The Long and Short of Steering in Computer Games

Simon L. Tomlinson

2 Campden Way, Handforth, Cheshire.
SK9 3JA, United Kingdom.
e-mail: DrSimonT@iclway.co.uk

Abstract - **Almost all types of computer game require some form of steering strategy to allow the AI agents to move around in the environment. Indeed such methods must form a solid base on which to build the remainder of the game AI. Some of the commonly used methods which can be categorised as short range steering (local steering) or long range (predominantly path finding) are reviewed and compared, with particular emphasis on how techniques are adapted for use within the games industry where the criteria for the algorithm may differ from other applications. Practical shortcomings of the methods are identified and solutions will be discussed with the objective of minimising resource usage whilst maximising the 'suspension of disbelief' which is essential for a successful computer game.**

Keywords - Computer Games, AI, Steering Methods.

I. INTRODUCTION

The objective of this paper is to provide an overview of techniques used in the games industry for steering methods, including extensive reference to detailed technical papers and an insight into the problems of designing and implementing a steering system. The principle application discussed here is the 1st/3rd person action game, although other genres of game will also be considered where they require alternative approaches. This type of game often requires not only movement on foot for a varied range of creatures, but also in more recent games such as Halo, Mace Griffin: Bounty Hunter and Conflict Desert Storm movement of ground and air/space vehicles. The main type of movement discussed here will be on foot, although issues associated with fast vehicle movement will also be briefly discussed. The subject of coordinated movement is beyond the scope of this paper although it forms a very important aspect of modern computer games.

A. Setting the Perspective

The main point to understand about programming for games is that the real time computing resources (both memory and processor power) can be very limited, especially for consoles such as the Sony Playstation 2™ and Microsoft Xbox™. As a result the AI programmer must strive to achieve optimum performance of the AI agents for minimum cost. A more detailed account of the Games Programmer perspective can be found in [Tomlinson et al, 2003], however it can be summarised by understanding that the emphasis is often on cheap and simple solutions which nonetheless convince the game player that the AI is a behaviourally complex unit.

This methodology has an interesting twist when applied to steering techniques. The main requirement for computer game AI is to improve the 'fun' for the player by providing immersion in the game world. It is often therefore more important for the AI to be believable than for it to attain high levels of intelligence. The AI must provide a challenge but, for a commercial product directed at a wide range of playing abilities, it need not be invincible. In general only very bad steering systems are actually noticed by the player, very good (accurate) steering systems are not readily distinguished from effective but mediocre ones. For example if a player is being chased by a monster, he will certainly notice if it gets stuck in the scenery, but he will rarely notice whether it caught up with the player a little earlier due to a highly efficient steering algorithm. Indeed small mistakes and a little fallibility can add to the sense of realism if adequately disguised [Tomlinson et al, 2003]. Thus it is often the programmer's strategy to minimise the resources used by the steering systems by compromising on their accuracy, so that systems which are more visible to the player such as tactics, strategy and personality can be maximised.

B. Steering Categorisation

Steering in computer games can be categorised as long range, exemplified by the A* path finding algorithm which is fundamentally a planned approach, and short range or local steering. Examples of local steering include force based methods such the flocking techniques popularised by Reynolds [Reynolds, 1987] and ray tracing methods where the agent tests and resolves potential lines of movement against the environment or some pre-computed representation of that environment. To be more rigorous the categorisation is not simply a question of distance, it is really a question of whether the route to the objective can be

seen and reached easily (a single line of movement can be established) or the route more convoluted – requiring twists and turns. The exact borderline is hazy, but generally a route which requires re-entrant turns (>180 degrees) can only be achieved using planning of some sort. As will be shown, it is really more a distinction of whether the steering solution is being planned, or is emerging from only local information.

II. LONG RANGE STEERING

A. Steering Using Pure A* Path Finding

Path finding relies on creating a graph of the viable movement areas as a set of nodes connected by arcs. The simplest type of graph is a square grid, although graphs based on triangulation are more efficient in terms of memory if the graph need only represent a minimal set of movement possibilities. However in practice it is often necessary to provide a more detailed graph to deal with behavioural issues [Tomlinson, 2003]. The 'industry standard' path finding technique is A*. This algorithm is well understood and easy to implement but also has the properties of being computational optimal – that is if there is a route to a target point then it will be found, and where there are multiple routes the best one will be found. An introduction to path finding techniques in computer games and A* in particular is discussed In reply to: [Stout, 2000; Matthews, 2002]. In general the path finding algorithm works by accumulating a score for the route. In it's simplest form this is distance travelled, but it is also possible to include other parameters in the state represented by each node, or on the arcs, such as terrain type or other tactical information.

