http://dx.doi.org/10.5370/JEET.2011.6.6.867

Reducing the Search Space for Pathfinding in Navigation Meshes by Using Visibility Tests

Hyungil Kim*, Kyeonah Yu and Juntae Kim†**

Abstract – A navigation mesh (NavMesh) is a suitable tool for the representation of a three-dimensional game world. A NavMesh consists of convex polygons covering free space, so the path can be found reliably without detecting collision with obstacles. The main disadvantage of a NavMesh is the huge state space. When the A^* algorithm is applied to polygonal meshes for detailed terrain representation, the pathfinding can be inefficient due to the many states to be searched. In this paper, we propose a method to reduce the number of states searched by using visibility tests to achieve fast searching even on a detailed terrain with a large number of polygons. Our algorithm finds the visible vertices of the obstacles from the critical states and uses the heuristic function of A^*, defined as the distance to the goal through such visible vertices. The results show that the number of searched states can be substantially reduced compared to the A^* search with a straight-line distance heuristic.

Keywords: Path finding, Navigation mesh, Visibility graph, A^* search

1. Introduction

An efficient pathfinding process is necessary to achieve autonomous movements of characters in a three-dimensional (3D) environment, and A^* algorithm is generally used for searching [7, 17]. In addition, elaborated terrain representation is required for the realistic movement of characters, and this increases the search time for path planning. Several methods for the representation of game space exist, such as regular grids, waypoint graphs, navigation meshes, and visibility graphs. The *regular grid method* [18, 19] decomposes the game space into tiles; the state space is the set of tiles and the operators are the moves to the next tile in each compass and each diagonal direction. This method is simple, but suffers from large state space. *The waypoint graph* consists of nodes and links manually placed by a level designer [16, 18]. There is a trade-off between the size of the search space and the path quality. *The visibility graph* (Vgraph) consists of a start node, a goal node, nodes placed at the vertices of the obstacles, and links connecting them, which include obstacle boundaries [3, 5, 11]. The strongest advantage of using Vgraph is the guarantee of the shortest path with the A^* algorithm [9, 14]. *The navigation mesh* (NavMesh) decomposes a terrain as a set of convex polygons on a walkable space [6, 15, 18]. The cells are irregular, instead of regular grids, and they can represent the various properties of the terrain. A popular type is the triangle-based NavMesh, where cells are represented by various triangles.

Among the four representative methods, the NavMesh is the most preferred for representing a 3D game world because it can simplify the representation of 3D game space by a warped two-dimensional (2D) field. A 3D game world is often represented by the waypoint graph. However, such representation has many limitations due to the placement of the manual node: it is difficult to implement on a huge map or to share the intentions of the placement among people in different levels of the design process. Moreover, it is hard to apply the regular grid method and the Vgraph in a 3D game world without additional processing. Path planning can be more easily performed on a NavMesh by projecting the 3D surfaces onto a 2D space. The NavMesh, which can be generated automatically, is more flexible in representing a 3D surface. One drawback is that pathfinding on a NavMesh can be inefficient due to the large search space. This can be solved by generating a simplified NavMesh [10] based on the quadtree method or by simplifying an existing NavMesh by merging adjacent polygons [4]. However, methods based on NavMesh simplifications have limitation when the terrain is represented with an elaborated mesh of a large number of triangles to achieve more realistic movements.

This paper proposes an efficient method to search the state space represented by a NavMesh. We incorporate the concept of the Vgraph into the NavMesh to reduce the search space with visibility tests, which increases the speed of searches even on an elaborated terrain with a large number of polygons. We first find the visible vertices of the obstacles from the current state and define the heuristic

† Corresponding Author: Department of Computer Engineering, Dongguk University, Korea (jkim@dongguk.edu).
* Department of Multimedia, Korea Nazarene University, Cheonan, Korea (hikim@kornu.ac.kr)
** Department of Computer, Duksung Women's University, Seoul, Korea (kyeonah@duksung.ac.kr)
Received: November 21, 2010; Accepted: September 1, 2011

function of A* as the distance to the goal through such vertices. With the proposed heuristic, the number of states visited decreases substantially compared to the plain search. The efficiency of the proposed algorithm is demonstrated through simulations with the plain A* and the A* with visibility tests, respectively.

