Arcade Vehicle System

Table of Contents

1. Description and general information

This arcade vehicle system is a UE4 plug-in that allows for simple, quick and painless vehicle creation. It implements UE4 physics in a way that allows predicting the networking accurately enough, so we could create reliable and smooth networking for the vehicles. Vehicles driving style is mostly arcade-like, although it uses physics to do calculations, it is pretty flexible and allows to tweak the system to your needs. It introduces a lot of curves that allow user to specify acceleration, braking, turning and more, all depending on the specific speeds.

The plug-in unlike other vehicle plug-ins, implements vehicle movement as pawn movement component. This allows to have total flexibility and allows to implement vehicle system easily even inside existing projects, as we do not need to destroy hierarchy of the blueprints, and can just add the vehicle movement component, and use it that way.

However, there is also vehicle C++ class, that users can derive from if they want to. It provides everything that is needed for the vehicles to work. There are also vehicle example blueprints that users can either derive from, or just take a look how they are implemented.

This plug-in does not limit users to how many wheels there are. There is separate setting for each suspension spring, so all springs can have different height and different stiffness or other settings. There is no limit to how many wheels/springs there are.

All vehicle settings are encapsulated by the separate structures in C++, which makes it easy to copy values from each settings category, or even all of them at once and paste it into other vehicle, or send to a friend or co-worker.

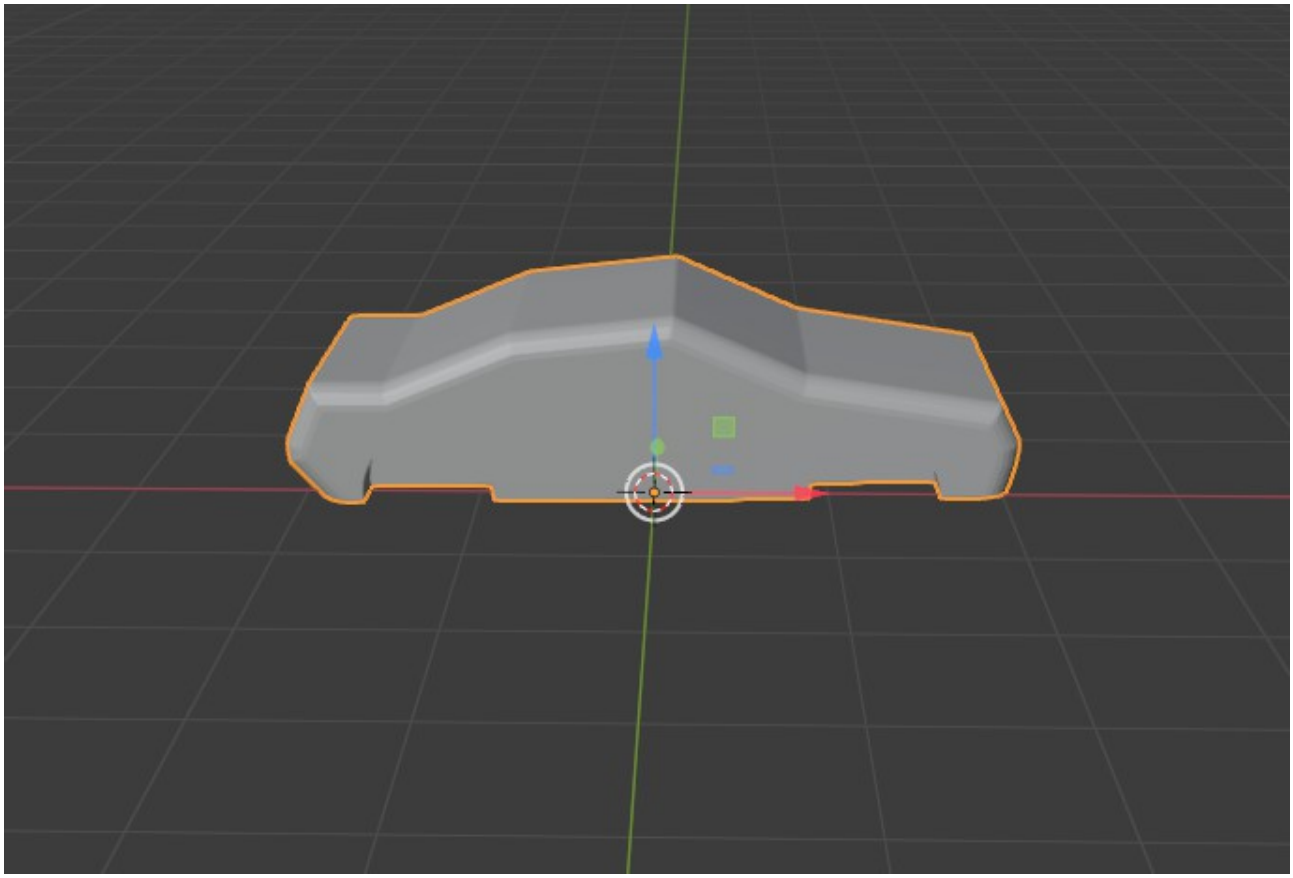
NETWORKING.
I just wanted to quickly tell about the networking. The plugin has networking implemented out of the box. However, the networking is Client-Authoritative. That is not an issue, because you can still create functionality on the server side that will allow you to verify that data for any sort of cheating.

2. Vehicle model preparation

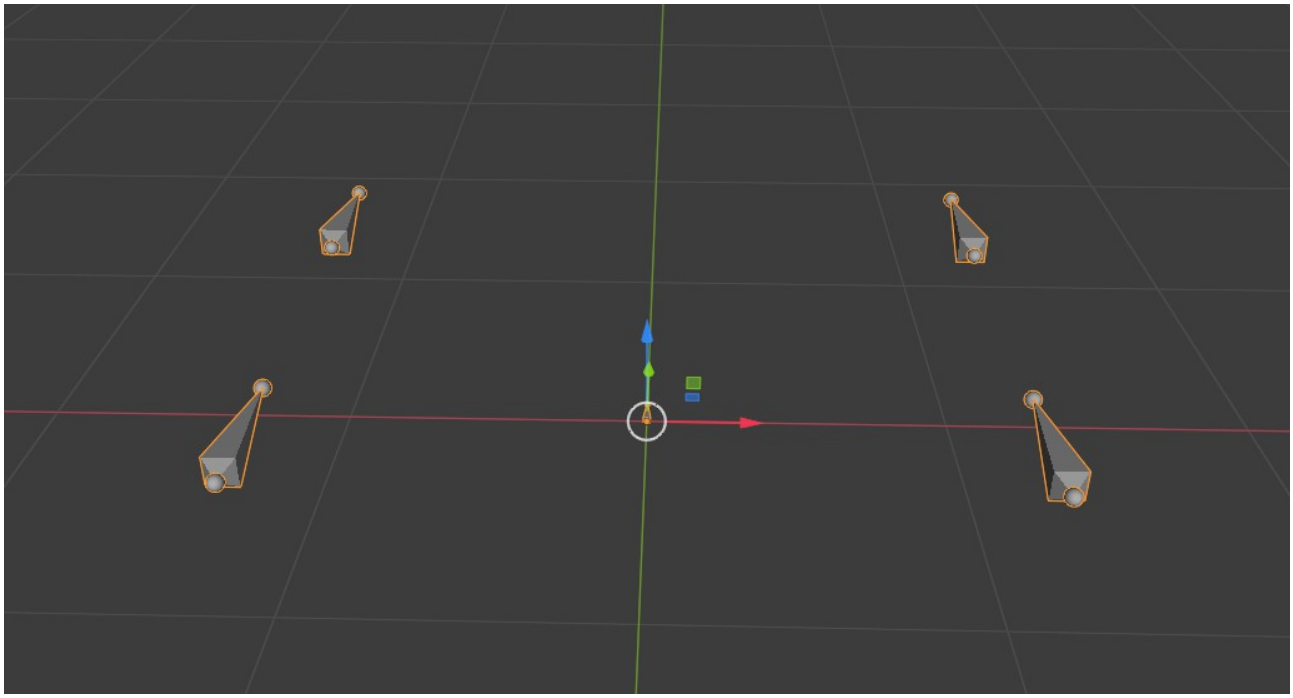
2.1. Rig creation in Blender.

