Randomation Vehicle Physics Manual

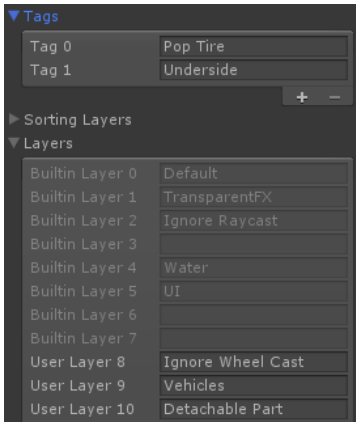
For questions or concerns, contact justin@justinvoke.com.
GitHub: github.com/JustInvoke/Randomation-Vehicle-Physics

Table of Contents

Project Settings

In this section you will find the recommended project settings. Settings not listed here can be adjusted freely.

Tags



Tags are case sensitive.

Pop Tire – Used on objects that pop tires.

Underside – Used on the underside colliders of vehicles, mainly for not detecting crashes on the undersides of vehicles.

Layers

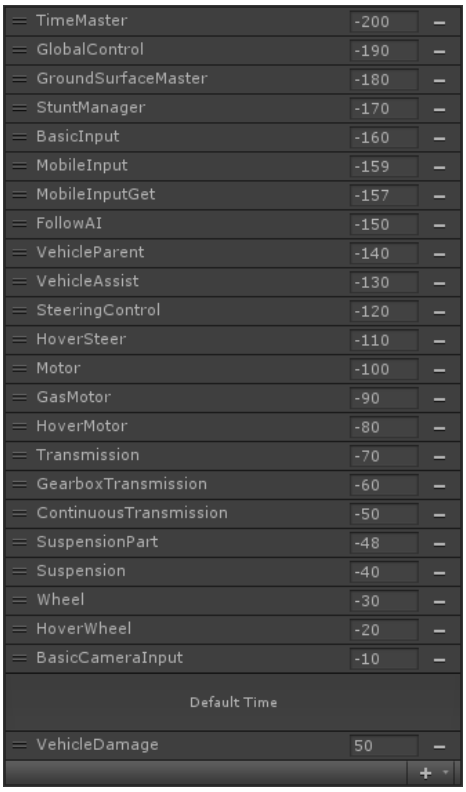
Ignore Wheel Cast – Used for objects ignored by wheels.

Vehicles – Vehicles and their colliders should be on this layer. Other child objects of vehicles do not need to be on this layer.

Detachable Part – Detachable parts and their colliders should be on this layer.

Script Execution Order

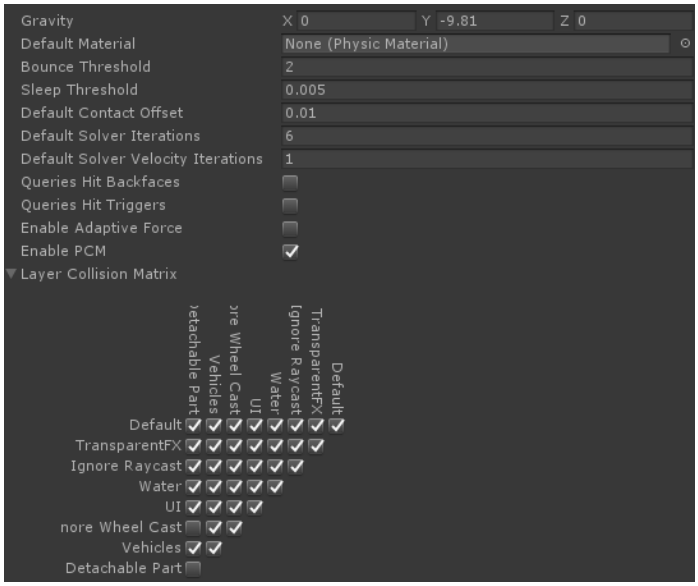
The image to the right shows the required script execution order. Scripts not shown in the list must execute around default time. Scripts cannot be modified in play mode.



Physics

Most of the settings are flexible, but “queries hit triggers” should be false unless you want vehicles to drive on triggers.

Detachable parts shouldn't be able to collide with other detachable parts or objects on the Ignore Wheel Cast layer.



Scene Objects

These are objects required to be present in each scene containing vehicles. Their prefabs are found in the prefabs folder.

GlobalControl – Object containing global variables, ground surface settings, and stunt settings.

TimeMaster – Adjusts the fixed timestep as the time scale changes and changes the pitch of all audio to match the time scale.

Setting Up a Vehicle

Start by finding a vehicle in the Prefabs > Vehicles folder which most closely represents the vehicle you wish to create. Make sure to duplicate the prefab so as to not overwrite the template. The default mass of the vehicle's rigidbody is 1. Acceptable ranges are generally from 0.5 to 10, values outside of this range can cause unpredictable behavior. The collision detection mode must be discrete, and the scale of the vehicle must be one on all axes. If you want to scale a vehicle, the meshes must be scaled up on import, and the suspensions moved to the correct positions.

Child Objects

Body – The mesh object.

Engine – The engine object.

Steering wheel – The object that steers wheels.

Transmission – The transmission object.

Suspension – Suspension objects will have letters indicating their position. F = front, R = rear/right, L = left. For example, suspensionFL would be in the front-left. SuspensionRR would be in the rear-right.

Wheel – The wheel objects, children of suspensions.

Rim – Rim meshes, must be the first child of their wheel. The forward direction of the mesh should be outwards from the face of the rim.

Tire – Tire meshes, must be the first child of their rim. If you don't want to have separate rim and tire objects, the rim mesh may contain both, and the tire object can be deleted.

Mid/front/rear collider – Colliders for different parts of the vehicle.

Under collider – Collider for the underside of the vehicle. Its friction is reduced and it does not count in crash detection.

Tire screech – Audio source for tire screeches.

