

# FAST MARCHING AND FAST DRIVING: COMBINING OFF-LINE SEARCH AND REACTIVE A.I.

Daniel Livingstone,  
School of Computing,  
University of Paisley,  
Paisley,  
PA1 2BE

Email: daniel.livingstone@paisley.ac.uk

Robert McDowell,  
Real Time Worlds,  
1 Courthouse Square,  
Dundee,  
DD1 1NH

Email: bert@realtimeworlds.com,

## KEYWORDS

AI, Fast-March Method, A-star, Path-Planning

## ABSTRACT

Fast Marching Methods, FMM, have a wide range of applications, including path planning and navigation, but rarely feature in surveys of path planning techniques. For some applications, however, FMM are more suitable than other popular techniques, such as A\*. This paper provides a brief outline of how FMM may be applied to path planning problems and notes the strengths and weaknesses of the method. Finally, an example application of the FMM is provided, derived from work that was carried out on a published game.

## INTRODUCTION

Fast Marching Methods, FMM, (Sethian, 1998; Sethian, 1999) are highly efficient numerical techniques for tracking the evolution of interfaces (such as wavefronts) with a wide range of applications. Aside from uses in fluid mechanics, FMM have been applied to problems in graphics, vision and imaging, as well as other topics that more typically work with evolving fronts (seismology, combustion) (Sethian, WW).

Although the most common applications of the FMM revolve around the extraction of shape or other information from three-dimensional data sets, FMM can also be applied to problems of search and Path planning (Kimmel and Sethian, 2001). Like any search-method, the FMM has some particular strengths and weaknesses, and these are discussed later.

We also provide an outline of an implementation of the FMM in a recent video game, illustrating how the combination of a simple reactive vehicle controller AI and a global search can generate interesting and surprising, yet life-like, behaviours.

But first, we provide some more detail on the workings of the FMM, illustrated with examples of how it performs in some example two-dimensional path planning problems.

In this paper we restrict our discussion to the use of the FMM for path planning in two dimensions, although the

method can easily be used for search problems in three dimensions. We assume the problem domain to be a two-dimensional grid of nodes, containing a given goal node. To apply these techniques to a continuous game world, a grid may be superimposed over the continuous problem domain.

## PATH PLANNING WITH THE FMM

The FMM works in the manner of an expanding wavefront. Initially, the wavefront is in a given position (which may be a single point). Over time, the wavefront expands, reaching more of the nodes. The time taken to reach any node, the travel-time, being determined by the distance from the start point and by resistance offered along the wavefront (e.g. obstacles and impeding terrain) as it progresses. Once the travel time is known, it is a simple matter to calculate the direction which should be followed to reach the goal from any given point (see below).

In its working, the FMM categorises all nodes as either known – for nodes with known travel-time values, near – for nodes adjacent to those already calculated, or far – for all other nodes. Starting with a single known node (the goal, with a travel-time of 0), the FMM works outwards in a manner broadly equivalent to a weighted breadth-first search, or Dijkstra's method (Dijkstra, 1959; Stout, 1996).

The general algorithm is as follows:

```
Repeat
  Select node with smallest
  travel-time value. Remove
  from near list, add to
  known.
  Compute travel-time values for
  each neighbour of the
  selected point (recalculates
  values for any neighbours
  already in near list)
  Add neighbouring points to near
  list (and remove from far
  list)
Until near list is empty
```

The use of an efficient insertion-sort algorithm is required to ensure that adding nodes to the near list, and selecting the node with the smallest value, does not adversely affect performance. An array based implementation of a 'min-

heap' structure, which guarantees that the root element is always the one with the (possibly equal) smallest travel-time value of is outlined in Sethian (1998,1999).

**CALCULATING THE TRAVEL TIME TO A NODE**

A number of calculations are required to compute the travel-time at a given point. For a considerably more involved description of the process than given here, including the derivation of the FMM, again see Sethian (1998).

For our purposes, every node has a known slowness,  $s$ , which is determined by the terrain cost at that node. The travel time,  $u_{i,j}$ , at point  $i,j$ , is determined from the neighbouring nodes in  $x$  and  $y$  with the minimum travel time values. For this calculation, any points in the far list, or off the edge of the grid, are treated as having infinite travel-time values. As an intermediary step, we can find the minimum travel times of the neighbouring nodes:

$$u_x = \min (u_{i-1,j} , u_{i+1,j} ) \tag{1}$$

$$u_y = \min (u_{i,j-1} , u_{i,j+1} )$$

The new travel time value is then found by:

$$( u_{i,j} - u_x )^2 + ( u_{i,j} - u_y ) = s^2 \tag{2}$$

Finally, solve for  $u_{i,j}$  using the quadratic equation. As any unknown travel time value must be greater than or equal to all currently known travel time values, the new travel time value will be the minimum result that satisfies the condition:

$$u_{i,j} \geq \max (u_{i-1,j} , u_{i+1,j} , u_{i,j-1} , u_{i,j+1} ) \tag{3}$$

Impassable terrain can be simply represented as points with infinite slowness.