I am not going to go through creation of the 3D meshes in the blender. This would be far beyond the scope of this document. However, I will provide simple steps and show how to get your vehicles to work with the plug-in. Vehicles utilize skeletal meshes. We do not use static meshes at the moment, however I am thinking of adding support for them too.

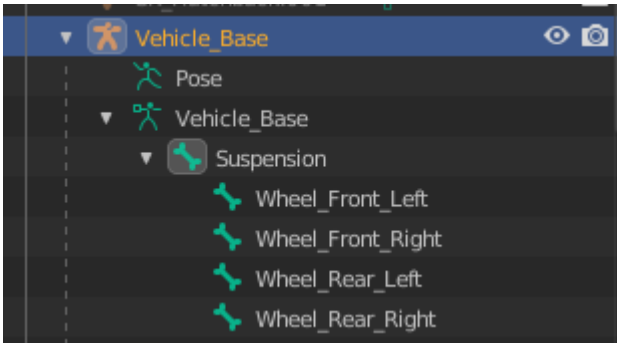
Firstly create your vehicle 3D mesh. Make sure the mesh lays on the positive X axis. Here's the example of the vehicle that I have:



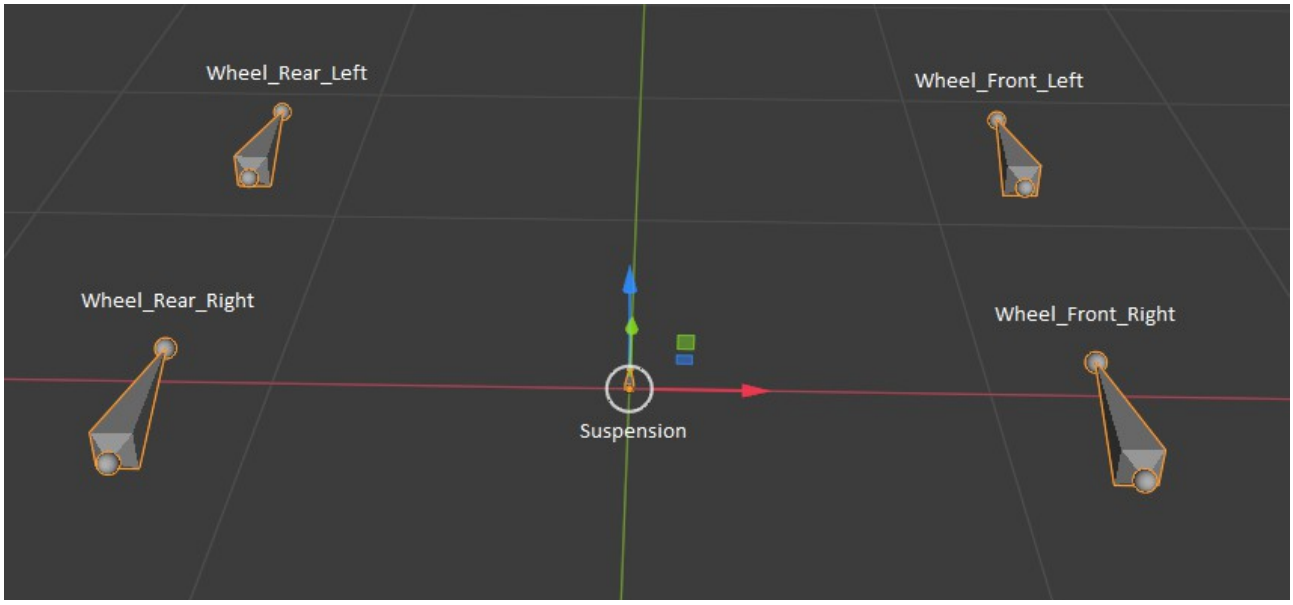
Once your vehicle is done, the next step is to create the skeleton for it, or the armature as Blender calls it. Your skeleton hierarchy should contain root of the skeleton, a single suspension bone, and the wheel bones. Of course you can add other bones if you wish! The system will only use those referred by you in the blueprints, so you are free to add more bones if you need them. However, I will show now how the bare minimum skeleton looks like:

