Implementation of Path Finding Algorithms in a 3-Dimensional Environment

Maulindu Sarkar 18EE3AI23 Vaibhav Vishal 18EE10055 Yogesh Yadav 18EE3AI25

# Introduction

Pathfinding is one of the most basic problems of Artificial Intelligence. The A\* algorithm provides an effective solution to the problem of pathfinding. The A\* algorithm is probably one of the most, if not the most used pathfinding algorithm AI domain today. What makes the A\* algorithm so appealing is that it is guaranteed to find the best path between any starting point and any ending point, assuming, of course, that a path exists. Also, it's a relatively efficient algorithm, which adds to its appeal. The A\* algorithm is efficient, but it still can consume quite a few CPU cycles, especially if it is needed to do simultaneous pathfinding for a large number of objects.

To find the shortest path between two nodes, there should be a way to put a weight for each connection between any two nodes. In this case, and because of the geographical nature of the simulation, the distance between each node and each of its neighbors is used as a weight. So in the code, the mesh of the nodes is defined as a data structure, and in the initialization, the weight (distance) between each node and its neighbors is calculated.

Accompanying the three-dimensional trend in computer society, three-dimensional A\* algorithm’s development has caught more attentions. To solve three-dimensional path finding problem, some path finding solution maps three-dimensional problem area to two dimensional expression in order to use traditional A\* algorithm solving the path finding request (Makanae and Takaki, 2004). Although the method of mapping 3D to 2D is working for path finding requirement under some simple 3D situations, the mapping method could not easily be used to finish path finding under complex situations. For example in the restricted spatial situations such as underground and inner   
building, the overlapping layers may appear frequently, and these situations seems impossible to take traditional A\* algorithm solution, for mapping 3D into 2D and deriving the optimum path are nearly impossible under these special circumstances Thus A\* algorithm should be improved to meet these routing requirements.   
  
The three-dimensional A\* algorithm is required to work the routing problem out under restricted situations. Several certain modifications should be taken for standard A\* algorithm and a new improved 3D A\* algorithm is introduced.

In the case of the 3D simulation, it is different in some aspects, the 3D environment must be planted with nodes and connections between those nodes must be made in a way that makes it possible for an object to move from one node to the other either directly or by passing through any number of nodes. It is better to use a few numbers of nodes but they must be sufficient relative to the environment size. By sufficient, it is meant that there must be a node in every corner or point of the 3D environment. All this is made to make the object able to reach any point in the environment.

# Problem Statement

Given a 3-Dimensional (3D) environment, propose a modified A\* algorithm for path finding and representation of the nodes in a 3D environment.

# AI Modelling

The path finding problem is modelled as an AI Search problem and the detailed mapping is as follows:

 State Definition: {Nodes, [X coordinate, Y coordinate, Z coordinate]}, A node in a 3D environment is represented by a bounding sphere.

 Start State: {Source Node S, [S.X, S.Y, S.Z]} The Source Node S is the starting Node.

 Goal State: {Goal Node G (not obstacle), [G.X, G.Y, G.Z]}. Any node from the Connectivity Graph that is not in the Closed List and does not have any children.

 Search Space: The entire Connectivity Graph.

As seen, the filled circles are the nodes (bounding spheres) with an appropriate radius. The radius is important because when the object moves in the environment, it may move in 'n' steps per frame. The more 'n' the more is the speed that the object appears to be moving. If the object is moving at a relatively high speed and passes over a node, the object's bounding box may not intersect with the node and in this case the path will not be calculated at that node and the object will continue moving in its direction which may lead it to an obstacle. For this reason, the bounding sphere that represents a node must have an appropriate radius, that is, it must be greater than the object's step. For example, if the object moves two units in a frame, the bounding sphere radius must be greater than two.   
In the simulation, each soldier has the capability to find the way to any point in the environment through the use of the nodes that are distributed in the environment. There are steps that the soldier in the simulation follows to go to a node.

