Simulating trains on railway tracks

Pijke Hochstenbach

August 2022

Summary

This article describes a way to simulate moving trains along a railway track, while using as little CPU resources as possible and appearing realistic.

Tracks are represented as lines or arcs, which are connected to a single other track at both ends. Switches have three or more tracks: a leading track which is connected to a single branch, and branches which all connect back to the leading track.

All tracks have a fixed length, points can be placed anywhere along the track as "measures".

Trains are represented by a set of bogies, with a fixed distance between them. Each bogie is a measure along a track. This means when updating the position of a bogie, it only has to work in 1 dimension.

Trains move by moving all it's bogies with the same speed. When any bogie leaves a track, it moves to the next track and continues.

If a train encounters an end of a track without another one connected to it, it will derail. This will also happen if different bogies of the same train start traveling on tracks the first bogie did not travel on. Trains can also collide with each other when the occupy the same portion of a track.

At each point along this simulation, it loses tiny bits of realism, but these are negligible and at every step the simulation has ways to correct itself.

1 Introduction

This article came to be from a research project to find options for a train dispatching simulation, simulating the old signalling systems of (Dutch) railways. In order to simulate these signalling systems with any level of accuracy, it is vital to provide a realistic simulation for train movements along railway tracks.

This article focuses on the interface between trains and railway tracks. The physics, traction, acceleration and deceleration of these trains are outside the scope of this article.

2 Real world characteristics

In real life, a train drives to whatever direction the track beneath it is going. The route of the train is determined by the geometry of these track, and the position of the railroad switches. Tracks have a fixed position and length, with switches being the only dynamic parts if the track.

Both the trains and track can be subdivided into their components. Each train consists of 1 or more carriages. These carriages (realistically) have 2 or more bogies. These are the "points" around which each carriage turns. The distance between the bogies within a single carriage is fixed. The distance between bogies from different carriages can differ in reality, but can be set to an average distance.

The railroad consists of different tracks. These can be straight or curved. A portion of the railroad consists of 1 or more tracks. A switch has 3 or more tracks: 1 for the leading portion, and 1 for each direction it branches.

3 Simulation design

The simulated world should be a simplified version of the real world, which mimics the characteristics as close as possible. Because this is intended for a railway signalling simulation, it's more important the train simulation consumes as little CPU resources as possible, so some trade-offs in realism will be made. Our train simulation has the following requirements:

1. It must be efficient for computers to calculate
2. To the end user, it must look as realistic as possible. This means that they shouldn't be able to notice any shortcuts the simulation takes by looking at a map view or their signalling panel

3.1 Track design

In the simulation, tracks can be simulated the same was as they exist in real life. A track is either of type line or arc. Real world tracks have transition arcs too, but these will be ignored. For both lines and arcs, all required attributes (for example position and length) can be calculated fast.

The geometry of the track is calculated using 2D coordinates. A possible simplification is to use a Cartesian coordinate system using meters as the x and y values, instead of a spatial reference system like WGS84.

Each track has a fixed length. We can make use of this for defining any point related to railway tracks. A point along a track is called a "measure". It's unit is distance, which means the distance along the track from it's starting point. A measure can always be calculated into a distinct 2D coordinate if need be, for example when rendering the railroad, trains and any other elements related to the tracks.

All tracks are connected to 1 other track at the start, and 1 other track at the end. This also applies to switches. The start of both branches connect to the same leading track, but the leading track only connects to at most 1 of the branches. It can also connect to none of the branches, if the switch is not in it's final position.

If the simulation encounters a train reaching the end of a track without another track connected to it, this can be counted as a derailment, and it's up to the specifics of the actual simulation on how to handle that.

3.2 Train design

Following a single bogie along a railway is pretty straight forward this way. The bogie has a track on which it starts, including a measure for the exact position along that track. When it starts to travel, at some point it will reach one of the ends of the track (it's measure will drop below 0 or become bigger than the track length). At this moment, it finds the track which is connected to that end of the current track, and continues it's journey on the new track. Rinse and repeat.