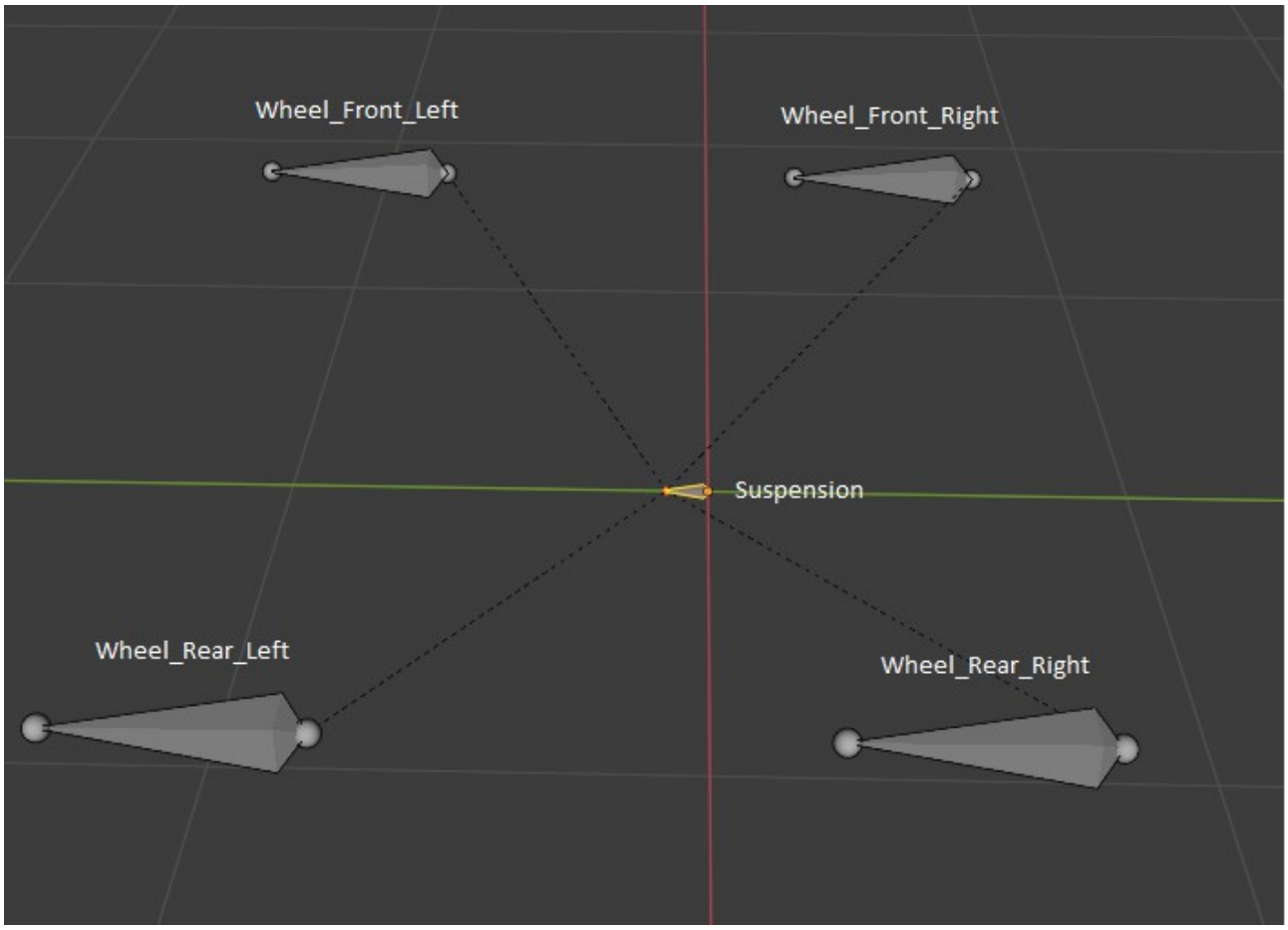


As you can see, all of the bones are oriented on the positive Y axis. That is **NOT A COINCIDENCE**. All of the bones, that you want system to use **MUST** be rotated as shown above. To back it up, I will now show the bones with their appropriate hierarchy.



Here are some more pictures of the skeleton to make everything clear for everyone.

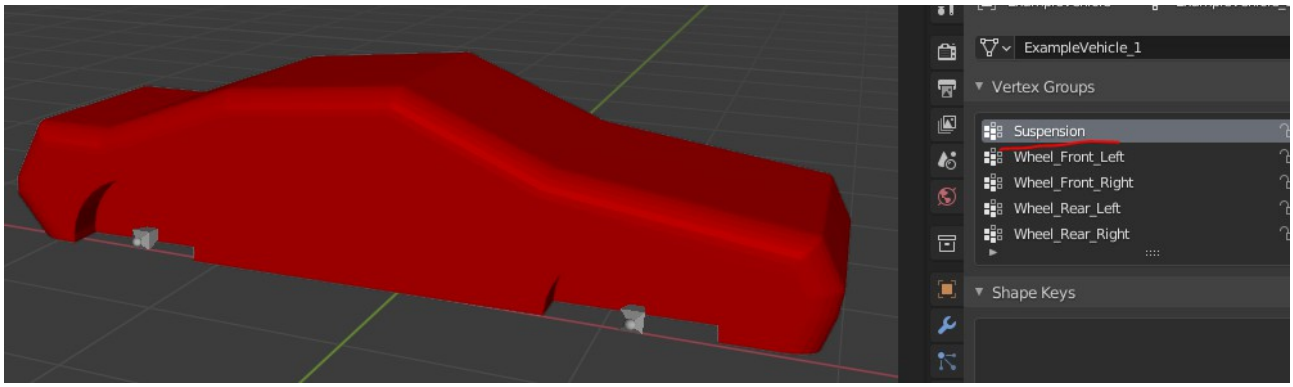




The wheels bones are simply used by the suspension system of the vehicle. Those bones are basically used to locate where the suspension raycasts should be sent from towards the ground. Those bones are also animated by the integrated animation system for the steering wheels as well as rolling them on the ground.

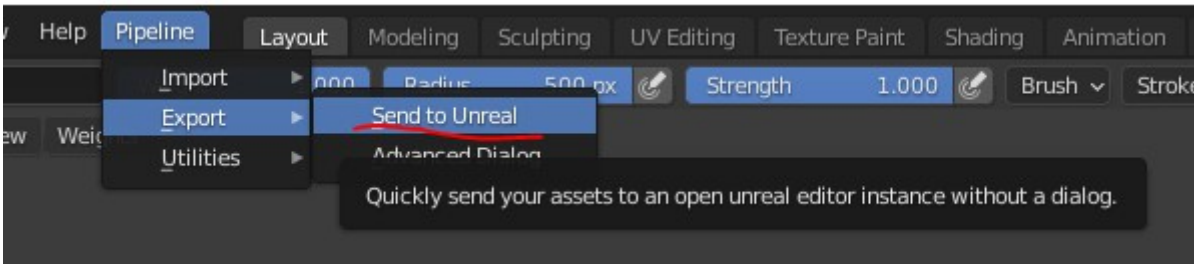
This is how your final rig should look like. Your wheel bones can be placed anywhere you want your future wheels to be **centered**. So I recommend attaching some temporary cylinder to them in Blender to see how they'd look.

Weight painting! Important step for each vehicle. If you want the vehicle body to properly utilize integrated animation system, you need to make vehicle body use **Suspension bone – NOT THE ROOT.**

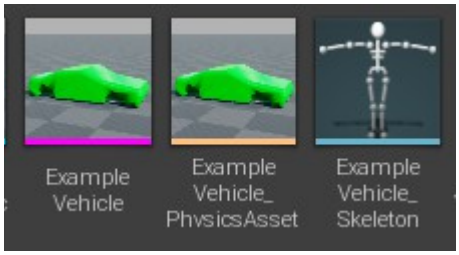


2.2. Importing to UE.

For this step I recommend using Blender to UE plugin available through Epic Games website. I use it, so having my UE4 project opened, I simply click this:

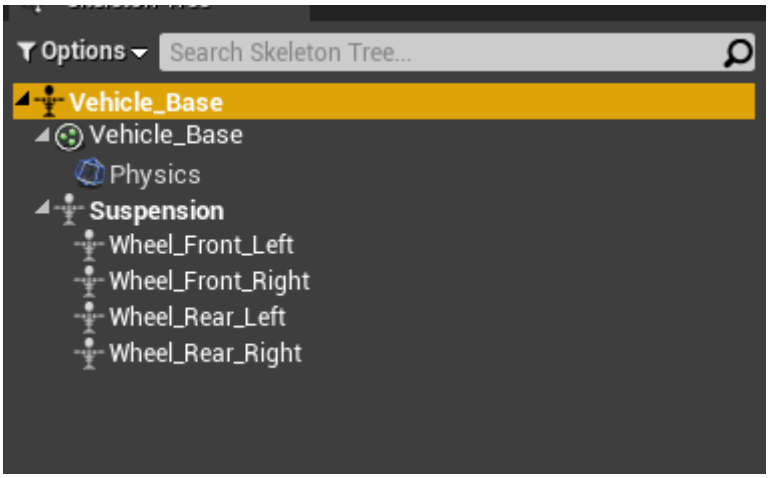


Now that we clicked **Send to Unreal** button, we should see three assets somewhere in the content of our project like so:



2.3. Physics asset setup.

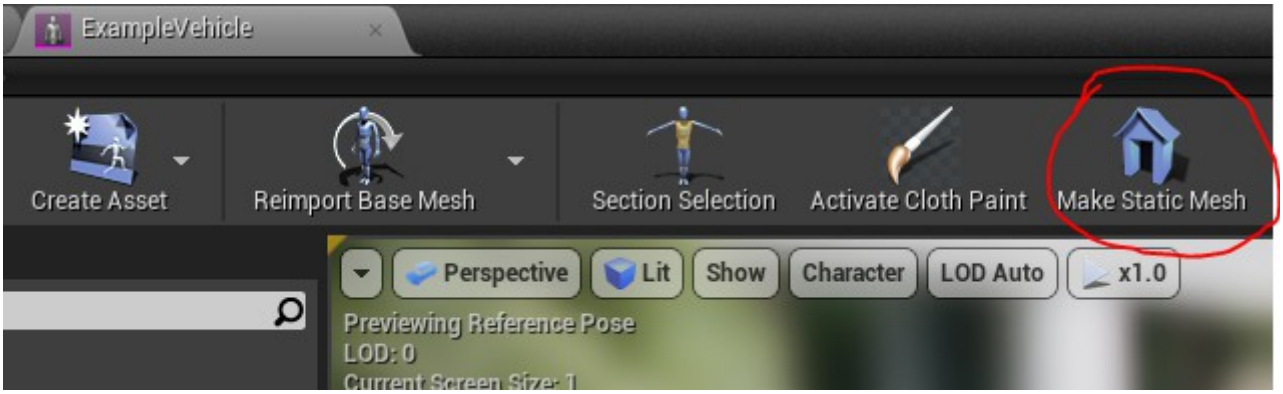
Physics asset for the vehicles should be relatively accurate. **Important! Your collision in the physics asset should always exist only inside root bone of the skeleton like so:**



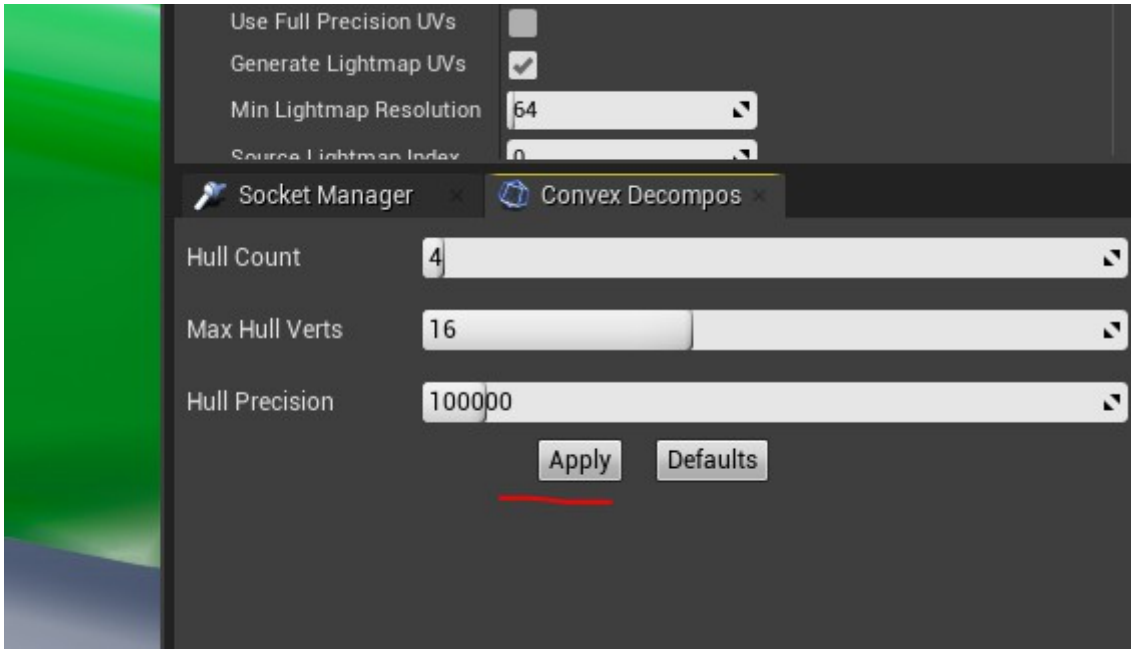
This is important, because suspension and wheels are bones that will be animated, and therefore we don't want animation to affect collision.

Your collision can be created manually, however I highly recommend following these steps:

- Create static mesh from your vehicle's skeletal mesh.

