

Vehicle Tools QuickStart Guide

Intro

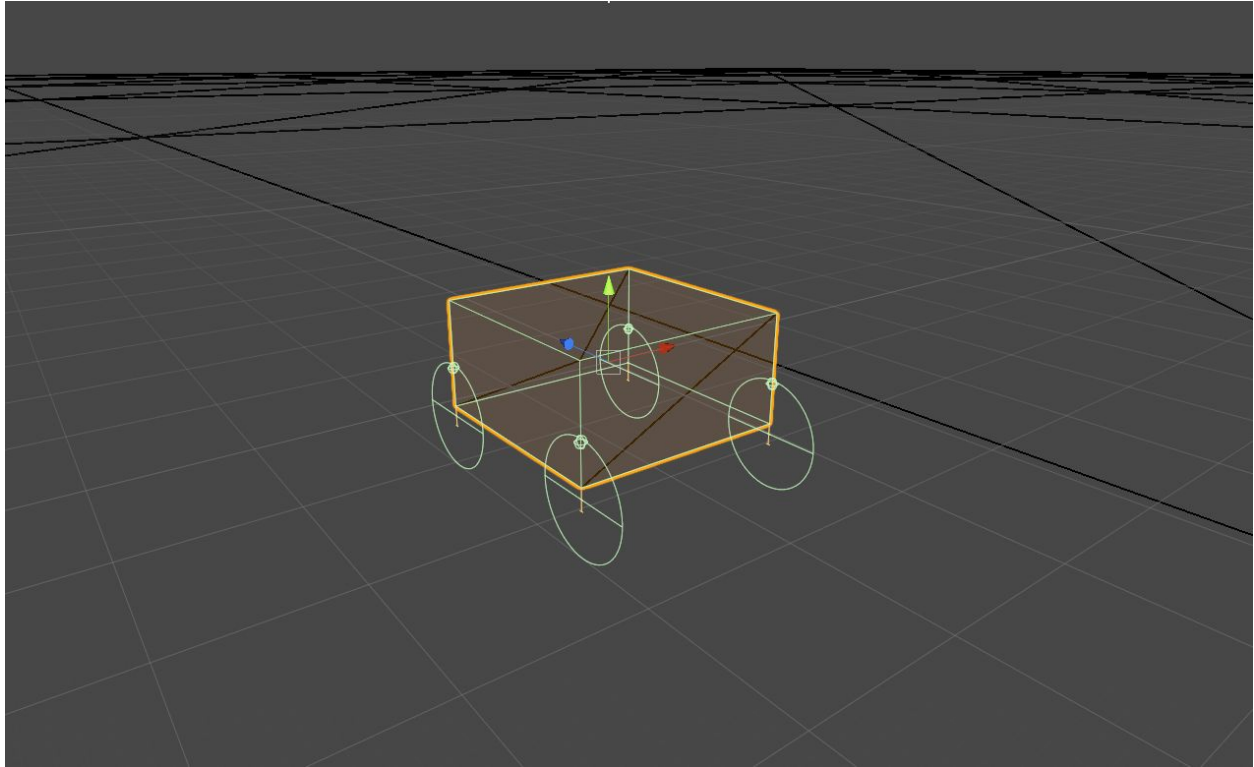
A default WheelCollider is set up for a standard use case: a 1.5 tonne car with 4 wheels, where all wheels are perfectly symmetrical relative to the car's centre of mass. If you have a different use case, such as a vehicle with more wheels, you risk violating implicit assumptions in the physics simulation code and may see unpredictable results.

In addition to this, the meaning of the explicit spring stiffness and damper values in the WheelCollider component are somewhat unclear. It is not obvious how to tweak these values to simulate different kinds of suspension, such as a stiff sports suspension or one of a family car.

Vehicle Tools solves these problems. This package provides you with a wizard to create a drivable skeleton of a car with any amount of wheels, and with suspension settings that are very easy to tweak. It also provides a tool for debugging problems with friction calculations, to help with debugging acceleration and cornering.