Wheel snd – Audio source for other wheel sounds.

Crash snd – Audio source for crash impact sounds.

Boost snd – Audio source for the nitrous boost sound.

Boost particle – Particle systems for boost, L/R indicates the side of the vehicle it's on.

Adding or Deleting Wheels

Duplicate the suspension objects within a vehicle and position them accordingly. Make sure to add the new wheels to the wheels array of *VehicleParent* and add the new suspensions to the output drives of the transmission if you want them to drive, or to the steered wheels array of the steering wheel if you want them to steer. For deleting wheels, do the opposite; make sure to remove references to the deleted wheels/suspensions.

Changing Performance

The estimated top speed of vehicle is shown on its engine (*Motor*) object.

The top speed of vehicle in meters per second can be calculated with the following formula:

max RPM of engine / last gear ratio of transmission / (Pi * 100) * circumference of driven wheels

Multiply this by 2.23694 to get miles per hour, or 3.6 to get kilometers per hour. Substitute “last gear ratio of transmission” for “max ratio” in the case of continuous transmissions. For hovering vehicles, the formula can be replaced by the x-value of the last key in the force curve of the *HoverMotor*.

The parts that affect top speed the most are the engine and transmission. The engine's RPM range can be modified by lengthening or shortening the torque curve. After doing this, make sure to call the GetMaxRPM() function on the *GasMotor*. When modifying the gears of *GearboxTransmission*, make sure to call CalculateRpmRanges() if you want the RPM ranges of the gears to automatically adjust. If changing the quantity or order of gears, make sure to call GetFirstGear() to establish which gear is the first. These functions only need to be called with changes in play mode.

A vehicle's acceleration is affected by the engine, transmission, and tire friction. The torque curve/force curve changes the torque/force output based on RPM/speed. The power variable multiplies this.

Handling is affected by the steering wheel (*SteeringControl*), suspension, and tire friction. Drifting is largely affected by the sideways friction of the rear wheels on a vehicle.

Damage

Damage is handled by the *VehicleDamage*, *DetachablePart*, and *ShatterPart* scripts. The muscle car, car damage, and hover car damage prefabs in the Prefabs > Vehicles folder are currently the only examples of damageable vehicles. Damageable vehicles don't have to be as complex as the muscle car, but having more colliders allows for greater deformation “resolution”.

Mesh Orientations

Mesh orientations must be corrected in modeling software.

The main body mesh(es) of a vehicle should be oriented such that the positive z-direction points out from the grille, and the positive y-direction points up from the roof.

Suspension objects (not the meshes) should have their positive z-directions point out from the sides of the vehicle they are on, and their positive y-directions towards the roof.

Moving suspension parts (the actual meshes) should have their positive z-directions facing in the direction they are meant to point in while moving. Their origins need to be at their pivot point.

Wheels should have their positive z-directions point out from the face of their rims.

Separate parts of damageable vehicles need to have their origins centered based on their local geometry, not at the center of the entire vehicle. This is necessary for deforming and displacing parts correctly.

Scripts

VehicleParent

This is the main script for controlling vehicles and must be on the same object as the vehicle's rigidbody at the top of the heirarchy.

Public Variables

Accel axis is brake – Does the acceleration input also function as the brake input when it's negative?

Brake is reverse – Does the brake input initiate reverse driving once the vehicle is stopped?

Hold ebrake park – If the vehicle is stopped and the ebrake is pressed, should it continue being held automatically? This is convenient for parking on slopes, where you just press the ebrake once when stopped rather than holding it down constantly.

Burnout threshold – How great both the acceleration and brake input must be to initiate a burnout. -1 prevents burnouts.

Burnout spin – Amount of force to apply for spinning during burnouts.

Burnout smoothness – Smoothness of starting and ending burnouts.

Engine – The object in the vehicle with the *GasMotor* or *HoverMotor* script attached.

Wheels – The wheels of the vehicle.

Hover wheels – The hover thrusters, referred to as wheels in the code.

Wheel groups – Each wheel group represents one fixed update and each wheel in the group will fire their raycast during their fixed update. For example, if there are 4 groups, each one containing a single wheel of a car, it will take 4 fixed updates for all of the wheels to check for their collisions. This can greatly improve performance at the cost of precision, so it should be reserved for AI vehicles.

Hover – Does the vehicle hover?

Suspension center of mass – Automatically lower the center of mass of the vehicle by the average suspension length.

Center of mass offset – Offset for the center of mass.

Wheel force mode – Force mode used for wheel driving and tire friction forces. This also applies to the driving force for hover wheels.

Suspension force mode – Force mode used for suspension support and hover float forces.

Tow vehicle – Vehicle to be instantiated for towing. It must have a joint component attached. The Big Rig in the Prefabs > Vehicles folder uses this.

Can crash – Can the vehicle crash?

Crash snd – The crash sound audio source.

Crash clips – Audio clips of crash sound effects. These are randomly chosen during crashes.

Camera distance change – Extra distance the camera should sit from the vehicle.

Camera height change – Extra height the camera should sit above the vehicle.

Public Functions

SetAccel(float) – Sets the accel input to the value ranging from -1 to 1.

SetBrake(float) – Sets the brake input to the value ranging from 0 to 1.

SetSteer(float) – Sets the steer input to the value ranging from -1 to 1.

SetEbrake(float) – Sets the ebrake input to the value ranging from 0 to 1.

SetBoost(Boolean) – Sets the boost input to the value.

SetPitch(float) – Sets the pitch input to the value ranging from -1 to 1.

SetYaw(float) – Sets the yaw input to the value ranging from -1 to 1.

SetRoll(float) – Sets the roll input to the value ranging from -1 to 1.

PressUpShift() – Shifts up a gear. (Gearbox transmission only.)

PressDownShift() – Shifts down a gear. (Gearbox transmission only.)

