symmetry

MDPI

Article

Navigating Non-Playable Characters Based on User Trajectories with Accumulation Map and Path Similarity

Jong-Hyun Kim [1], Jung Lee [2] and Sun-Jeong Kim [2,*]

1 School of Software Application, Kangnam University, Yongin 17021, Gyeonggi-do, Korea; jonghyunkim@kangnam.ac.kr
2 School of Software, Hallym University, Chuncheon 24252, Korea; airjung@hallym.ac.kr
* Correspondence: sunkim@hallym.ac.kr

Received: 4 September 2020; Accepted: 21 September 2020; Published: 25 September 2020

check for updates

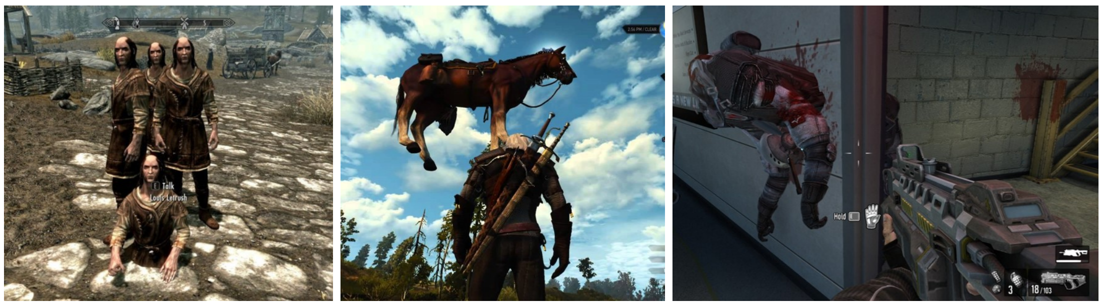
Abstract: In this paper, we propose a method to efficiently control the path of non-playable characters (NPC) in an interactive virtual environment such as a game or virtual reality (VR) by calculating a weight map and path similarity based on the user's path. Our method automatically constructs a navigation mesh that provides a new route to the NPC by referring to the user's trajectory. Our method finds more paths that users usually go through as time passes, and the number of users increases. Accordingly, the paths that NPCs can traverse automatically are updated adaptively to the virtual environment. In addition, NPC agents can move smartly by assigning high weights to the user's preferred paths. We tested the usefulness of the proposed method through several example scenarios in an interactive environment such as a video game or VR, and this method can be easily applied to various types of navigation based on the interactive environment.

Keywords: computer graphics; character movement; character animation; path finding; user's trajectory

1. Introduction

In the real world, there are no roads in areas where people do not often go. As time goes by, people explore and mark these areas with routes, creating roads where people can travel. Others follow the path created by those who have been there earlier, and as a result, the path gradually expands and becomes clear. Inspired by this natural path-creation process, this paper introduces a new way to explore non-playable character (NPC) paths in an interactive virtual environment. Using existing static navigation mesh-based methods, when NPCs reach a dead end, that is, without a navigation mesh, even if the destination is nearby, it may fall into unexpected situations such as NPCs returning or being caught between obstacles (see Figure 1). This situation occurs because the NPC's movement depends only on the static navigation mesh, which in turn leads to another problem in which the NPC behaves differently from the developer's intention. This problem is important because it reduces immersion in the game and eventually leads to a decrease in profits from the game. The proposed system samples the user's movement in an interactive environment and then generates a new path that was not available before. By using the newly generated paths, the existing static navigation mesh can be expanded and the inefficient route of NPCs can be improved.

Recent developments in graphic hardware have led to the development of interactive applications related to a number of NPCs. As a result, users experience many interactions with NPCs in virtual environments such as MMORPG (Massively Multiplayer Online Role-Playing Games) or virtual reality (VR) contents. The method of finding a path in a virtual environment in real time has been studied in the field of computer graphics and robotics because it can adaptively and dynamically find the optimal path to move the NPC. Most real-time path-finding approaches have focused on how to globally optimize paths based on roadmaps or graphs to reduce space complexity. Kavraki et al. presented a graph-based path-finding technique using a stochastic roadmap in a static world [1]. They pre-embedded the roadmap for the robot's movement through the given environment in the preprocessing stage and searched for the path in the roadmap using fast graph search when a specific path needed to be queried. This approach has been successfully applied to solve various issues related to computer animation [2,3]. This stochastic roadmap approach works well in high-dimensional static environments, but in real-time processes, low-quality paths are often created. The two-level path-planning method was proposed to satisfy both the performance and quality of path generation [4,5]. This method calculates the global path to the destination at the first level and the local collision avoidance path at the second level. The global path calculates a roadmap graph representing static objects and then calculates the distance from the destination to the graph nodes using a search algorithm.



(**a**) The Elder Scrolls V: Skyrim (**b**) The Witcher 3: Wild Hunt (**c**) F.E.A.R. 3

Figure 1. Inappropriate behavior of non-playable characters (NPCs) in various games.

As the size and complexity of the virtual environment increases due to the development of processors and memory, a solution to search a path in a large-scale virtual environment in real-time is required in many fields. Pettr et al. dealt with the path-finding problem in crowd simulation [6]. They used spatial structuring techniques to generate navigation graphs in crowd simulations to create complex paths in real-time. They extended this technique to a predictive approach and proposed an autonomous navigation method considering interactions between pedestrians [7]. Since this method supports scalable simulation loops, it is possible to update crowd situations while distributing computational resources over space and time, resulting in efficient path-finding in large virtual worlds. Recently, virtual worlds are changing into dynamic environments, and Sud et al. introduced a new approach to explore the paths of many NPCs in this dynamic environment [8]. They used MaNG (Multi-agent Navigation Graph) based on a Voronoi diagram. In addition, they proposed AERO (Adaptive Elastic ROadmaps), a responsive roadmap graph using particle-based simulation [9]. Traditional path exploration techniques based on roadmaps or navigation graphs focused on using the data of the environment itself. Studies trying to control the movement of NPCs in a virtual environment have grown exponentially in recent years as more and more users want natural interactions with NPCs. In the virtual world, users can control and interact with specific NPCs to a certain level. However, due to the irregularity and complexity of the interactive environment, there is a limit to processing with the existing path search method alone. To solve the path-finding problem in this environment, this paper proposes a new adaptive navigation method based on user interaction data.

Techniques for modeling the independent movement of individual agents as well as generating crowd movement have been proposed [10–12]: this includes agent-based methods, methods using cellular automata, and methods using continuous flow.

Agent-based methods generally determine the direction and behavior rules based on local information of individual agents. Reynolds used simple local rules to group them effectively [13]. The model has been extended to fields such as analyzing movements according to psychological or sociological preferences [14,15]. In addition, Berg et al. proposed a method to improve collision avoidance by including the velocity of obstacles in the motion rules [5]. The cellular automata approach proposed an evolutionary model for agent location by solving cellular automata. This has been extended to a variety of approaches, including techniques using static and dynamic fields [16], grid-based behavioral rules [17], and rules based on behavioral models [18]. Since these approaches are not physical based, they are only used in certain fields under specified conditions. The flow-based approach calculates the movements of various agents in a similar way to the physical-based flow. In the process of discretizing a 2D simulation space, the agent is treated like particles covered by SPH (Smoothed Particle Hydrodynamics) [19,20]. These particles are usually advected according to the potential field to create the motion of the agent. In addition, studies have been proposed to control motion based on continuous granular flow [21] and continuum mechanics [22]. Most of these methods considered the local movement of the target, and some approaches even modelled the group of agents in consideration of global movement [8,9].

A^* search algorithm is widely used for real-time path finding in an interactive environment [23]. D^*(Dynamic A^* algorithm) is a highly accurate method in a partially known or constantly changing environment [24]. A^*-based algorithms require a static quantized research space depending on the nature of the environment. Stout et al. proposed a method to quantize the environment using rectangular grids, quad trees, convex polygons, and navigation meshes [25]. Navigation mesh is a data structure used by agents to find paths in a large virtual space. This is generally implemented as a graph structure, and this structure has been modified and improved by numerous algorithms. Recently, it is widely used in games [26,27] and commercial path-finding software [28,29].