Creating a drivable car

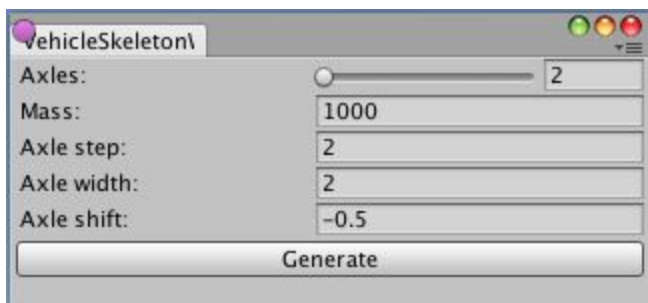
- In Unity, open the Vehicle Tools project
- Create an empty Scene
- Create a simple plane to use as a surface to drive on
- Set the scale of the plane to 100, 1, 100
- From the top menu, select **Vehicles > Create skeleton**
- A wizard will appear, where you can set the values for your vehicle
- For now, simply leave the default values as they are
- Select **Generate** to generate a vehicle



The Vehicle Skeleton Wizard

The Create skeleton wizard allows you to configure your drivable car.

Unless otherwise specified, units of measurement are [SI units](#).

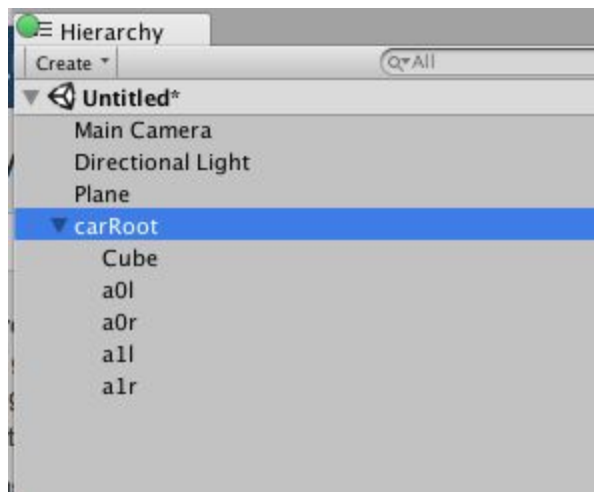


Axes	The number of axles in the car
Mass	The mass of the car

Axle step	The distance between axles
Axle width	The width of the car
Axle shift	The distance axles are offset above the car's center along the local Y axis

Examining a generated drivable car

The car generated by the wizard is a fully functional and drivable skeleton, based on the settings chosen in the wizard.



If you examine the hierarchy of a generated car, you can see that it consists of a parent GameObject named *carRoot* and child GameObjects that represent the wheels and body of the car.