SetUpshift(float) – Sets the upshift input to the value ranging from 0 to 1. (Continuous transmission only.)

SetDownshift(float) – Sets the downshift input to the value ranging from 0 to 1. (Continuous transmission only.)

Inspector Buttons

Get Engine – Sets the engine variable automatically with the first instance of *Motor* in a child object.

Get Wheels – Populates the wheels/hover wheels array with all of their instances in the vehicle.

VehicleDamage

This script damages vehicles. Meshes that are deformed should not be in the displace parts array, otherwise they will be displaced twice as much.

Public Variables

Strength – Resistance to damage.

Damage factor – Multiplier for collision magnitude.

Max collision magnitude – Maximum allowed magnitude for collision velocity.

Max collision points – Maximum contact points that can be evaluated for a collision. This should be 2 in most cases, as it drastically affects performance.

Collision ignore height – Collisions under this local y-position will be ignored. This is identified by a red lateral cross gizmo.

Ignore grounded wheels – If true, grounded wheels will not be damaged.

Collision time gap – Minimum time gap in seconds between collisions evaluated for damage.

Seamless deform – If true, adjacent deforming parts will keep their edges aligned.

Use Perlin noise – If true, some Perlin noise will be added to the mesh deformation.

Calculate normals – If true, normals will be recalculated on deformed meshes.

Damage Parts – Array containing damageable parts. Only parts with motors or transmissions are affected.

Deform meshes – Meshes that are deformed by collisions. Shattering parts should be in this array.

Deform colliders – Mesh colliders that are deformed by collisions. This should not be used unless absolutely necessary because altering mesh colliders is slow.

Displace Parts – Parts that are displaced by collisions. Colliders should be in this array, unless they are in the deform colliders array. Detachable parts, suspensions, and hover wheels should be in this array.

Public Functions

ApplyDamage() – Damages the vehicle, there are six different constructors for this function:

> **ApplyDamage(*ContactPoint* Collision point, *Vector3* Collision velocity)** – Applies damage at the contact point with the specified collision velocity. Useful for collision damage.
>
> **ApplyDamage(*ContactPoint* Collision point, *Vector3* Collision velocity, *float* Velocity limit)** – Same as above, but the collision velocity is clamped to the limit.
>
> **ApplyDamage(*Vector3* Damage point, *Vector3* Damage force)** – Applies damage at the damage point with the specified damage force.
>
> **ApplyDamage(*Vector3* Damage point, *Vector3* Damage force, *float* Force limit)** – Same as above, but the damage force is clamped to the limit.
>
> **ApplyDamage(*Vector3[]* Damage points, *Vector3* Damage force)** – Applies the specified damage force at each point in the array.
>
> **ApplyDamage(*Vector3[]* Damage points, *Vector3* Damage force, *float* Force limit)** – Same as above, but the damage force is clamped to the limit.

Repair() – Repairs all damage.

DetachablePart

This script is for parts that can detach. Detachable parts and their colliders need to be on the Detachable Part layer. Detachable parts must have at least one collider on the object itself or as a child. Vehicles cannot be teleported while loose parts are attached by joints.

Public Variables

Mass – Mass of the rigidbody once detached.

Drag – Drag of the rigidbody.

Angular drag – Angular drag of the rigidbody.

Loose force – Collision force required to make the part come loose with a hinge joint. Leave at -1 to not make a joint and just fall off.

Break force – Collision force required to detach the part completely.

Joints – Array of instances of *PartJoint* for the hinge joint. One is randomly chosen to use for the hinge joint.

PartJoint Class

This is the class for individual hinge joint properties. Variables in this class reflect those of the *HingeJoint* class.

Public Functions

Detach(Boolean) – Detaches the part. The Boolean variable determines whether or not to create a hinge joint.

Reattach() – Reattaches the part.

ShatterPart

This script shatters parts such as windows and lights.

Public Variables

Break force – Collision force required to shatter.

Seam keeper – Transform to reference when deforming for maintaining seams. This is useful for instances where a window breaks, but leaves visible shards around the edge. With the seam keeper set, the shards will stay attached to the edge of the window.

Broken material – Material to change to upon shattering.

Shatter particles – Particle system to play when shattering.

Shatter snd – Sound to play upon shattering.

Public Functions

Shatter() – Shatters the part.

VehicleAssist

This script makes vehicles easier to drive and perform slightly less realistically. It must be on the same object as *VehicleParent*.

Public Variables

Based on wheels grounded – Should the power of assisting forces be based on how many wheels are grounded?

Drift spin assist – How much to assist with rotating the vehicle while drifting.

Drift spin speed – How fast to spin the vehicle while assisting.

Drift spin exponent – Exponent for the sideways speed of the vehicle when sampling from the curve.

Auto steer drift – Should the vehicle use auto-steer drifting? This is useful for situations where counter-steering is difficult, such as on mobile platforms with accelerometer steering.

Max drift angle – Maximum angle in degrees for auto-steer drifting. The vehicle will steer itself to match the angle multiplied by the steer input.

Drift spin curve – The curve used for calculating the assisting forces based on the sideways speed of the vehicle. X-axis = speed, y-axis = force.

Drift push – How much to push the vehicle forward while drifting.

Straighten assist – Should the vehicle automatically straighten out if it's just barely sliding?

Downforce – The downforce of the vehicle.

Invert downforce in reverse – Should downforce be inverted in reverse?

Apply downforce in air – Should downforce be applied in the air?

Downforce curve – Curve for calculating the downforce based on the forward speed of the vehicle. X-axis = speed, y-axis = force.

Auto roll over – Should the vehicle automatically roll over if flipped?

Steer roll over – Should the vehicle be able to flip over with steer input?

Roll check distance – Raycasts are projected from the sides and top of the vehicle to determine if it needs to roll over. This variable sets the distance of the raycasts.

Roll over force – Strength of the roll over forces.