Methods based on navigation mesh typically include preprocessing and runtime steps. During the preprocessing step, (1) loading terrain data and (2) generation of navigation mesh are performed. Using the loaded terrain data, the area in which the agent is allowed to move is made into a navigation mesh. When finding such a movable area, it is identified by using the slope information of the terrain or manually by the developer. In the runtime step, (1) destination setting, (2) shortest path search, and (3) agent movement are performed. Movement of the agent is specified according to the event trigger or user interaction. In the shortest path search step, the A^* algorithm is applied to calculate the shortest path to the destination included in the navigation mesh. In the movement step, the NPC moves along the calculated shortest path, avoiding collision with other agents due to local deviations inside the navigation mesh.

In an interactive virtual environment, the agent is classified as playable characters (PCs) and non-playable characters (NPCs). The movement of NPCs is simpler than that of the PC and moves along the shortest path mentioned above. However, due to the nature of the navigation mesh, the area not covered by the navigation mesh becomes an area where the NPC cannot move. The process of moving a PC in an interactive environment is slightly different from that of an NPC. A PC is usually moved by point designation or direction change determined by the user. Point designation means that the user directly designates the destination of the PC and is generally done through a mouse click. The directional change is determined by user input on the direction to be moved from the current position of the PC and is generally done through the direction key. A PC with a destination designated as a point uses the navigation mesh to find a path in the same way as an NPC, and when a direction key is used, the PC moves at a predetermined speed in a predetermined direction without referring to the navigation mesh.

The structure of this paper is as follows: Section 1.1 explains the research on planning NPC behavior in games and virtual environments, and Section 1.2 explains the limitations of previous research and the necessity of the proposed method. Section 2.1 explains how to create an accumulation map from user's path data, and Section 2.2 explains how to obtain the path data necessary to verify the proposed method. Finally, the experimental results are explained in Section 3. In particular, Section 3.1 explains the difference between the proposed method and the grid-based approach and the improvement in our study, and Section 3.2 explains the scalability of the proposed method.

1.1. Specific Technical Survey: NPC Behavior Planning in Games and Virtual Environments

Since the proposed technique is used to control NPC behavior not only in game but also in virtual environments, we review the latest methods related to this.

NPC behavior planning generally follows a hard-coded method or works to minimize a given objective function: hard-coded methods include rule-based systems, finite state machines [30], or behavior trees [31], and objective functions include the POPCoP [32] and Goal-oriented action planning (GOAP) [33] techniques. Rule-based systems (RBS) are used to manage AI behavior according to a set of rules. These are the basic forms of artificial intelligence systems [34] and are one of the most frequently used methods in games [35]. Finite state machines (FSMs) [36] provided a more sophisticated method to control NPCs in video games. Agents of the FSM model go through a set of distinct states during game play. For example, the guard NPC is usually in the post state, but when a sound is heard nearby, it switches to the searching state and moves. As such, FSMs are widely used in game development because they are easy to understand and run efficiently. However, it is cumbersome to create detailed FSMs for various NPCs, so a method of automatically generating them was also proposed [30].

Behavior tree is also a method used to control NPC behavior [31]. Each node allows NPCs to perform specific tasks or visits other nodes to control various actions. In this method, the behavior of an NPC for each node is activated while searching the behavior tree from the root node to the child node. For example, (1) the character moves to the door, (2) checks whether the door is locked, (3) unlocks the door if it is locked, and (4) decides the action to take after opening the door. This method has a problem that the behavior tree becomes too large when there are too many actions to be considered. To solve this problem, Shoulson et al. introduced a method of parameterizing the behavior tree by encapsulating a subtree [37].

The search-based planning technique is also used to generate NPC behavior in games, and in most games, it is used to control the behavior of NPC agents by using A* graph search similar to the behavior tree concept. The navigation mesh [38] is used to calculate the search-based path-planning structure in video games, and this is a mesh structure that encodes the part that can be explored in the game environment.

Goal-oriented action planning (GOAP) [33] is an expanded NPC behavior-planning method based on STRIPS, a classic planning method. GOAP is a graph approach that allows us to select potential NPC actions using abstraction of game elements and state space. A low-level behavior using GOAP is controlled by a simple three-state FSM. Geib et al. [39] proposed a way to express both plan recognition and planning so that we can understand what the PC is trying to do and can create a plan to support the PC. This method is similar to ours in that it observes the behavior of the PC, but it is a system that predicts and guesses the plan of the PC. In other words, this study is a system that proposes and plans a series of subgoals to help the PC and is different from our method. As the latest research on tactical behavior planning, N. Sturtevant proposed a way to consider the relationship with NPCs during path planning [40]. This technique focused only on path planning and used interactions with NPCs as a way to supplement the search cost function by classifying it separately without considering the distance of movement. Uriarte and Ontanon used past game play experience data when constructing policies that can be used to plan new game play decisions [41]. They used the Monte Carlo tree search technique to construct a policy and applied the policy to the $StarCraft$ game to demonstrate the results.

1.2. Problem Statement

As mentioned earlier, there are various methods for NPC behavior planning, but most of them are methods of searching for NPC behavior in a quantized space considering predefined rules or obstacles. In general, the path of an NPC is determined by searching the space created by the static surrounding environment. The problem with these approaches is that it can only be used in a static environment. Of course, it can be extended in the form of Dyanmic NavMesh, but in this case, it is ambiguous to present the relevant conditions in virtual environments and it is difficult to verify its accuracy.

If an NPC moves along NavMesh, which is based on the surrounding environment, regardless of static or dynamic, its path will not take into account the movement created by interactions with people. Although there is no guarantee that the user's path always interacts with the surrounding environment, we propose a framework for NPC behavior planning based on the user's path and search for a path that is close to the correct answer from the perspective of an actual user. In addition, our method improves the quality and stability of the navigation mesh based on path similarity so that the path is not broken.

2. Method

In this paper, we propose a novel method to find the path of the agent using trajectory data in which the user moves to solve the path-finding problem in an interactive environment. The key idea in this study begins with the assumption that the user controls the character more efficiently than the algorithm chooses. From this assumption, considering the trajectory in which the character moves that the user controls, a path similar to the player's choice is generated when the agent moves. In the early days when the virtual environment was operated, there was no accumulation of trajectory information from user movement, so it relied only on the original navigation mesh to find a path for the NPC to move along. As time goes by, users become familiar with the environment and discover new routes such as shortcuts, resulting in better paths as data accumulates. For example, PCs can move to unplanned routes on the system by jumping off cliffs or by specifying where the path is cut off, but NPCs do not. If the system learns paths optimized by the PC's experience and can integrate these learning results into path finding of the NPC, the NPC will be able to move more efficiently and naturally. This process is called path expansion calculated based on physical attributes, and in this paper, path expansion is designed based on statistical data based on physical attributes of actual users.

In an interactive environment, PCs generally adapt to the environment through their own play experience, improving their ability over time and showing smarter movements, but NPCs always show constant movement because there is no change in ability or movement. Users often exploit these agents' vulnerabilities during games, and this problem is a factor that degrades the quality of content. Evolutionary algorithms or artificial neural networks may be the solution, but to properly control the movement of the NPC using these techniques, an appropriate objective function based on various parameters is required. On the other hand, the proposed method allows the NPC to make smarter decisions when searching the path by learning only location information among various physical attributes of the user. In this paper, when the start point and destination of the PC agent are determined, path data are collected by calculating the behavior pattern, the path of the NPC is improved based on the collected data, and the path is finally adaptively changed.

Before calculating the trajectory of the PC, we calculate the path of the NPC under the assumption that we already have user's path data or that it is stored in the server (see Section 2.1). In the future, in order to verify the validity of the proposed NPC path control method, a method of obtaining path data in consideration of the user's behavior pattern will be described, and finally, experiments will be conducted using these data (see Section 2.2). A list of symbols is available in Table 1.

2.1. Generation of Accumulation Map with User's Path

We used 1000 path data to find the paths that users frequently use and build η, a density map, by accumulating the paths in a histogram format. Figure 2a is the accumulation map according to the movement path, and the closer the color is to red, the more users pass by.

Table 1. List of symbol used in the paper.