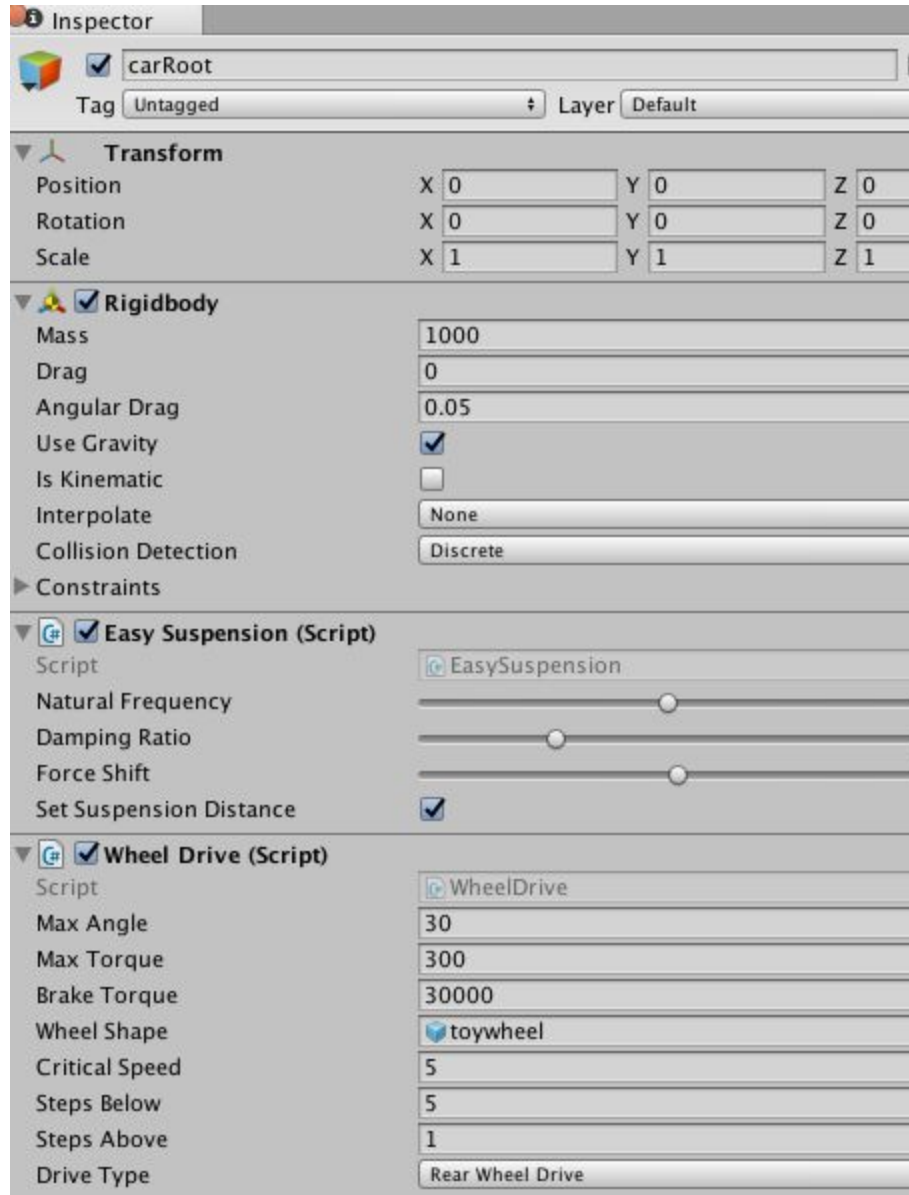
The body of the car

By default, the GameObject called *Cube* serves as both the graphics and the collision geometry of the generated car. Any WheelCollider-based car requires a Collider to represent the body of the car. This Collider is used by the physics engine to compute the distribution of masses over the wheels.

Cube can be replaced with non-placeholder graphics and collision geometry as needed.

The carRoot GameObject

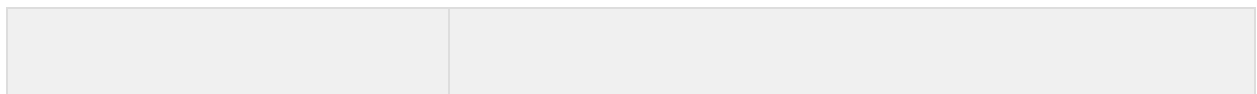
The GameObject called *carRoot* is the root of the generated car, and has the following components: an EasySuspension script, a WheelDrive script and a Rigidbody.



EasySuspension

The EasySuspension script is a utility script that updates suspension settings across all child WheelCollider components. Simply enter the mass, natural frequency and damping ratio for the wheels, and the EasySuspension script will take care of updating each WheelCollider component with these settings.

Unless otherwise specified, units of measurement are [SI units](#).



