39

Representing and Driving a Race Track for AI Controlled Vehicles

Simon Tomlinson and Nic Melder

39.1	Introduction	39.5
39.2	Physical Track	39.6
	Representation	
39.3	Calculating Track Data	39.7
39.4	Driving the Track	39.8

- 9.5 Racing Lines
- 39.6 Alternate Lines and Other Tactical Information
- 9.7 Using Splines
- 9.8 Conclusion

39.1 Introduction

One of the most important elements in an AI racing system is the representation of the track. The amount and diversity of data contained in this structure will depend on the genre, but there will always be a representation of the physical layout. There will also usually be some sort of racing line—a guide that is used by the AI as a primary (fastest) route along the track. However, the nature of the racing line and other data depend on the details of the genre.

An arcade style game may require detours to retrieve pickups and, in this case, the racing line may be less about the fastest route and more about choosing the best tactical route; that is, the track will be heavily branched with alternative routes. In a very tightly defined game, such as a Formula 1 simulation, there will be a single, highly optimized racing line that contains detailed tactical information all the AI vehicles can reference. Because all the vehicles have similar characteristics, the optimal speed for any corner can be "baked-in" to the track data during development. In a game where a wide range of vehicles can race on the same track, there may be too much data to bake-in and so techniques are required that allow parameters such as the optimal speed to be evaluated in real time. Indeed, real-time evaluation will account for variations in the current situation (such as a non-ideal corner entry) and is therefore preferred if computing resources allow. Ideally,

any AI solution should use both approximate baked-in data at the strategic (behavioral) level and detailed real-time evaluation at the tactical (driving) level.

39.2 Physical Track Representation

As a minimum, the objective of the track representation is to define the boundaries of the racing area. A good standard representation is to mark up the track as a series of nodes along the nominal track center, with perpendicular and normal vectors defining the width of the track and its orientation. These nodes are then arranged in an array or bi-directional linked list to form a series of quadrilateral segments, as shown in Figure 39.1. This is quite similar to previously published representations [Biasillo 02a] but is strongly node-centric rather than focusing on the area of the segment. Note that the track is normally projected onto the 2D ground plane; height variations in the track are important, but are dealt with separately. However, this is not possible if the track is truly 3D and has loops or many over-/underpasses.

The track width need not be symmetric and rarely is in practice. It is more important to keep the path defined by the track nodes and links reasonably smooth. The definition of track width is also fluid. A good definition is the main area of tarmac on which the vehicles race. However, in real-world tracks, there are often rumble strips, runoff areas, road junctions in a street track, and so on, all of which might be drivable to some extent. Therefore, the edge of the track is best considered to be *fuzzy* and represented by three values on each side: the main track boundary (w_t), the extended runoff boundary (w_r), and the hard width, which defines solid walls or barriers (w_w) as shown in Figure 39.1. These widths are defined separately for each side of the track and can vary independently. However, on a track with close-in walls, these three widths will be the same. Also, the track width should be smooth, as sudden changes or kinks can prove troublesome in later processing.

Note that the node orientation as defined by the normal vector, \mathbf{n} , is not the same as the link vector between nodes, \mathbf{l} . Because the representation is node-centric, this should be





derived from the track tangent at the node which is the average of the forward and backward links, as shown in Equation 39.1.

$$\underline{t} = \left[\left(\underline{N}_{i+1} - \underline{N}_i \right) + \left(\underline{N}_i - \underline{N}_{i-1} \right) \right] / 2$$
(39.1)

39.2.1 Generating the Track

It is possible to generate the physical track representation of nodes and widths by processing the "polygon soup" that defines the track surface. However, this method is not recommended. It can become confused at street track junctions or tarmac runoff areas, for example, and will often require designer intervention to correct such issues.

It is much better to allow the track designers to mark up the track nodes, links, and different widths, within a track editor tool. Using a tool has large advantages. Such a tool can provide a lot of programmatic assistance in placing the nodes—for example, by suggesting widths for a new node placement or supplying metrics, such as link and edge smoothness. Last minute edits to the track are easier to deal with, and the human eye is a much better arbiter of the fuzzy track edges than a simple algorithm. The designer will also have the option to adjust the track in order to affect the AI behavior—for example, by narrowing the main track and hard track widths as a last resort to reduce AI collision avoidance failures in difficult areas.