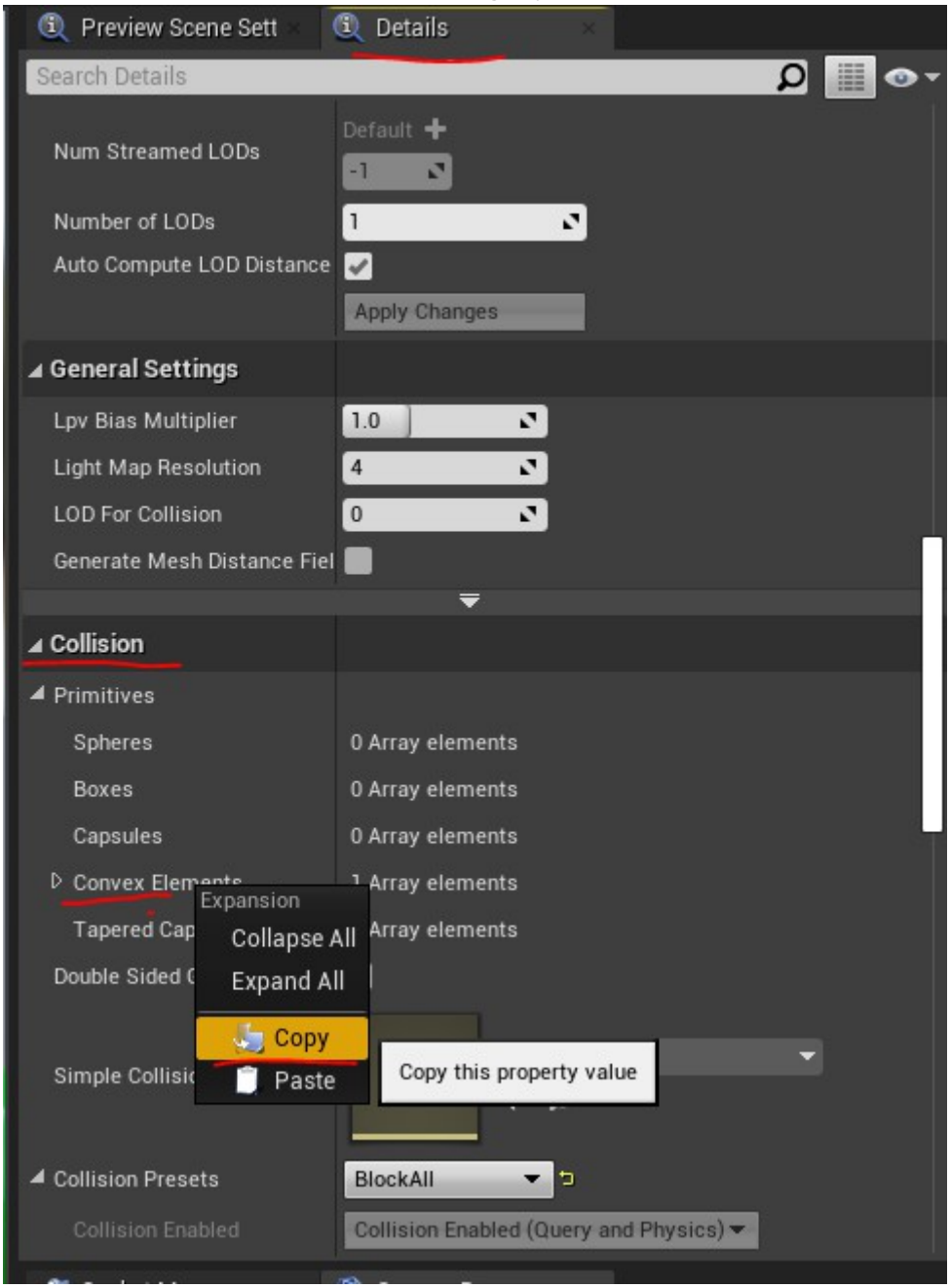


This should generate static mesh for your vehicle. We will only use vehicle static mesh for a few extra steps, and if you wish, it can be later removed. We will use static mesh of the vehicle to generate the collision for the vehicle like so:

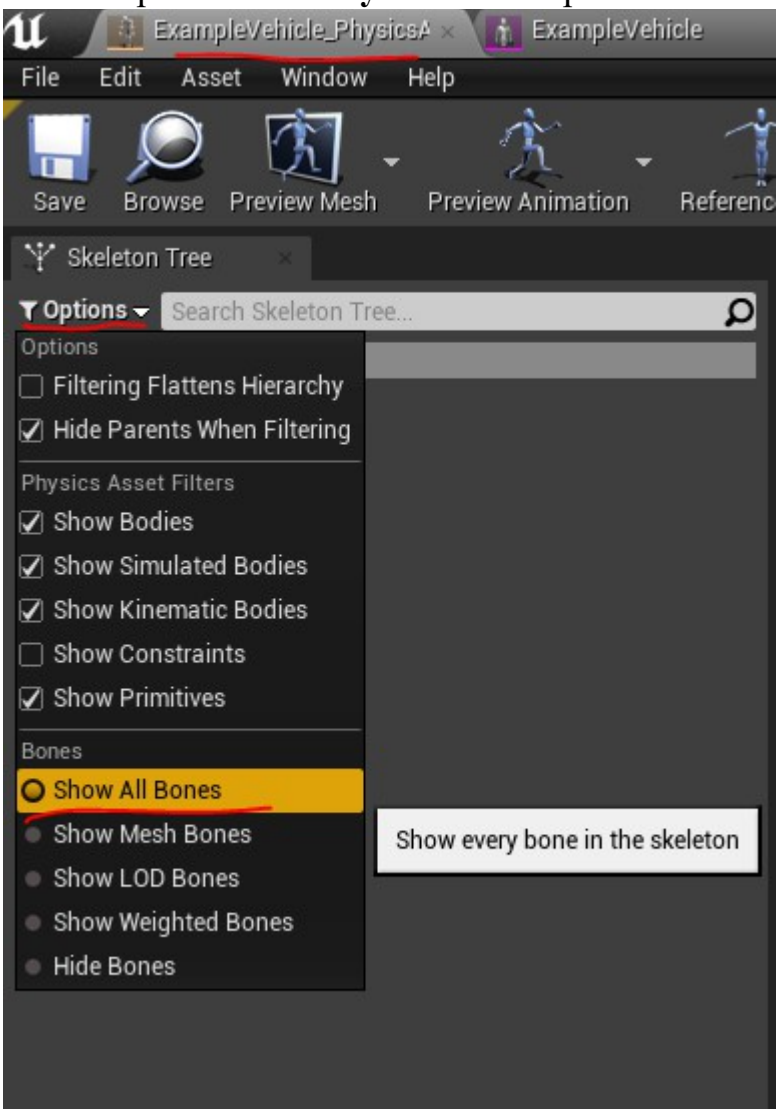


Note that this option is in **Static Mesh – NOT THE SKELETAL MESH.** So if you can't find it, you missed or misunderstood previous step.

Now that the collision is generated for the vehicle, we will simply copy it.
- Go to the **Static Mesh details panel** and right-click and copy **Convex Elements** list from the **Collision** category.