A1. Optimising A*:
The primary method of optimising A* in computer games is to minimise its use (see [Higgins 2002] which also discusses other implementation optimisations) by using graphs with a small number of nodes or even a hierarchical graph and search. Clearly long range searches will be more costly, and possibly geometrically so. The programmer must therefore think carefully about the target node for the search, clearly the farther the target, the less likely it's tactical significance will remain for the increased duration of the path traversal. The implementation of the algorithm is also important. In particular A* depends heavily on the accuracy of the 'heuristic function' which estimates the cost from a current search node to the target. If this is particularly poor then A* will become a less efficient breadth first search. Interestingly slightly over-estimating the heuristic can reduce search time at the risk of making it 'inadmissible' [Rabin, 2000a]. The second point is that using larger amounts of memory not only risks compromising the total resources of the system, but on most consoles can produce very poor cache behaviour [Tomlinson, 2003]. Thus careful

design of the memory used by the algorithm is essential. It should be noted that where memory is critical the method known as IDA* [Korf, 1985]. This method uses a process known as iterative deepening to avoid having to store explicit open and closed lists and hence reduces the memory required compared to A* for a small cost in performance. Finally including a greater amount of information in the heuristic will not only make the evaluation at each node more costly, but may also broaden the search through the graph.

A2. Potential Disadvantages:
Although A* is a simple solution there are two potential disadvantages with using path finding alone as the total steering method. The first is that A* in itself is just too good. If there are multiple agents needing solutions to a similar target point or through a common choke point then the routes will overlap and mutually interfere. As a result some form of arbitration protocol is required to deal with queuing. In other words a simple path finding system will only deal with static issues, not dynamic ones. The other problem is that the routes may appear quite mechanical. If the routes are followed rigorously then the agents will tend to follow straight lines and make turns in places which do not look natural. Furthermore the quantised nature of the map can result in correct but non-obvious routes being chosen (See Figure 1).
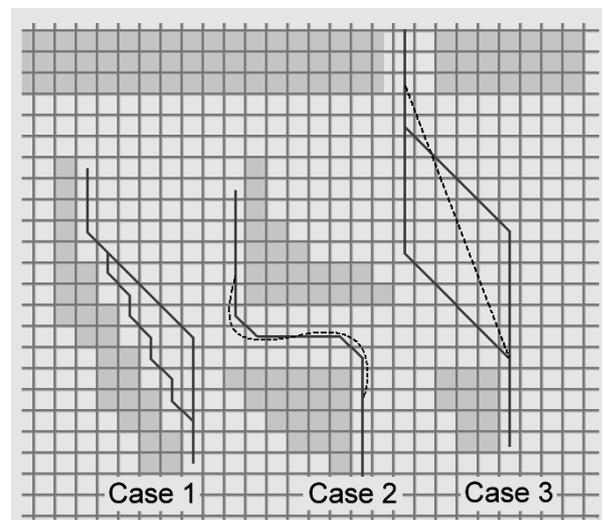

Figure 1. Case-1: two paths of equal cost but one zigzags and looks unnatural; Case-2: An optimal path may still require smoothing (dashed line); Case-3: Neither path is the intuitive route but this (dashed line) is not available due to quantisation.

B. Smoothed Path Finding

One way to improve the aesthetics of path finding is to smooth the route to produce a more natural, less quantised behaviour. There are several ways to achieve route smoothing but whichever is used a fundamental issue must be dealt with. A path finding route is a solution to a

problem, but only on the route itself, if the agent begins to deviate from the route then it is not clear whether what emerges will correctly avoid all obstacles. A simple example is one of cutting off a corner too closely which leads to the agent getting stuck thus defeating the object of the path finding. Figure 2 shows a simple 'walker' arrangement: the walker agent precedes the AI object by some distance which achieves smoothing, but can cause intersections with the scenery. Thus the algorithm must properly account for this using one of two broad strategies: either include factors for optimum aesthetic route finding in the A* cost function or accept line of movement tests against the environment. Cost functions include penalising turns to try to avoid unnecessary zigzags and or very sharp turns. Using LOM tests essentially involves iterating along the forward path to find the farthest visible point on the route or perhaps iterating back from an upper range limit. Either method has a processing cost and also brings into play the issue of the width of the object, which cannot always be assumed to be constant in modern games. In general LOM tests can deal with any value of width, but path cost solutions need to store data in the node such as bits which confirm if a member of a given width category can pass. It might also be tempting to try to pre-empt smoothing by attempting to find path routes which guarantee the deviation from the arc is available (for smoothing purposes), that is we increase the effective width of the traversing object. However this is dangerous because routes may be disqualified due to insufficient width when in fact they are viable because they have no significant turns and so do not require smoothing. Thus if this approach is used it must be done by calculating the effective width based on the arc entry and exit angles from the node. All this will in general make the path finding algorithm more cumbersome to solve and may not guarantee that no impacts with the scenery will take place. Often therefore the best solution is a combination of simple A* cost function modifications and limited LOM tests.

B1. Path Smoothing: Walkers and LOM tests are not the only methods for path smoothing. Rabin has proposed another simple method of fitting a Catmull-Rom spline to the set of nodes [Rabin, 2000b]. Another type of smoothing is the technique of finding geometric solutions to the known set of path nodes which can also account for the dynamic constraints of the agent such as turn radius [Pinter, 2001; Pinter, 2002], although in this case Pinter chose to ignore partial scenery penetrations due to smoothing. Whilst it is true that this approximation will not often be noticed by the player it does raise an additional question: how do the AI agents deal with collision detection and resolution. The implication of ignoring smoothing penetrations is that the hard collision is also ignored, which might be a good optimisation but could ultimately lead to issues of the AI completely and noticeably penetrating the environment. This is nonetheless acceptable in some games. Whichever

technique is used clearly the validity of the adjusted route must remain a question that the AI programmer must deal with.

