Exploring 3D Shortest Distance using A* algorithm in Unity3D

*Kyoung Hwan Kim, Sumin Sin and Wonhyung Lee**

Chung-Ang University / Seoul, Korea
**Corresponding Author (whlee@cau.ac.kr)*

Abstract: In a 3D game environment, pathfinding or graph traversal is performed using the A* algorithm with 2D layers. However, in a 3D virtual environment or space, A* Algorithm that uses 2D layers is limited. Therefore, it is necessary to use an algorithm that uses 3D layers in a 3D game environment. This study shows that a 3D A* Algorithm may configure the path between two nodes faster than an A* Algorithm that uses 2D layers in a 3D game environment.

Keywords: A* Algorithm, Heuristic Function, Unity3D, 3D

Received May. 31, 2015; revised manuscript received Aug. 06, 2015; accepted for publication Aug. 19, 2015; published online Aug. 28, 2015. DOI: 10.15323/techart.2015.08.2.3.1/ ISSN: 2288-9248.

1. Introduction

The similarity between strategic simulation games, such as StarCraft or League of Legends, and first person shooter games, where the enemies are rushing into a maze, is to achieve a certain objective (e.g., attack) while simultaneously looking for an optimized path to move from one point to another point in the 3D virtual game environment. The A* Algorithm falls in the category of graph/tree exploring algorithms, and it was created by Peter E. Hart, Nils John Nilsson, and Bertram Raphael by using Heuristics [1].

Currently, A* Algorithm is used for determining the shortest path in 2D and 3D games [3], and this paper aims to demonstrate that using a 3D A* Algorithm in a 3D environment facilitates more efficient pathfinding as compared to an A* Algorithm with 2D layers.

2. Related Research

A. Heuristic Method

A Heuristic Method is used to obtain a satisfactory approximate value for a given problem. Considering the limited time and resources that are available in the computer-engineering field [1], using a computation intensive method to obtain a highly accurate value is resource intensive and can be avoided.

B. Heuristic Function

Heuristic Function, h(n), is a function which estimates the cost required, regardless of random wall, to traverse from a random node (n) to the target node [1,14,15].

$$h = g \times (\text{abs}(\text{currentX} - \text{targetX}) + \text{abs}(\text{currentY} - \text{targetY}))$$

Where,

currentX : x value of the starting point
targetX : x value of the target point
currentY : y value of the starting point
targetY : y value of the target point
abs(a-b) : distance value between a and b.
g : weight value

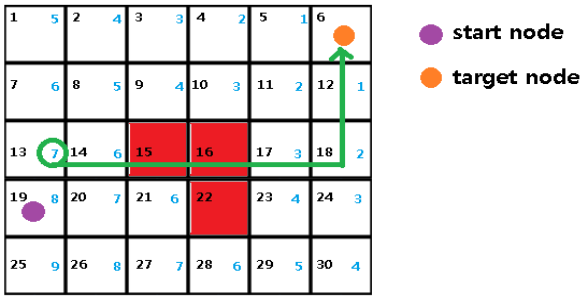


Fig. 1. Heuristic Function

C. Studies on Obtaining Shortest Path Associated with 3D

In [13], 5 plane corners were used instead of 3 plane corners in the 3D space to find the shortest path, and in [3], the shortest path was calculated using an A* Algorithm utilizing multi-layers in 3D space in the real game.

