virtual Worlds **17**, 445–455 (2006)
13. Pettr'e, J., Grillon, H., Thalmann, D.: Crowds of moving objects: navigation planning and simulation. In: 2007 IEEE International Conference on Robotics and Automation, vol. 1, pp. 3062–3067 (2007)
14. Sas, C., O'Hare, G., Reilly, R.: Virtual environment trajectory analysis: a basis for navigational assistance and scene adaptivity. Future Gener. Comput. Syst. **21**, 1157–1166 (2004)
15. Sud, A., Andersen, E., Curtis, S., Lin, M., Manocha, D.: Real-time path planning for virtual agents in dynamic environments. Proc. IEEE Virtual Real. **1**, 91–98 (2007)
16. Sud, A., Gayle, R., Andersen, E., Guy, S., Lin, M., Manocha, D.: Real-time navigation of independent agents using adaptive roadmaps. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, vol. 1, pp. 99–106 (2007)
17. Stentz, A.: Optimal and efficient path planning for unknown and dynamic environments. Int. J. Robot. Autom. **10**, 89–100 (1993)
18. Stout, B.: The basics of A^* for path planning. In: Game Programming Gems, pp. 254–263. Charles River Media, Hingham (2000)
19. Valve: Counter-strike: source. http://developer.valvesoftware.com/wiki/ (2009)
20. Van Den Berg, J., Patil, S., Sewall, J., Manocah, D., Lin, M.: Interactive navigation of multiple agents in crowded environments. In: Symposium on Interactive 3D Graphics and Games, pp. 139–147 (2008)



Shin-Jin Kang received his M.S. degree from the Department of Computer Science Engineering from Korea University in 2003. After graduation, he joined Sony Computer Entertainment Korea (SCEK) as a video game programmer. Since 2006, he has worked at NCSoft Korea as a lead game designer in various MMORPG projects including AION. He is now working toward his Ph.D. degree in Computer Science and Engineering at Korea University. He is also a professor at the School of Games at Hongik University.

YongO Kim received his M.S. degree from the Department of Media Engineering at Korea University. He had worked on four commercial games as a server programmer before, and was principal architect on two of them. He is currently working on the MMOG project as a lead programmer at NCSoft Korea. His research interests are artificial intelligence, mix with MMO and MO, and artificial life.

Chang-Hun Kim is a professor in the Department of Computer Science and Engineering at Korea University. He received his B.A. degree in Economics from Korea University in 1979. After graduation, he joined the Korea Advanced Institute of Science and Technology (KAIST) as a research scientist, where he was involved in many national research projects in the area of computer-aided design and geometric modeling. He received his Ph.D. from the Department of Electronics and Information Science, Tsukuba University, Japan, in 1993. During 1993–1995, he headed the Human Interface and Graphics Laboratory for the System Engineering Research Institute (SERI). His current research interests include fluid animation and mesh processing. He is also a member of the IEEE Computer Society and ACM.