2. Pathfinding on a NavMesh

A NavMesh is the 2D projection of a 3D space represented with a set of small convex polygons that comprise the geometry information, such as the inclination of the surface or the type of terrain. We can use 2D search for pathfinding once a terrain is represented by a NavMesh. This paper focuses on the A* search algorithm for pathfinding on the triangle-based NavMesh.

2.1 NavMesh

A NavMesh decomposes the walkable 3D terrain into a set of convex polygons. It should satisfy three conditions [15]. First, it must consist of triangles, each of which is contained only in a single plane. Second, all adjacent meshes should have two vertices and one side in common. Third, no two meshes should share a common plane. Fig. 1 shows an example of a terrain with obstacles represented using a NavMesh. The NavMesh can be applied to various types of terrain structures. By modeling the NavMesh to be mutually exclusive of the static obstacles, the collision of characters to the obstacles can be easily detected by the examination of whether it reaches the boundary of the mesh.

When a character moves on a mesh, the movement vector is projected onto the plane of the corresponding triangle, and a 2D algorithm checks the intersection of this vector and the sides of the triangle. The character will collide with an obstacle if the vector intersects with a side that is not shared with another triangle.

2.2 A* Algorithm on a NavMesh

The pathfinding on a NavMesh can be performed with the A* search algorithm. The state space in a NavMesh is the set of triangles, and the operators are the moves to the next triangles. In Fig. 1, the start and the goal states are the triangles that include the start node and the goal node, respectively. Three triangles – A, B, and C – around the start state are its child states. The cost from one state to its child state is defined as the distance from the center of the state to the center of the child state through the middle point of the sharing side. The evaluation function $f(n)$ of a state n is the sum of the distance from the start state to state n and the heuristic value of the distance from the state n to the goal state. The most widely used heuristic in pathfinding is the Euclidean distance, which is defined as the straight-line distance between n and the goal state. The Euclidean distance heuristic is admissible.

Fig. 1 shows an example path found by the A* search. The search result is a sequence of adjacent triangles that is indicated as the colored triangles. The dashed line indicates the trajectory connecting the centers of the adjacent triangles through the middle point of the sharing side. The actual movement trajectory can be made more straight and smooth by using various techniques [8, 12]; however, it is not shown in the figure.

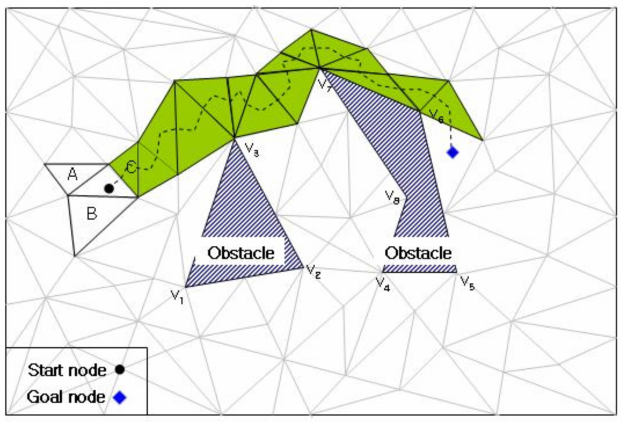


Fig. 1. An example of a terrain represented in a navigation mesh and the result of the search; the sequence of the shaded triangles is a path from the start node to the goal node

3. Pathfinding with Visibility Tests

Pathfinding on a NavMesh becomes inefficient when many triangles are used for fine granularity. This is because each triangle corresponds to a state in the state space search. A large number of states is sometimes visited by the search algorithm due to obstacles. This can be avoided if the algorithm utilizes the information that the shortest path must go through one of the visible vertices of the obstacles. This section explains the method of using the visibility information to reduce the size of the state space searched.

3.1 Visibility tests

Every visible vertex of polygonal obstacles can be reached by a straight line, and the shortest path between the start and the goal consists of visible edges - the edges between two vertices that can see each other. This set of visible edges is named by the *visibility graph*. The visibility graph is widely used for pathfinding in robot path planning and is recently applied for point-of-visibility pathfinding in computer games because it guarantees the shortest path [13, 20]. However, when the visibility graph is used directly in path-finding, it cannot use the detailed information kept in every cell of a NavMesh because it provides only a raw connection between nodes. In this paper, the visibility information is used to compute a superior heuristic function for the A* algorithm while the search is performed on a NavMesh. The visibility is tested in critical states, including the start state, as A* search proceeds. The next section will explain how these critical states are determined.

