New Ideas In Pathfinding

Peter Yap
GAMES Research Group
Department of Computing Science
University of Alberta
Edmonton, Canada
T6G 2E8
peteryap@peteryap.com

Abstract

Pathfinding is a computationally expensive component of many computer games. Typically, a (rectangular) tile grid is superimposed over the region, then some sort of graph search is used to find the optimal path. In this paper, we mathematically prove and empirically show that the hexagonal grid is superior to the conventional tile grid for IDA*. Pathfinding using a hexagonal grid will generate smoother and shorter paths. Furthermore, searching on a hexagonal grid instead of a tile grid will result in exponentially faster searches. Although the hexagonal grid is better than the tile grid in many ways, it is difficult to implement in practise. Consequently, we introduce the tex grid which retains the advantages of a hexagonal grid but is easier to implement.

1 Introduction

Pathfinding in computer games may be easy, but to do it well is hard. Consider Diablo II, a successful multi-player game. To be very brief, the player basically runs around and kills hordes of demonic minions... over and over again. To finish the game, the player usually exterminates a few thousand minions. The game involves quite a lot of pathfinding, since each of these minions either chases the player, or (less commonly) runs away from the player. In the meantime, the player is rapidly clicking on the screen in an effort to either chase the minion, or (more commonly) to run away from the minion and company. All this frantic running and chasing requires pathfinding computations. To complicate the matter, the player is allowed to hire NPCs (called hirelings) or play with other humans in an effort to kill more minions. This significantly adds to the complexity of the game in terms of pathfinding. Consider this complicated situation, where a party of human players with hirelings is attacked by a very large horde of minions. Being very sensible, the human players all independently run in different directions to escape the horde. In this state, pathfinding is done on each fleeing player by interpreting the player's mouse clicks, pathfinding must be done on each minion so that they give chase to the human players, and pathfinding must be done on each hireling so that they flee with their respective employers. On a slow computer or in a network game, this computationally intensive scenario reduces the game to a slide show, often leading to a player's untimely death, since the minions still attack even while the game appears "frozen" to the human players. One of the solutions applied by Blizzard was to magically transport the hireling to beside the player instead of calculating a path to move the hireling to the player. Another solution was to program the minions such that they lose interest in chasing the player if the player is too far away.

Consider The Sims, where the player controls a family in a household environment. Often, the house is cluttered with obstacles like furniture, making pathfinding slightly tricky. Typical pathfinding problems involve deadlocks when two Sims are trying to occupy the same space at the same time. A common situation is when a Sim has just finished using the bathroom and is trying to leave through the bathroom door. Simultaneously, another Sim desperately needs to go and rushes towards the bathroom. The two Sims collide at the bathroom door and a deadlock ensues. Often the player must personally resolve the issue. This situation could be avoided if the pathfinding is better and if the Sims could learn simple courtesy.

Many computer games like The Sims, the Civilization series, and the Baldur's Gate series conduct pathfinding on a (rectangular) tile grid. Each tile has a positive weight that is associated with the cost to travel into that tile. Typically the pathfinding algorithm is some type of A* search on the tiles. Some games like BioWare's upcoming Neverwinter Nights use IDA* (Iterative Deepening A*), which avoids the open list used in A* and its associated memory and maintenance costs. While IDA* avoids the overhead of maintaining and storing the open list, it is typically slower than A*. For the rest of this paper, we assume all pathfinding searches are done with IDA*.

Copyright 2002, American Association for Artificial Intelligence
(www.aaai.org). All rights are reserved.

There are two ways to conduct pathfinding searches on tiles. Traditionally, one is allowed to move in the four compass directions on a tile; however, it is possible to also include the four diagonal directions (so eight directions in total). We call the latter the octile grid and the former the tile grid. We shall prove that the octile grid is inferior to the tile grid for IDA* searches. Other than the tile and octile grids, the hexagonal (or hex) grid is also known, although it is only seen in war strategy games. This is very unfortunate since choosing the hexagonal grid instead of the tile or octile grid results in exponentially faster pathfinding searches.