Natural Frequency	<p>The natural frequency of the suspension springs. Natural frequency might be thought of as a mass-independent analogue for stiffness.</p> <p>Typical range [4..20]. A family car might have springs with the natural frequency of about 10.</p>
Damping Ratio	<p>The damping ratio of the suspension springs. This parameter sets how fast the oscillations in the suspension comes to rest.</p> <p>Typical range [0..1]. A family car might have springs with the damping ratio of about 0.8.</p>
Force Shift	<p>The distance to the point where the tyre forces are applied, starting from the center of mass of the vehicle, along the local Y axis.</p>
Set Suspension Distance	<p>Whether or not to adjust the travel distance of the suspension springs. This setting helps to avoid having physically incorrect configurations that still apply force at maximum suspension elongation.</p>

WheelDrive

The WheelDrive component is a minimalistic vehicle controller. This script allows for quick testing out of the box, and provides a good start for your own customized vehicle controller.

Unless otherwise specified, units of measurement are [SI units](#).

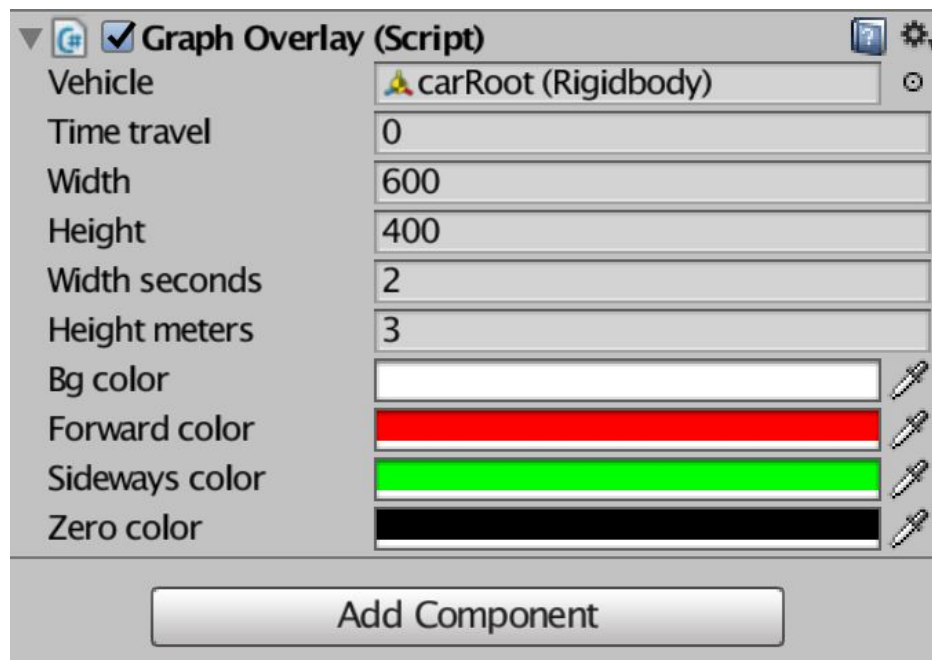
Max Angle	The maximum steering angle of the front wheels.
Max Torque	The maximum engine torque applied to the driving wheels.
Brake Torque	The braking torque.
Wheel Shape	If a prefab of a wheel is assigned here, wheels will be instantiated automatically when we enter Play Mode. The provided <i>toywheel</i> prefab can be used as an example.

Critical Speed, Steps Below, Steps Above	<p>The parameters of the vehicle substepping algorithm.</p> <p>If the speed of the vehicle is below the speed specified in Critical Speed, the vehicle integrator will perform the number of substeps specified in Steps Below.</p> <p>If the speed of the vehicle is at or above the speed specified in Critical Speed, the vehicle integrator will perform the number of substeps specified in Steps Above.</p>
Drive Type	<p>Choose the drive type from front wheel drive, rear wheel drive or all wheel drive. Front wheels are those having their local z coordinate positive.</p>

Debugging friction with the GraphOverlay component

Issues with friction are the most frequent cause of simulated vehicles failing to accelerate and corner correctly. Most often, this happens because the physics code is not performing enough substeps - that is to say, it is not making frequent enough calculations.

The GraphOverlay component helps us to debug the friction calculations of our generated drivable cars.



With the information from this component we can decide whether we need to adjust the number of substeps.