Symbol	Description	Value
η	Accumulation map	–
κ	Navigation mesh	–
p^*	Jittering movement of NPC	–
p_{start}	Starting point of NPC	–
p_{cur}	Current location of NPC	–
p_{dest}	Destination of NPC	–
p_{mid}	Transit point of NPC	–
Q	Shortest path search algorithm (i.e., A*)	–
R	Stochastic motion of NPC	–
θ	Altered direction	–
ϵ	Uniform random number	–
k	Scattering coefficient	$k > 0$
δ	Smoothed Heaviside function	–
h	Size of the grid cell	–
Ω	Cumulative map of NPC	–

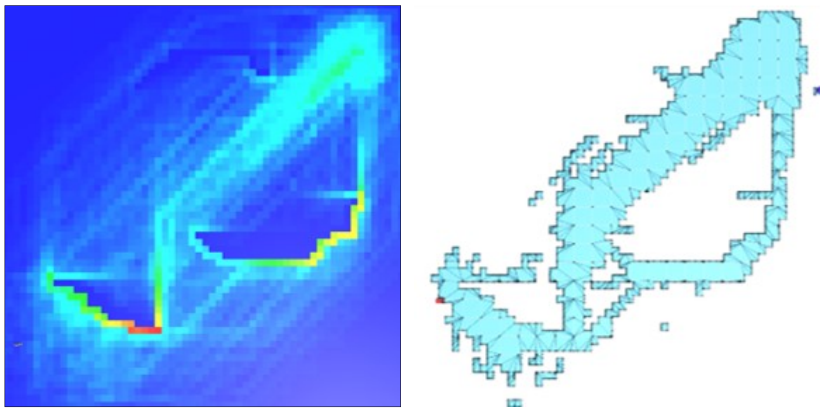
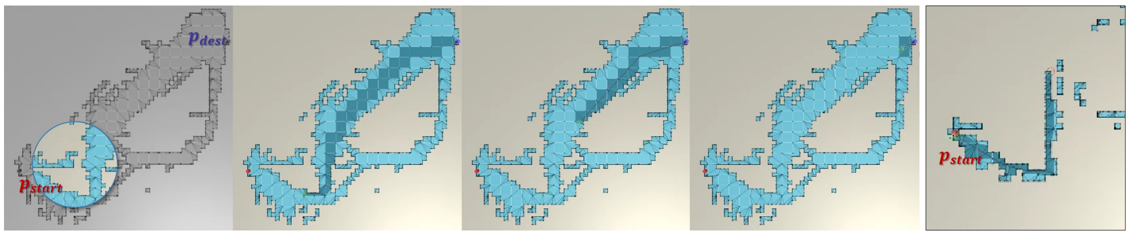

(**a**) Accumulation map (**b**) Refined navigation mesh

Figure 2. Visualization of the accumulation map and refined navigation mesh.

Based on this map, a navigation mesh, κ, was created using NavMesh provided by Unity3D (see Figure 2b). κ was normalized to extract the region near the route upon which the user mainly traveled, and κ was created only for regions with a density value of 0.1 or higher. In this way, a navigation mesh was built around the path upon which the user mainly traveled. The most stable result was obtained when the threshold of η for pruning the navigation mesh was set to 0.1 after several experiments.

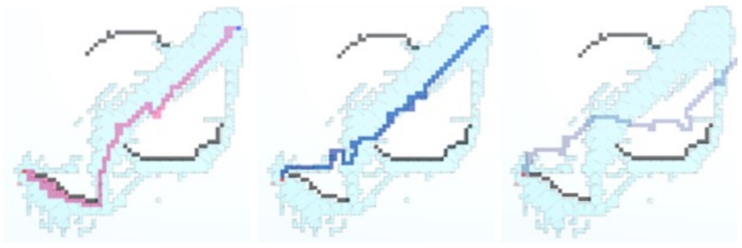
After κ was generated by the process described above, the NPC could be moved using it. Figure 3a shows the navigation mesh constructed from p_{start} to p_{dest}, and the path that the NPC actually moved along was marked as a dark square. This method was meaningful in that it was built in consideration of the user's path, but as shown in Figure 3a, the path was sometimes cut off (see the circled region in Figure 3a) because it was excessively dependent on the threshold of η when creating the navigation mesh. This method obtains different paths each time according to the user's threshold selection. If the threshold is too small or large, κ is unstablely pruned, resulting in a disconnected path, and the NPC cannot moving properly (see Figure 3b).



(**a**) NPC's path (threshold of η: 0.1) (**b**) Unstable navigation mesh

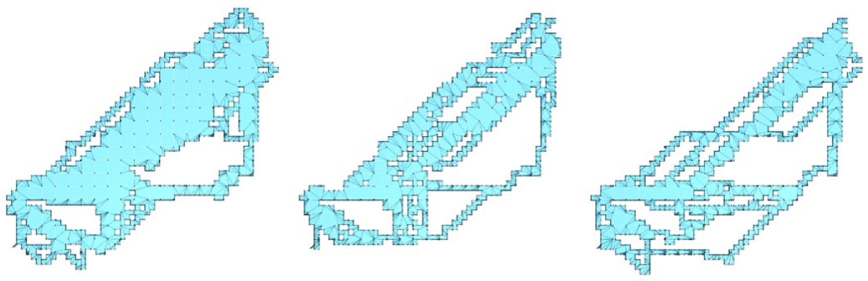
Figure 3. Visualization of the path of an NPC with η and the unstable case (circled region in (**a**): discontinuous path).

In order to generate a navigation mesh based on the path that the user frequently uses without sensitively responding to η, valid paths are collected based on the path similarity between the PC route and κ. In this process, if the PC path and the coordinates of κ overlap, its similarity value is high, and in this paper, PC paths with a similarity value of 70% or more were collected. Taking Figure 4 as an example, Figure 4a,b show paths with high similarity and Figure 4c shows paths with low similarity, which are significantly different from the navigation mesh. Figure 5 shows how the number of collected paths and similarity affect the formation of a navigation mesh. Compared to the left side of Figure 3a, a stable path was created without any broken areas. In the case of Figure 5b,c, different types of navigation mesh were created as the number of collected paths and similarity were changed, but the difference was not large in terms of the direction of the path itself and the navigation mesh was created a without disconnected path (see Figure 5).



(**a**) Valid (**b**) Valid (**c**) Invalid

Figure 4. Valid and invalid paths of playable characters (PCs) with path similarity.



(**a**) 70%, num.: 50 (**b**) 70%, num.: 15 (**c**) 80%, num.: 15

Figure 5. Visualization of navigation meshes with different path similarities and collected paths (%: similarity, num.: number of collected paths).

In the process of implementing the accumulation map, the path-similarity could be easily measured by checking whether the entire NPC path exists on the refined navigation mesh. We sample the $\eta(p^*)$ value corresponding to the NPC position p^*, and if this value is above the threshold, it is regarded as a valid position (threshold of η : 0.1). After eta is investigated for all positions constituting the path, if valid positions are more than 70% of the total path, it is regarded as a valid path (see Equation (1)).

$$\frac{|\eta(p_j^*) > 0.1|}{|p^*|} > 0.7 \tag{1}$$

where j is the index of the position constituting the path, and $|p^*|$ is the number of all positions constituting the path. As a result, we used the NavMesh supported by Unity3D to construct a navigation mesh, and we built it in the form of our proposal by delivering only the valid paths using flag.