Finding all visible vertices from a point *p* is computationally expensive because every line connecting *p* and each vertex must be checked to see if it intersects with any obstacle edge. The time complexity is O(n^2), where *n* is the number of vertices of the obstacles. We use the rotational sweepline algorithm for a rapid visibility test [2]. According to the rotational sweepline algorithm, as we sweep the plane in cyclic order, we can use the information of the previous test for the rest. In Fig. 2, the plane is swept by the half-line around *p*.

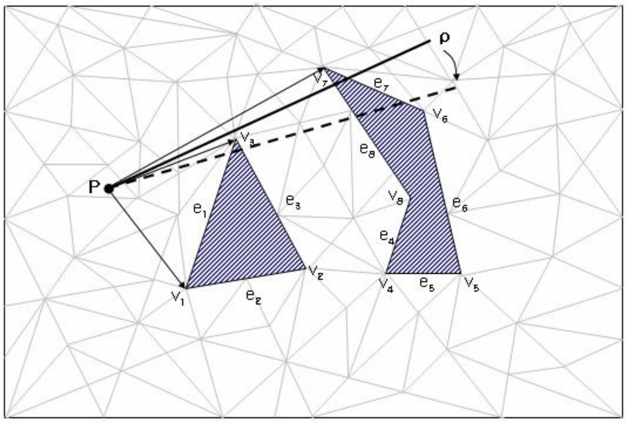


Fig. 2. The visibility test process: v_1, v_3, v_7 are the visible vertices from *p*; the whole plane is swept by the half-line around *p* in cyclic order

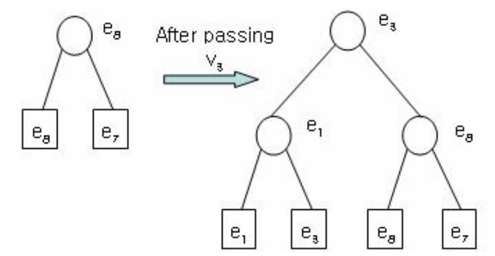


Fig. 3. The binary search tree on the intersected edges; the leaves of the tree store the intersected edges in order

The information about obstacle edges intersected by ρ is maintained in a binary search tree in order. The obstacle vertices are sorted in order of the clockwise angle and stored in set *W* before the sweep starts: $W = \{v_7, v_3, v_6, v_8, v_5, v_4, v_2, v_1\}$. The sweeping stops at every event (vertex) in *W* and checks whether it is visible from *p* or not. This can only be accomplished by checking the intersection between the line segment pv_i and the edge of the leftmost leaf node of the binary search tree.

For example, in Fig. 2, the second event is v_3, and ρ intersects with the edges e_8 and e_7 before v_3 and with the edges e_1, e_3, e_8, and e_7 after v_3. This information is stored in the binary search tree (Fig. 3). The visibility of v_3 from *p* can be easily verified by checking whether the line segment pv_3 intersects with e_8. Since pv_3 does not intersect with e_8, v_3 is visible from *p*. The binary search tree is then updated by the deletion of all edges on the counter-clockwise side of v_3 and by the insertion of new edges on the clockwise side of v_3. This sweeping process continues until ρ visits all the events in *W*. When there are *n* obstacle vertices, the number of events is *n*, the initial sorting takes O(*nlogn*) time, searching the leftmost edge in the tree takes O(*logn*), and the status update operations takes O(*logn*). Therefore, the overall time complexity is O(*nlogn*).

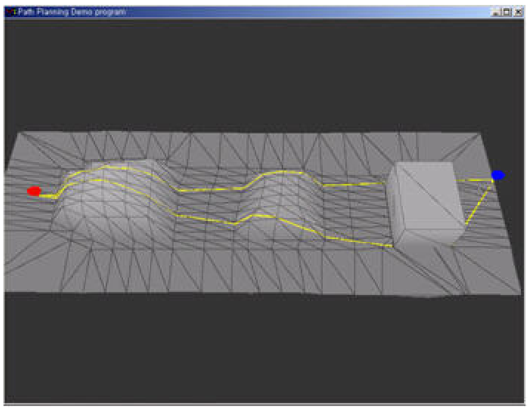


Fig. 4. An example of a visibility test in a 3D terrain; the cube at the right is an obstacle

In a 3D terrain, the visibility test can be done on a 2D projection of the mesh. In such case, the vertices found may not be actually visible, but reachable without changing the direction of movement [1]. Fig. 4 shows the result of the visibility test and the line drawn on the 3D surface.