C. Representing the World

The issue of agent width raises a secondary question associated with steering, which is how the world should be represented. Thus far it has been assumed that the path finding network consists of an arbitrary graph of nodes and arcs, but the ability to move in regions off the arcs is implicit and restricted by the scenery through additional tests. However it is also possible to assert that the polygonal areas bounded by arcs can be stood upon by the AI, that is the graph also represents a legally usable set of areas. With this in place other techniques can be employed. For example if all agents have the same or similar width we can bevel the edge arcs away from the scenery. This means that LOM tests are significantly simplified, as they no longer need to account for width, and they need only test against the simplified 2D map rather than the full 3D geometry. The disadvantage is the fact that this method is less able to cope with varied object widths.

C1. An Alternate Approach: Bounded polygons derived from the render geometry are not the only solution to areal representation. An alternate approach was discussed by Hancock [Hancock, 2002] where the nodes in the network are assigned a radius. Arcs therefore have an effective width which is based on a linear interpolation between the nodes at either end of the arc. This method is very attractive as it is simple and requires minimal data, although it can fail to fully represent all the walk-able areas due to discretisation issues. Other methods to represent the environment such as quad trees and convex hulls can also be used and have various advantages and disadvantages [Stout, 2000]. If non-areal graphs are sufficient a method to know as 'Points of Visibility' produces a simple network with a minimal number of nodes using an easy to understand construction [Rabin, 2000a; Young 2001]. Simple rectangular grids can also have advantages in terms of implicit node connectivity and can represent either networks or areas. The decision for the world representation facing the AI programmer will often also need to account for tactical or behavioural issues which may add nodes to the graph and mitigate the storage advantages of some solutions compared with others [Tomlinson, 2003].

C2. Graph Generation: It is also worth considering at this point how the graph is generated during development. The options can be summarised as automatic generation or manual generation by level designers. Automatic solutions include dividing the 3D render environment into a minimal set of convex hulls [Tozour 2002]. However this method can produce non-intuitive node distributions which, whilst being correct, can lead to unusual path finding routes that

require a substantial smoothing effort. Manual solutions using effective design tools are often superior as the level architect can include substantially more context in the graph (e.g. preferred routes for tactical or other behavioural reasons) but these can be more labour intensive. Thus the make-up of the team and the structure of the development pipeline can influence the programmers choices as much as technical issues.
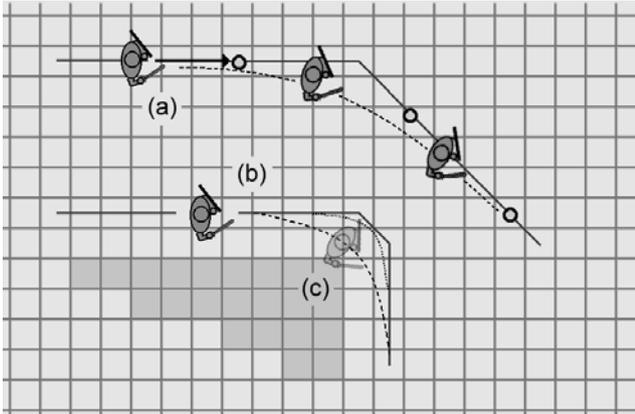

Figure 2. (a) The soldier is led by the walker for natural smoothing; (b) longer lead distance means greater curvature around kinks in the path; (c) This risks the soldier intersecting the environment (shaded squares).

D. Path Finding: Dealing with Dynamic Objects

Path finding is clearly capable of dealing with static avoidance, but can it deal with dynamic objects? The short answer is that it cannot, and a short range steering method will be required. Although path finding algorithms designed for changing environments do exist in robotics [Stentz, 1994] to the authors knowledge D* has not been used in games and may be too cumbersome for this application. This section will explore some of the possible techniques for games where aesthetics are less of an issue and agents only move along arcs.

D1. Naive Approach: A naive approach is to simply disallow nodes to be used which are already part of the active route of another agent, but there are two problems with this. A long range path route is temporal as well as physical, it is a plan that must be followed over a period of time. The longer the time period is, the less accurately we can estimate whether it will be blocked by a dynamic object at a particular place and time. Thus the algorithm may adjust or even fail to find a route based on a weakly anticipated future collision. The method described by Cain [Cain, 2002] neatly alleviates this to some extent by using iterative deepening to spread the processor effort over several frames which also allows the dynamic situation to evolve in the meantime. The second issue is one of data – for multiple objects to avoid each other there must be a sufficient density of alternative routes available, which will

increase both memory and solution time significantly. Even then we may encounter problems where routes must cross, which will require some sort of traffic manager. Nonetheless this might be appropriate in some applications such as air and ground traffic control in a flight simulator game for example.

D2. Realistic Approach: A more realistic solution is to apply scores so that path queries will attempt to avoid nodes which have been chosen by other agents by increasing their cost. Nodes are never disqualified due a weakly anticipated collision, but do become less likely. A modest number of alternative routes are made available so that groups of agents are encouraged to spread out, which is a significant aesthetic bonus. However this will not entirely solve the problem in choke points for example, and inevitably some additional form of arbitration will be required.