2.2. Collecting User's Path Data

The jittering movement p^* is calculated by using the following equation in order to collect metadata for the evolution of the agent (see Equation (2)).

$$p^*(x) = (1 - \delta(w))\, Q(x) + \delta(w)\, R(x) \tag{2}$$

$$\delta(w) = \begin{cases} 0 & w < -\epsilon, \\ \frac{1}{2} - \frac{w}{2\epsilon} - \frac{1}{2\pi} sin\left(\frac{\pi w}{\epsilon}\right) & -\epsilon \le w \le \epsilon, \\ 0 & \epsilon < w. \end{cases} \tag{3}$$

In Equation (3), $w = \frac{\|p_{start} - p_{dest}\|}{\|p_{cur} - p_{dest}\|} - 1$, and p_{start}, p_{cur}, and p_{dest} indicate the starting point, current location, and destination, respectively. Q is the location obtained through the shortest path search algorithm such as the A^* algorithm, and R is the stochastic motion calculation based on the discrete random walk to model the movement pattern of the PC agent. When users find their way, they usually build zigzag paths because, at first, they are more concerned about their surroundings than about distant destinations. In order to model this, we set up θ to randomly turn the direction slightly (see Figure 6). When the NPC is far from the destination (i.e., starting point), they are strongly influenced by the movement caused by R, so they move freely regardless of the location or direction of the destination. This pattern is calculated in the method as follows (see Equation (4)).

$$cos\theta = \frac{2\epsilon + k - 1}{2k\epsilon - k + 1} \tag{4}$$

where $\theta, \epsilon \in [0,1]$, and $k \in [-1,1]$ represent the altered direction (in radian), a uniform random number, and the scattering coefficient, respectively. If $k = 0$ is isotropic scattering movement, $k > 0$ is forward scattering movement and $k < 0$ is a backward scattering movement.

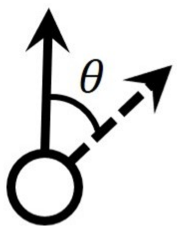


Figure 6. Zigzag pattern of an NPC's path: θ indicates the altered direction (see Equation (4)).

In this paper, we applied k to the positive number to show a zigzag path toward the destination. Usually, when a PC agent is near the starting point, the shortest path to the destination is not fixed, so zigzag paths appear, while the closer it gets to the destination, the faster and more accurately it finds the destination. The weight that affects this motion is δ (see Equation (3)). δ is a smoothed Heaviside (step) function, and when the NPC is near the starting position, the path with an irregular pattern appears because δ grows. As the NPC gets closer to the destination, δ becomes smaller, irregular movement decreases, and it moves straight toward the destination. We employed $\epsilon = 1.5h$, where h is size of the grid cell. Depending on the shape of Kernel, the NPC's path appears to be different, and in this paper, it was experimented and shown in Figure 7b.

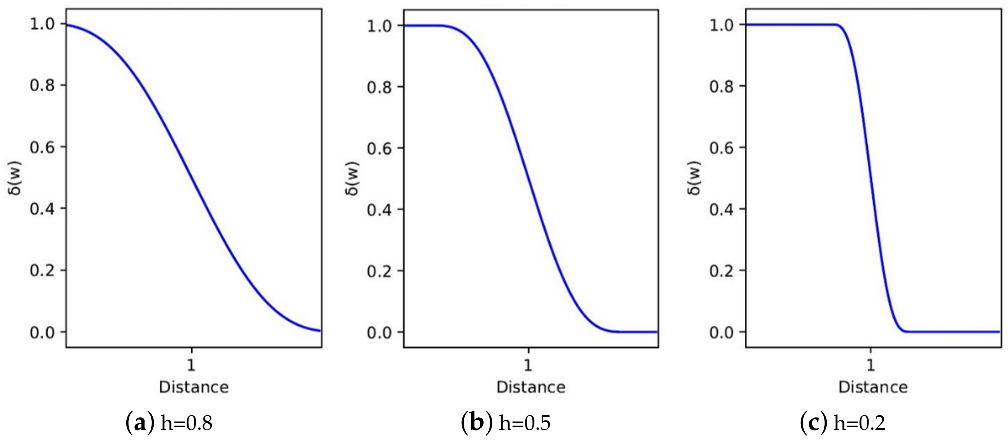


Figure 7. The graph of weight parameter δ with Equation (3).

As shown in Figure 8, in the process of proceeding from p_{start} to p_{dest}, R causes an irregular path near the starting point. However, when the obstacle is concave, the NPCs sometimes get stuck and cannot escape the obstacle. To avoid this problem, we conducted experiments under four conditions and showed two results with slightly different settings for each condition (see Figure 9). The locations of p_{start} and p_{dest} are randomly set to be slightly different for each experimental scene. *cond.* 1 is not allowed to go back the way the NPC has passed, and this condition has been suggested not to repeatedly hover over the local area. The NPC does not hover in the same area entirely, but sometimes, the R that produces the irregular movement of the NPC used by Equation (2) causes it to be stuck and unable to escape (see Figure 9a). If the obstacle is close to the starting position, δ becomes larger, so the movement of the NPC becomes more unstable due to the random pattern.

cond. 2 allows the NPC to go back on the path it went through, and because of this condition, when the destination is obscured by a concave obstacle, sometimes it cannot escape the obstacle and continues to hover near the corner of the obstacle (see Figure 9b). *cond.* 3 uses the cumulative map, Ω, to allow the NPC to pass through the same place again up to three times. Excluding the place where the NPC once went in the path calculation, it often goes past the obstacle too far, so we made it possible for the NPC to repeatedly pass the same place up to three times to compute path flexibly.

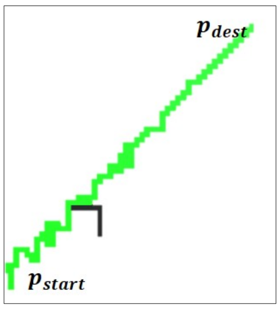
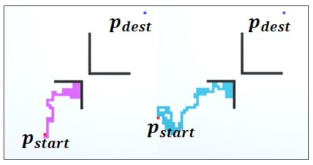
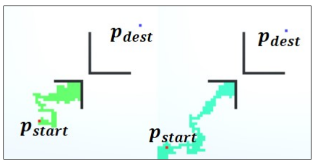


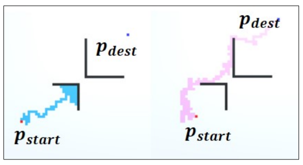
Figure 8. NPC's path with Equation (2).



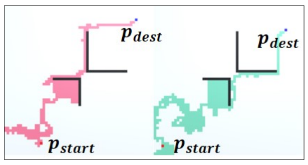
(**a**) Path using *cond*.1



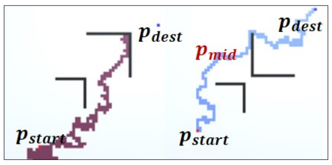
(**b**) Path using *cond*.2



(**c**) Path using *cond*.3



(**d**) Path using *cond*.4



(**e**) Path using *cond*.5

Figure 9. Visualization of the path of NPC with various conditions.

This value was obtained by experimenting with various scenes, and in most cases, a stable path was found. *cond*. 3 had a higher probability of avoiding obstacles than the two conditions previously introduced, but there were occasional situations in which it could not take into account irregular obstacle patterns or find a path by hovering around (see Figure 9c). *cond*. 4 uses R and Ω together when NPC enters the convex region surrounding the obstacle (see Figure 10). When the NPC is near a concave obstacle, it often tries to move in the direction of the destination and fails to escape in the opposite direction, so we move the NPC in various directions based on the convex region R, making it easier to get out of the obstacle. We randomly set the locations of p_{start}, p_{dest}, and obstacles for the four conditions mentioned above and conducted 1000 experiments. NPCs reached their destinations avoiding obstacles with success rates of $cond.1 : 10\%$, $cond.2 : 13\%$, $cond.3 : 20\%$, and $cond.4 : 96\%$, respectively. In this paper, the data of *cond*.4, which has the highest success rate, is used as metadata to control the movement of the NPC (see Figure 9d).

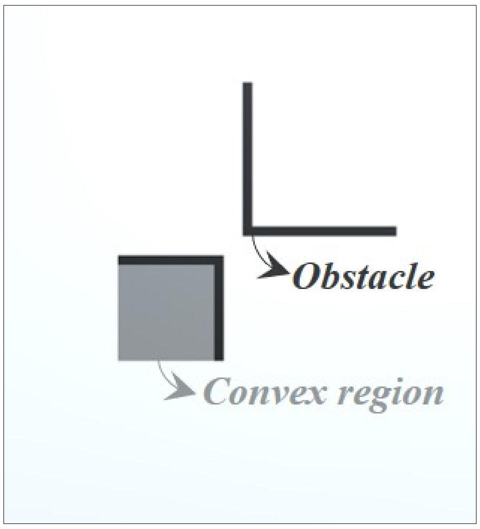


Figure 10. Convex region of obstacle.