The nearest node to the source and destination will be the origin node and the destination node respectively. As it is impossible to cover the whole environment with nodes, the concept of nearest node has to be implemented. In the practical 3D simulations like game (eg. Lost in Space), military environment, the desired point is always an entity (game object, military etc.) and most of the time they are prone to move.

When the bounding box of the object intersects with a bounding sphere, it is known now where the object is and the path finding routine can be activated again to find the path to another node.

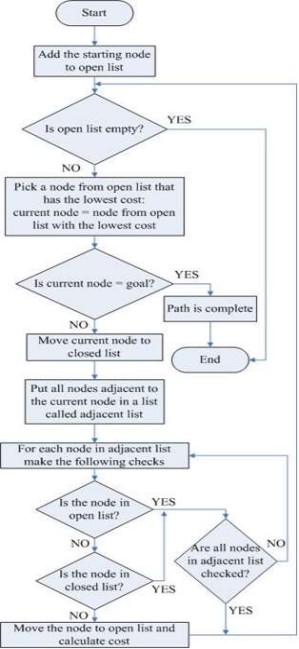
# Improvement Proposal

As seen, the soldier turns to the nearest node to itself and then runs toward it to be able to locate itself and this is to calculate the path to the destination node. In some situations, the soldier happened to be outdoor, that is, it is not inside a building, and the nearest node is inside the building. In the traditional case, the soldier will turn to the nearest node and run toward it. This sometimes will result in a non-realistic effect when there is an obstacle "the building's wall" between the soldier and the indoor node, the soldier will run through the wall, which is not realistic. For this reason, a solution was proposed, that is, the indoor nodes are marked with a flag by the designer when coding and in run-time each soldier is tracked to know that it is indoor or outdoor. If the soldier is outdoor, then when finding the nearest node to it, the indoor nodes are neglected and the nearest outdoor node is returned by the routine. This simply solves the problem.

# Algorithm

Below is the pseudocode of the proposed A\* algorithm:

1: **procedure** Main()  
2: open := closed := φ;  
3: g(sstart) := 0;  
4: parent(sstart) := sstart;  
5: open.Insert(sstart,g(sstart) + h(sstart));  
6: while open 6= φ do  
7: s := open.Pop();  
8: if s = sgoal then return ”path found”;  
9: closed := closed ∪{s};  
10: for each s′ ∈nghbr(s) do  
11: if s′ /∈closed then  
12: if s′ /∈open then  
13: g(s′) := ∞;  
14: parent(s′) := NULL;  
15: UpdateV ertex(s,s′);  
16: return ”no path found”;  
17: **procedure** UpdateVertex(s,s’)  
18: gold := g(s′);  
19: ComputeCost(s,s′);  
20: if g(s′) < gold then  
21: if s′ ∈open then  
22: open.Remove(s′);  
23: open.Insert(s′,g(s′) + h(s′));  
24:  
25: **procedure** ComputeCost(s,s’)  
26: / ∗ Path 1 ∗/  
27: if g(s) + c(s,s′) < g(s′) then  
28: parent(s′) := s;  
29: g(s′) := g(s) + c(s,s′);



# Code and Conclusions

**Code Link:** https://github.com/Yogesh7023/AIFA-assignment

After experimenting in the test area, following advantages of the improved A\* algorithm has been found:

1. As is known to all, A\* algorithm’s open list consumes huge size of cache. Although effort has been made to reduce the cache cost of A\* algorithm in the past, the result is not very significant. The region using idea in the improved algorithm partly overcome the cache consuming shortage of A\* algorithm. Because of introducing of “Region”, smaller amount of storage is provided for each region’s processing.

2. Paralleling computing of the algorithm calculation is introduced by using the improved A\* algorithm. For each region is an independent path finding area, it makes program own the ability to use multi-thread to calculate each segment of the optimum path.

3. The “Cell” structure makes the improved A\* algorithm works more flexibly than ever. Any change of the corresponding territory will be represented in the cell. Small modification in the experiment area will only affect the related cells in one region instead of reconstructing the total problem area.