An introduction to the slip-based friction in Unity

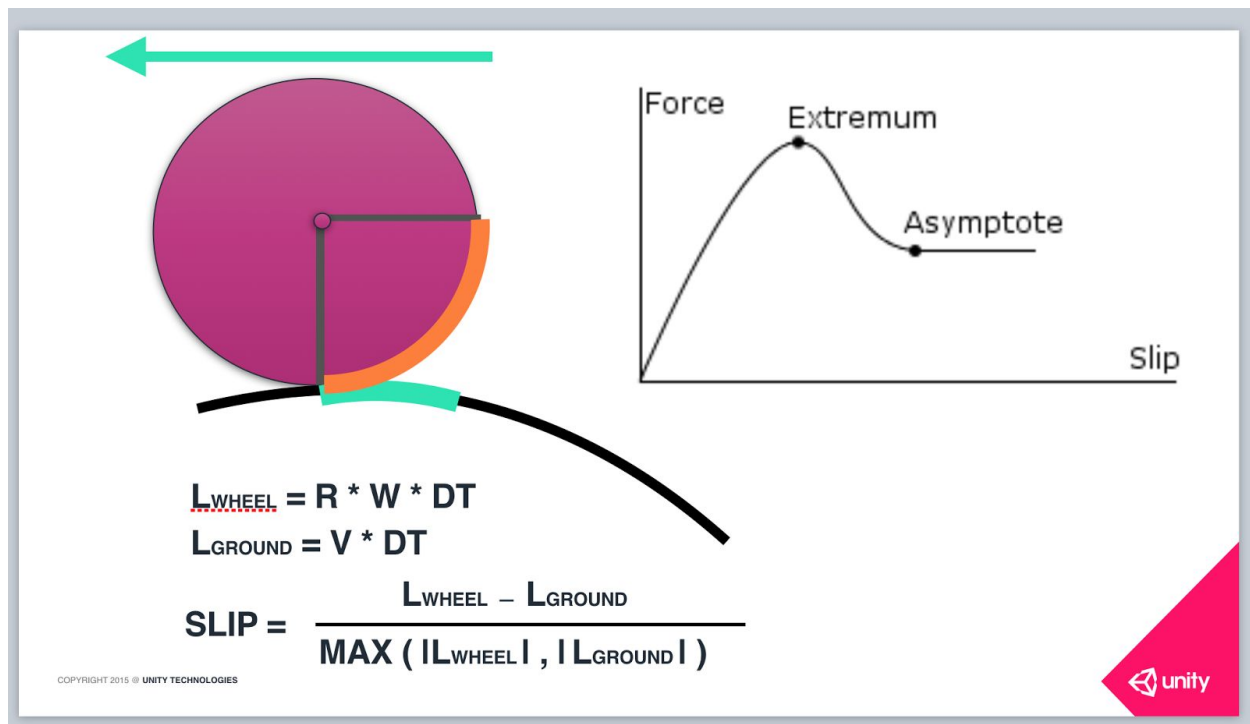
Unity uses the slip-based friction model to calculate how much longitudinal and lateral force should be applied to a vehicle that uses WheelColliders. Those forces determine how fast the vehicle reacts to acceleration as well as steering. In order to compute the forces, the physics engine works out the longitudinal and lateral slip values for each wheel first. Thus, all the computational inaccuracies that occur during the slip computation, have a direct impact on the forces produced by wheels, and, in the end, on the handling of your vehicle. The slips are computed from values like the angular speed of the wheel that are obtained an iterative integration process, and are sensitive to the amount of simulation sub-steps, and the natural frequency of the spring attached to the wheel. If it happens that there are not enough sub-steps for this given natural frequency of the spring, then one can observe a wheel oscillating instead of rotating smoothly. This is clearly visible on the slip graph shown by the GraphOverlay component, as explained below.