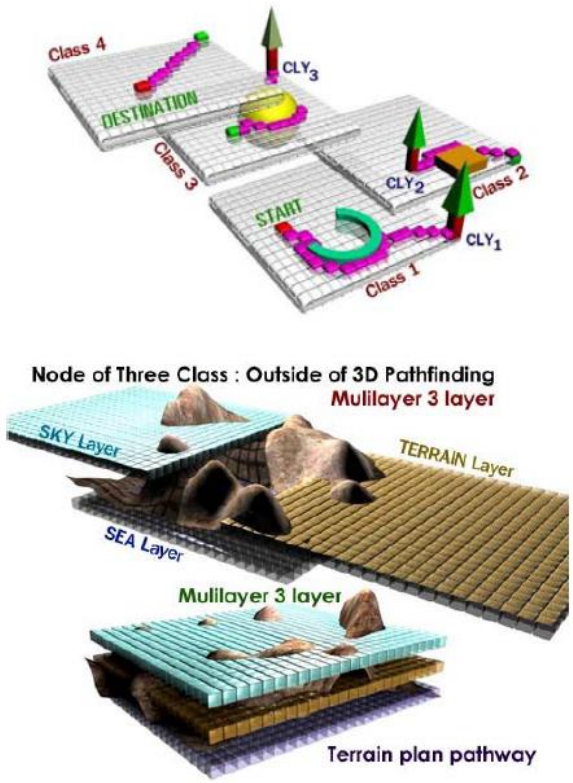


Fig. 2. Navigating the Shortest Distance in a 3D Environment

Fig. 2 shows the methodology to enable multi-layered bro wsing to obtain shortest distance in a 3D environment.

3. A* Algorithm

A. Weight in a 2D Node

Weight of the portion from A to B on the grid shown in Fig. 3 is calculated using the Pythagorean Theorem as follows:

$$c^2 = a^2 + b^2$$

Since a and b are the same in the square grid,

$$c^2 = a^2 + b^2 = 2a^2$$

$$c = \sqrt{2}a$$

Then, the value of C will be $c = 10 \times \sqrt{2} \approx 14.142$ which is a little bigger value than 14 if a value is 10.

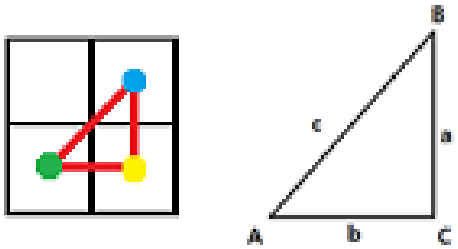


Fig. 3. Weight of 3D Node in a Grid

B. Weight in a 3D node

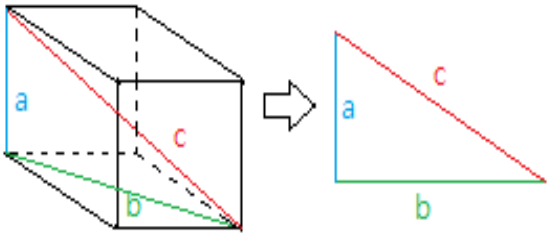


Fig. 4. Weight in a 3D Node

In 3D space, weight becomes a rectangle as shown in Fig. 4 and has a value of b ≈ 14. If a = 10, then the weight will have the following value:

$$c^2 = 10^2 + 14^2 = 296$$

$$c = \sqrt{296} \approx 17.2$$

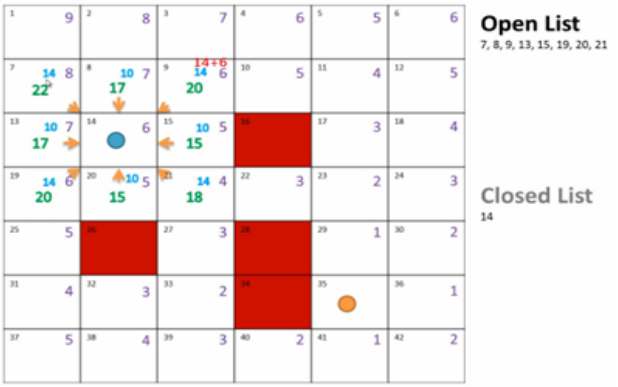


Fig. 5. A* Algorithm Pathfinding Tutorial [15]

C. Overview of the A algorithm*

The h(n) value is the Heuristic Estimate, which is the value taken from start point to target point. g(n) is the weight of the movement by a Guidance function, and the diagonal direction becomes 14 when a value of 10 is assigned to left/right/up/down direction.

f(n) is the summation of h(n) and g(n).

$$f(n) = g(n) + h(n)$$

g(n): Load from the start node to n node.
h(n): Estimated load from n-node to target node

The open list in Fig. 5 is the list of nodes that need to be checked, and the list includes the surroundings of nodes that need to be further checked, starting from the point that was currently checked. The closed list refers to the list of nodes that were already checked.

D. Implementation of the A algorithm in Unity3D*

A* Algorithm was applied and implemented in Unity3D [8, 14]. The methods to implement A* Algorithm in 2D use 4 or 8 directions; we implemented the A* algorithm using 8 directions in this paper.

