An Overview of Pathfinding in Navigation Mesh

Xiao Cui and Hao Shi,

Victoria University, School of Engineering and Science, VIC, AUSTRALIA

Summary
Pathfinding is a fundamental problem that most commercial games must deal with. Due to the increase in game complexity, early solutions to the problem of pathfinding were soon overwhelmed. A* alone, a classic search algorithm, is no longer sufficient to provide the best solution. The popularization of using navigation mesh in pathfinding makes A* search a very small proportion of pathfinding implementation. In this paper, it systematically reviews the entire process of using a navigation mesh to find an optimal path. First a general pathfinding solution is described. Then examples of using A* in a navigation mesh are given. Additionally, implementation details of using funnel algorithm in both triangulation and polygonization are given, which is a major contribution of this paper.

Key words:
Pathfinding, A, navigation mesh, triangulation, funnel algorithm*

1. Introduction

For most commercial games, especially real-time strategy games, gameplay experience heavily relies on a brilliant artificial intelligent system. Pathfinding, as a fundamental part of artificial intelligence, is critical to the success of a game. In this multi-billion-dollar industry, game designers have put enormous amount of efforts to improve the pathfinding performance. As a computational intensive task, pathfinding often requires huge amount of computational resources. However, in practice, the major part of these resources is allocated to graphics. Because only limited CPU time and memory space are available, it is very important to improve efficiency of a pathfinding solution.

Navigation mesh is a technique to represent a game world using polygons. Due to its simplicity and high efficiency in representing the 3D environment, navigation mesh has become a mainstream choice for 3D games. According to the number of sides of polygons, navigation mesh can be categorized into triangulation and polygonization. Implementation details of using funnel algorithm in both of them are given. A general pathfinding solution is described in Section 3 and intuitive examples of pathfinding in both triangulation and polygonization are given in Section 4.

2. Pathfinding

Generally speaking, pathfinding is a process of determining a set of movements for an object from one position to another, without colliding with any obstacles in its path. Obviously, selecting a reasonable path for each moving object is often considered the most fundamental artificial intelligence task in a commercial game.

A ‘reasonable’ path must have two properties. The first property is called validity which is the most common measure to indicate whether or not the path is collision free. The second property is called optimality which is measured normally by either a distance metric or the time required for travelling through the path. Using a distance metric, an optimal path is simply the shortest path. It means the distance between start and goal in such a path is no greater than any other routes. It is an intuitive requirement. For example, if someone travels from London to Pairs, a route passing through New York would not be considered optimal.

Time is another widely used measure. It defines an optimal path as the fastest route. Simply it means the time required for an optimal path to be travelled through is always less than any other routes. In most cases, the shortest path is often the fastest one. . However, there are some special cases. For example, when travelling between two locations, the travel time for following a highway is obviously less than it for driving on a rough road even the distance may be shorter. In the context of commercial games, measuring optimality by time is a better choice, especially in real-time strategy games such as StarCraft and Age of Empires, where the time to reach a destination is more important than the distance travelled.

Regarding efficiency of a pathfinding solution, generally it is measured by execution time and memory usage. According to existing research, finding a nearly optimal path only requires a small part of the resources that are needed to find an exactly optimal path. As commercial games often impose strict requirements on both execution time and memory usage, finding an exactly optimal path is not always worthwhile. As long as the path appears ‘reasonable’, a suboptimal approach is usually acceptable.

Manuscript received December 5, 2012
Manuscript revised December 20, 2012

3. Approach to pathfinding

Generally speaking, finding an optimal path for a game character requires at least 3 stages. At the 1st stage, a game world is transformed into a geometric representation such as a navigation mesh. There are many ways to approach but they are not discussed in this paper. Once a navigation mesh is generated, no matter it is a triangulation or a polygonization, at stage 2 A* search is performed in such a mesh. Because A* search could not give a real path (see Section 4.1 for detail), further processes are required to find a real path, which is the 3rd stage. A popular algorithm, simple stupid funnel algorithm, is used to overcome the issue. Demonstrations of the simple stupid funnel algorithm in both triangulation and polygonization are given in Section 4.2 and Section 4.3 respectively.