39.2.2 Branches and Open-Ended Tracks

Normally, a racing track will be a single list of nodes, where the last node connects back to the first node to form a closed loop. However, other configurations are also possible. A track may have one or more branches that will require multiple link references in the data. If the branches are high speed, it is a good idea to make the branch in the track representation some distance before the actual physical branch on the track, so that the most appropriate racing lines on the approach to the branch can be stored. Bear in mind that the two routes overlap physically, so code that considers the two vehicles to be immune to collision, based on the track frame of reference, should be avoided. Tracks can also be point-to-point rather than looped, in which case there will be null links at the first and last node on the track. This can be problematic for the AI if they drive beyond the first/last track node, but the physical track should always contain a run-out distance beyond the finish line for cars to slow down and stop or drive through. Once off-camera, they can then be removed, so the AI should never actually reach the final AI track node.

39.2.3 Track Node Spacing

There is a trade off with the track node spacing between accuracy and development time. Shorter spacing means that the angle change between any pair of nodes is smaller, that is, the track is more linear, so any calculations are more accurate. It also means variations in curvature or other data, such as track width, are better represented. However, longer segments mean less work for designers in laying out the AI data. Furthermore, any kind of automated learning relating to the track will be more tractable if there are fewer nodes.

A good compromise is variable spacing, concentrated in highly curved regions and more widely separated on long straights. If this method is used, there is some extra overhead in the AI since distances along the track must always be summed segment by segment; they can no longer be derived as a number of nodes multiplied by a constant separation. Also, sudden changes in length between adjacent segments will lead to difficulties with the tangent/normal evaluations, which will become dominated by the longer link. These undesirable effects can be reduced by gradually transitioning the length over several segments (preferably only on straights) and by precalculating and caching the link lengths.

39.3 Calculating Track Data

There are two principle types of track data. First, the AI needs to know where it is on the track; essentially its position in the frame of reference of its nearest track nodes. This can be called the "track registration" and from this, other information such as the vehicle orientation relative to the track can be derived. This data will almost always be baked-in. The other type of information is driving hints, such as the maximum viable speed in an upcoming track section. This can also be baked-in but, since it potentially depends on vehicle specific parameters, it must often be calculated in real time. Instead, we can bake-in solid physical information on which the corner speed depends, such as the radius of curvature of the track, the racing line, or both.

39.3.1 Track Registration

Track registration involves calculating three things: the nearest consecutive pair of track nodes which define the link segment that the vehicle is on, the distance along that segment, and the perpendicular distance from a reference line, such as the link vector between nodes.

Finding the nearest node pair is a simple process of stepping through the nodes and seeking the smallest squared distance to the vehicle position. The second node in the pair depends on whether the vehicle is in front or behind the nearest node, as determined using a dot product with the node normal. Coherence may be used [Biasillo 02b] where the nodes are searched outwards from a previously known nearest node. If no previous node exists, a full search is necessary, but this can be sped up using a preliminary coarse search of every nth node. Once a node pair is identified, it is customary to store the lowest index node as the registered node, with the vehicle being in the segment between that node and the next node in the list.

The distance along the segment is then calculated using the vector between each node and the vehicle position, **R**. Because corner track segments are not simple rectangles, projecting a position vector onto the link vector between the nodes will be inaccurate—indeed, it could show the vehicle as within a segment when in fact it is not. Similarly, projecting onto the \mathbf{n}_i normal alone may be inaccurate if the vehicle is at the far end of a segment. To resolve this issue a combination of projections can be used as a ratio, as shown in Equation 39.2.

$$d = \frac{(\underline{R} - \underline{N}_1) \cdot \underline{n}_1}{(\underline{R} - \underline{N}_1) \cdot \underline{n}_1 - (\underline{R} - \underline{N}_2) \cdot \underline{n}_2}$$
(39.2)



Figure 39.2

Track registration.