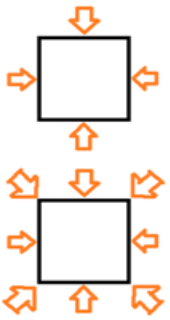


Fig. 6. Search Direction

Fig. 7. A* Algorithm Implemented in Unity3D

4. 3D A* Algorithm

A. Overview of 3D A algorithm*

A* algorithm is designed to check a total of 8 directions i ncluding left, right, up, down, left-up, left-down, right-up and right-down; it may also check up to 26 directions including front, rear of 2D A* Algorithm, front, rear, front-left, front-right, front-up, front-down, rear-left, rear-right, rear-up, rear-down, front-left-up, front-right-up, front-right-down, front-left-down, rear-left-up, rear-left-down, rear-right-up, and rear-right-down.

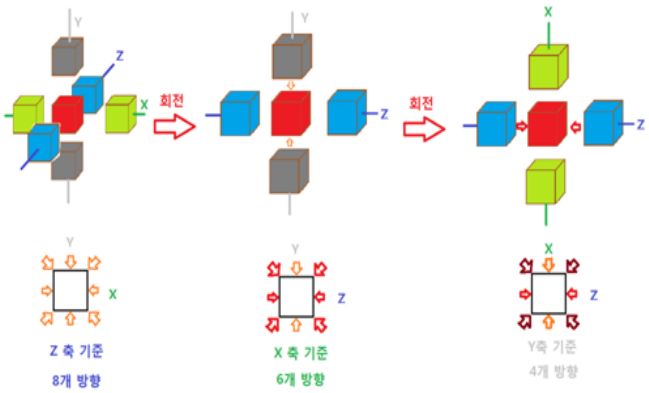


Fig. 8. Directions Required in 3D A*

B. Implementation of the 3D A algorithm in Unity3D*

The A* Algorithm was modified in order to check 26 directions, up from the existing 8 directions, by using the program used in 2D because it was used for comparison with 2D and implemented as shown in Fig. 9.

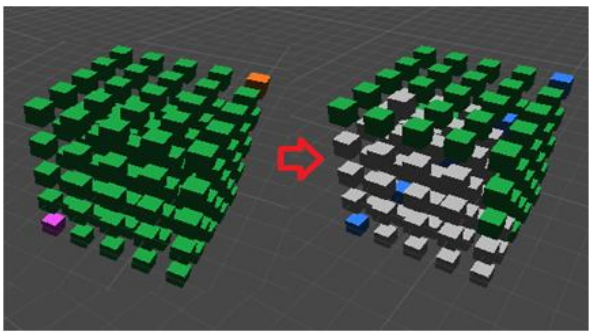


Fig. 9. 3D A* Algorithm Implementation in Unity3D

5. Experimental Methods

A. Test Environment

The experiment was conducted on a MacBook Air computer using an Intel Core i5 1.8 GHz CPU, 4 GB RAM, and Unity Pro v4.5. The 3D A* Algorithm was created by extending the 2D A* Algorithm

B. Method to Measure 2D and 3D simultaneously

The test was conducted in order to search for Multi-Layers in 2D as a Multi-Layer based shortest-distance exploration [3].

The value of 5x5x5 3D form is same as the form wherein 5 pieces of 2D (5x5) were aggregated. The start point and the target point (as shown in Fig. 10) were created in order to select the most distant point as the base, and the 2D form was again changed to 5x25 for the test. The difference between the search times for the individual 5x5 sections calculated consecutively and for the combined sections (as shown in Fig. 10) is negligible. Therefore, the test performed as shown in Fig. 10 is identical to the test performed by creating the layers.

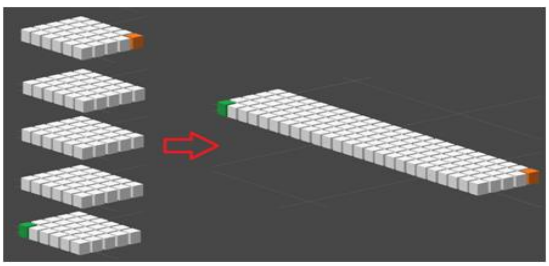


Fig. 10. 3D Developed in 2D

C. 2D and 3D Implemented in Unity3D

Results of 2D and 3D implemented in Unity3D based on the idea of 5.2 are as follows:

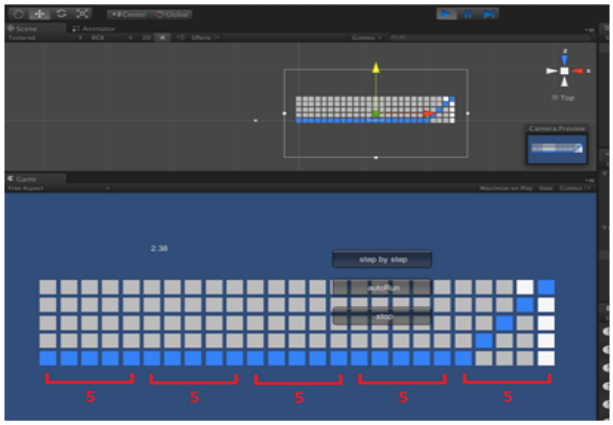


Fig. 11. 2D A* algorithm Implemented in Unity3D

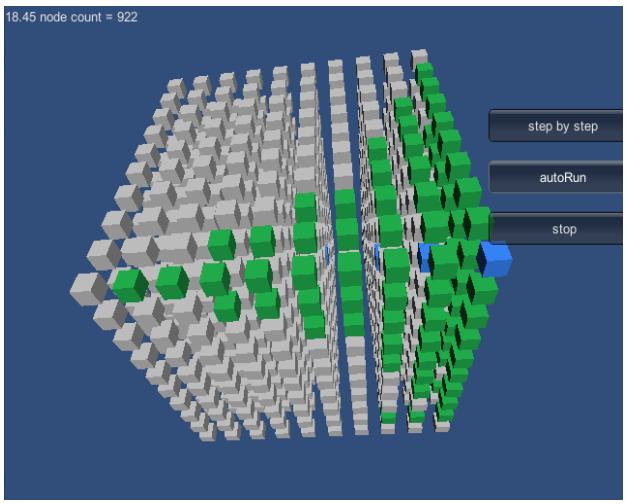


Fig. 12. 3D A* Algorithm Results Implemented in 3D

6. Results and Discussion

A. Measurement of Exploration in Environments with and without a Wall

The environment with wall showed a little faster exploration in the node where 5 pieces of 5x5 were arranged with and without wall. The result appeared in 10 pieces of 10x10, means the case with wall showed a little faster exploration. The below test results indirectly indicate a faster exploration when a random wall exists.

Table 1. 2D Test Results

	Exploration Time (s)	Number of nodes explored
Five 5x5 without random wall	2.3–2.38	118
Five 5x5 with random walls (7)	2.24–2.25	111
Ten 10x10 without random wall	19.88–19.89	993
Ten 10x10 with random walls (28)	19.53–19.54	975

Table 2. 3D Test Results

	Exploration Time (s)	Number of nodes explored
Five 5x5 without random wall	1.39–1.40	69
Five 5x5 with random walls (7)	1.35–1.37	111
Ten 10x10 without random wall	18.46–18.50	922
Ten 10x10 with random walls (28)	17.44–17.48	871

The results after measuring the relative efficiency of 3DA* based on 2D A* are as below. As the number of nodes increase, the efficiency improves in principle, although the difference in efficiency was minor.

Table 3. Comparison between 2D A* and 3D A* algorithms based on relative efficiencies.

	2D A*	3D A*
Five 5x5 without walls	1.00	1.395–1.411
Five 5x5 with random walls (7)	1.00	1.391–1.397
Ten 10x10 without random wall	1.00	1.069–1.076
Ten 10x10 with random walls (28)	1.00	1.105–1.107

7. Conclusion

Applying the 3D A* Algorithm to a 3D environment may result in obtaining the optimized path a little faster than applying the 2D A* Algorithm with multi layers to a 3D environment, provided that applying the 2D A* Algorithm could be more useful depending on the environment of creating the game. For example, the current A* Algorithm is more efficient for 3D games based on the ground, but the 3D A* Algorithm is more efficient for a 360 degree 3D space such as the sky or outer space. Thus, there is the need for additional research on the optimized game environment wherein the 3D A* Algorithm is more efficient.

Acknowledgements

This research was supported by the Chung-Ang University's Cross Functional Team (CFT) Program under the Brain Korea 21 PLUS Project in 2015.