2 Tiles, Octiles, and Hexes. Oh my!

The speed of the IDA* search depends on the number of nodes it needs to examine. Analysis has shown that the size of the nodes to be searched is $O(b^D)$ [Korf *et al.*, 2001], where b is the average branching factor and D is the depth of the search. Intuitively, this is because IDA* checks every path of length D, and at each depth, each node branches into b more nodes. A search on a tile grid has four degrees of movement. Hence for a pathfinding search, we need to check which of the four adjacent tiles we wish to explore next. Since we never backtrack in an optimal path, we need not check the direction we came from. All in all, we need to check at most three adjacent tiles at every tile except for the start tile. We say the branching factor of a tile grid is three, $b = 3$, since we need to check three adjacent tiles per tile visited. If the length of the path from the start tile to the goal tile is D, then we say the depth of the search is D. In conclusion, the size of the nodes to be searched in a tile grid is $O(3^D)$.

Now consider a hex grid with six degrees of movement. Using a similar argument, one might deduce that the branching factor of a hex grid is five; however, we can do better and reduce the branching factor to three. Let N, E, S, W be four hexagonal tiles (corresponding to the compass directions) such that every tile is adjacent to the other three. Clearly an optimal path from S to E would not be SNE using positively weighted edges. In terms of the graph search, if one was to enter hex N from S, then the next search would review the three hexes adjacent to N but not S. Clearly, S can be omitted as an optimal path would never backtrack. It follows easily that we need not consider E or W because SNE or SNW are more costly than SE or SW (and SE and SW were already considered as path candidates when we searched at S). In summary, at each non-root hex, we need only examine three hexes and hence a branching factor of three.

We have established that the branching factor of both the tile grid and the hex grid is three. For comparison purposes, we equate the area of the hex with that of a tile. Given the same distance, on average, a path

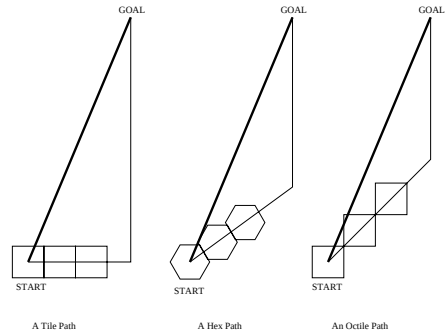

Figure 1: Optimal Paths on Different Grids

represented by the hex grid is shorter than the path represented by the tile grid. It follows that because the hex path is shorter, one doesn't need to search as deep. In fact, one can mathematically show that given the same distance, a tile grid will search with depth D while a hex grid will search with depth $0.81D$ [Yap, 2002]. If we combine our knowledge of the branching factors and the depths of both grids, it follows that if the tile grid searches through $O(3^D)$ tiles in a search, then the hex grid searches through $O(3^{0.81D}) \approx O(2.42^D)$. This result is very significant because it proves that a hex grid is exponentially faster than a tile grid for an IDA* search.

We can now consider the octile grid, which has eight degrees of movement. Using similar arguments, we can conclude that the branching factor of an octile grid is five. One can also mathematically show that if the tile grid is searched for D depth, then the octile grid is searched for $D/\sqrt{2}$ depth. Hence an octile grid searches $O(5^{\frac{1}{\sqrt{2}}D}) \approx O(3.14^D)$. Intuitively, the depth for the octile should be less than that of a tile because one diagonal octile move is equal to two tile moves (see Figure 1). Although an octile grid need not search as deep as a tile grid, it is clear that the octile grid is the worst of the bunch due to its branching factor of 5. While one may claim that the octile grid produces smoother and shorter paths, this can be accomplished more efficiently by smoothing the tile path. A tile search with path smoothing produces the same result as an octile search but is exponentially faster. In terms of IDA* search speed, tiles are better than octiles, and hexes are better than tiles. Do hex grids have any disadvantages? Well, a hex grid is slightly more difficult than the rectangular grids to implement due to its hexagonal shape.