4. Navigation Mesh

Navigation Mesh (NavMesh) is a method for representing a game world using polygons. Polygons on a NavMesh must be convex. The properties of a convex polygon could guarantee a free-walk for a game character as long as such a character stays in the same polygon [1]. Triangulation is a special case of NavMesh as all of its polygons on the NavMesh are triangles. In most cases, the number of sides of polygons on a NavMesh varies from 3 to 6; as in practice, over 6 could result in a significant increase in memory usage [2].

4.1 A* in a NavMesh

Below is a brief example of how A* works in a NavMesh. Let G be a graph with P polygons which are walkable and B polygons which are blocked as shown in Fig. 1.

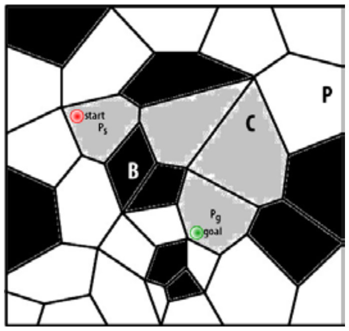

Figure.1 Construct a NavMesh

First, we must find a set of polygons $C \subseteq P$ where an optimal path will go through. If P_s is the polygon where the start is, then P_s must be the first polygon in C. If P_g is the polygon where the goal is, then P_g must be the last polygon in C. To find the remaining parts of C, we must make some changes in the map. Each polygon in G is mapped to a node in G'. For instance, P_s is mapped to N_s and P_g is mapped to N_g. Each edge shared by two polygons in G is mapped to an edge connecting two nodes in G' as shown in Fig. 2.

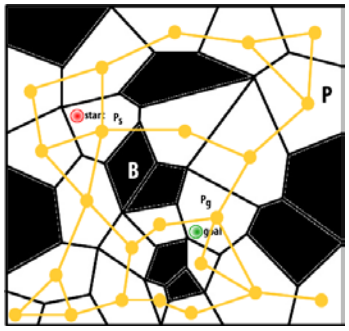

Figure.2 Transform polygons to nodes

Then, an optimal path p from N_s to N_g in G' can be found with A*. As each node in G' corresponds to a polygon in G, each node in p corresponds to a polygon in C as shown in Fig. 3.

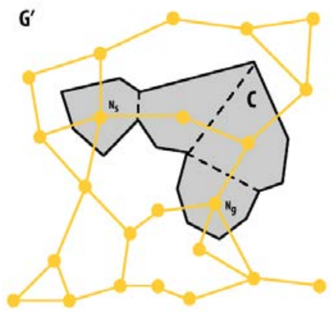

Figure.3 Run standard A*

C is not a real path; instead, it is a set of polygons. Fig. 4 illustrates three different ways to find a real path in C. No matter which method we use, none of them could guarantee an optimal path. Thus, further process is required.

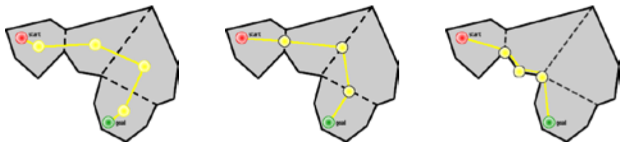

Figure.4 Three different ways to find a pathin C

4.2 Triangulation

Triangulation is a special case of NavMesh where polygons are replaced with triangles. The minimum angle of a triangle must be maximized. This property guarantees an optimal path will not cross any triangle more than once [3]. Using A* alone could not yield a real path in a triangulation; instead, a set of triangles with a start and a goal is given. A channel is formed, a simple polygon with start and goal as two nodes and which traces perimeter of triangles in between as shown in Fig. 5. We aim to find a real path within it to use for object motion.