3.2 Integrating visibility information into the A* search

The terrain is decomposed into cells in a NavMesh, each of which has meaningful information about the terrain. Therefore, we search cells on a NavMesh and check the visibility from a state whenever necessary to integrate the visibility information into the heuristic function, instead of constructing the whole visibility graph. The simplest admissible heuristic for pathfinding is the Euclidean distance between a state *s* and a goal *G*, *d*(*s,G*). This heuristic is admissible [9], and we modify it by integrating the visibility information as follows.

We assume that the visible points (the visible vertices of the obstacles and a goal if visible) from a state *s* are v_1, v_2, …, v_n. Since the shortest path to the goal must go through one of the visible points to avoid the obstacles, the new heuristic function $h_v(s)$ can be defined as in (1).

$$h_v(s) = \min[d(s,v_1)+d(v_1,G), d(s,v_2)+d(v_2,G), \ldots, d(s,v_n)+d(v_n,G)] \quad (1)$$

The estimated distance to the goal from a state s is the distance of the shortest path among the paths from s to G through visible points. This new $h_v(s)$ is larger than the straight-line distance from s to the goal $d(s,G)$, but it is smaller than the actual optimal distance $h^*(s)$ found on the NavMesh, which may consists of more line segments. This leads to the relation $d(s,G) \le h_v(s) \le h^*(s)$. This implies that the proposed heuristic is more informed than the Euclidean distance heuristic and that the search algorithm using $h_v(s)$ visits less number of states.

It is unnecessary to check the visibility for every state. As the search continues, the visibility information is passed to the selected child state, unless the search reaches a critical state that is adjacent to one of the visible points. We assume that s_1 is selected after s, where s_1 is not adjacent to any visible point from s. Because every cell is convex, there must be at least one child on the way to each visible point from s. The selected child s_1 must be closer to a visible point than the parent s, but not on the straight line to it, which leads to the equation $h_v(s) \le h_v(s_1) + d(s,s_1)$. Therefore, it is clear that the heuristic is monotone and admissible. When the search reaches one of the visible points, the visibility is retested. The new list of visible points is passed to its children and used to compute a heuristic value. The A* search algorithm with the visibility test is presented in Fig. 5.

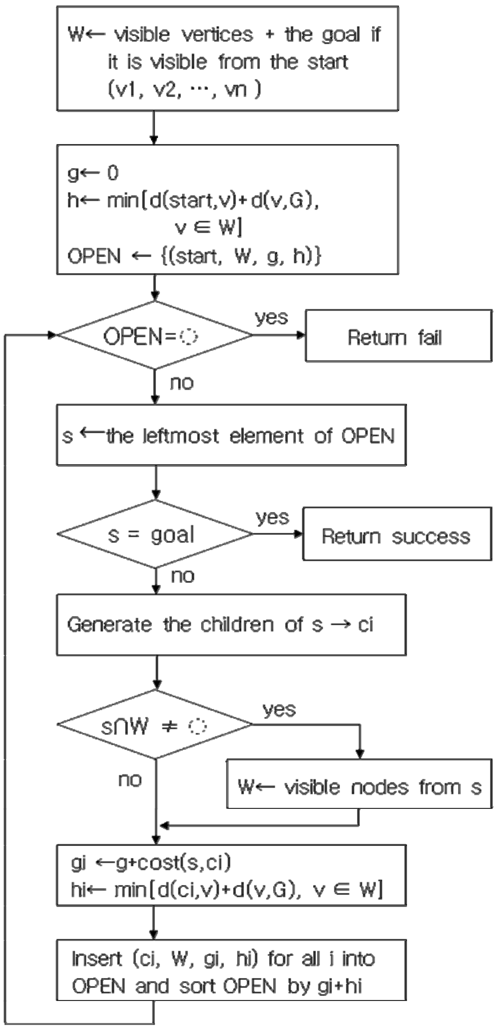
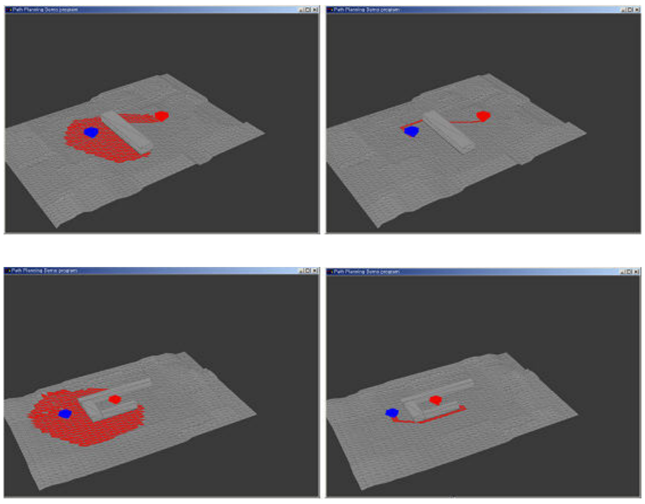