This research was supported by the Basic Science Research Program through the national Research

Foundation of Korea (NRF) funded by the Ministry of Education (2014R1A1A2058248).

References

[1] P. E G. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, SSC4, 1968.

[2] S. Cho, " Pathfinding Algorithm Using Path Information," *Journal of Korean Society for Computer Game*, vol. 25, no. 4, pp. 27-28, 2013.

[3] K. Khantanapoka and K. Chinnasarn, "Pathfinding of 2D & 3D Game Real-Time Strategy with Depth Direction A* Algorithm for Multi-Layer," *IEEE*, 2009.

[4] K. Min, "An A*-based Technique for Extracting Feature Curves form Game Characters," *Journal of Korean Society for Computer Game*, vol. 3, no. 21, 2010.

[5] S. I. Kang and H. K. Hong, "Real-time Location Information Matching for AR Service in Outdoor Game," *Journal of Korean Society for Computer Game*, vol. 27, no. 1, 2014.

[6] L. Niu and G. Zhuo, "An Improved Real 3D A* Algorithm for Difficult Path Finding Situation," *XXI Congress of the International Society for Photogrammetry and Remote Sensing*, 2008.

[7] T. Sintaamrongruk, K. Mahakitpaisarn, and W. Manopiniwes, "A Performance Comparison between A* Pathfinding and Waypoint Navigator Algorithm on Android and iOS Operating System," *IACSIT International Journal of Engineering and Technology*, vol. 5, no. 4, August 2013.

[8] J. Hu, W. G. Wan, and X. Yu, "A Pathfinding Algorithm in Real-time Strategy Game based on Unity3D," *IEEE Audio, Language and Image Processing (ICALIP) International Conference*, 2012.

[9] Y. Ryu, and Y. Park, "A Study on A* Algorithm Applying Reversed Direction Method for High Accuracy of the Shortest Path Searching," *The Journal of The Korea Institute of Intelligent Transport Systems*, 2013.

[10] M. Kwon, Y. Kang, C. H. Kim, and G. Park, "A Path & Velocity Profile Planning Based on A* Algorithm for Dynamic Environment," *Journal of Institute of Control, Robotics and Systems*, 2011.

[11] Y. Kwon, "A Path Generation Method for a Autonomous Mobile Robot based on a Virtual Elastic Force," *The Journal of the Korean institute of electronic communication science*, 2013.

[12] J. Kang, H. Yoon, "A Heuristic Algorithm to Find the Critical Path Minimizing the Maximal Regret," *Journal of the Society of Korea Industrial and Systems Engineering*, vol. 34, no. 3, 2011.

[13] I. Prak, D. Baek, "Methods for Approximating Shortest Paths on Convex Polytope in R," *Journal of the Institute of Electronics Engineers of Korea*, vol. 40, no. 2, 2003.

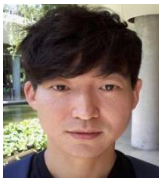
[14] B. Seet, G. Liu, B. Lee, C. Foh, K. Wong, and K. Lee, "A-STAR: A Mobile Ad Hoc Routing Strategy for Metropolis Vehicular Communications," *Lecture Notes in Computer Science*, vol. 3042, 2004.

[15] P. Singh, "A Star pathfinding algorithm," https://www.youtube.com/watch?v=Gw18AtiupCs, 2013.

[16] Stephenpanagiotis, "A* Pathfinding Tutorial," https://www.youtube.com/watch?v=KNXfSOx4eEE, 2012.

[17] "Monte-CarloMethod," from http://en.wikipedia.org/wiki/A*_search_algorithm, 2013.

Biographies



Kyounghwan Kim is an M.S. candidate at the Graduate School of Advanced Imaging Science, Multimedia, and Film, Chung-Ang University, Seoul Korea. His research interests include computer-game technology, human-computer interaction, and game AI.

Sumin Sin is an M.S. candidate at the Graduate School of Advanced Imaging Science, Multimedia, and Film, Chung-Ang University, Seoul Korea. His research interests include computer-game technology, human-computer interaction, and game AI.

Wonhyung Lee is the president of the Korean Society for Computer Game and a professor at the Graduate School of Advanced Imaging Science, Multimedia, and Film, Chung-Ang University, Seoul, Korea. He leads the Culture technology/Game Lab. His research interests are computer games, including computer game technology, information protection and game AI and Game Art.