This provides the longitudinal registration (d) as a normalized value between 0.0 and 1.0 along **l**. To find the lateral position, simply calculate z, and hence the length of x, as shown in Figure 39.2.

Note that the registration should be considered undefined if a vehicle is beyond the inner intersection of the two perpendiculars (point C in either Figure 39.1 or Figure 39.3) as the logical forward direction of the track is no longer consistent. Furthermore, while the registration to the track center line is useful at a strategic level, the tactical driving calculations need registration to the racing line. This means that a track node will ideally need to store not only a position along the perpendicular where the racing line passes into the segment, but also the racing line normal (tangent) and perpendicular at this point. Using a fixed lateral offset from each node combined with the primary node orientation vectors is an alternative, but this can be inaccurate as the racing line angle will not be the same as the node normal angle around corners.





39.3.2 Radius of Curvature

Calculating the radius of curvature requires an angle and a distance. Various constructions are possible, but the preferred one is based on the node using the tangent of the track curve rather than the middle of the segment, as the situation is better defined as shown in Figure 39.3. The tangent at **N** is assumed to extend to form a simple arc to **P** (the next track node) from which the length of **d** can be found. The angle is found from the dot product of the tangent and link vectors at the node. Then, using similar triangles, the radius of curvature, **r**, is shown in Equation 39.3.

$$r = d/\sin(\theta) \tag{39.3}$$

The radius of curvature of the racing line is generally more useful than that of the center of the track, as this is used to calculate maximum cornering speed. However, where the AI driver is off the racing line, some other radius is apparent and the center track radius is then useful as an upper value in approximating a revised radius.

39.3.3 Curved Nature of the Track

Although the track representation is piecewise linear, being simple straight links between nodes, it should be understood that any segment of track is in fact curved to some degree. What this means is that any reference to position within the track frame of reference is approximate. We can still interpolate information along the linear link, such as the track width, but these calculations are only accurate close to the line across the track at a node formed by its perpendicular. For this reason you should also avoid using the track frame of reference for fine-grain collision avoidance, if possible. For vehicle-to-vehicle avoidance, always use physical world coordinates, but for strategic planning, the approximate track registration is sufficient. Similarly, the radius of curvature, defined in Equation 39.3, is strictly the instantaneous value at **N**. On a real track, the tangent at **P** will differ, so the radius of curvature will vary throughout the segment.

39.4 Driving the Track

In this section we discuss the frame-by-frame driving along the main racing line. However, the AI code should also account for situations where it is not on the ideal line and be prepared to make approximate adjustments to the calculations. Indeed, because the practice of a race will never match the ideal of the racing line, one should be aware that ultimate accuracy within the driving calculations may in fact be superfluous.

39.4.1 Steering the Line Using a Look-Ahead or Runner

Track steering is based on using a *racing line* as a guide. This is defined as a series of positions associated with each track node, but can be interpolated between nodes using a lateral distance within the segment. If the steering is calculated, based on the racing line immediately in front of the car's track registration position, the AI will tend to make a lot of small corrections, which results in weaving left and right across the line. In the worst case, this can build up, become noticeable to the user, and ultimately cause the vehicle to spin out.

Instead, it is better to steer towards an aiming point some distance ahead of the vehicle, with steering based on the angle between the vehicle's current direction and the vector

between the car center and the aiming point. This tends to smooth out the steering. However, it can lead to unusual behavior; on a sharp corner, a larger look-ahead will make the AI cut across the inside of the corner. To counter this, the look-ahead distance should be related to the track curvature and/or the current speed; the exact formulation tends to be a matter of trial and error.

This technique can be further improved by making the steering aim at an actual AI object called a *runner* or *rabbit*. On every frame, the runner is updated along the racing line by a distance equal to the current speed of the vehicle, with a correction for any re-evaluation of the look-ahead distance. Using a runner can have other advantages, as it is an object that can be operated upon to affect the steering. It is also possible to drive parallel to the racing line, but at a small offset distance, by using interpolation of the scaled node perpendicular vectors through the segment.