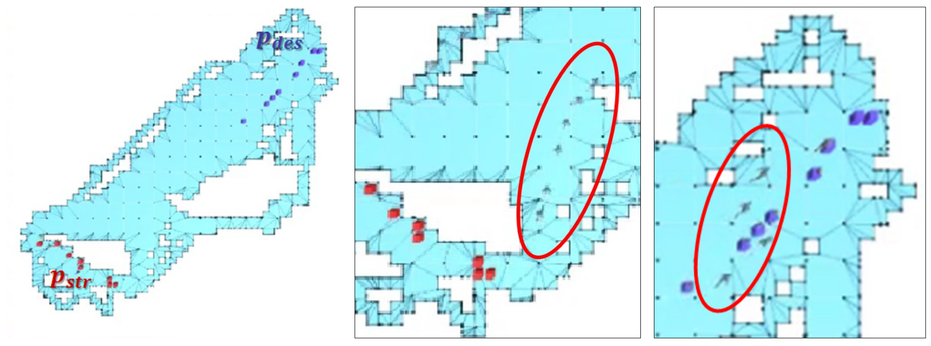
In addition, we can set the intermediate position, p_{mid}, to pass through to obtain different path patterns for several obstacles. As shown in Figure 9e, the left figure has no transit point (p_{mid}), so paths are set only in a similar pattern, but the right figure can consider a more diverse user path because a slightly different form of the path can be created each time due to p_{mid}. We created p_{mid} in the user-designated or random location and we determined that this method was similar to the actual PC behavior pattern because users move under different judgments. At the beginning of the game, different path types will appear due to the randomness inherent in the algorithm, but as users' data accumulates, the paths that they frequently go to will become clear. In this paper, we simulated the path data of users by the method described above because it is difficult to secure a large amount of user data.

Among the various conditions defined above, the reason that the success rates of *cond*.1 and *cond*.2 are low is because there is no action to observe while searching the surroundings like *seeking state*. For stable navigation, it is necessary for NPC to look around, not just towards destination, and this action increases the chances of finding the best path. When an NPC observes only a local area near itself, it often fails to find an optimal path, and this problem also occurs in a simple Q-Table-based reinforcement learning or Markov decision process. We are trying to find the optimal path in the end by randomly generating movements that seem to observe in all directions, rather than the NPC moving only toward the maximum reward. However, in *cond*.1 and *cond*.2, they are only to prevent or allow NPC to go back the way it already passed, so NPC tends to move around the same place or to care about only the destination. This often leads to a situation where the NPC is stuck in an obstacle and cannot escape, resulting in a low success rate.

3. Results and Discussion

The experimental results of this study were performed using a computer equipped with an Intel Core i7-7700K CPU, 32 GB RAM, and nVidia GeForce GTX 1080Ti GPU. To demonstrate the excellence of the proposed method, we checked the paths of NPCs in various environments, and it was confirmed that the unbroken path was stably generated. We propose a dynamic navigation mesh that is automatically updated in consideration of the user's path, so that the movement of the NPC can be easily improved without manual intervention by a developer or designer.

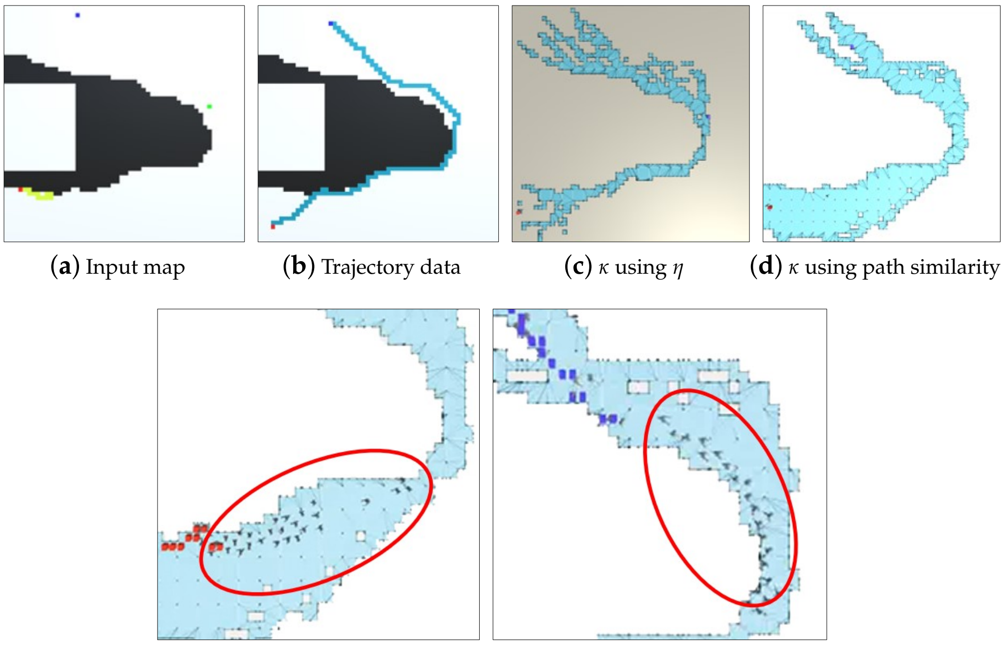
Figure 11 is the result of setting p_{start} and p_{dest} in the diagonal direction and observing the movement of NPCs. In this experiment, p_{start} and p_{dest} were set to multiple points to prevent NPCs from being concentrated in one place. As shown in Figure 11b,c, NPCs in the red ellipses moves smoothly to p_{dest} by using the navigation mesh created based on the user's path.



(**a**) Entire navigation mesh (**b**) Frame: 21 (**c**) Frame: 65

Figure 11. Scene 1—movement of NPCs with the navigation mesh (red dot: p_{start}, blue dot: p_{dest}, and red wire: NPCs).

Figure 12 shows the result of the NPCs moving according to the navigation mesh created based on the curved paths of PCs. In Figure 12a,b, black areas represent obstacle regions that NPCs or PCs cannot access. Figure 12b is the result of collecting path data using the equation, and 1000 path data were used in this paper. It takes about 12 min to calculate the 1000 paths required for NPC path control, and it is the longest time-consuming process in the proposed framework. We performed this process to obtain path data similar to the user's path, and if there are enough user data in the game server, we can skip this process and proceed in real-time. Figure 12c is a navigation mesh created by accumulating frequently passed paths in histogram format, and Figure 12d is an updated navigation mesh based on path similarity to prevent path breakage. By using this updated mesh, as shown in Figure 11, it worked stably not only in straight but also curved paths. Figure 13 shows the results of experiments in different environments. The number of branching points and their distributed shapes may vary depending on the threshold of the η proposed in our method, but the NPCs moved stably without breaking the path from the starting point to the destination.


(**a**) Input map (**b**) Trajectory data (**c**) κ using η (**d**) κ using path similarity

(**e**) Frame: 25 (**f**) Frame: 73

Figure 12. Scene 2—curved path of an NPC.

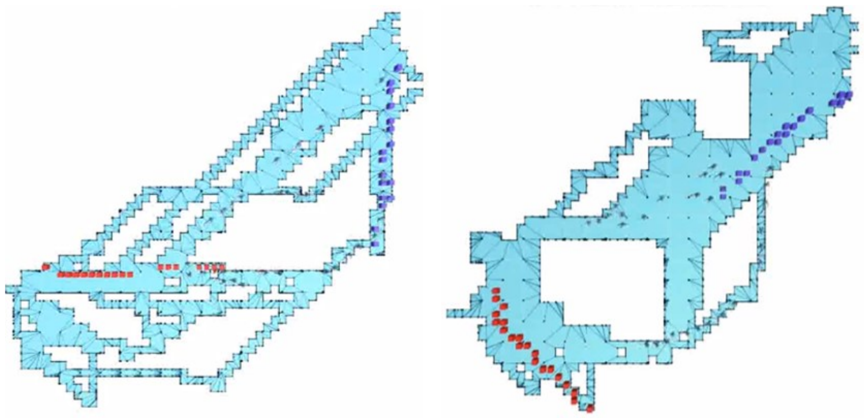

(**a**) 80%, num.: 15 (**b**) 70%, num.: 50

Figure 13. Scene 3—movement of an NPC with various navigation meshes (%: similarity, num.: number of collected path data).