D3. Dynamic Collisions Ad Hoc: A second type of strategy is to deal with dynamic collisions ad hoc, that is when imminent collisions are detected, based on the priority of the agent. Strategies range from very simple ones such as one object stopping when a potential collision is detected to re-checking sections of route, re-planning and splicing in sections due to local blockages. Of course simply stopping will not work if the two agents are following the same path in the opposite direction. Re-pathing when such a contention arises may be possible where an alternate route exists but if none is available a 'false' route is needed to force the submissive agent to back up and clear the route. This problem is not trivial for two agents in contention and with many agents it can be extremely difficult.

So to conclude, although use of an increased number of routes to reduce the number of dynamic potential collisions is useful, it is impractical to assume that this will solve the entire problem without additional detection and resolution schemes. However providing a higher density of routes will allow more tactically appropriate behaviour to be introduced using adaption of the heuristic.

E. Summary

It is apparent that practical considerations of usage, aesthetics, tactics and so forth make the solution to long range steering not only considerably more complex in games that it may be in more abstract applications, but it is also highly dependent on the style of the game and what assumptions can be made or short-cuts accepted. It should also be noted that although this section began by looking at long range steering, many of the issues discussed overlap into what is really short range steering. This is an important point, for the vast majority of computer games long range steering alone is not sufficient, short range steering will also be required even if it is only applied during the interpretation of the long range path finding route.

III. LOCAL STEERING

A. Emergent Steering (Flocks, Swarm, Herds etc.)

Emergent steering schemes in general are very different from path finding. Not only do they use only local environmental information, but also the solution is usually built up from a number of different terms in a parallel fashion, and as such the final route is developed over time and cannot be calculated a-priori. That is local steering techniques are 'emergent'. Emergent steering techniques are much more responsive to dynamic changes in the environment. The best known example of these techniques is 'flocking' [Reynolds, 1987] which can be used directly in games to model the movement of groups of birds, fish or similar creatures. The basic method is on each iteration to evaluate a number of force components which are combined to ultimately derive an acceleration to be applied to the agent. There are four principle sources of force: Separation designed to maintain the distance between the flock mates, Alignment which is the steering guidance term (for a flock this is just the average heading of the group), Cohesion which brings the flock together into a unit and Avoidance to prevent collision with static scenery or dynamic objects. However the beauty of force based techniques is that they are infinitely variable. Forces can be adapted or added as required. For example different formulations might be appropriate or convenient for the avoidance of dynamic objects, small static objects and large static objects and an interesting model [Woodcock, 2001] adds predator-prey seek behaviour. If large numbers of creatures are required (known as a swarm or herd) the rules can be simplified to achieve group behaviour within a limited resource [Scutt, 2002]. Flocking has also been used for Monsters in Unreal and Half-Life and where there are large numbers of human AI such as real time strategy (RTS) games. However all these systems are primarily designed to simulate animal like behaviour which is generally not a primary part of the game, in this paper we are concerned with highly motivated intelligent AI agents which should be moving around the environment with definite purpose.

A1. Clarification: Thus before progressing a point of clarification should be made. Since the term flocking has become intimately associated with A-life type applications with larger numbers of creatures I will draw a distinction and define the term 'Force Based Emergent Steering' as the calculation and combination of forces in order to guide steering for more individual autonomous agents, although they may indeed be acting as a group. The main difference between this and flocking is that although the separation and avoidance terms remain, the alignment and cohesion terms are replaced by a single target seeking force. This is designed to provide a directional motivation which can be more readily controlled by higher level AI goal setting algorithms. This is not really new, it is simply defining a

seek behaviour as a force component to be used in a combined force based approach [Reynolds, 1999]. Reynolds also defines a number of other behaviours such as 'forage' and 'wander', but here the more motivational aspects are assumed to be controlled in higher AI layers and are converted to a seek target. This approach has been used effectively by the author in the Mace Griffin: Bounty Hunter game for space craft.

A2. Main Disadvantage: In terms of game design the main disadvantage of FBES is that it does not actually plan – so it may be prone to going up dead ends for example, and may fail to find a route. Thus FBES is better suited to environments which consist of relatively large open spaces with occasional isolated obstacles rather than networks of narrow corridors for example, such as space combat games and RTS games which generally work on an open landscape. This is in fact only one example where the agent will become trapped in a local force minimum (and hence be unable or unmotivated to move), in this case where the seek force is balanced by the wall avoidance force. Other possibilities include clusters of agents with mutually opposing seek directions which balance against separation forces to achieve a minimum. This can easily occur if a group of agents seek a common goal for example.

A3. A Second Disadvantage: A second but almost equally important disadvantage is that emergent systems are less controllable: a typical comment is that a game agent must be able to fully follow the designer's script but emergent systems do not guarantee this. In the previous paragraph for example no agents would actually reach the common seek goal, and so a scripted trigger based on that event may fail to fire properly. This point is true, but not insurmountable. Firstly it has to be noted that as games become more complex the ability of designers to fully control the entire game world at an intimate level will become less viable in any case. Indeed if games are going to begin to use better AI to improve both game play and productivity designers will have to accept some loss of control. Secondly where it is absolutely necessary to achieve certain goals it is always possible to bypass the FBES system by replacing it with a more directed navigation agent. Thirdly it should certainly be a goal for the AI programmer to deal with these issues such that the seek force is paramount in the combination of the forces, even if this requires state changes such as re-calculating long range paths.