39.4.2 Corner Speed and Braking Distance

The maximum cornering speed at any node is based on a simple function of the local radius of curvature (*r*), the vehicle mass (*m*), and its velocity (*v*) compared to the limit of grip force available from the tires (G_{max}), as shown in Equation 39.4.

$$\frac{mv^2}{r} \le G_{max} \tag{39.4}$$

However, there are other factors—the vehicle suspension, weight distribution, down force, and even steering lock can play a roll. For that reason a preferred method is to request the maximum speed from the vehicle physics interface, based on a value of radius. This request might use Equation 39.4 with modifications or it might use a lookup table.

In an ideal corner, the radius of curvature would change from infinity to a fixed value for the direction of the bend and all braking would happen in the preceding straight. However, real corners are not like that, and the best drivers can brake into the start of a bend before committing fully to steering.

A good solution is to predict the vehicle's speed under braking and compare this to the maximum viable speed at each future track node on the racing line. If the predicted speed with full braking is greater than the maximum speed at a node, then the driver must brake. If the speed prediction reaches zero or less without exceeding a node speed, the car is safe to continue accelerating. Recalculate this every few frames in the tactical layer and, if possible, factor in the braking rate at each node (which will not be 100% if the vehicle also needs to steer). This can be expensive computationally, so alternative methods include baking in the maximum speeds at each node or having a designer define the speeds in an editing tool. A comparison of "distance to the speed hint" with "braking distance to that speed" can then be used as a braking initiator.

39.4.3 Wall Avoidance

Wall avoidance can be considered as either a medium or short-range problem. The medium range problem is to extrapolate the future vehicle position and take corrective action if that position is outside the track [Biasillo 02b]. However, there are some caveats. First, if a vehicle is successfully steering around a bend, then the extrapolation should account for the turn rate; otherwise, a vehicle on the outside of a bend may conflict with

a wall without reason. Also, any turning should be "under control," that is, within grip limits. Furthermore, the corrective action should be based on available spare grip; if extra steering would force the vehicle beyond the grip limit then braking might be preferred (even though this still may not help, it is more realistic as the inevitable wall impact should be at a lower speed). If the extrapolation is in conflict with a non-hard track boundary, the reaction might be less severe, thus allowing the car to run off and recover rather than risk a spin. If a runner object is used, this can also be validated to be within the track, and this can be used alongside extrapolation.

However, it is also a good idea to look directly sideways as well as ahead; that is, make small corrections based on the proximity to any wall alongside the car in order to maintain a safe buffer distance. This can be more effective in tight tracks where the vehicle must travel close to walls, as extrapolation is an approximation and can be overly conservative. This kind of short-range adjustment works particularly well with a PID-based control layer [Melder and Tomlinson 13], when mixed into the main steering target as a correction.

39.5 Racing Lines

What is the optimal racing line? Strictly speaking, it is the resultant path driven by a vehicle that is optimized to complete the track in the minimum amount of time. This is not necessarily the same thing as the fastest line in any local region, as it may be more important to set up the entry for a future corner than pass through the current corner at top speed in complex sections.

Even where this line can be defined, the use of a runner usually means the actual path steered by a vehicle can be different from the racing line guide. This apparent disconnection between the stored racing line and emergent path should be borne in mind when editing the data; the most important thing is the final resultant driving performance, not necessarily what the stored racing line data looks like. In code, the racing line should be stored as a width offset at each track node. This can easily be converted to a real-world position using the reference node position and the normalized perpendicular.

Generation of the racing line can be a complex process and space here is insufficient to explore all of the issues and techniques; however, here are some general points. Asking designers to include an initial estimate of the racing line in the track editing tool, manually moving points and measuring the lap times, is a time consuming and tedious process. Another method is to record a racing line as driven in-game by an experienced user. This does work, but the recorded route can be prone to kinks and weaker areas. A human does not steer absolutely smoothly and almost always tends to include increments or corrections in the steering graph. Any kinks in the recorded line will lead to an overestimate of the radius of curvature in that track section and hence a lower speed than may actually be possible. Furthermore, it is unlikely that any one lap will be "perfect"; even the best player will make one or more small mistakes, which will invalidate some sections. This is not to say the recording method is to be avoided, but it should be backed up with some postprocessing to average or patch over several laps and generally smooth out kinks. No matter which of the above approaches is used, remember that the AI uses the racing line as a guide only and therefore even the most "perfect" stored racing line will not result in the most perfect driven line.