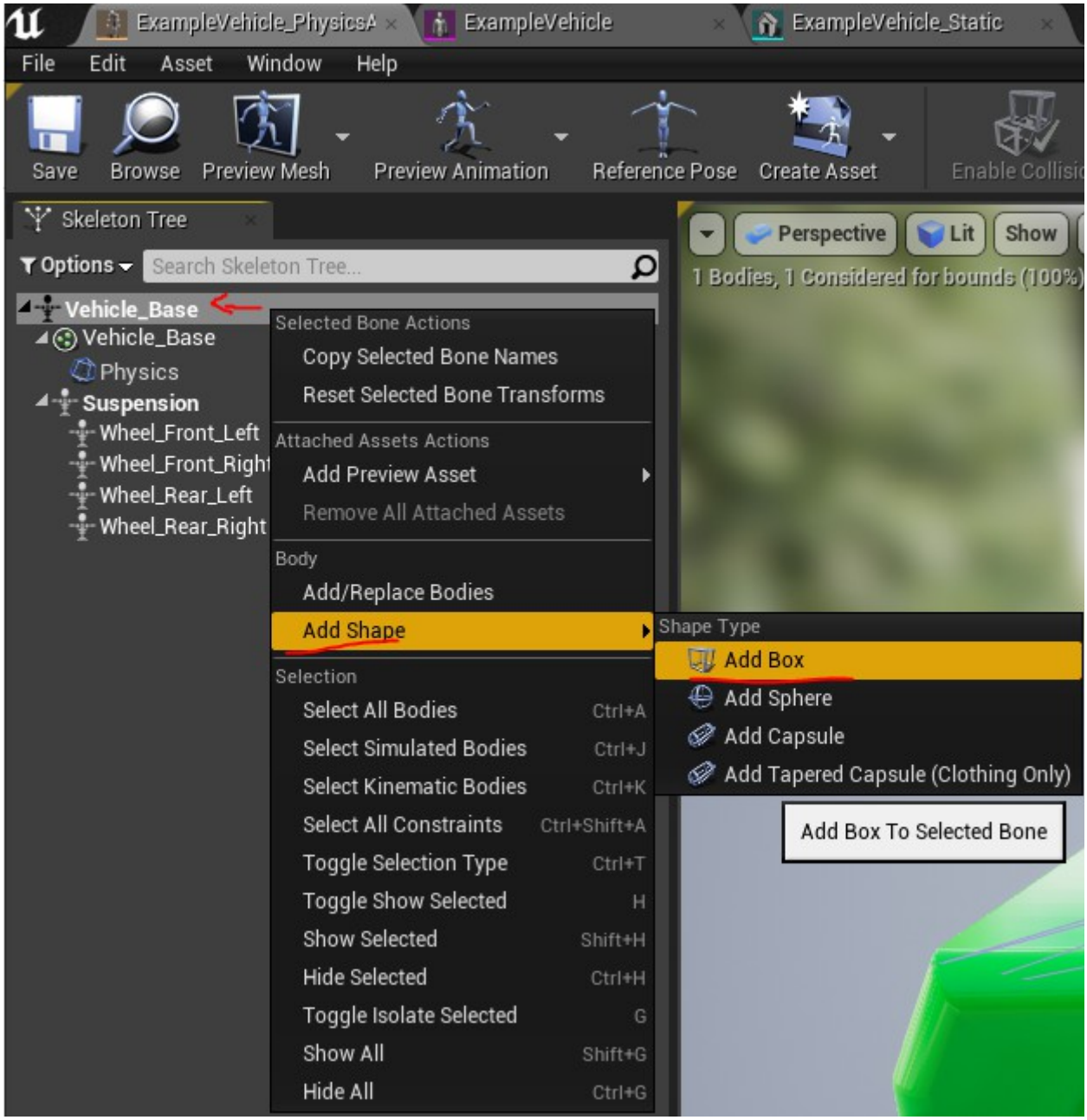


When the values are copied, return to your **Physics Asset**. Unreal always generates some rubbish collisions inside newly imported physics assets by default. We will replace them with our copied **Convex Elements**. However, we need to make sure, that the collision is assigned to the correct bone. Mark **Show All Bones** option in the **Physics Asset** Options context like so:

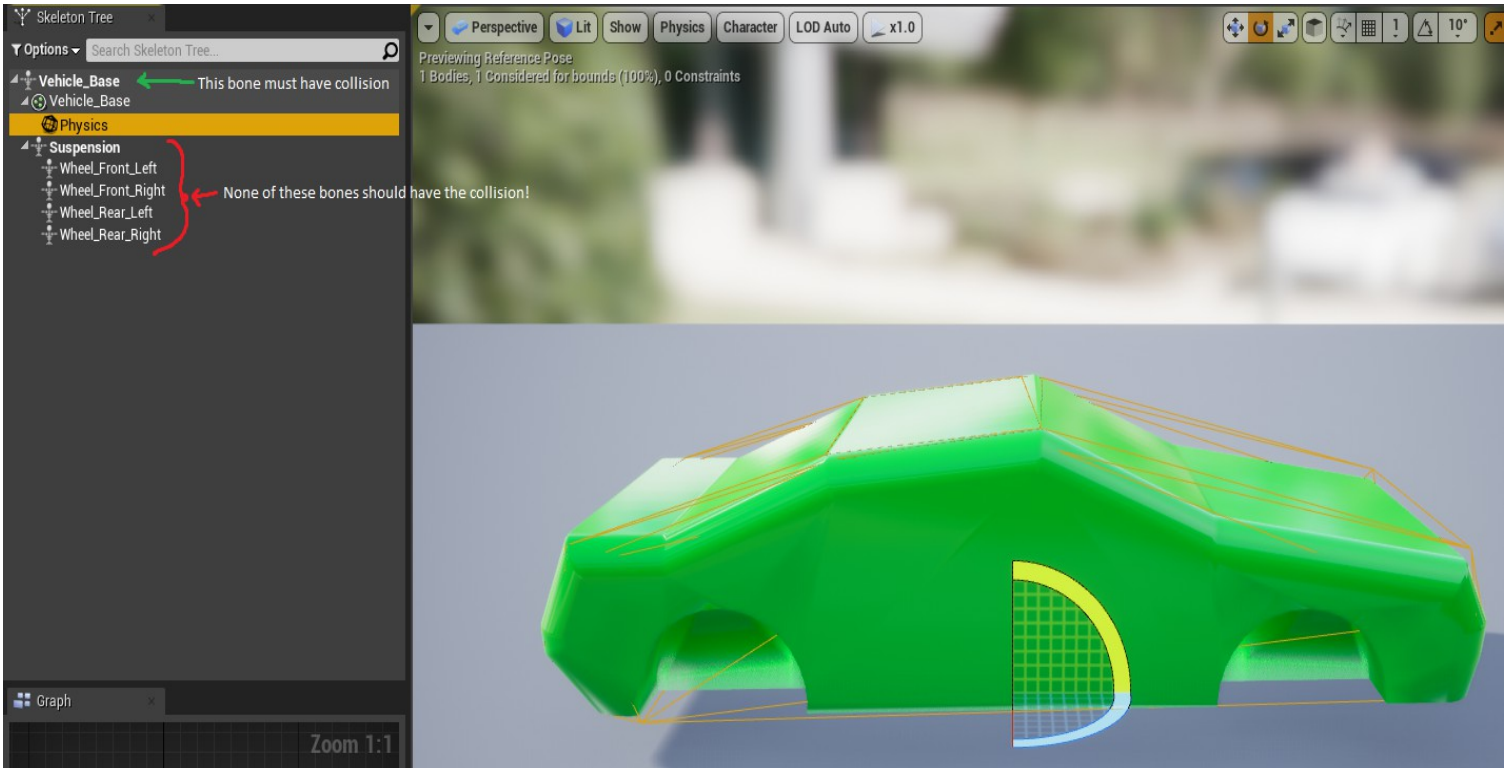


IMPORTANT! Remove any collisions that you see under Suspension or any of the wheel bones!!! If you don't, it will cause issues for the raycasts and the physics simulation. **The best, remove all collisions you see in the hierarchy.**

Now you need to right-click **Vehicle_Base** bone and go for **Add Shape and Add Box:**



This box is just temporary. It will allow us to access vehicle bone physics. Highlight the **Root bone** of the skeleton and go to the **Details** panel. Find **Collision** category. Right-click **Convex Elements** and paste what we earlier copied from the static mesh. **Now we can remove the Box shape we added previously.** It was just a placeholder. End result should look like this:



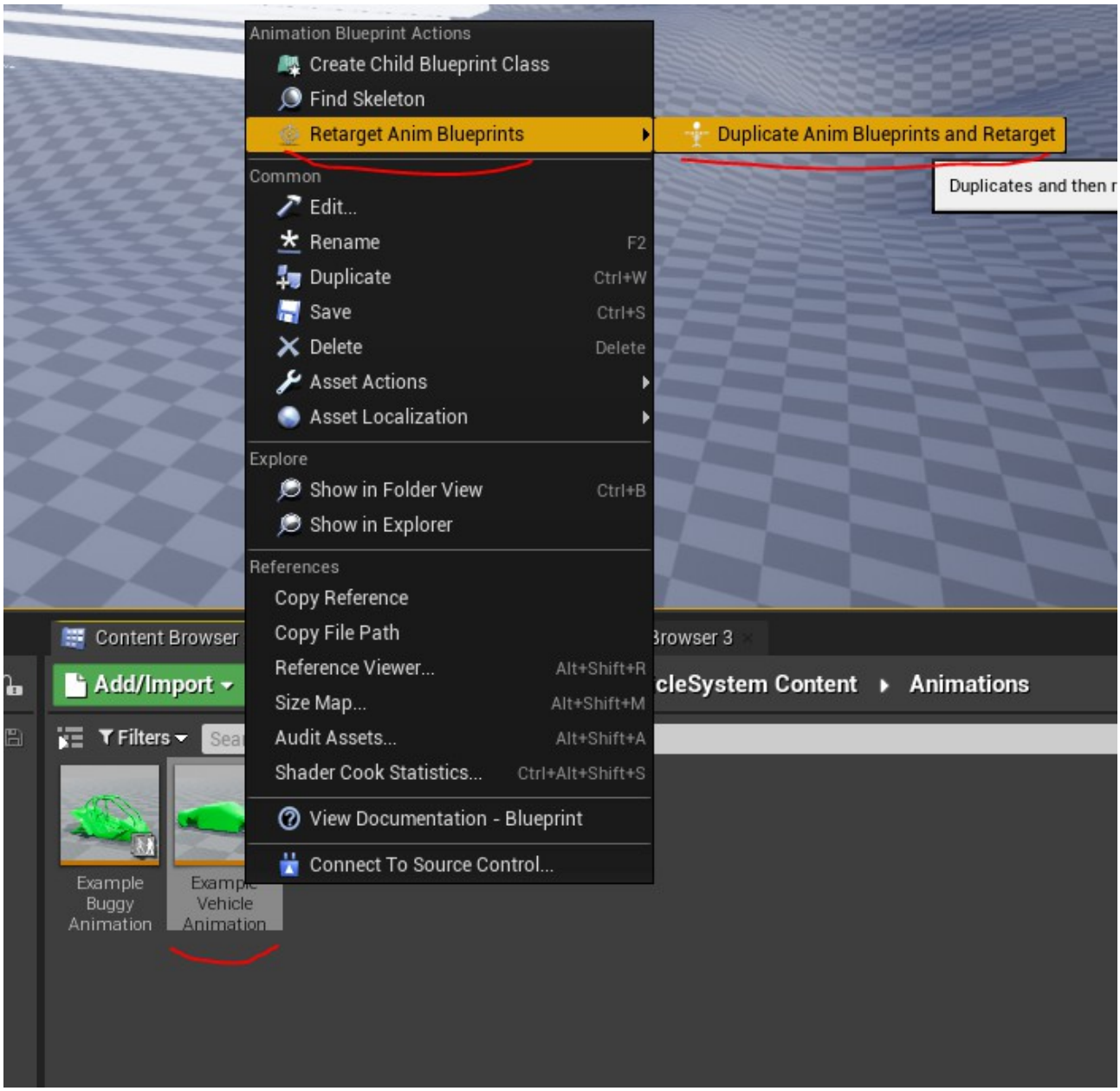
Now we have full setup of the physics asset for the vehicle.

2.4. Animation Setup.

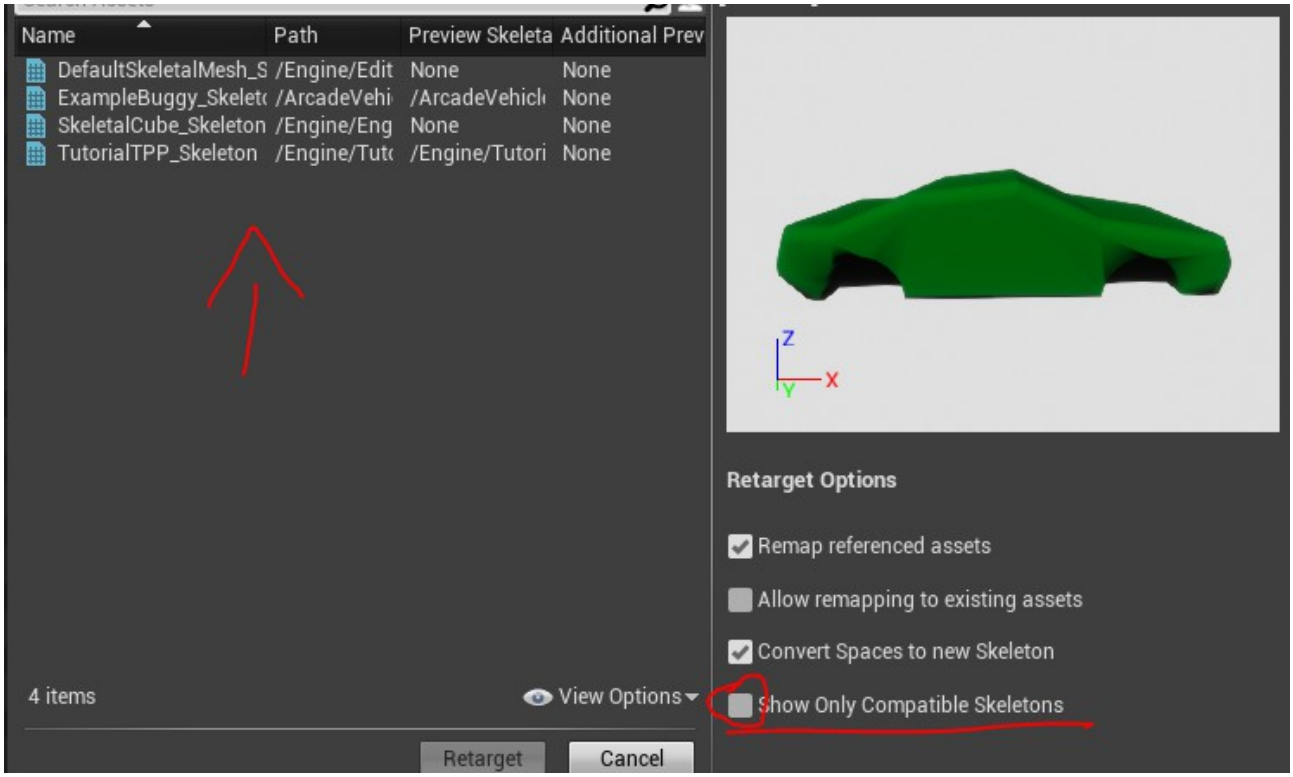
Animation setup is very straightforward if you have your bones named the same way as the bones inside the plugin content vehicles. Otherwise you'll need to fix them up later.

For the animations, go to the **ArcadeVehicleSystem Content/Animations.** You'll find there animations prepared by me, that you can freely used with your own skeleton by simply retargetting them.