The algorithm given calculates the travel-time from every point in the world back to the goal – starting with the points next to the goal, and working outwards. To find the optimal path from any given point back to the goal is then a simple matter of calculating the travel-time gradient at each point, which can be done as the travel-times are derived.

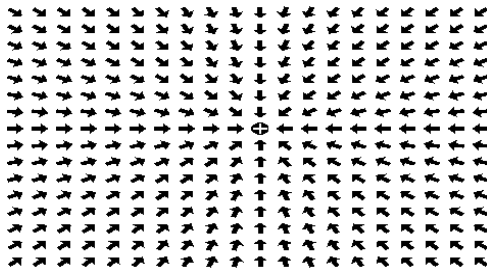


Figure 1: The Travel Time Gradient at Each Point Gives The Direction Back To The Goal. The direction of travel is not constrained by grid connectivity

**STRENGTHS AND WEAKNESSES OF THE FMM**

As noted above, this method is similar to Dijkstra's, which also starts at a single point and, expanding outwards, visits each neighbouring node in turn and keeps track of the cost to reach each point. However, the optimal path found by Dijkstra's method is a solution that is constrained to following the existing network connections. In contrast, the FMM can find a path which follows any arbitrary diagonal (Figure 1). This advantage of FMM is also true for comparisons against most other popular search methods, which require that additional smoothing operations be carried out in order to produce realistic paths without a noticeable 'zig-zag' pattern.

Perhaps the most widely used search methods for path-planning are those based on the A\* algorithm. A\* is a heuristic search which tries to find the shortest path from a given start point to a goal point, while exploring as little of the search space as possible. Much has already been written about A\* (see, for example, Ginsberg, 1993; Stout, 1996), and a description of its operation will not be repeated here. In most cases we would expect A\* to find a path from a given start point to a goal using far less operations than would be used by the FMM. Accordingly, in most applications where we need to calculate in real-time the path to be followed to reach a set goal from a set start point, A\*, or one of its variants, would be preferred over FMM.

While it is possible to modify FMM such that it terminates its search as soon as it has found a path from the goal back to the required start point, saving some processing time, the breadth-first nature of the FMM search means that it would still require more operations than the equivalent A\* search. This is assuming that the heuristic function,  $h'(n)$ , of the A\* search has been set to a value such that  $0 < h'(n) < h$ , where  $h$  is the perfect heuristic. Where  $h'(n) = 0$ , A\* also performs as a breadth-first search.

However, in any case where the goal is shared by a number of different units, which may have different start points, A\* requires a different search for each pair of start and goal points. FMM would only have to be run once to find the route for each of the units to follow. These strengths and weaknesses of the FMM for path-planning are summarised in Table 1.

Table 1: Strengths And Weaknesses Of FMM For Real-Time Path-Planning

Strengths	Weaknesses
Guaranteed to find optimal path	Slow (In comparison to A* using good heuristic)
Generates 'smooth' paths	
Finds path to goal from ALL points	

FMM has only one notable weakness, which is its computing overhead is high relative to the popular A\*. This alone is enough reason to rule out the use of FMM in many game applications, where computing time resources are precious and performance is always a high priority. In the next section we outline one application where the benefit gained by using FMM outweighed what turned out to be a negligent cost of the slow processing speed.

While the FMM as presented, and as used in our work, relies on a regular grid-like search space, the method can be extended for applications using triangulated-meshes (Kimmel and Sethian, 1998).

### OFF-ROAD RACING WITH REACTIVE A.I. VEHICLES

In this section we briefly outline some issues relating to the design of off-road racing games, which led to the selection of FMM search for use in a commercially released racing game.

The majority of video high-speed racing games, including those that market themselves as accurate simulations, possess incredibly simplistic vehicle controller AI for the computer controlled cars. For most titles, the AI is limited to a ‘catch up-slow down’ speed controller and a fixed route round the track for the AI cars to follow.

‘Catch-up/slow-down’ is implemented to make the game more interesting to the player, although it is highly unrealistic. When the player is behind the computer controlled cars, they slow down in order to let the player try and overtake them. When the player is ahead, the AI cars speed up. Indeed, it is interesting, and very simple, to test this out. In a racing game, try slowing down to a stop, and observe the effect this has on the time it takes the AI cars to complete a lap. It also explains why, in many racing games without any apparent difficulty settings, AI cars ‘improve’ as players get better.

The AI track following is commonly so strict that while collisions between player and AI vehicles may have a dramatic effect on the player vehicle, knocking it considerably off course, the converse is usually not true. Again, this can be easily tested, and the outcome can be quite striking. We suggest trying deliberately colliding with computer-controlled cars in Gran Turismo 3.