Figure 14 shows the navigation mesh of NPC moving according to various colliders. Figure 14a,b have the same init. state, only their start positions and destinations are different. In the navigation mesh to which path-similarity is not applied, the problem of path break occurs, but the proposed method generates a navigation mesh stably. In particular, even though only 15 collected path data were used in Figure 14a, the path to the destination avoiding obstacles was well extracted.

Figure 15 is the result created using the path data of real users randomly selected, not the method introduced in Section 2.2. As shown in the figure, the navigation mesh was stably generated.

The proposed method was able to obtain similar results when using real user path data as well as when using virtual user path data.

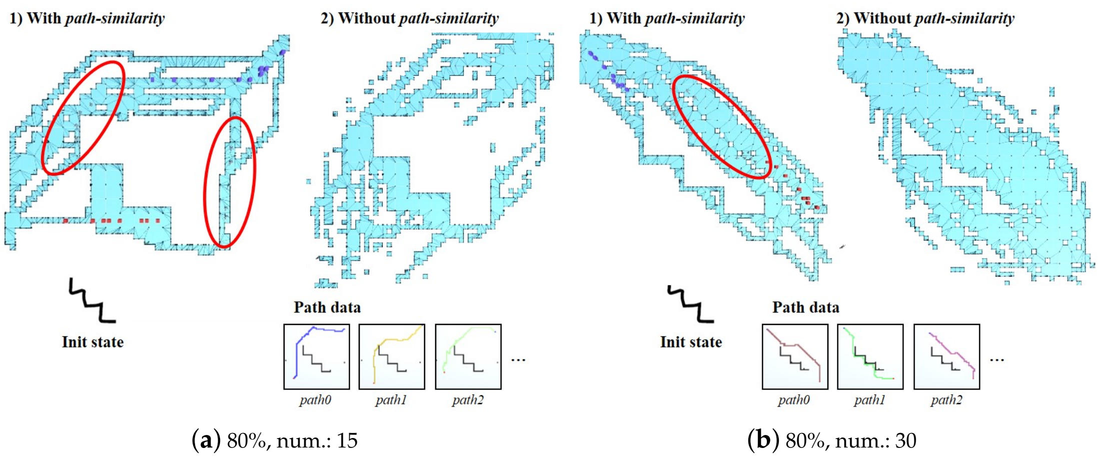
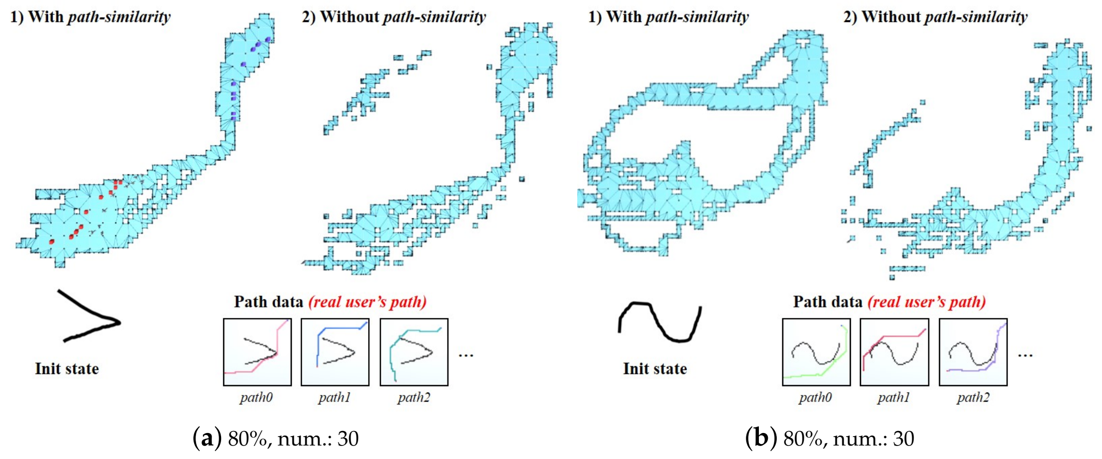


Figure 14. Scene 4—movement of an NPC with various navigation meshes (%: similarity, num.: number of collected path data).

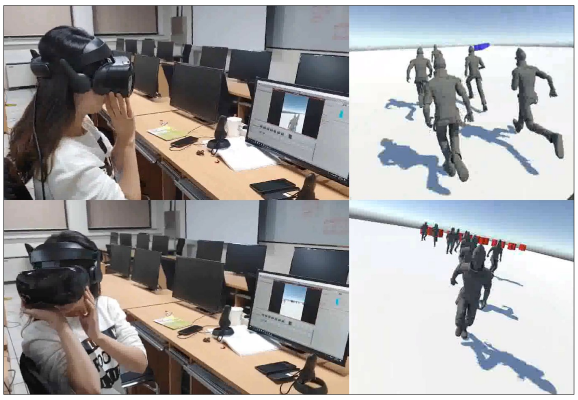


1) With *path-similarity*
2) Without *path-similarity*
Init state
Path data *(real user's path)*
path0
path1
path2
(**a**) 80%, num.: 30
(**b**) 80%, num.: 30

Figure 15. Scene 5—movement of an NPC with various navigation meshes (%: similarity, num.: number of collected path data).

Figure 16 is a VR demonstration to check whether each NPC moves naturally according to the navigation mesh from the first-person perspective. Like the previous results, the red and blue boxes represent p_{start} and p_{dest}, respectively, and it was confirmed that NPCs naturally ran to the destination. Since it is a first-person view using VR HMD (Head-Mounted Display), we turned around while running to observe the movement of the following NPCs and confirmed that they move naturally along the navigation mesh. As shown in this result, our method can be applied not only to games but also to applications that control agents based on navigation mesh, such as VR contents.

As confirmed from the results so far, if the proposed method is used, the NPC moves depending on the path the PC has passed, so different types of navigation meshes can be generated in the same map. Therefore, it is possible to overcome the limitations of existing NPCs, which always showed uniform movement on the same map, and because creation and update of the dynamic navigation mesh can be efficiently performed, user's intervention and time required for game production or maintenance can be greatly reduced. Many interactive graphic applications, such as games and VR contents, use numerous maps, and in particular, identifying the movable area on each map takes a long time and requires a lot of labor. Our method can greatly shorten this process, and since the navigation mesh is automatically generated based on the user's movement, all processes can be easily updated. In this paper, 1000 paths were calculated to obtain virtual user's path data, and most of them took 10 to 15 min. The accumulation map calculated during this process can be calculated much faster by using the actual user's data stored in the game server. In the process of creating a navigation mesh, not all of the 1000 path data but only the path selected by path similarity is used, so it is possible to quickly create a stable navigation mesh. In this paper, about 15 to 50 path data were selected for each scenario, and after the navigation mesh was created, the movement of an NPC could be controlled in real time. In addition, in order to measure the time required in the process of updating the navigation mesh using the proposed method, the navigation mesh was periodically updated using a path created differently in the same map, and it usually took 3 to 7 s to build η.


(**a**) Base navigation mesh: Figure 12

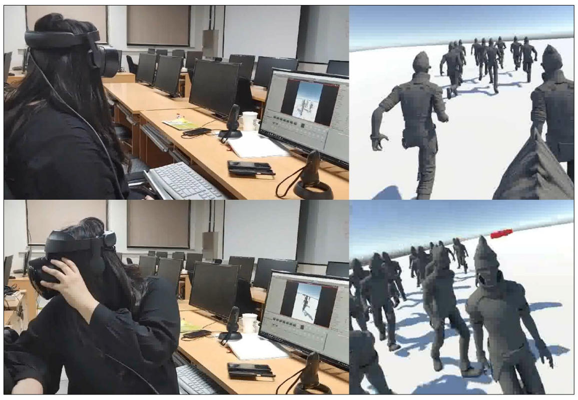

(**b**) Base navigation mesh: Figure 13a

Figure 16. Demonstration of NPC movement in a virtual reality (VR) environment.