A4. The Main Advantage: of an emergent system is that very point – much of the behaviour emerges from relatively simple processes without the need for in-depth analysis. Indeed by creatively adapting the rules many interesting behaviours can be created. For example for A* the issue of providing realistic routes rather than all agents following the same route was identified. In FBES this tends to occur

spontaneously due to the separation forces combined with obstacle avoidance (which tends to seed the choice of alternate routes depending on how the agent first approaches it) so that a group moving through a forest for example will tend to flow through the obstacles following a variety of routes.

B. FBES Implementation Issues

The reasons why the FBSE techniques have not been widely used may also be associated with implementation problems. These are varied, but the main points are environmental representation, dynamic stability and behavioural realism.

B1. Environmental Representation Problems: Problems with environmental representation are two fold, the overall volume of data and how to generate the forces from that data. Dynamic objects tend to be less of a problem since their numbers are modest (or can be limited by simple proximity tests) and sizes similar and can thus be represented using simple shapes such as a sphere, which generates a spherical force. The problem tends to be with the static render geometry. This usually exists in a 3D scene graph and so extracting appropriate local information is not always simple, and must be properly managed to avoid a data bottleneck. Solutions include a similar strategy to the A* above, generate intermediate data either as simplified 3D geometry or 2D navigation surfaces where edges define the presence of obstacles. The remaining issue is then how to evaluate the forces. There are many solutions to this if one looks for analogies from physics such as electric or magnetic field calculations. These vary significantly in complexity but all produce a repulsive force which will act to prevent the agent intersecting with the geometry. In a computer game application the best solution is probably to use the formulation based on the simplest geometry – such as line or point charges. Even then careful coding is required to ensure the evaluation of this force does not dominate the entire AI budget. It is also important to understand that a repulsive force alone is not sufficient, a secondary force is required to provide steering around the object, generally in a direction parallel to the surface. It is the authors experience that objects which are a similar size as an agent or smaller do not require this 'circumferential force' but large objects where the surface is almost flat when viewed by the agent do require it. Reynolds [Reynolds, 1987] used a slightly different approach by using a single force designed to attract the agent to the edge of the object's silhouette.

B2. Dynamic Stability: The most difficult issue is probably dynamic stability which in turn results in behavioural problems. Flocking was originally designed to models 'boids' which are continually moving. However a generic AI agent will often need to come to a halt (indeed

its primary goal should be to halt at the seek target) which can only be achieved in an FBES system by reaching a minimum in the total force map. For a single agent there are several difficulties. Firstly the locomotion rules must recognise that the agent is about to reach a target and make appropriate changes to its speed – see 'Arrival' in [Reynolds, 1999]. Unfortunately however it is not clear where that position will be a-priori, the seek target may be specified but other forces may create a balance such that the actual local minimum falls short of this. As a result the stopping speed is only approximate and simple agents can over-shoot the target or even orbit it. If there are multiple moving agents the problem is only worsened since this can result in multiple moving local minima. Figure 3 shows a situation which can arise where the force arrangement combined with dynamic lag (more on this later) results in oscillation rather than steady movement.
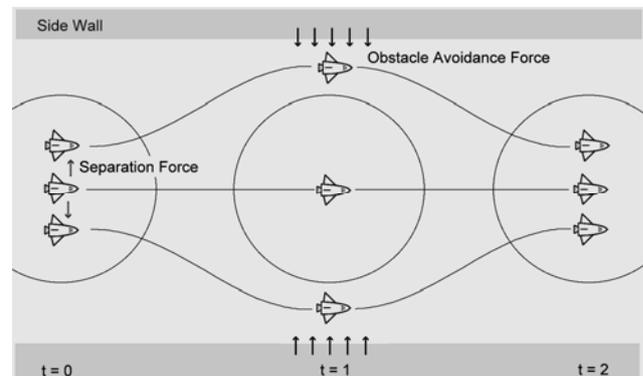

Figure 3. At t = 0 the two outer craft experience a separation force which pushes them apart, at t = 1 they are outside of the separation influence but have a large avoidance force from the walls and are so are pushed back to the centre to repeat the cycle at t = 2. This problem is particularly bad if the integration rate is poor or the craft have a large lag between the instruction to steer and the realisation due to dynamic constraints.

B3. Controlled and Intelligent AI Objects: This is not significant for birds or fish, but is highly significant for controlled and intelligent AI objects as it does not look like suitable behaviour. Note that these problems are not caused by poorly designed force field distributions, it should be possible to use any form of field, but are more to do with the way the object integrates the effect of the forces from frame to frame. Thus there are two routes to a solution as with any integration problem: the first is to use smaller integration time steps (maybe multiple updates per render frame); the second is to use higher order time differential components to better estimate the integral (e.g. Runge-Kutta integration or some other form of engineering control approach). In either case a certain amount of care and balancing is required but in the authors experience stability can be achieved by careful design.