Following this logic, a train would be a set of bogies connected together. What options there are for simulating a train are discussed in the next chapter.

4 Simulating trains

For simulating trains, the least CPU intensive way is to split the train into it's carriages, split those into their bogies and represent these bogies as measures along a track. This way, trains are simulated in a 1D environment, without the need of calculating 2D coordinates for each position update.

4.1 The shrinkage problem

But if we assume a train can be simulated as a set of bogies, and these bogies are purely represented by a measure on a track, we run into a problem. Representing the bogies by a measure on a track means the distance between these bogies is constant along the center line of the track. But when the train is in a curve, this doesn't ensure the straight distance between these bogies is always the same. This is caused because the distance between the bogies along the track is greater than the straight line distance between them. Figure 1 shows an exaggerated version of this. In this figure, distance a (the distance between the bogies along the center line of the track) is constant, which means distance b (the actual length of the carriage) has to shrink.

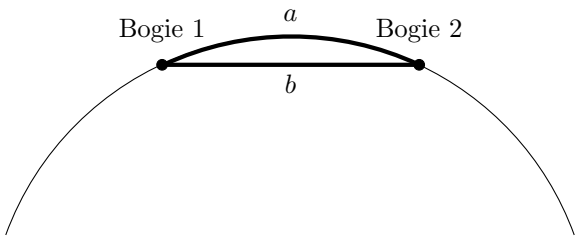

Figure 1: Railway carriage on a track with a narrow curve

For a real carriage, the straight line distance between it's bogies is constant. When they go through curves, both bogies can have a different speed to account for this phenomenon and keep the distance between it's bogies constant.

Simulating carriages this way means the carriage will become shorter when going through curves, and correct itself again when back on straight track. This doesn't have to be a problem, as long as this shrinkage is not noticeable in the simulation.

To determine if this is acceptable, the worst case scenario for a Dutch railway was calculated. Normal curves are at least 2000m, but inside railway stations, when going through switches, they can become as little as 195m. A normal carriage has a maximum length of 25m. The distance between the bogies will be less, because they usually are not at the far ends of a carriage, but this calculation didn't take that into account, as that will only make the effect less pronounced. With these values, the result is as follows:

$$\begin{aligned} b &= 2r\sin(\frac{a}{2r}) \\ &= 2 \cdot 195 \cdot \sin(\frac{25}{2 \cdot 195}) \\ &= 24.983m \end{aligned} \tag{1}$$

This means that in the worst case scenario for a Dutch railway, the carriage will be 1.7cm shorter than it should be. The bogies on the track should be further apart to compensate for the curve. When calculating a carriage or train from the front and working backwards, this will have no consequences for when a carriage enters a new track, but if the track is split in the middle of such a curve, this means the carriage will leave the last track sooner.

Assuming the carriage is travelling at $40km/h$ through this curve, it will leave the track 0.0016 seconds earlier than it should. If this would be a long passenger train, and this error would stack up over all carriages, a train of 12 carriages would leave the last track 0.02 seconds earlier than it should.

In the simulation, the position of each train will be updated periodically. The tick rate (the number of times the train position will be updated every second) for the simulation has not been determined yet. The higher this tick rate, the more sensitive it becomes to little anomalies like this shrinkage. As

of writing, the assumption is the tick rate for the simulation will be no bigger than $25t/s$. With this tick rate, there will be 0.04 seconds between each position update. Both previously calculated values are within this time window. That means for this scenario, the limiting factor on how fast other components of the simulation will receive information from the trains is the tick rate, and not the shrinkage of carriages.

The advantage of simulating trains this way is that to the simulation, the train has a constant length (according to it's 1D simulation) and all bogies and carriages have the same speed. This simplifies the interaction with these trains greatly. Apart from that, it should be one of the most CPU efficient ways to calculate, because simulating it in more complex ways requires calculating 2D coordinates for the carriages and/or bogies, and simpler simulations start taking shortcuts that will be noticeable to end users. Finally, simulating this way gives a lot of options to the simulation in the way of exchanging information between the train and it's tracks, and rendering trains for end users.