Creating a realistic AI for an off-road racer, which does not possess these flaws, presents a number of challenges. AI cars should react to the presence of other cars – swerving to avoid collisions if necessary. As a result of collisions (or attempts to avoid collisions), cars may leave the track. In some cases they could be knocked considerably off course, down gullies, or over other impeding terrain. As such, the best route to get to the race end may not be to try to rejoin the track as soon as possible, but to follow an alternative route – perhaps rejoining the track at some later point (Figure 2).

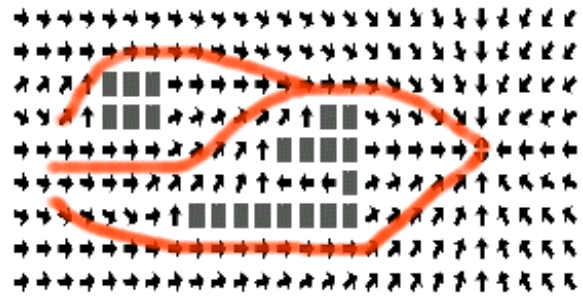


Figure 2: Small Perturbations Can Lead To Dramatically Different Routes

Accordingly, we note that this problem has the following features:

- All AI vehicles share a common goal
- The goal is known at compile time – real-time search is not required
- We need to know paths to the goal from a wide variety of points – including many ‘off-track’.

This makes off-road racing an ideal task for solving using FMM. The slow compute speed is not an issue, as the paths to the goal can be computed at compile time.

In use, some tweaking of the slowness values of terrain may be required – in practice the FMM search succeeded in many cases to find short cuts that omitted large portions of the track. Adjusting impedance values was sufficient to keep cars closer to the track, while allowing them the freedom to take different routes when forced off it.

Finally, it should be noted that a general awareness of the working of the FMM can be exploited by level designers. Careful design can lead to the deliberate inclusion of a number of key intersections in a level, where small variations in AI vehicle position can lead to the vehicles taking different routes to reach the goal. There is little ‘off-road’ about a racing game where all vehicles are forced to take the same route.

### CONCLUSIONS

Game players have ever increasing expectations. To date in racing games, realistic AI has not been in great demand – the lack of it has not adversely affected sales. Indeed, the simple, yet unrealistic, catch-up/slow-down is often used to guarantee that players are not left alone on a stretch of track. Ensuring that other cars keep pace with the player (and vice-versa) also ensures that the race remains exciting.

However, as newer games innovate, the audience are likely to become more aware of shortcomings of existing games and their demands are likely to rise. We are going to see more reactive AI in racing games – and this will introduce an opportunity to think differently about the search and path-planning methods used.

More generally, the FMM has obvious application in off-line search – which may be applicable to a wider range of game applications. The directions followed by agents towards a goal are not constrained to a superimposed grid,

providing for realistic path following when applied to a continuous game world.

On line search applications are also possible, particularly in any situation where it is required to find multiple paths for multiple units to a single target – such as can easily occur in many strategy games – where FMM may be more efficient than multiple A\* searches. Future studies might look at a comparison of the performance of FMM against that of A\* for solving such problems.

For many games A\* will remain the search of choice, but it is not the only algorithm worth considering – new and innovative search techniques continue to be developed, and many of these should, by right, find a place in a programmers repertoire.

### ACKNOWLEDGEMENTS

Robert would like to acknowledge Gordon Yeoman for introducing him to the FMM. We would also like to thank Darryl Charles and the anonymous reviewers for their comments on earlier drafts.

### REFERENCES

- Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. Numer. Math. **Vol.1**: 269-271.
- Ginsberg, M. (1993). Essentials of Artificial Intelligence, Morgan Kaufmann.
- Kimmel, R. and J. A. Sethian (1998). Computing Geodesic Paths on Manifolds. Proceedings of National Academy of Sciences, 95(15):8431-8435, July.

- Kimmel, R. and J. A. Sethian (2001). Optimal Algorithm for Shape from Shading and Path Planning. Journal of Mathematical Imaging and Vision **14**(3): 237-244.
- Sethian, J. A. (1998). Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision and Materials Sciences. Cambridge, UK, Cambridge University Press.
- Sethian, J. A. (1999). Fast Marching Methods. SIAM Review **41**(2): 199-235.
- Sethian, J. A. (WWW). [Http://Math.Berkeley.Edu/~Sethian/](http://Math.Berkeley.Edu/~Sethian/). (Accessed: 14/09/2003)
- Stout, B. (1996). Smart Moves: Intelligent Pathfinding. Game Developer (October).

**DANIEL LIVINGSTONE** is a lecturer at the University of Paisley, where he recently completed his PhD. He also holds an MSc from the University of Essex and a BEng (Hons) from the University of Strathclyde. His current research interests include Artificial Life, and almost anything games related.

**ROBERT MCDOWELL** is a games programmer working for Real Time Worlds. He has been a part of the games industry since his graduation from the University of Paisley a few years ago. He has a number of published titles under his belt, that have provided him the opportunity to work in many areas of games development. This has included A.I., engine development and general game programming.