The slip-based friction model simulates a simplified version of the processes that take place in a real tyre. When a wheel starts turning, the rim of the wheel slips relative to the ground as the rubber of the tyre stretches and produces momentum. The more torque one applies to the wheel, the faster it rotates and the bigger slip it produces. Bigger slip stretches the rubber even more, producing more grip. This continues until the extremum slip is reached, after which the tyre loses the stiff contact with ground at the contact patch. At this point, the rubber stretch gets relaxed and thus produces less momentum. After the slip becomes bigger than the asymptote slip, it no longer matters how high the slip is: the rubber of the tyre reaches the state where the remaining slight contact with the surface would no longer let it compress further, and thus the momentum produced by the wheel remains constant no matter how much torque is applied.

As an example, let's look at how the longitudinal slip works.

The picture below shows a wheel rolling to the left. Once the angular momentum (acceleration due to the engine; shown in cyan) is applied, the wheel starts spinning, and slips a little as the rubber gets compressed. Because of that, the distance along the surface travelled by the centre of the wheel will be different from the distance the tyre travelled along the surface (shown in orange). The physics engine computes the longitudinal slip as the difference between those two distances over the one that has the biggest absolute value. The result is normalised to be in $[-1;1]$.

The lateral slip is computed in a similar way.



How to use the GraphOverlay component

- Add the GraphOverlay to any GameObject in your Scene
- Drag the carRoot GameObject of the car that you wish to debug into the Vehicle field of the GraphOverlay component
- Enter Play Mode and observe the graph of wheel slips in the Game view
- Accelerate the car (for example, from 0 to 100 km/h) and observe the changes in the graph

Interpreting the graph generated by the GraphOverlay component

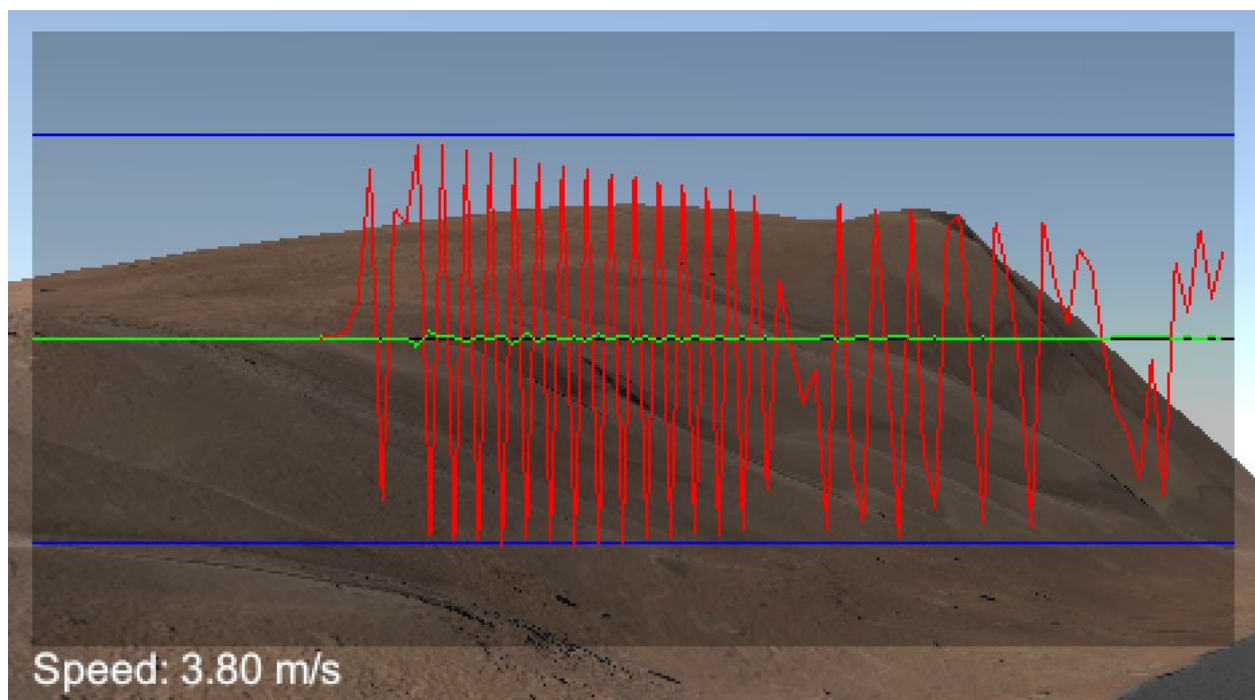


The x axis of the graph represents time and the y axis represents the slip value. The red and the green lines represent the longitudinal and lateral friction over time, respectively.