W← visible vertices + the goal if it is visible from the start (v1, v2, ···, vn)
g← 0
h← min[d(start,v)+d(v,G), v ∈ W]
OPEN ← {(start, W, g, h)}
OPEN=∅
yes
Return fail
no
s ←the leftmost element of OPEN
s = goal
yes
Return success
no
Generate the children of s → ci
s∩W ≠ ∅
yes
no
W← visible nodes from s
gi ←g+cost(s,ci)
hi← min[d(ci,v)+d(v,G), v ∈ W]
Insert (ci, W, gi, hi) for all i into OPEN and sort OPEN by gi+hi

Fig. 5. The A* search algorithm with the visibility test

4. Simulation Results

We implemented the A* algorithm with the visibility test using C++ and the OpenGL to support the theoretical results in the previous section. In the demos, 3D terrain and obstacles are modeled using the 3D Max. The mesh representing the terrain surface is converted into the OpenGL coordinate and used as a NavMesh. Fig. 6 shows examples of pathfinding on a 3D terrain. Each example has an obstacle, a starting point, and a goal point. The dot on the left side is the start point. The shaded area indicates the states (triangles) visited during the search from the start point to the goal point. Fig. 6(a) is the result of the A* search with the straight-line distance heuristic, and Fig. 6(b) is the result of the A* search with the proposed heuristic using the visibility information. The figures show that the A* search with the proposed heuristic visits a less number of states compared to the original A* search.



(a) A* with the straight-line distance heuristic (b) A* with the heuristic using visibility tests

Fig. 6. Comparison of the search areas in 3D terrain; the nodes expanded by the search are shaded

We also performed simulations to compare the results quantitatively. We run the A* algorithm with the Euclidean distance heuristic (plain A*) and A* with the visibility test heuristic (Vtest A*) with two obstacles. We then compared both the number of nodes visited during the search and the execution time for various arrangements of obstacles. Fig. 7 shows the case of two fully overlapped obstacles.

Tables 1 and 2 show the results of running plain A* and Vtest A* as we increase the overlapping of two obstacles. The results show that, as the obstacles are overlapped more, the number of nodes visited and the execution time of plain A* rapidly increase due to the increased search space; however, those of Vtest A* does not increase much when they are compared to plain A*. When the obstacles are fully overlapped, the search space of Vtest A* is only 6.1% of the search space of plain A*, and the execution time of Vtest A* is 10.6% of the execution time of plain A*. The execution time of Vtest A* includes the visibility computation time in addition to the search time. Therefore, the execution time of Vtest A* is slightly higher than that of plain A* when there is no overlap between two obstacles (the start and the goal are visible to each other). This is due to the overhead of visibility tests. This overhead is minor compared to the search time, thus it minimally affects the total execution time.