Once stability has been achieved the only remaining issues are minor behavioural ones. As noted above agents will not come to a halt precisely at the seek target, but this can be designed around or the force system modified using

AI state changes to deal with this. In general FBES offers more advantages behaviourally: it is possible to apply any form of goal seeking through the seek target and group dynamics can be applied by adding additional cohesion terms. For example pairs of soldiers can prefer to work together by applying attractive forces. These behavioural terms can be switched on or off based on AI state. It is also worth noting that in some cases a temporary lack of stability can lead to interesting group dynamics – for example if agents slow too slowly they can push others forcing the group as a whole to re-arrange itself.

C. Other Local Steering Methods

FBES is by no means the only local steering method. Ray casting (LOM checks) has already been mentioned but can be costly. Another lightweight method based on robotics has also been described previously [Mika and Charla, 2002]. However these methods can be costly in a 3D environment and are also prone to stability issues. In some games such as RTS constructional approaches which account for the dynamic constraints of the agent's movement can be very effective [Jakob, 2003] and Green has also applied Reynolds' ideas in the game of Dungeon Keeper [Green, 2000]. In practice these methods are very similar to the FBES and suffer from many of the same problems. A particular danger is state changes such as avoiding only the nearest obstacle which in a crowded environment could lead to the agent constantly re-focussing it's attention and again seeming to oscillate. A slight variation on local steering component combination is to use a pipeline, where a steering goal is presented from the high level AI and modified sequentially to satisfy each movement constraint, although care must be taken with designing the pipeline prioritisation.

D. Summary

Almost all local steering behaviours are characterised by the combination of one or more simple components into a single goal steering instruction which is evaluated regularly (usually every frame). While the components are simple they must be properly used. The combination of components without causing instability is a challenge, but not an impossible one. Local steering can also be widely adapted to exhibit a range of agent behaviour.

IV. HYBRID STEERING USING PRE-COMPUTED METHODS

It is possible to identify several other methods which we might consider as a hybrid of short and long range steering. Such methods are essentially planned off-line and interpreted in real time in such a way that only local data is queried. The most obvious game genre to benefit from this method is the racing game where the track is essentially a single continuous looped spline with width information. Reynolds has discussed a simple example of track following behaviour [Reynolds, 1999], although his method is not really suitable for a racing simulation which needs to follow a racing line rather than simply remain inside the track. A simple system was discussed by Biasillo [Biasillo, 2002] although more advanced techniques are emerging for modern games particularly simulation games where players are growing to expect more realism in the AI vehicle. Such a method was briefly outlined in [Tomlinson et al, 2003].

A. Action Games: Returning to action games, there are three related methods which pre-compute navigation hints into a grid covering the entire playable area. Potential fields (or flow fields) encode data which indicates a direction of movement for both agents on foot (see Baert web site in further reading) and vehicles [Egbert and Winkler, 1996]. In it's simplest form a potential field is simply a pre-calculation of the forces generated by static objects which is a welcome and significant optimisation since it vastly reduces the number of environmental queries required in real time. However because the potential field can be manipulated in any way, it is possible to encode additional behavioural hints. This idea is called influence mapping [Tozour, 2001]. Tactical information is represented on the grid which can also account for terrain and can therefore be used as a navigation system, typically in RTS games. Tozour illustrates that pre-computed path finding data can be used as part of the map. Another technique is the Fast March Method which has recently been suggested for navigation in free roaming vehicle games [Livingstone and McDowell, 2003] which is effectively generated by inferring data from a Dijkstra search. However there is one problem with all of these methods for long range navigation. They tend to require that the target point is defined to perform the pre-computation. Of course in a game world there will be many target points and so one grid of data might be required for each target which is not feasible with a limited memory resource. An exception has been proposed [Surasmith, 2002] in which path finding data is pre-computed and stored in a 'connectivity table' although as the author notes memory could be an issue. Therefore at the moment pre-computed navigation methods are limited to a few game types with a minimum of goals, or as a simple optimisation for local steering methods where the data is not goal based but represents only avoidance information.

V. VEHICULAR STEERING

As noted above steering for racing games in confined tracks requires a special approach, but steering for other vehicles (or large creatures) in free roaming worlds also presents additional problems. These arise from two main sources, differences in scale and the dynamic constraints of the vehicle such as acceleration and turn rate (which could

also apply to creatures on foot). For path finding larger vehicles may not be able to traverse all routes and so width must be accounted for. Other routes might be problematic because they cannot be navigated within the constraints of the vehicle – for example if the vehicle must move to turn and has a limited turn rate (e.g. an aircraft) then very sharp changes of direction on the route are not possible, which must also be dealt with in the A* cost. For FBES dynamic constraints act as another potential source of instability. The reason for this is that limited turn rates and accelerations effectively result in a temporal lag between the resolution of the forces and the actuation of the locomotion. For example forces indicate that the vehicle must brake to a halt, but the dynamic module is slow to respond, the vehicle may overshoot the target leading to various forms of undesirable oscillation or orbiting behaviour. Again engineering control type approaches can deal with these issues. Scale affects local steering more through speed than size. Very fast vehicles must be aware of potential collision objects much earlier, which not only increases the amount of environment they must query, but can necessitate alterations to the avoidance constructions or force formulation to favour the forward direction.