The blue lines on the graph represent the corridor $[-1, 1]$. The white line at the centre corresponds to the slip of 0.

Ideally, both curves should be smooth and fall close to 0, right where the value doesn't yet reach the extremum.

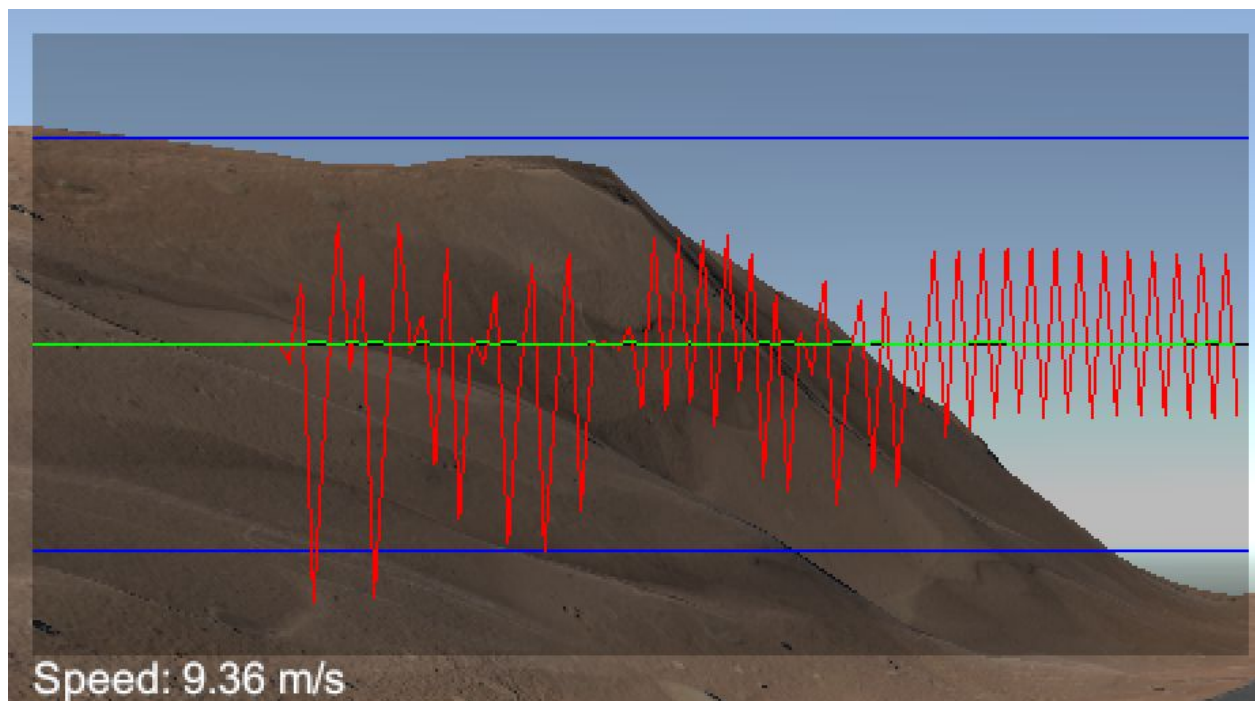
Consider the following graph, produced by a test car at low speed.



At these low speeds, the slip (the red line) repeatedly goes above and below zero (the green line). The oscillations are a direct indication that not enough substeps are taking place. This means that the code fails to work out the longitudinal slip, which results in force being applied in opposing directions. This results in the vehicle struggling to accelerate.