Roll speed threshold – Speed above which the vehicle cannot be rolled over with forces.

Angular drag on jump – Should the angular drag of the vehicle be temporarily increased after jumping to reduce the likelihood of it flipping over in the air?

Fall speed limit – How fast the vehicle is allowed to fall, more specifically the limit on the vehicle's local y-velocity.

Apply fall limit upwards – Should the fall limit be applied when the vehicle is flying upwards?

FlipControl

This script controls the rotation of vehicles in the air. The variables in *VehicleParent* that control flipping are pitchInput, yawInput, and rollInput. This script must be on the same object as *VehicleParent*.

Public Variables

Disable during crash – Disables all flipping while the vehicle is crashing.

Flip power – The flipping power along each local axis.

Free spin flip – Should the vehicle continue spinning if flip input is released? If this is false, flip speed will be directly linked to flip input.

Stop flip – Should the vehicle automatically stop flipping when upright?

Rotation correction – How much the vehicle should attempt to rotate upright. The y-axis indicates forward rotation towards the direction the vehicle is traveling.

Ground check distance – Maximum distance to check for valid ground normal to use for the rotation correction. This uses the “ground mask” layermask on *GlobalControl*.

Ground steepness limit – Minimum dot product between the ground normal and the global up direction.

Dive factor – How much the vehicle should dive in the direction it's traveling.

BasicInput

This script fetches input from axes in the input manager and sends them to *VehicleParent*. It must be on the same object as *VehicleParent*.

Public Variables

Accel axis – The axis for acceleration input.

Brake axis – The axis for brake input.

Steer axis – The axis for steer input.

Ebrake axis – The axis for ebrake input.

Boost button – The boost button.

Upshift button – The button for upshifting gearbox transmissions.

Downshift button – The button for downshifting gearbox transmissions.

Pitch axis – The axis for pitch input.

Yaw axis – The axis for yaw input.

Roll axis – The axis for roll input.

VehicleDebug

This script is used for easily resetting vehicles. The input axes for resetting rotation and position are hard-coded as “Reset Rotation” and “Reset Position” respectively. The script must be on the same object as *VehicleParent*.

Public Variables

Spawn Pos – The position to reset to.

Spawn Rot – The rotation to reset to. This is the forward direction the vehicle will face.

Fall limit – If the vehicle falls below this y-position, it will automatically reset.

PropertyToggleSetter

This script allows for easily changing between “modes” of a vehicle such as skid steering and crab steering. This works in conjunction with *SuspensionPropertyToggle*. The Special Car in the Prefabs > Vehicles folder showcases this component.

Public Variables

Steerer – The *SteeringControl* component of the vehicle.

Transmission – The transmission of the vehicle.

Suspension properties – Suspension properties to be changed.

Presets – Presets for different modes, array of the *PropertyTogglePrese*t class.

Current preset – Index of the current preset being used.

Change button – The button in the input manager that cycles through different presets.

PropertyTogglePreset Class

This is the class for individual presets.

Variables:

Limit steer – Limit the steering range.

Skid steer transmission – Adjust the transmission to output skid steer driving.

Wheels – Array of an array of Boolean variables. The first array corresponds to the number of wheels in the suspension properties array. The second array, the one of Boolean variables, determines if that property in the corresponding *SuspensionPropertyToggle.properties* array is enabled.

Public Functions

ChangePreset(int) – Changes the current preset to the one in the presets array identified by the integer.

Inspector Buttons

Get Variables – Automatically sets the “steerer”, “transmission”, and “suspension properties” variables based on components in child objects.

DriveForce

This is the class that carries RPMs and torque through the drivetrain of a vehicle. Its variables are automatically set by other components.

Public Variables

RPM – The target RPM of the drive.

Torque – The torque of the drive.

Curve – The torque curve of the drive.

Feedback RPM – The rpm being fed backwards through the drivetrain. This can be thought of as the actual RPM of the wheels. It's used for sampling from the torque curve and adjusting engine audio.

Active – Is the drive active?

Public Functions

SetDrive(DriveForce) – Sets the variables of this DriveForce to those in the argument.

SetDrive(DriveForce, float) – Same as above, but the torque is multiplied by the float.

Motor

This is the abstract class for motors from which *GasMotor* and *HoverMotor* inherit their properties. A motor's output torque/energy is dependent on how damaged it is. Once fully damaged, it will no longer be capable of driving. The estimated top speed of a vehicle is shown at the top of the inspector.

Public Variables

Ignition – Is the motor turned on?

Power – Multiplier for the torque output.

Input curve – Curve for adjusting throttle based on the acceleration input of *VehicleParent*.
X-axis = input, y-axis = throttle.

Min pitch – Minimum pitch of the motor's audio source.

Max pitch – Maximum pitch of the motor's audio source.

Can boost – Is the motor capable of nitrous boost?

Boost – How much boost the motor starts with.

Boost power curve – Percentage of how much the motor's RPM and torque is increased while boosting. X-axis = local z-velocity of vehicle, y-axis = power.

Max boost – Maximum possible boost.

Boost burn rate – How quickly the boost amount decreases.

Boost loop snd – The audio source for the boost's sounds.

Boost start – The audio clip for boost starting.

Boost end – The audio clip for boost ending.

Boost particles – The particle systems representing boost.

Strength – Resistance to damage.

Damage pitch wiggle – How much the pitch of the motor's audio source fluctuates based on its damage.

Smoke – Particle system for smoke from the engine. The emission rate is based on the damage.

GasMotor

This is the *Motor* subclass for internal combustion engines.

Public Variables

Torque curve – The engine's torque curve. The x-axis is the RPMs in thousands, the rightmost key represents the maximum RPM of the engine. If this is changed at runtime, call GetMaxRPM().

Inertia – How much resistance the engine has to changing RPMs.