The traditional approach for vehicles in character based actions games has been to treat them as simple special cases - a so called 'smoke and mirrors' approach [Tomlinson, 2003]. In it's simplest form this means running the vehicle on a spline, for example a tank may be hiding in the woods and when the player approaches it pulls out on a fixed path for a few tens of yards then starts blasting. But this type of approach must be questioned in terms of replay value of the game. In some ways repeating the same action on every play is good because the player can develop a strategy in subsequent plays if he is killed by the tank the first time. This is a common design stance in puzzle games. But if the player is to be attracted to repeat the mission after successful completion the tank must change tactics. Thus slightly more advanced solutions include multiple fixed paths which the tank can choose, or even restricted navigational areas designed especially for the vehicle within which it can freely roam. Of course if the vehicle or craft is common in the game then it is worthwhile making it's steering abilities an integral part of the AI system rather than an exception.

VI. CONCLUSION AND FUTURE WORK

This paper has reviewed two main groups of steering strategies for AI objects in computer games. It transpires that both techniques have shortcomings in a computer game in which both static scenery and dynamic objects contribute to the total environment. Although path finding algorithms can be made very efficient, and it is plausible to use such an algorithm alone for static object avoidance, the presence of dynamic objects and the need for aesthetic smoothing can significantly complicate both the interpretation of the path

route and it's solution in an A* type algorithm. On the other hand local steering methods are excellent for dynamic avoidance and safe path smoothing and can be used creatively to induce emergent behavioural traits, but they are unable to anticipate blocked routes in highly corrugated maps or find optimum routes where information is outside their limited environmental range. The solution therefore may well be a combination of the two techniques, indeed it should now be apparent that the two methods overlap to some degree (long range methods must have a short range interpretation). However hybrid algorithms based on pre-calculating path finding data are of limited application. Thus the most practical choice is to use both short and long range steering in harmony, or find short cuts in the application which favours one method and ameliorates the risks. Often one of the most powerful methods in computer game development is to design the world such that it helps the steering algorithms rather than presenting a significant and unjustifiable challenge in terms of added game play value by asking the AI to deal with a arbitrarily difficult problem.

Future Work: It has been suggested that most game developers now believe that the basics of steering are well understood [Woodcock et al, 2000], and any future work in the games industry will concentrate on special cases such as choke points and extensions such as tactical factors. Indeed it seems developers are now beginning to feel that even computing resources are becoming less of a problem. The emphasis for the future will therefore be one of behaviours and creativity in AI rather than dealing with the purely mechanical issues of steering. Having said that the exact steering solution will vary significantly between applications and platforms; PCs are generally resource rich and game consoles have also improved significantly but applications such as handhelds (e.g. Gameboy), PDAs and mobile phones will continue to require lightweight and ingenious solutions.

REFERENCES

[1] Cain T. 2002, "Practical Optimisations for A* Path Generation", In AI Game Programming Wisdom, Charles River Media, pp146-152.

[2] Egbert P. and Winkler S. 1996 "Collision-Free Object Movement Using Vector Fields", IEEE Computer Graphics and Applications, 16(4), pp18-24.

[3] Biassilo G. 2002, "Representing a Racetrack for the AI", "Racing AI Logic", Training an AI to Race". AI Game Programming Wisdom, Charles River Media, pp437-455.

[4] Green R. 2000, "Steering Behaviours" from Course 39: "Games Research: The Science of Interactive Entertainment", In Proc ACM SIGGRAPH '87 Conf.

[5] Hancock J. 2002, "Navigating Doors Elevators, Ledges and Other Obstacles", In AI Game Programming Wisdom, Charles River Media, pp193-201.

[6] Higgins D. 2002, "How to Achieve Lightening Fast A*". In AI Game Programming Wisdom, Charles River Media pp133-145.

[7] Jakob M. 2003, "Directional Extensions of Seek Steering Behaviour". In Proc. Game-On 2003 Conf. (London), Eurosis, pp187-191.

[8]   Korf R.E. 1985, "Depth-First Iterative Deepening: An Optimal Admissible Tree Search". In Artificial Intelligence, 27, pp97-109.

[9]   Livingstone D. and McDowell R. 2003, "Fast Marching and Fast Driving: Combining Off-Line Search and Reactive A.I.", In Proc. Game-On 2003 Conf. (London), Eurosis, pp197-200.

[10] Matthews J. 2002, "Basic A* Pathfinding made simple". In AI Game Programming Wisdom, Charles River Media, pp105-113.

[11] Mika M. and Charla C. 2002, "Simple and Cheap Path Finding", AI Game Programming Wisdom, Charles River Media, pp155-160.

[12] Pinter M. 2001 "Toward More Realistic Pathfinding", Game Developer Magazine (April), CMP Media LLC, pp54-64.

[13] Pinter M. 2002, "Realistic Turning Between Waypoints", In AI Game Programming Wisdom, Charles River Media pp186-192.

[14] Rabin S. 2000a, "A* Speed Optimisations". In Game Programming Gems, Charles River Media,, pp272-287.