3 Welcoming the Tex Grid

In addition to the exponential search advantage that the hex grid enjoys over the tile and octile grids, hexes have very nice geometric properties. For example, each hex is perpendicular and equidistant to each adjacent

hex; furthermore, a hex shares exactly one side with each adjacent hex. For these reasons, it is common to see hexes used in war strategy games. Unfortunately, because of the regular hexagon's shape, the hex grid is hard to implement. As such, we introduce the tex grid which is topologically equivalent with a hex structure but uses tiles. One can imagine a tex grid as a tile grid such that the odd columns are moved up by half the height of a tile (see Figure 2). A bricked wall is another example of a tex grid. Tex grids are more manageable than hex grids since space is represented as rectangles. Additionally, each tex is equidistant and share exactly one side to each adjacent tex; but more importantly, texes have a branching factor of three. Theoretically, texes are only slightly slower than the hex grid on average; $O(3^{0.809D})$ instead of $O(3^{0.805D})$. Another obvious advantage that texes have over hexes is that every tex path is shorter than a tile path, whereas some hex paths are longer than some tile paths (but on average is shorter). All in all, tex grids are exponentially faster than tiles and octiles (and slightly slower than the hex grid), produces smoother and shorter paths, and are easy to work with. The attributes for the choice of grid is summarized in Table 1.

Table 1: Summary of Grids

Grid Type	Branching Factor	Average Depth	Nodes Examined
Octile	5	$0.71D$	O(3.14^D)
Tile	3	$1.00D$	O(3.00^D)
Hex	3	$0.81D$	O(2.42^D)
Tex	3	$0.81D$	O(2.43^D)

4 Empirical Results

This paper has shown the asymptotic theoretical results, but it is unclear how tex grids behave for the grid sizes normally used in practise. This section contrasts the costs of IDA* searches on tile grids, against comparable tex grids. By comparable we mean that the tile grid and its equivalent tex grid can be fairly compared. For example in Figure 2, the tex grid can't be compared to the tile grid since the tile grid can not reach A while the tex grid can. All test cases where the tex grid has an unfair advantage over the tile grid are removed.

The tile grid and its comparable tex grid are compared in 10^4 independent trials (see Table 2). In each trial, a fixed number of obstacles are randomly placed on the grid, after that the start and the goal are randomly placed. Each tile or tex has a weight of either 1 or 2, to denote the cost to enter that tile or tex. Additionally, a path exists between the start and the goal in every trial. A glance at the $\frac{TexPath}{TilePath}$ column show that it reaffirms the theoretical prediction of 0.81. The $\frac{TexNodes}{TileNodes}$ and $\frac{TexTime}{TileTime}$ clearly show that searches on tex grids are faster and examine less nodes. It is also clear that tex

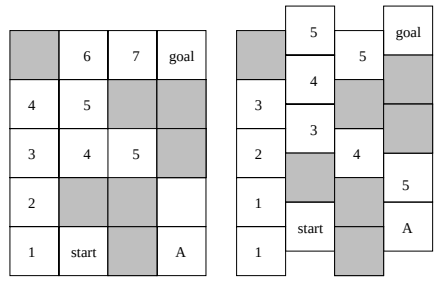

Figure 2: A tile grid and the corresponding tex grid

grids perform even better in the presence of obstacles and variable terrain weights. One curious result from these trials is the fact that the tex grid does better than the tile grid on the 10x10 grid, but not as well on the 20x20 grid (in contrast to the theory); this is probably because the small 10x10 grid limits the exponential size of the search tree. In general, we would expect tex grids to do better than tile grids exponentially as the grid size increases.

Table 2: Empirical Results

Size	Weights	Obstacles	$\frac{TexPath}{TilePath}$	$\frac{TexNodes}{TileNodes}$	$\frac{TexTime}{TileTime}$
10^2	1	20%	0.80	0.1	0.2
20^2	1	20%	0.81	0.3	0.4
10^2	1,2	10%	0.83	0.2	0.3
10^2	1,2	20%	0.82	0.001	0.002

5 Conclusion

While we have shown the many advantages of hexes including its exponentially faster search speed, it is the more manageable tex grid that we expect future games to use. These results are only applicable to games which use IDA*. Current research is focused on pathfinding on grids using A*.

6 Acknowledgments

I am indebted to Mark Brockington of BioWare, who patiently explained to me the problems of pathfinding in commercial games. I would also like to thank my supervisor, Jonathan Schaeffer, who helped with everything despite his busy schedule.

Financial support was provided by NSERC and iCORE.

References

[Korf *et al.*, 2001] R. Korf, M. Reid, and S. Edelkamp. Time complexity of iterative-deepenng-A*. *Artificial Intelligence*, 129(2):199–218, 2001.

[Yap, 2002] P. Yap. Pathfinding on Different Grids. To appear