Can reverse – Can the motor turn backwards? Reverse turning requires negative acceleration input.

Output drives – Where the drive is distributed.

Drive divide power – Exponent for output torque; the exponent applied to 1 / the number of outputs. If there is only one output this will not have any effect.

Transmission – Optional reference to a gearbox transmission if the vehicle possesses one, used only for increasing the pitch between shifts.

Pitch increase between shift – Should the pitch of the engine audio briefly increase while shifting?

Public Functions

GetMaxRPM() – Recalculates the maximum RPM, call this if the torque curve is changed.

HoverMotor

This is the *Motor* subclass for hovering vehicles.

Public Variables

Force curve – Curve for adjusting the driving force based on the speed of the vehicle. X-axis = speed, y-axis = force.

Wheels – The hover “wheels” (thrusters).

Inspector Buttons

Get Wheels – Populates the wheels array with instances of *HoverWheel* in child objects.

SteeringControl

This script acts as a steering wheel.

Public Variables

Steer rate – How quickly the vehicle steers.

Steer curve – Limits the steering range based on the speed of the vehicle.
X-axis = speed, y-axis = multiplier.

Limit steer – Should the steering be limited by the curve?

Steer curve stretch – Horizontal stretch applied to the curve.

Apply in reverse – Should steer limits be applied while driving in reverse?

Steered wheels – Wheels that are steered by this component. Wheels can only be steered if their suspensions' steer ranges are set to allow it.

Rotate – Rotate the steering wheel?

Max degrees rotation – Maximum degrees the steering wheel can rotate.

Rotation offset – Degrees added to the rotation.

HoverSteer

This script steers hovering vehicles. Hovering vehicles are capable of turning on the spot, and their steering is not inverted in reverse.

Public Variables

Steer rate – How quickly the vehicle steers.

Steer curve – Adjusts the steering rate based on the speed of the vehicle.
X-axis = speed, y-axis = multiplier.

Steer curve stretch – Horizontal stretch applied to the curve.

Steered wheels – Wheels that are steered by this component.

Rotate – Rotate the steering wheel?

Max degrees rotation – Maximum degrees the steering wheel can rotate.

Rotation offset – Degrees added to the rotation.

Transmission

This is the abstract class for transmissions from which *GearboxTransmission* and *ContinuousTransmission* inherit their behavior. Ratios are represented by floats. 1 = Unchanged RPM and torque, greater than 1 means more torque but lower RPM, and less than 1 means less torque but greater RPM. Once a transmission is fully damaged, it will no longer be able to shift.

Public Variables

Strength – Resistance to damage.

Automatic – Is the transmission automatic?

Skid steer drive – Distribute the drive to the outputs in a special way for skid steering?

Output drives – Where the drive is distributed.

Drive divide power – Exponent for output torque; the exponent applied to 1 / the number of outputs. If there is only one output this will not have any effect.

Public Functions

ResetMaxRPM() – Recalculates the maximum RPM. This needs to be called if the input torque curve is changed.

GearboxTransmission

This is the *Transmission* subclass for gearbox transmissions. For automatic transmissions, note that if the engine is too powerful and the tires slip, the transmission may upshift to compensate, but shift down again if the new ratio is too low for the RPM range. This can result in oscillations between gears. One way to avoid this is by increasing the “feedback RPM bias” variable of the driven wheels.

Gear Class

This is the class for individual gears.

Variables:

Ratio – The ratio of the gear.

MinRPM – The minimum RPM acceptable for automatic transmissions.

MaxRPM – The maximum RPM acceptable for automatic transmissions.

Public Variables

Gears – The array of gears in the transmission. Negative ratios indicate reverse gears. There should be at least one gear with a ratio of zero to represent the neutral gear. This is necessary for determining which gear is the first.

Start Gear – The gear that the transmission starts on. This is the index of the gear in the array, not the number of the gear.

Skip Neutral – Should neutral be skipped when shifting? This is always true for automatic transmissions.

Auto calculate RPM ranges – Should the RPM ranges for each gear be automatically calculated at the start of the game?

Shift delay – Number of physics steps a shift should take.

Shift threshold – Multiplier for comparisons in automatic shifting calculations, should be 2 in most cases.

Public Functions

Shift(int) – Shifts up or down a gear based on the sign of the integer.

ShiftToGear(int) – Shifts to the gear in the gears array identified by the integer.

CalculateRpmRanges() – Recalculates the RPM ranges of the gears. Call this if the ratios are changed.

GetFirstGear() – Gets the index of the first gear, not as a return type. This is the first gear after neutral, the gear with a ratio of zero. Call this if the gears array is changed.

Inspector Buttons

Calculate RPM Ranges – Calls *CalculateRpmRanges()* in the editor.

ContinuousTransmission

This is the *Transmission* subclass for continuously variable transmissions.

Public Variables

Target ratio – How much to linearly interpolate between the min ratio and maximum ratio.

Min ratio – The minimum ratio.

Max ratio – The maximum ratio.

Can reverse – Can the output be reversed?

Manual shift rate – How quickly the ratio changes with manual transmission.

Suspension

This script controls the suspensions of vehicles and the positions and rotations of wheels.

Public Variables

Wheel – The wheel of the suspension, must be a child object.

Generate hard collider – Generates a capsule collider to represent the suspension in hard collisions. Sometimes this can cause the vehicle to be launched by curbs. This object will be on the Ignore Wheel Cast layer.

Hard collider radius factor – Multiplier for the radius of the hard collider.

Brake force – Strength of the brake.

Ebrake force – Strength of the ebrake.

Steer range min – Minimum steer angle in degrees.

Steer range max – Maximum steer angle in degrees.

Steer factor – How much the wheel is steered. Negative values invert steering.

Steer angle – The current steer angle. This can be adjusted in edit mode to visualize the steering.