In this case, the Critical Speed value for the WheelDrive component is set to the default value of 5 m/s. When the speed of the vehicle is below the Critical Speed value, the number of substeps is determined by the Steps Below value. As this problem occurs speeds below the Critical Speed, increasing the Steps Below value of the WheelDrive component is advised.

Now consider the following graph, taken from a test car at a high speed.

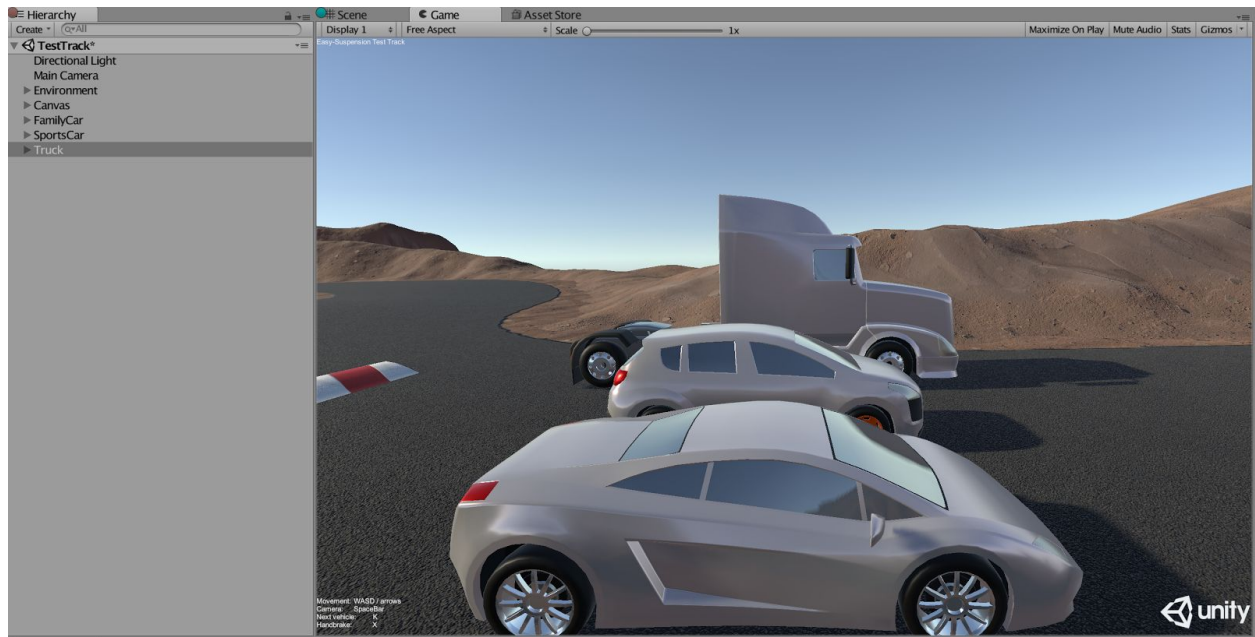


In this case, the slip does not jitter at above and below 0 moderate speeds. However, when the vehicle reaches a speed of 5m/s the slip began to jitter above and below 0.

Again, the Critical Speed value for the WheelDriver component is set to the default value of 5 m/s. Once the speed of the vehicle reaches the Critical Speed, the number of substeps is determined by the Steps Above value. In this case, increasing the value of Steps Above is advisable.

Sample Scene included with the package

This package contains a Scene named **TestTrack**. This Scene contains a race circuit and 3 pre-configured vehicles to try out. The Scene also includes a working example of the GraphOverlay, and code to switch between vehicles at run time.



Further reading

Learn more about why natural frequency and damping ratio is better than having to specify the raw spring values here: <https://youtu.be/WGvuP6vV6j4?t=17m>

Having a working skeleton of a car is just one step from having a full vehicle working reasonably well. Learn about the steps involved in adding dynamics to any graphics model over here: <https://www.youtube.com/watch?v=xQcJAa6Ooa4>

More in-depth information about how the simulation code works internally: <https://www.docdroid.net/vd5x/wc-info.pdf.html>