4.2 Navigating trains

Now that trains can be placed onto the track as a set of bogies, the next step is to make the trains move. There are two options to achieve this.

The first option is to only move the first bogie. Every tick, this bogie is moved forward the correct distance. After that, all other bogies are placed behind it, separated by the correct distance. It works it's way backwards from the first bogie. If it reaches the end of a track, it places the next bogie on the next track, and so on. This has the advantage all bogies will always be the exact same distance from each other along the center line of the track. The disadvantage is, it assumes the track behind the first bogie is the track all parts of the train will use, and that working back from the first bogie will lead you to the correct tracks.

Nearly always, that will be the case, but in certain edge cases this will lead to unexpected behaviour. If the simulation would use dynamically generated tracks, this can become unpredictable. Secondly, if going back onto the switch leads a different way or a railway switch would change position while the train is passing, the train would follow the wrong path.

Therefor, a better way is to move each bogie individually. Using this method, once all bogies are placed, every tick they all will be moved forward at the exact same speed. When any bogie reaches the end of a track, it will go to the next track. Because they are all moving at the same speed, they will stay at the exact same distance from each other along the center line of the track, even though each bogie moves as if it's on it's own. The advantage is all bogies will follow the track from their current position, like they would in real life. If a railway switch would change position while the train is passing, all the bogies that come after will go a different route or will derail of the switch is not in it's final position.

Using this method to simulate train movement, error could stack up over time due to rounding errors, floating point errors, etc... This should be on the order of millimeters, but can be solved if needed. Periodically, when all bogies are on the same track, the simulation can re-calculate the position of each bogie by using the method described before: update the position of the first bogie, and work it's way backwards placing all other bogies at the correct distance. When all are on the same track, this can cause no unexpected behaviour.

5 Detecting problems

5.1 Derailment

Just like real life trains, simulated trains should be able to derail. In the simulation, a derailment can be simplified into two possible causes:

1. A bogie leaves a track, and there is no track connected to that side
2. Bogies on a train start taking different paths, where the distance between the bogies of a carriage starts to increase

The first item is trivial to calculate, this will be noticed immediately by the simulation when updating the position of a bogie and not finding a track to go to. It's up to the simulation how it handles this event.

The second item is more CPU intensive to calculate, because it will need to calculate the 2D coordinates for each bogie. However, this only has to be calculated when a bogie that is not the front one goes onto a track that the front bogie has *not* accessed.

There are more possible causes for train derailment in real life, but these will either be ignored (for example, G-force induced derailment when going through a curve too fast) or can be simulated using the first cause (for example, a switch that malfunctions).

5.2 Collisions

In order to defect if trains have collided with each other, the simulation should keep track which train occupies which portions of track. For every train, this would be a list of tracks with two measures each: from and to. If a train or carriage spans over multiple tracks, these measures will go to the end of this track (either to 0 or to the track length). The simulation can compare the results from each train to defect if they overlap. If they do, it can calculate the impact of the collision by the speed and 2D direction of each train. It's up to the simulation how to handle this event.

The main advantage of this method is that it doesn't need to use 2D calculations as long as no collision is detected. To further optimize, this check doesn't have to occur every tick.

The disadvantage is that very short tracks can go unnoticed. If a track is shorter than the maximum distance between bogies, it can go unnoticed. But this will only be for brief moments, and all other tracks the train is present on will be taken into account. It is also possible to remedy this problem with a slightly more complicated way of calculating what tracks the train is on. The simulation can find out what tracks are in between the first and last bogie using a route finding algorithm.

6 To close off

While this article outlined a way to simulate trains along a railroad, simulating them doesn't stop there. To make the movement itself look realistic, their traction and breaking systems need to be simulated. Furthermore, an AI needs to drive the train according to the signs and signals along the track. That AI also needs to be able to find their way on a shunting ground, communicate to the dispatcher, and much more.

Acknowledgement

I would like to thank Amber Albers for her help with calculating the impact of the shrinkage problem. Her calculations ultimately determined the impact of this problem to be acceptable.