[15] Rabin S. 2000b, "A* Aesthetic Optimizations", In Game Programming Gems, Charles River Media,, pp264-271.

[16] Reynolds C. 1999, "Steering Behaviors For Autonomous Characters". In Proc. Game Developers Conference (San Jose, California). Miller Freeman Game Group, San Francisco, California. pp763-782.

[17] Reynolds C. 1987, "Flocks, Herds, and Schools: A Distributed Behavioral Model", In Proc ACM SIGGRAPH '87 Conf., Computer Graphics, 21(4) pp25-34.

[18] Scutt T. 2002, "Simple Swarms as an Alternative to Flocking", In AI Game Programming Wisdom, Charles River Media, pp202-208.

[19] Stentz A. 1994, "Optimal and Efficient Path Planning for Unknown and Dynamic Environments", In Proc. IEEE Int. Conf. On Robotics and Automation, (May).

[20] Stout B. 2000, "The Basics of A* for Path Planning",. In Game Programming Gems, Charles River Media, pp254-262.

[21] Surasmith S. 2002, "Preporcessed Solution for Open Terrain Navigation", AI Game Programming Wisdom, Charles River Media, pp161-170.

[22] Tomlinson S.L., Davies A. and Assadourian S. 2003, "Working at Thinking About Playing …". In Proc. Game-On 2003 Conf. (London), Eurosis, pp5-12.

[23] Tozour P. 2001, "Influence Mapping", In Game Programming Gems 2, Charles River Media, pp287-297.

[24] Tozour P. 2002, "Building a Near Optimal Navigation Mesh", In AI Game Programming Wisdom, Charles River Media, pp171-185.

[25] Woodcock S., Laird J. and Pottinger D. 2000, "Game AI: The State of the Industry", Game Developer Magazine (August), CMP Media LLC, pp24-39.

[26] Woodcock S. 2001, "Flocking with Teeth: Predators and Prey", In Game Programming Gems 2, Charles River Media, pp330-336.

[27] Young T. 2001, "Expanded Geometry for Points-of-Visibility Pathfinding", In Game Programming Gems 2, Charles River Media,, pp317-323.

FURTHER READING AND LINKS

[1]   www.gameai.com/pathfinding.html - path finding resources for game AI.

[2]   www.gamasutra.com/features/19990212/sm_01.htm - Stout paper on the basics of A*.

[3]   www.gamedev.net/reference/articles/article1125.asp – Baert's paper on potential field planning.

[4]   www.gamedev.net/reference/list.asp?categoryid=18#94 - general AI resources for games.

[5]   www.riversoftavg.com/flocking.htm - download of flocking based on Woodcocks' algorithm.

[7]   www.gamasutra.com/features/20010314/pinter_pfv.htm - more realistic path finding (expanded version of Pinter paper).

[8]   www.ai.mit.edu/people/lpk/mars/temizer_2001/Method _Comparison/ - potential field methods compared to a model of human movement.

[9]   www.red3d.com/siggraph/2000/course39/ - Robin Green's paper which is an excellent example of local steering based on combined constructional behaviours.

[10] www.steeringbehaviors.de/ - description of a complete model based on the work of Reynolds and Green.

[11] www.student.nada.kth.se/~f93maj/pathfinder/index.html - "An optimal pathfinder for vehicles in real-world digital terrain maps" includes techniques such as simulated annealing and other interesting alternative path finding methods.

[12] www.red3d.com/breese/navigation.html - part of the Reynolds site with many links to further more academic papers.

[13] www.gamasutra.com/features/19990122/movement_01.htm - Co-ordinated unit movement by Dave C Pottinger.

[14] www.igda.org/ai/report-2003/report-2003.html - interesting set of working groups including path finding and steering with the objective of finding agreed standardisations in the game AI community.

[15] www.gamasutra.com/features/20000619/wright_pfv.htm - explores resource usage for game AI and the balance between accuracy and believability.

[16] www.gamasutra.com/features/20020405/smith_01.htm - "GDC 2002: Polygon Soup for the Programmer's Soul: 3D Pathfinding" – looks at how to extract path finding data from 3D geometry.

[17] www.gamasutra.com/features/19991110/guy_03.htm - ray casting for navigation.

[18] www.gamasutra.com/features/19990820/game_ai_01.htm - Game AI state of the industry in 1999 by Steve Woodcock – bit dated but possibly still relevant, with interesting sections on which technology is waning and which evolving, and how academics view computer game AI.

[19] www.gamasutra.com/features/20010423/dybsand_01.htm- Overview of game AI from GDC 2001.

[20] www.gamasutra.com/features/20000626/brockington_pfv.htm - an explanation of IDA* for path finding.

AUTHOR BIOGRAPHY

Simon Tomlinson was born in Southport, England in 1965. He gained a BSc in Physics and PhD in Electrical Engineering from Manchester University. After continuing an academic career with research into solid state magnetic materials, he fulfilled a lifelong interest in games by moving to Mirage in 1997. He has since worked on the AI and dynamics of a variety of successful products including Pool Shark, World Championship Snooker, Mace Griffin Bounty Hunter and Battlestar Galactica. Simon has most recently worked at Warthog where he developed AI and advised local Universities on Games education.