Table 2 shows the configuration and computation time of the experimental results in this paper. In all examples, 1000 path data were used, and the calculation took about 12 minutes without significant influence by the shape of the collider. Actually, it takes a lot of time to obtain user path data, and the process of calculating the navigation mesh η does not take long. The proposed method showed that if a sufficient amount of user data is stored in the DB, the navigation mesh can be updated in about 3∼5 seconds. Since Figure 15a,b use real user path data, there are no path DB number and computation cost. We used a grid resolution of 50×50 to calculate η in all experimental examples, and since sufficient navigation meshes were built under this configuration, we did not increase the resolution any more. If the resolution of the grid is increased, the overall calculation time will be increased.

Table 2. Computation time and size of our example scenes.

Figure	(Selected Number of Path Data)	(Number/Time of Path DB (to Get Path Data for Calculating) η)	(Computation Time for Calculating η)	Grid Resolution of η
Figure 11	30	1000/12 min	5 sec	50×50
Figure 12	30	1000/12 min	5 sec	50×50
Figure 13a	15	1000/12 min	3 sec	50×50
Figure 13b	50	1000/13 min	7 sec	50×50
Figure 14a	15	1000/12 min	3 sec	50×50
Figure 14b	30	1000/13 min	4 sec	50×50
Figure 15a	30	–	4 sec	50×50
Figure 15b	30	–	4 sec	50×50

3.1. Comparison between Our Method and Previous Approaches

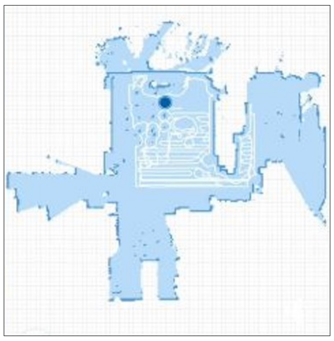
Heuristic search finds the path from the start state to the goal state in the graph. Heuristic search techniques such as A* [25], similar to ours, are based on Dijkstra's algorithm, but use a heuristic method to guide the search process. This algorithm discretizes the space in the form of an implicit graph that connects the space of the game world state to the surrounding state in order to apply graph search to the behavior planning domain of NPC. Many games use the A* algorithm, but since only the surrounding environment is statically considered, the NPC's movement is also static. Weighted A* [42] is a heuristic search technique that is biased toward a destination more than the A* algorithm. Actually, this method can search a path to a destination within a calculation time shorter than A*. However, since this method is also based on the A* algorithm, the movement of the NPC only considers the surrounding environment, but not the movement of the PC.

The ARA* (Anytime A*) [43] technique quickly finds an initial solution and then continues to improve it until the given time is over. Therefore, this method can adjust the degree of performance and stability according to the specified search time. Since this technique also focuses on improving performance, behavior planning is not much different from the previously mentioned techniques. D* [44] and D* Lite [45] solved the limitations of static A* approaches by efficiently replanning the search space when the surrounding environment changes dynamically. These methods reuse the search tree obtained from previous planning episodes and update the graph structure in consideration of the changes. Compared to the traditional A* and Weighted A* techniques, dynamic surroundings can be considered, but these techniques also have a structure that is difficult to reflect the user's movements or behavior patterns according to events. Independent Multi-Heuristic A*, Shared Multi-Heuristic A* [46], and improved Multi-Heuristic A* algorithms [47] are heuristic search techniques used for the search process. These methods have been used to deal with difficult topographic search spaces or large terrain, and do not adaptively update the navigation mesh based on the user's movement. The aforementioned techniques actually calculate a path based on the movement of the NPC focused on the terrain change, and our method creates a path by analyzing the motion of the PC. NPCs in games or VR contents should react to the user's motion rather than the surrounding terrain to make the user more immersed in the contents. However, since the process of constructing a navigation mesh in consideration of the surrounding environment and the user's movement can be easily integrated with existing techniques, the portability of our technique is good.

3.2. Development Potential

The proposed navigation mesh can be used in robotics as well as the movement of NPCs in games and VR contents. Robot vacuum cleaner is a representative product that moves in consideration of the surrounding environment (see Figure 17). A robot vacuum cleaner cleans a given indoor space by scanning surrounding furniture or room structures. In this process, the robot vacuum cleaner builds a navigation mesh through the scanning function (see Figure 17a), and cleans the interior by moving as if an NPC moves in games. Sometimes, it is necessary to intensively clean a certain area by analyzing the trajectory that the user moves as well as the designated space. If the user's path is analyzed by measuring sound or connecting to external devices such as a smartphone, and the navigation mesh is periodically updated based on this, the robot vacuum cleaner will be able to show smarter performance.

Recently, robotic path planning techniques based on machine learning have been continuously proposed [48]. These are techniques to find map features or map coordinate space in the process of scanning space [49]. In addition to research that scans space while tracking based on ray-tracing, there are also C-space techniques that accurately scan space based on CAD data [50]. Recently, methods for finding a path based on supervised learning (i.e., *Reinforcement Learning*, *Support Vector Machines*, etc.) have also been proposed [49]. By integrating our method with the techniques studied focusing on the spatial accuracy mentioned above, it can be extended to a more user-friendly path-finding technique.


(**a**) Robot vacuum cleaner

(**b**)The navigation mesh obtained through the scanning function of a robot vacuum cleaner

Figure 17. A robot vacuum cleaner that moves according to the navigation mesh.

4. Conclusions and Future Work

In this paper, we proposed a framework that can control the NPC's movement path by analyzing the user's path. In particular, even with the same map, the NPC moves in various changed paths according to the change in the user's movement trajectory rather than a fixed path. A stable navigation mesh can be created by adding a path-similarity technique so that the paths that users mainly use are not cut off. The data used in all results were 1000 path data, and after the construction of the navigation mesh, the NPCs moved in real time. In the future, we will apply this result to contents with a vast map such as VR and expand the technique so that NPC scan automatically determine the pattern of running, walking, climbing, or descending while interacting with the user. In addition, we plan to study a navigation mesh generation technique that more accurately controls the movement of NPCs by reflecting various physical properties such as sound, speed, and viewpoint as well as location when real users navigate the map.

Author Contributions: Conceptualization, J.-H.K. and J.L.; methodology, J.-H.K.; software, S.-J.K.; validation, J.-H.K., S.-J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by a Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (NRF2020R1A2C101519511).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kavraki, L.E.; Svestka, P.; Latombe, J.-C.; Overmars, M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **1996**, *12*, 566–580. [CrossRef]
2. Choi, M.G.; Lee, J.; Shin, S.Y. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph.* **2003**, *22*, 182–203. [CrossRef]
3. Kamphuis, A.; Overmars, M.H. Finding paths for coherent groups using clearance. In Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Grenoble, France, 27–29 August 2004; pp. 19–28.
4. Li, Y.; Gupta, K. Motion planning of multiple agents in virtual environments on parallel architectures. In Proceedings of the 2007 IEEE International Conference on Robotics and Automation, Roma, Italy, 10–14 April 2007; pp. 1009–1014.
5. Berg, J.V.D.; Patil, S.; Sewall, J.; Manocha, D.; Lin, M. Interactive navigation of multiple agents in crowded environments. In Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, Redwood City, CA, USA, 15–17 February 2008; pp. 139–147.
6. Pettré; Ciechomski, P.D.H.; Maïm, J.; Yersin, B.; Laumond, J.-P.; Thalmann, D. Real-time navigating crowds: scalable simulation and rendering. *Comput. Animat. Virtual Worlds* **2006**, *17*, 445–455.