Ackermann factor – How strong the approximation of Ackermann steering geometry is. This basically makes it so wheels on the inside of a turn steer sharper.

Camber curve – Camber angle of the wheel as it travels. X-axis = suspension compression, y-axis = angle in degrees.

Camber offset – Forced offset to the camber angle.

Solid axle camber – Should the camber adjust as if the wheel was connected to a solid axle? The opposite wheel cannot be null.

Opposite wheel – The wheel on the opposite side of the axle. This is only necessary if solid axle camber is true.

Side angle – Angle in degrees that the wheel extends to the side. This does not affect the camber angle.

Caster angle – The caster angle of the wheel in degrees.

Toe angle – The toe angle of the wheel in degrees.

Pivot offset – Offset for the steering pivot point of the suspension.

Suspension Distance – How far the suspension extends.

Target compression – Target position of the wheel along the suspension. This should only be used for visualizing suspension travel in edit mode, as having it less than 1 can cause unpredictable behavior.

Spring force – The strength of the suspension.

Spring force curve – The strength of the suspension based on how compressed it is.

X-axis = compression, y-axis = force.

Spring exponent – Exponent used on the compression of the suspension, which is then multiplied to the force of the suspension. Higher values basically make the suspension sit lower.

Spring dampening – Resistance of the suspension to compression and extension. This helps prevent jittering and over-corrections.

Extend speed – How quickly the suspension extends when the wheel is not grounded.

Apply hard contact force – Should hard forces be applied when the suspension is compressed further than its limit?

Hard contact force – The strength when applying hard contacts.

Hard contract sensitivity – Sensitivity to the travel velocity of the suspension for hard contacts.

Apply force at ground contact – If true, applies suspension forces at the ground contact point of the wheel. Otherwise it will apply them at the wheel's position. (Not the visual wheel.) This should be true unless the vehicle depends on it.

Leaning force – If true, suspension forces will be applied in the local up direction of the vehicle instead of the ground normal.

Damage pivot – Local point around which the suspension will rotate when damaged.

Detached compression – Compression the suspension will stay at when the wheel is detached.

Jam force – Collision force require to jam the wheel.

Public functions

UpdateProperties() – Updates the toggleable properties of the suspension.

Inspector Buttons

Get Wheel – Gets the wheel attached to the suspension.

Get Opposite Wheel – Gets the suspension of the wheel on the opposite side of the vehicle. This only works if the “wheels” array of *VehicleParent* is set correctly.

SuspensionPropertyToggle

This class toggles certain properties of suspensions to allow vehicles that have different “modes” of steering and driving such as skid steering and crab steering. The Special Car in the Prefabs > Vehicles folder showcases this component.

SuspensionToggledProperty Class

This class represents individual properties to be toggled.

***Properties* enumerator values:**

steerEnable – Is steering enabled?

SteerInvert – Is steering inverted?

DriveEnable – Is driving enabled?

DriveInvert – Is driving inverted?

EbrakeEnable – Is the ebrake enabled?

SkidSteerBrake – Adjust braking for skid steering?

Variables:

Property – The property to use.

Toggled – Is the property enabled?

Public Variables

Properties – The array of properties.

Public Functions

These automatically call UpdateProperties() on the corresponding *Suspension*.

ToggleProperty(int *index*) – Toggle the property at *index* in the properties array.

SetProperty(int *index*, Boolean) – Set the property at *index* in the properties array to the value.

SuspensionPart

This script controls moving suspension parts. The Car RWD, F1, and Monster Truck prefabs in the Prefabs > Vehicles folder use this for their suspensions. Suspension parts must be oriented such that their forward direction is the direction they point in. Suspension part movement must be approximated because wheels travel along a straight line; unlike real suspensions where they move along a curve.

Public Variables

Suspension – The suspension this is a part of. Can be null for solid axle parts.

Is hub – Is this the hub of a wheel?

Connect obj – The object this part is pointing at, can be null for solid axles.

Connect point – The local space point to point at in connect obj. This is identified by a green sphere gizmo.

Rotate – Should this part rotate towards the connect point?

Stretch – Should the part scale to reach the connect point? Scaling is based on the distance to connect point when Start() is called, so make sure parts are lined up correctly at first.

Solid axle – Is this a solid axle?

Invert Rotation – Invert the local z-rotation? (Only for solid axles.)

Solid axle Connector – Is this part connected to a solid axle?

Wheel 1 – One of the wheels a solid axle is connected to, order doesn't matter.

Wheel 2 – Other wheel a solid axle is connected to, order doesn't matter.

Wheel

This script controls wheels and their friction, and is the final destination for drive forces. It also sends the feedback RPM up the drivetrain. Each wheel must be a child of its *Suspension*.

Public Variables

Generate hard collider – Generate a sphere collider to follow the wheel and represent side collisions? This can be disabled to improve performance as moving a collider in local space is slow. This object will be on the Ignore Wheel Cast layer.

Feedback RPM bias – Variable for linearly interpolating between the raw RPM and the actual RPM of the wheel for sending the feedback RPM up the drivetrain. Raw RPM refers to what the RPM would be based purely on the velocity of the wheel, ignoring driving and braking forces.

RPM bias curve – Curve representing how the final RPM of the wheel should be set based on the torque and braking force. By default, this scales linearly, but can be tweaked such that the wheel will have a greater bias towards spinning at it's target RPM, regardless of the torque being applied. X-axis = torque/brake force, y-axis = linear interpolation between raw RPM and target RPM.

RPM bias curve limit – The RPM bias curve is interpolated with a linear curve based on the raw RPM of the wheel divided by this variable. As the raw RPM of the wheel approaches this value, the RPM bias curve has less of an effect.

Axle friction – Friction generated by the wheel's axle. Increasing this causes vehicles to slow down when coasting. Another option for achieving this effect is increasing the drag of the vehicle's rigidbody.