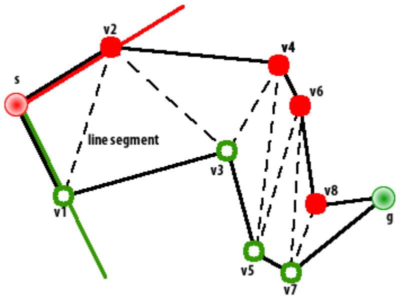

Figure.5 Construct a funnel

For a point object, such path could be found with funnel algorithm in linear time [3]. Instead of describing the original funnel algorithm, here we prefer to present a simplified version, simple stupid funnel algorithm (SSF) which was first time introduced by Mononen in 2010 [4].

For example, as shown in Fig. 5, a channel is formed from 8 triangles with 7 line segments. A funnel is constructed using the node *s* (*s* represents the start) and the endpoints ($v_1$and v_2) of the first line segment (from left to right). $\overrightarrow{sv_2}$ represents the left funnel edge and $\overrightarrow{sv_1}$ represents the right funnel edge. Assume v_2 is the left node and v_1 is the right node. Then, the next left and right nodes are v_2 and v_3 respectively. The rest could be done in the same manner.

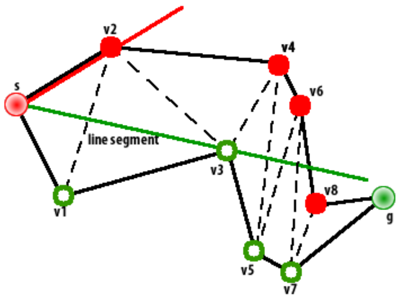

Figure.6 Update the funnel

All of the left nodes are represented by solid circles and all of the right nodes are represented by hollow circles. If the next left and right nodes are inside the funnel, for instance as shown in Fig. 5, v_2 and v_3 are both inside the funnel, simply narrow the funnel as shown in Fig. 6. If the next left node is outside the funnel, do not update the funnel.

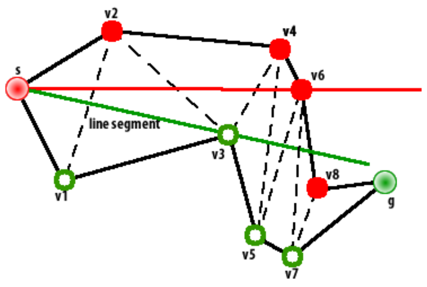

Figure.7 Finding the turning point on the path

If the next left node is over the right funnel edge, for instance as shown in Fig. 7, v8 is over the right edge, set v3 as an apex in the path and construct a new funnel using v3, v4 and v5 and restart the above steps as shown in Fig. 8.

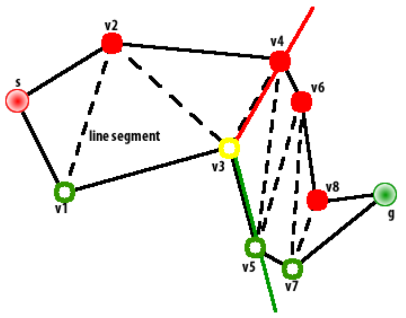

Figure.8 Restart the search

The same logic goes for right nodes as well. These steps are repeated until all the line segments are processed. Then, a shortest path from s to g is found. Instead of using a double-ended queue in the original funnel algorithm, SSF runs a loop and restarts the loop from earlier location when new apex is added. Some line segments may be calculated more often than others because of the restart, but the calculations are too simple to impair the execution time. This simplified version is much easier to implement and in practice is even faster as well [4]. Compared with the original funnel algorithm where a triangulation is required [3], SSF provides the possibility to be deployed in a polygonization.

4.3Polygonization

If a NavMesh is not a triangulation, it must be a polygonization. Such a NavMesh must have a property of at least one of its polygons that the number of sides is greater than 3. All polygons on a NavMesh must be convex. This property guarantees a game character could move anywhere it likes to in a straight line as long as it stays in the same polygon. A* alone could not generate a real path as stated before; instead, a series of polygons are given. Similar with pathfinding in a triangulation, SSF can be used to find a real path in the channel. As described in Section 4.2, apexes can be found by analyzing the relative position between nodes and edges. Thus, being able to identify the left or right nodes correctly is critical to SSF.