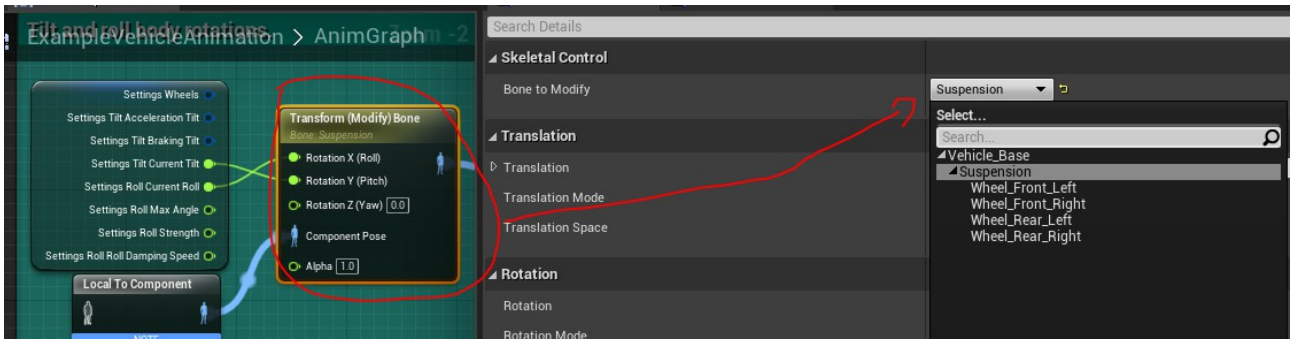
Right-click **ExampleVehicleAnimation** and select **Retarget Anim Blueprints → Duplicate Anim Blueprints and Retarget** like so:



You should now get the window as below. You probably do not see your skeleton in the list. **Make sure you have Show Only Compatible Skeletons DISABLED.** Select your skeleton from the list and click **Retarget.** This will generate identical animation to the one I created, but it will use your skeleton!

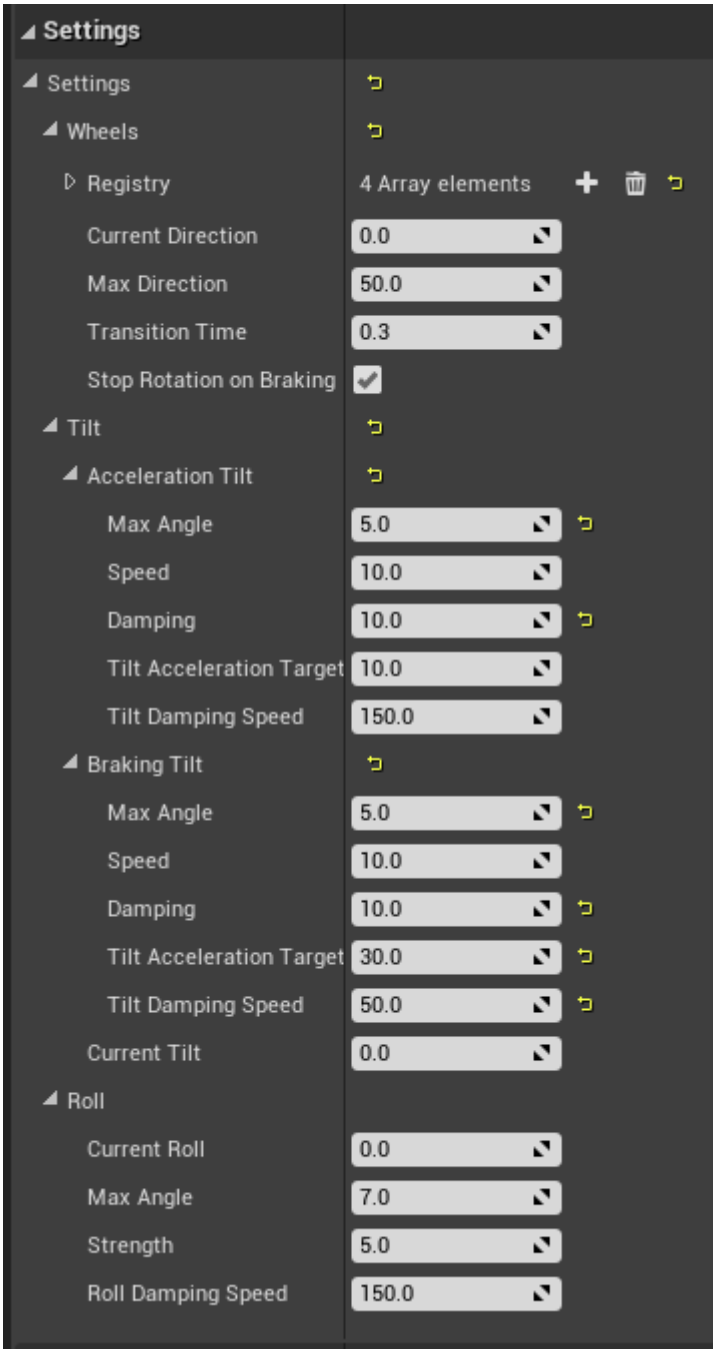


Note! If your bone names are different than what I displayed earlier in this document, you'll need to fixup bone names in the animation like so:



That is for each bone that causes yellow exclamation mark **warning** in the UE. If your bones have the same names, you can literally do nothing.

This plugin allows you to tweak vehicle animations to your needs. So there are a lot of animation options for various situations, so please take a look at those and make them work for your project! **All the variables are well commented in the code, so if you hover over them, each will contain useful tooltip!**



3. Vehicle blueprint setup

3.1. Blueprint creation.

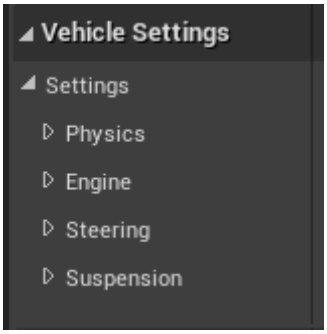
It's extremely easy to set up vehicle blueprints. You can choose one of following ways, it's just depends on what you need

- **Derive from ExampleVehiclePawn** – this option is good if you want to make your vehicle work quickly for testing, where you do not have much of your own logic yet.
- **Derive from ArcadeVehiclePawn –** this is pure C++ class. It does not implement input actions, but provides bare minimum to begin working with vehicles.
- **Create empty pawn –** this is possible to have simple pawn created. All you need it to have is the **SkeletalMeshComponent** and **ArcadeVehicleMovementComponent**.
- **Use your existing vehicle pawn** – this option is most suitable for people that already used other vehicle systems in the project and want to migrate to this plugin. In this case you have your pawn already, so all you need to do is to add **ArcadeVehicleMovementComponent** to it.

For the sake of simplicity, I recommend to use **ExampleVehiclePawn** to as a parent for your vehicle and just play around with values to make it fit your needs.

3.2. Blueprint settings overview.

ArcadeVehicleMovementComponent has a lot of various options to play around with. They can be easily be copied around and are categorized and organized to help users to know what's what.



I am not going to explain them here, that would take me a year to do so. However, I made sure they are very well commented in the code so, if you want to know what those settings do, **HOVER OVER THEM AND READ THE TOOLTIP!**

4. Note for programmers.

If your are a programmer, know that all the code in the plugin is well commented, so if you do not know how something works, just take a look at it, and it should be clear. Feel free to modify the code and play around with it. It's really extensible as it is, but I do not see anything wrong with you modifying the codebase of the plugin to make it fit your needs.