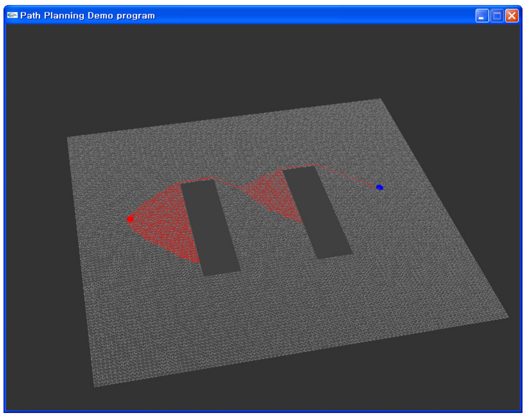


Fig. 7. An example of fully overlapped obstacles; the nodes expanded by A* are shaded

Table 1. The number of expanded nodes as the arrangement of two obstacles changes

Obstacles overlap	A*	Vtest A*	Vtest A* / A*
No	141	141	**100.0%**
1/3	562	163	**29.0%**
2/3	1,120	173	**15.4%**
Full	2,927	179	**6.1%**

Table 2. The execution time (sec) for search as the arrangement of two obstacles changes

Obstacles overlap	A*	Vtest A*			Vtest A* / A*
	Total	Total	Vtest	Search	
No	0.1888	0.1964	0.0088	0.1876	**104.0%**
1/3	0.7446	0.4276	0.2016	0.2260	**57.4%**
2/3	1.8409	0.4966	0.2050	0.2916	**27.0%**
Full	4.8401	0.5132	0.2081	0.3051	**10.6%**

Tables 3 and 4 show the results of running plain A* and Vtest A* as we increase the number of obstacles. Since the search space and the execution time depend on the configuration of the obstacles, they do not increase linearly. When there are five obstacles, the search space of Vtest A* is 8.2% of the search space of plain A*, and the execution time of Vtest A* is 15.2% of the execution time of plain A*. These results show that the A* search with visibility information visits a substantially less number of states compared to plain A* search, and the performance of search is more improved as the complexity of environment increases. This is because the proposed method recognizes the obstacles during the search and eliminates unnecessary visits to the polygons near the obstacles. Visibility test time is negligible compared to the time for searching a large number of polygons, because the visibility test is performed only when the search visits the state adjacent to a vertex of an obstacle. In general, when a highly detailed 3D terrain represented with navigation mesh contains many obstacles, Vtest A* will be very efficient. Fig. 8 shows how much the search space and the execution time are reduced by using Vtest A* as the number of obstacles increase.

Table 3. The number of expanded nodes as the number of obstacles increases

Number of obstacles	A*	Vtest A*	Vtest A* / A*
1	1,216	216	**17.8%**
2	1,529	328	**21.5%**
3	3,254	349	**10.7%**
4	2,637	276	**10.5%**
5	4,349	356	**8.2%**

Table 4. The execution time (sec) for search as the number of obstacles increases

Num. of obstacles	A*	Vtest A*			Vtest A* / A*
	Total	Total	Vtest	Search	
1	1.9317	0.4286	0.1369	0.2917	**22.2%**
2	2.1283	0.5289	0.1905	0.3384	**24.9%**
3	4.0913	0.7855	0.3282	0.4573	**19.2%**
4	3.8652	0.6356	0.2437	0.3919	**16.4%**
5	5.9156	0.9021	0.3895	0.5126	**15.2%**

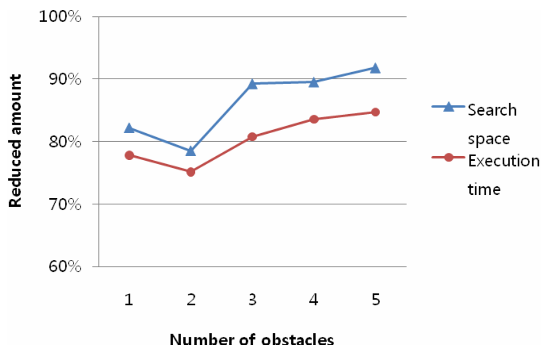


Fig. 8. The search space and execution time reduction obtained by Vtest A* as the number of obstacles changes

5. Conclusion

This paper proposed an efficient pathfinding method on a 3D terrain represented with navigation meshes. When path planning is performed on a highly detailed terrain, the search can be inefficient due to the large search space. By finding visible vertices of obstacles and defining heuristics as a distance of the path that goes through such vertices, we could reduce the search space by avoiding unnecessary searching. The experiments show that the proposed method outperformed the plain A^* search in terms of both the number of nodes visited and the execution time for searching.

The proposed method finds the shortest path in a 2D projected terrain, but does not guarantee the optimal path in 3D. Finding an efficient method to get a near-optimal path by combining terrain-specific factors such as height with the visibility information would be the future research direction of this work.