In a triangulation, the left and right nodes can be easily found. For example, as shown in Fig. 5, v3 is a right node because v2 is a left node which is set at the beginning. As v3 is a right node, v4, another endpoint in the same line segment with v3, must be a left node. However, in a polygonization, it is slightly different. In a polygonization, as long as the vertices in the polygons are stored in a counter clockwise (CCW) order, the first vertex on the line segment is always the right node and the second vertex, another endpoint on the same line segment, is always the left node.

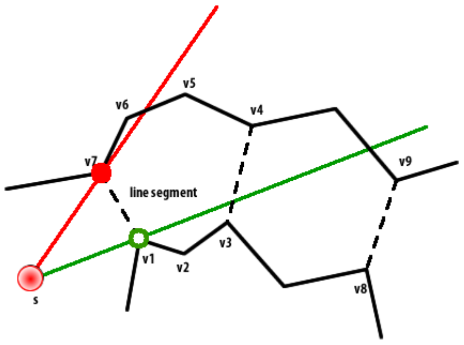

Figure.9 Identify the left and right nodes

The difference is as shown in Fig. 9, setting v2 is the left node and v1 is the right node. Then, the endpoints v3 and v4 on the second line segment are the right and left nodes respectively. Once the next left and right nodes are identified, the remaining steps are exactly the same as SSF in a triangulation, where the relative positions between the left and right nodes and the funnel edges are used to identify the apexes on the optimal path.

A polygonization provides a possibility to use fewer large polygons to represent a game world. Especially, in real-time strategy games, large open areas are everywhere. In this case, a large polygon could cover the whole open area while a triangle could not able to cope due to its geometric properties. Regarding efficiency, using fewer large polygons could reduce both memory footprint and search space, especially the reduction in the search space could significantly speed up the search.

5. Conclusion

This paper reviewed a general approach to finding an optimal path in a navigation mesh and illustrated how to achieve it in both triangulation and polygonization. The solution described in this paper is an A* search with funneling based on a navigation mesh. Although SSF is applicable to both of triangulation and polygonization, a further modification is required when identifying the left and right nodes in a polygonization. A major contribution of this paper is to give implementation details of both of them in an intuitive way. Further work is required to explore applicability of both of triangulation and polygonization to classic game maps as well as compare their performances.

References

[1] S. Rabin, “A* speed optimizations”, in Game Programming GEMS, pp.264-271, Charles River Media, America, 2000
[2] M. Mononen, “Recast and Detour, a navigation mesh construction toolset for games”, http://code.google.com/p/recastnavigation/, accessed September 20, 2012.
[3] D. Demyen and M. Buro, “Efficient triangulation-based pathfinding”, in the 21st National Conference on Artificial Intelligence, pp.942-947, Boston, 2006.
[4] M. Mononen, “Simple stupid funnel algorithm”, http://digestingduck.blogspot.com.au/2010/03/simple-stupid-funnel-algorithm.html, accessed August 4, 2012.

Xiao Cui is a Ph.D student in School of Engineering and Science at Victoria University, Australia. He completed his master degree in the area of Software Engineering at Australian National University and obtained his Bachelor of Computer Science degree at Victoria University. His research interests include game artificial intelligence and social network computing.

Hao Shi is an Associate Professor in School of Engineering and Science at Victoria University, Australia. She completed her PhD in the area of Computer Engineering at University of Wollongong and obtained her Bachelor of Engineering degree at Shanghai Jiao Tong University, China. She has been actively engaged in R&D and external consultancy activities. Her research interests include p2p Network, Location-Based Services, Web Services, Computer/Robotics Vision, Visual Communications, Internet and Multimedia Technologies.