Friction Smoothness – How smoothly the applied friction force changes. This can be increased to reduce jittering/shaking.

Forward friction – The forward friction of the wheel.

Sideways friction – The sideways friction of the wheel.

Forward rim friction – Forward friction of the wheel when the tire is popped.

Sideways rim friction – Sideways friction of the wheel when the tire is popped.

Forward curve stretch – Horizontal stretch applied to the forward friction curve.

Sideways curve stretch – Horizontal stretch applied to the sideways friction curve.

Forward friction curve – Adjust how much friction the wheel has based on how much it's slipping. X-axis = slip, y-axis = friction.

Sideways friction curve – Adjust how much friction the wheel has based on how much it's slipping. X-axis = slip, y-axis = friction. You might want the rear wheels to have a little more sideways friction than the front to reduce spin-outs and fish tailing.

Slip dependence – Enumerator that configures the relationship between forward slip and sideways slip. Sideways is usually the ideal setting.

Dependent – Both the forward and sideways slip are related, in that if one increases, both do. For example, if the vehicle is sliding sideways, then the wheels will not be able to grip the ground in order to accelerate or brake.

Sideways – Only the sideways slip is dependent on the forward slip.

Forward – Only the forward slip is dependent on the sideways slip.

Independent – Neither slips are dependent on each other. For example, if the vehicle is sliding sideways, it can still accelerate or brake as the forward grip remains unchanged.

Forward slip dependence – The lower this is, the more the forward friction will be affected by the sideways slip. Most noticeable when the slip dependence is set to *Independent*.

Sideways slip dependence – The lower this is, the more the sideways friction will be affected by the forward slip. Most noticeable when the slip dependence is set to *Independent*.

Normal friction curve – Adjusts the friction of the wheel based on the dot product of the ground surface normal and the global up direction. X-axis = normal dot product, y-axis = friction multiplier. This is useful if you don't want vehicles to grip walls or steep slopes easily.

Compression friction factor – How much the compression of the suspension affects the friction of the wheel. The more the suspension is compressed, the more friction the wheel will have. 0 = full wheel friction, no change based on suspension compression. 1 = friction completely dependent on compression and less friction overall as a result.

Tire radius – The radius of the tire.

Rim radius – The radius of the rim.

Tire width – The width of the tire.

Rim width – The width of the rim.

Tire pressure – The pressure of the tire, basically a linear interpolation between the rim radius and tire radius.

Popped – Is the wheel popped?

Can pop – Can the wheel pop? Tires will pop if this is true and the wheel drives over an object with the tag “Pop Tire”.

Deform amount – How much the tire deforms. Tires must use the “Tires” shader to deform.

Rim glow – How much the rim glows when it scrapes and the tire is popped. The rim's shader must have a property named "_EmissionColor". This works with the standard shader.

Apply force at ground contact – Determines whether friction forces are applied at the ground contact point, or the center of the wheel.

Impact snd – The audio source for wheel sounds other than tire screeches.

Tire hit clips – Array of audio clips to use in wheel collisions.

Rim hit clip – Audio clip for rim collisions.

Tire air clip – Audio clip for the tire letting out air.

Tire pop clip – Audio clip for tire popping.

Detach force – Collision force required to detach the wheel.

Mass – Mass of the detached wheel.

Tire mesh loose – Collision mesh for the detached wheel if it isn't popped.

Rim mesh loose – Collision mesh for the detached wheel if it is popped.

Detached tire material – Physic material for the detached wheel's collider if the wheel isn't popped.

Detached rim material – Physic material for the detached wheel's collider if the wheel is popped.

Public Functions

Deflate() – Deflates the tire.

FixTire() – Fixes the tire.

Detach() – Detaches the wheel.

Reattach() – Reattaches the wheel.

GetWheelDimensions(radiusMargin, widthMargin) – Automatically sets the dimensions of the wheel based on the rim and tire meshes. RadiusMargin is added to the radius and widthMargin is added to the width.

Inspector Buttons

Get Wheel Dimensions – Calls the *GetWheelDimensions()* function in the editor.

WheelContact Class

This is the class for a wheel's contact point.

Variables:

Grounded – Is the wheel grounded?

Col – The collider the wheel is touching.

Point – The contact point position.

Normal – The normal of the contact point.

Relative velocity – Relative velocity between the wheel and contact point object.

Distance – Distance from the suspension to the contact point minus the wheel radius.

Surface friction – The friction of the contact surface.

Surface type – The surface type of the object identified by the surface types array of *GroundSurfaceMaster*

HoverWheel

This script makes hovering vehicles hover by controlling each thruster.

Public Variables

Hover distance – Distance the wheel should hover above the ground.

Buffer distance – Distance at which the wheel will apply stronger forces to avoid touching the ground.

Float force – Floating strength.

Buffer float force – Buffer floating strength.

Float force curve – Adjusts the floating force based on how compressed the “suspension” is.
X-axis = compression, y-axis = force.

Float exponent – Exponent used on the compression of the “suspension”, which is then multiplied to the float force. Higher values basically make it float lower.

Float dampening – Resistance of the wheel to vertical movement. This helps prevent jittering and over-corrections.

Brake force – Brake strength of the wheel.

Ebrake force – Ebrake strength of the wheel.

Steer factor – Adjusts how much the wheel steers.

Side friction – Resistance to sideways movement.

Visual wheel – The visual representation of the wheel. This tilts based on how the vehicle is driving.

Visual tilt rate – How quickly the visual wheel tilts.

Visual tilt amount – How much the visual wheel tilts.

Detach force – Collision force required to detach the wheel.

Mass – Mass of the rigidbody of the detached wheel.

Wheel mesh loose – Mesh for the detached wheel's collider.

Detached wheel material – Physic material for detached wheel.

Public Functions

Detach() – Detaches the wheel.

Reattach() – Reattaches the wheel.