References

[1] H. Alt, M. Godau and S. Whitesides, "Universal 3-dimensional visibility representations for graphs," *Computational Geometry: Theory and Applications* (9), 1998.

[2] M. de Berg, M. van Kreveld, M. Overmas, and O. Schwarzkorf, *Computational Geometry-Algorithms and Applications*, Springer-Verlag, New York, 2000.

[3] G. Dehghani and H. Morady, "An Algorithm for Visibility Graph Recognition on Planar Graphs," *Proceedings of the ICFCC2009*, 2009.

[4] F. Farnstorm, "Improving on Near-Optimality: More Techniques for Building Navigation Meshes," In: Rabin, S. (eds.): *AI Game Programming Wisdom* 3, Charles Rive Media, 2006.

[5] B. Gao, D. Xu, F. Zhang, and Y. Yao, "Constructing Visibility Graph and Planning Optimal Path for Inspection of 2D Workspace," *Proceedings of the ICIS2009*, 2009.

[6] D. H. Hale, G. M. Youngblood, and P. N. Dixit, "Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds," *Proceedings of the fourth Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2008.

[7] D. Higgins, "Generic A^* Pathfinding," *AI Game Programming Wisdom*, Charles River Media, 2002.

[8] G. Johnson, "Smoothing a Navigation Mesh path," In: Rabin, S. (eds.): *AI Game Programming Wisdom* 3, Charles Rive Media, 2006.

[9] J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[10] C. McAnlis and J. Stewart, "Intrinsic Detail in Navigation Mesh Generation," In: Rabin, S. (eds.): *AI Game Programming Wisdom* 4, Charles Rive Media, 2008.

[11] M. NouriBygi and M. Ghodsi, "3D Visibility and Partial Visibility Complex," *Proceedings of the ICCCSA2007*, 2007.

[12] M. Pinter, "Towards more realistic pathfinding," *Game Developer Magazine*, April, 2001.

[13] S. Rabin, "A^* speed optimizations and A^* Aesthetic Optimizations," In: Deloura, M. (eds.): *Game Programming Gems*, Charles Rive Media, 2000.

[14] N. Sariff and N. Buniyamin, "An Overview of Autonomous Robot Path Planning Algorithms," *Proceedings of the 4th Student Conference on Research and Development*, 2006.

[15] G. Snook, "Simplified 3D Movement and Path-finding Using Navigation Meshes," In: Deloura, M. (eds.): *Game Programming Gems*, Charles Rive Media, 2000.

[16] W. Sterren, "Terrain reasoning for 3D action games," *Proceedings of the CGF-AI Conference*, 2001.

[17] B. Stout, "The Basics of A^* for Path Planning," In: Deloura, M. (eds.): *Game Programming Gems*, Charles River Media, 2002.

[18] P. Tozour, "Search Space Representations," In: Rabin, S. (eds.): *AI Game Programming Wisdom* 2, Charles Rive Media, 2004.

[19] P. Yap, "Grid-based Path-finding," *Lecture notes in Artificial Intelligence*, Vol. 2338, 2002.

[20] T. Young, "Expanded Geometry for Points-of-Visibility Pathfinding," In: Deloura, M. (eds.): *Game Programming Gems* 2, Charles Rive Media, 2001.

Hyunil Kim He received his Ph.D. degree from the Department of Computer Engineering at Dongguk University in 2004. He is currently a professor in the Department of Multimedia at Korea Nazarene University. His research interests include game AI, intelligent agent, machine learning, and recommender systems.

Kyeonah Yu She received her B.S. and M.S. degrees from the Department of Control and Instrumentation Engineering at Seoul National University in 1986 and 1988, respectively, and her Ph.D. degree from the Department of Computer Science at University of Southern California in 1995. She joined the Department of Computer Science at Duksung Women's University in 1996. Her current research interests include robot algorithms, game AI, and path planning.

Juntae Kim He received his B.S. degree from the Department of Control and Instrumentation Engineering at Seoul National University in 1986, and his M.S. and Ph.D. degrees from the Department of Electrical Engineering at University of Southern California in 1990 and 1993, respectively. He is currently a professor in the Department of Computer Engineering at Dongguk University, and a member of the Korean Institute of Information Scientists and Engineers, Korea Business Intelligence Data Mining Society, IEEE Computer Society, and Association of Computing Machinery. His current research interests include intelligent agent, data mining, and social network analysis.