By far the best approach is some form of automated learning [Tomlinson and Melder 13]. Not only does this speed up the optimization process, but it can also nullify artifacts within the tactical driving calculations as the learning is based on the output results and not the input racing line format. However, bear in mind that the nodes in the line are correlated, that is, if one is moved, those around it should move in a coherent fashion. For example, if the learning algorithm moves a node 0.5m laterally (which is the only degree of freedom if the racing line is constrained to be on a track node perpendicular), then forward and backward nodes should be moved a smaller amount based upon a function related to their distance from the primary perturbation node. Also, make sure the metric (lap time) is averaged over several laps; in particular, the AI should have half a lap run-up on a looping track, rather than a standing start that will not be representative. Randomization in the AI should be switched off, or rather set to fixed mid-values. Even where learning is used, the generated racing line only really applies to that particular car's capability, as differences in braking or cornering grip will subtly affect the line. It is impractical to record a line for each and every vehicle, but some averaging with different car types or using two or three lines for different car groups should be considered.

39.6 Alternate Lines and Other Tactical Information

In a competitive race, the actual racing line may also account for strategic considerations, for example, making it more difficult for an opponent to overtake where small losses in time or speed are tolerable. This means there is conceptually a *best defending* line, which might be stored in the same way as the racing line. Similarly, one or more overtaking lines might be stored. Typically, there are multiple possible lines in a corner, including pushing up the inside of a slightly wide opponent or entering a corner wide in order to gain speed earlier on the exit ("the undercut"). Of course, all these lines can be a lot of additional work to optimize and additional data to store, so emergent solutions may be preferred that use lateral offsets from the single racing line. Markers at track nodes can still be useful though, for example, to indicate to the AI what overtaking entries might be available at a future corner.

39.7 Using Splines

In the discussion of track segments, it was noted that the piecewise linear approximation is only accurate near the nodes. This can be improved by using a spline between the nodes, so that the curve of the track center, edge, or racing line is fully defined throughout the segment. Various types of splines are available, such as Catmull–Rom or Bézier. However, splines are a mixed benefit. Registration is not direct and must generally be done using an iterative process such as bisection. Similarly, local tangents must be estimated using a finite difference.

39.8 Conclusion

We have discussed a highly detailed representation of a race track and how to use that track in real time to guide AI vehicles at their limit of speed and performance. The level of detail and complexity here is probably only appropriate for simulation type racing games, but the methods described can be simplified for more arcade style games. It has also been

suggested that the piecewise linear representation can be improved upon by using splines. Further areas to explore include learning techniques to improve the racing line and more subtle driving considerations such as the effect of height variations, in particular crests, which can cause a vehicle to temporarily lose downward force and hence grip.

References

- [Biasillo 02a] G. Biasillo. "Representing a race track for AI." In *AI Game Programming Wisdom*, edited by Steve Rabin. Hingham, MA: Charles River Media, 2002, pp. 439–443.
- [Biasillo 02b] G. Biasillo. "Racing AI logic." In *AI Game Programming Wisdom*, edited by Steve Rabin. Hingham, MA: Charles River Media, 2002, pp. 444–454.
- [Melder and Tomlinson 13] N. Melder and S. Tomlinson. "Racing vehicle control systems using PID controllers." In *Game AI Pro*, edited by Steve Rabin. Boca Raton, FL: CRC Press, 2013.
- [Tomlinson and Melder 13] S. Tomlinson and N. Melder. "An architecture overview for AI in racing games." In *Game AI Pro*, edited by Steve Rabin. Boca Raton, FL: CRC Press, 2013.