7. Pettré, J.; Grillon, H.; Thalmann, D. Crowds of moving objects: Navigation planning and simulation. In Proceedings of the 2007 IEEE International Conference on Robotics and Automation, Roma, Italy, 10–14 April 2007; pp. 3062–3067.
8. Sud, A.; Andersen, E.; Curtis, S.; Lin, M.; Manocha, D. Real-time path planning for virtual agents in dynamic environments. *IEEE Virtual Real. Conf.* **2007**, 91–98. [CrossRef]
9. Sud, A.; Gayle, R.; Andersen, E.; Guy, S.; Lin, M.; Manocha, D. Real-time navigation of independent agents using adaptive roadmaps. *ACM SIGGRAPH* **2008**, 56. [CrossRef]
10. Ashida, K.; Lee, S.-J.; Allbeck, J.; Sun, H.; Badler, N.; Metaxas, D. Pedestrians: Creating agent behaviors through statistical analysis of observation data. In Proceedings of the Computer Animation 2001 Fourteenth Conference on Computer Animation (Cat No 01TH8596) CA-01, Seoul, Korea, 7–8 November 2001; pp. 84–92.
11. Shao, W.; Terzopoulos, D. Autonomous pedestrians. *Graph. Model.* **2007**, *69*, 246–274. [CrossRef]
12. Thalmann, D. Populating virtual environments with crowds. In Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications, Hong Kong, China, 14–17 June 2006; Volume 69, p. 11.
13. Reynolds, C.W. Flocks, Herds and schools: A distributed behavioral model. In Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, Anaheim, CA, USA, 27–31 July 1987; pp. 25–34.
14. Pelechano, N.; O'Brien, K.; Silverman, B.; Badler, N. *Crowd Simulation Incorporating Agent Psychological Models, Roles and Communication*; Pennsylvania Universityhiladelphia Center for Human Modeling and Simulation: Fort Belvoir, VA, USA, 2005.
15. Musse, S.R.; Thalmann, D.; Panne, M. A model of human crowd behavior: Group inter-relationship and collision detection analysis. *Comput. Animat. Simul.* **1997**, 39–51. [CrossRef]
16. Schreckenberg, M.; Wolf, D.E. *Traffic and Granular Flow'97*; Springer: Singapore, 1998; p. 301.
17. Dadova, J. Cellular automata approach for crowd simulation. In *Faculty of Mathematics, Physics and Informatics*; Comenius University: Bratislava, Slovakia, 2012.
18. Tu, X.; Terzopoulos, D. Artificial fishes: Physics, locomotion, perception, behavior. In Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, Orlando, FL, USA, 24–29 July 1994; pp. 43–50.
19. Helbing, D. Self-Organized pedestrian Crowd Dynamics and Design Solutions.*Transp. Sci.* **2005**, *39*, 1–24. [CrossRef]
20. Lakoba, T.I.; Kaup, D.J.; Finkelstein, N.M. Modifications of the Helbing-Molnar-Farkas-Vicsek social force model for pedestrian evolution.*Transp. Sci.* **2005**, *81*, 339–352. [CrossRef]
21. Cristiani, E.; Piccoli, B.; Tosin, A. Multiscale modeling of granular flows with application to crowd dynamics. *Multiscale Model. Simul.* **2011**, *9*, 155–182. [CrossRef]
22. Treuille, A.; Cooper, S.; Popović, Z. Continuum crowds. *ACM Trans. Graph. (TOG)* **2006**, *25*, 1160–1168. [CrossRef]
23. Hart, P.; Nilsson, N.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
24. Stentz, A. *Optimal and Efficient Path Planning for Unknown and Dynamic Environments*; Carnegie-Mellon University: Pittsburgh, PA, USA, 1993.
25. Stout, B. *The Basics of A* for Path Planning, Game Programming Gems*; Charles River Media: New York, NY, USA, 2000.
26. Thompson, C. Halo 3: How Microsoft labs invented a new science of play.*Wired Mag.* **2007**, *15*, 1–7.
27. Guzman, E. Valve thrills FPS fans with new Counter-Strike game. *2D-X-The Best Damn Video Game Site Period* **2011**.
28. Intel Corporation. *Havok, Intel To Acquire, en Lınea*; Intel Corporation: Santa Clara, CA, USA, 2007.
29. Comeau, G.; Paramonoff, A. Data Path Engine. U.S. Patent Application 20,020,016,869, 6 December 2001.
30. Coman, A.; Muñoz-Avila, H. Automated generation of diverse npc-controlling fsms using nondeterministic planning techniques. In Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Snowbird, UT, USA, 14–18 October 2013; pp. 121–127.
31. Damian, I. Handling Complexity in the halo 2 ai. Available online: https://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php (accessed on 15 March 2020).

32. Macindoe, O.; Kaelbling, L.P.; Lozano-Pérez, T. Pomcop: Belief space planning for sidekicks in cooperative games. In Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford, CA, USA, 8–12 October 2012.
33. Jeff, O. Symbolic representation of game world state: Toward real-time planning in games. In Proceedings of the AAAI Workshop on Challenges in Game AI, Valencia, Spain, 4–8 June 2012.
34. Donald Kehoe, Designing Artificial Intelligence for Games. 2009. Available online: https://software.intel.com/content/www/us/en/develop/articles/designing-artificial-intelligence-for-games-part-1.html (accessed on 25 July 2020).
35. James, W. *Artificial Intelligence in Games: A Look at the Smarts Behind Lionhead Studio'S "Black and White" and Where It Can and Will Go in the Future*; University of Rochester Press: Rochester, NY, USA, 2002.
36. Jack, B.; Albert, G.H.; Allen, K. Encyclopedia of Computer Science and Technology. Available online: https://www.e-reading-lib.com/bookreader.php/135785/harry-henderson-encyclopedia-of-computer-science-and-technology.pdf (accessed on 30 July 2020).
37. Shoulson, A.; Garcia, F.M.; Jones, M.; Mead, R.; Badler, N.I. Parameterizing behavior trees. In *Motion in Games*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 144–155.
38. Greg, S. Simplified 3d movement and pathfinding using navigation meshes. In *Game Programming Gems*; Charles River Media: New York, NY, USA, 2000; pp. 288–304.
39. Christopher, G.; Janith, W.; Sergey, M.; Pavan, K.; Bart, C.; Ronald, P.A. Petrick, Building helpful virtual agents using plan recognition and planning. In Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Burlingame, CA, USA, 8–12 October 2016.
40. Nathan, R. Sturtevant, Incorporating human relationships into path planning. In Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Boston, MA, USA, 14–18 October 2013.
41. Alberto, U.; Santiago, O. Improving monte carlo tree search policies in starcraft via probabilistic models learned from replay data. In Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Burlingame, CA, USA, 8–12 October 2016.
42. Ira, P. Heuristic search viewed as path finding in a graph. *Artif. Intell.* **1970**, *1*, 193–204.
43. Likhachev, M.; Ferguson, D.; Gordon, G.; Stentz, A.; Thrun, S. ARA*: Anytime A* Search with Provable Bounds on Sub-Optimality. Available online: https://cse.sc.edu/~mgv/csce580sp17/gradPres/pohl_HeuristicSearch_Weighted1970.pdf (accessed on 30 June 2020).
44. Anthony, S. Optimal and efficient path planning for unknown and dynamic environments. In Proceedings of the Conference on Neural Information Processing Systems, Denver, CO, USA, 14–16 September 1993; Volume 10, pp. 89–100.
45. Koenig, S.; Likhachev, M. Fast replanning for navigation in unknown terrain. *IEEE Trans. Robot.* **2005**, *21*, 354–363. [CrossRef]
46. Sandip, A.; Siddharth, S.; Venkatraman, N.; Victor, H.; Maxim, L. Multi-heuristic A*. *Int. J. Robot. Res.* **2015**, *33*, 305–320.
47. Venkatraman, N.; Sandip, A.; Maxim, L. Improved Multiheuristic a* for Searching with Uncalibrated Heuristics. Available online: https://www.researchgate.net/publication/278965100_Improved_Multi-Heuristic_A_for_Searching_with_Uncalibrated_Heuristics (accessed on 25 June 2020).
48. Otte, M.W. *A Survey of Machine Learning Approaches to Robotic Path-Planning*; University of Colorado: Boulder, CO, USA, 2015.
49. Thrun, S.; Montemerlo, M.; Dahlkamp, H.; Stavens, D.; Aron, A.; Diebel, J.; Fong, P.; Gale, J.; Halpenny, M.; Hoffmann, G.; et al. Stanley: The robot that won the DARPA Grand Challenge. *J. Field Robot.* **2006**, *23*, 661–692. [CrossRef]
50. LaValle, S.; Keffner, J. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*; AK Peters: Wellesley, MA, USA, 2001; pp. 293–308.