Path Planning in 3-Dimensional Space Using the A* Algorithm

John Beauregard

CS 485

Professor Kosecka

May 13, 2012

ABSTRACT

The problem that interested me was path planning in 3-dimensional space. My approach was to expand the A* path planning algorithm to function on a 3-dimensional grid with obstacles. This resulted in a path finding algorithm that worked well at finding the most optimal path, but is slow to process and lacked scalability.

INTRODUCTION

The broad problem is simply finding a path panning algorithm to find the most optimal path between two points in a 3-dimensional space. My approach to the problem was to map an area as a 3-dimensional grid, place obstacles on the grid, and use the A* algorithm to find the most optimal path between two points on the grid. The results regarding being able to find the path were good. The paths that were found seemed to have the lowest cost without any unexpected errors. However, the resources required in finding the path were far less then optimal. Finding the path took much longer than it would have on a 2-dimensional grid, and used too much memory to be useful in a practical application.

APPROACH

In my approach, I implemented the A* algorithm in Java and OpenGL for the display to make it usable in common applications, using the process as described by Patrick Lester [1]. A 3-dimensional grid was used to represent the space and obstacles. This was done using a 3-dimensional array of integers with the value on each point being 1 for an obstacle and 0 for traversable space. An open list and closed list is created for all nodes used in the process of determining the optimal path, with each

node consisting of its location on the grid, the cost g (the cost to get to that node from the starting point), the estimated cost h (the cost from that node to the goal node), the total cost f(the total cost of g and h), the parent node (the previous node before traversing to this node), and a pointer to the next node on the list.

From the starting point, each neighboring space is checked. If the neighbor is an obstacle, it is ignored, but if the node is traversable its costs g, h, and f are calculated and stored in the node. The neighbor's parent node is set as the node whose neighbors are being checked and neighbor is than added to the open list. Once this is completed for all neighboring nodes, the original node is add to the closed list and not checked again. The next step is to iterate through each node in the open list, keeping track of which node has the lowest cost f. The node with the lowest cost f than has its neighboring nodes checked as before, with the addition of checking whether or not a neighboring node is already on the open list. If it is and the neighboring node has a lower cost g, then the parent node is replaced with that of the current node whose neighbors are being checked. This process continues until the goal node is found (the cost h is 0), and from that node each parent node leading to the start point while each node's position is pushed on to a list. The result is a list of points representing the optimal path from the start to the goal.

The results were consistent and not only demonstrated the A* algorithm's ability to find the optimal path, but identified an issue preventing it from being applicable in large scale or other more practical applications. The algorithm takes a significantly larger amount of time when the grid is increased in size. When the path was generated on a grid with a size of 100x100x100, the time to complete just the path finding

operation generally took between 5.01 to 5.80 seconds. The maximum size of the grid that I was able to achieve was 192x192x192, anything higher than that would cause the operation to run out of memory. At the maximum size of 192x192x192, the time that it would take to complete the operation was generally between 2.878 to 3.043 seconds. Because of this, it would be difficult to implement this in a system in which a grid would need to represent every possible point in an area that is being simulated. While it may be possible to use it with the grid representing points each at a constant distance from each other in the area being simulated, the time it would take to process each path would make it impractical for processing multiple consecutive paths or multiple simultaneous paths in real time.

RESULTS

In order to find the shortest path between two points in a 3-dimentional space, the A* search algorithm was used. This was done by expanding the grid to 3-dimensions, and implementing the rest of the algorithm the way it would be done on a 2-dimensional grid. This resulted in optimal paths at a severe cost in performance efficiency. The biggest open question is how to improve the algorithm in a way to make it more practical to use in situations where memory and processing power are incredibly limited.

REFERENCES

[1]P. Lester, “A* Pathfinding for Beginners,” *Almanac of Policy Issues*, 18-Jul-2005. [Online]. Available: http://www.policyalmanac.org/games/aStarTutorial.htm. [Accessed: 13-May-2012].