HoverContact Class

This is the class for a hover wheel's contact point.

Variables:

Grounded – Is the wheel grounded?

Col – The collider the wheel is touching.

Point – The contact point position.

Normal – The normal of the contact point.

Relative velocity – Relative velocity between the wheel and contact point object.

Distance – Distance from the wheel to the contact point.

Inspector Buttons

Get Visual Wheel – Gets the transform with the mesh representing the wheel.

TireMarkCreate

This script generates tire marks, affected by the tire mark variables in *GlobalControl*.

Public Variables

Slip threshold – Minimum slip required for tire marks to be created.

Calculate tangents – Should tangents be calculated for tire mark meshes with normal mapping?

Tire mark materials – Materials used for the tire mark based on the surface the wheel is driving on. The material in the array corresponds to the surface type in the surface types array of *GroundSurfaceMaster*.

Rim mark materials – Same as above, but for when the tire is popped.

Debris particles – Particles such as dirt or tire smoke left by the wheel based on the current surface type.

Sparks – Particle system for sparks left by rims.

TireMark Class

This is the class used for tire mark instances.

TireScreech

This script plays tire screeching sounds based on which wheels are sliding.

GlobalControl

This script stores certain global variables.

Public Variables

Quick restart – Allow reloading the scene with the press of a button named “Restart” in the input manager.

Wheel cast mask – Layermask for objects wheels collide with.

Ground mask – Layermask for objects vehicles check against to see if they're rolled over.

Damage mask – Layermask for objects that damage vehicles.

Frictionless mat – Frictionless physic material used for generated wheel and suspension colliders.

Tire mark length – Number of segments in each tire mark.

Tire mark gap – Time gap between tire mark segments.

Tire mark height – Height of tire marks above their surface normals.

Tire fade time – Lifetime in seconds for tire marks.

TimeMaster

This script adjusts the fixed timestep according to the time scale and adjusts the pitch of all audio sources to the time scale.

Public Variables

Master mixer – The master audio mixer.

Destroy on load – Should the object be destroyed between scenes?

StuntManager

This script contains information about stunts and the points and boost they give.

Public Variables

Drift score rate – How many points drifting awards.

Drift connect delay – Delay during which subsequent drifts will count as a single drift.

Drift boost add – How much boost is awarded by drifting.

Jump score rate – How many points jump distance awards.

Jump boost add – How much boost is awarded by jump distance.

Stunts – Array of stunts that can be performed.

Stunt Class

Variables:

Name – The name of the stunt.

Rotation axis – Local rotation axis for the stunt.

Precision – Minimum dot product between the vehicle's rotation and the stunt direction.

Score rate – How many points the stunts awards based on how complete it is.

Multiplier – Multiplier to the score for multiple completions of the same stunt in a single jump.

Angle threshold – Minimum angle of rotation in degrees for a stunt to count.

Boost add – Amount of boost rewarded based on stunt completion.

StuntDetect

This script detects stunts and awards the player with points and nitrous boost. Crashing will cancel stunts if canCrash on VehicleParent is set to true. A string is generated showing which stunts are being performed.

Public Variables

Detect drift – Should drifting be detected?

Detect jump – Should jumps be detected?

Detect flips – Should flips be detected?

Engine – The engine to give boost to after successful stunts.

GroundSurfaceMaster

This script manages the different ground surfaces. Ground surfaces determine the friction of the surface, the tire mark, and tire sound.

Public Variables

Surface types – The *GroundSurface* array of surface types.

GroundSurface Class

Class for individual ground surface types.

Variables:

Name – The name of the surface type.

Use collider friction – Should this surface type use the friction of the collider it's attached to?

Friction – The friction of the surface, overridden by the collider friction if the previous variable is true.

Always scrape – Should tire marks always be left while driving on this surface even when not sliding?

Leave sparks – Should rims scraping on this surface leave sparks?

Tire snd – The sound of tires sliding on this surface.

Rim snd – The sound of rims sliding on this surface.

Tire rim snd – Audio clip combining both of the previous sounds. This is necessary because tire screech sounds are played by a single audio source for each vehicle, and this particular clip is for when only some of the tires on a vehicle are popped.

GroundSurfaceInstance

This script is used on objects that are meant to utilize certain surface types.

Public Variables

Surface type – Name of the surface type used from the surface types array in *GroundSurfaceMaster*.

TerrainSurface

This script is for associating surface types with painted terrain textures. You can attach it to any terrain object. The inspector automatically builds an array of surface types based on the number of textures on the terrain. You can select a surface type for each one.

Public Functions

GetDominantSurfaceTypeAtPoint(Vector3) – Gets the index of the dominant surface type at the world-space point specified. This is the index of the surface type in the surface types array of *GroundSurfaceMaster*.

GetFriction(surfaceType) – Gets the friction of the indicated surface type.

LightController

This script controls lights on a vehicle.

Public Variables

Headlights on – Are the headlights turned on?

High beams – Are the high beams on? This will activate the actual lights, not just change the light mesh's material.

Brake lights on – Are the brake lights on?

Right blinkers on – Are the right blinkers on?

Left blinkers on – Are the left blinkers on?

Blinker interval – Time in seconds the blinkers are on or off.

Reverse lights on – Are the reverse lights on?

Transmission – Transmission used to turn the reverse lights on and off.

Headlights – Array containing the headlights.

Brake lights – Array containing the brake lights.

Right Blinkers – Array containing the right blinkers.

Left Blinkers – Array containing the left blinkers.

Reverse lights – Array containing the reverse lights.

VehicleLight

This script is for individual lights on a vehicle, controlled by *LightController*.

Public Variables

On – Is the light turned on?

Half on – Is the light halfway on? The difference between *on* and *half on* is comparable the difference between a brake light coming on when the headlights are on, and becoming brighter when the brakes are engaged. Another example is the difference between headlights and high beams.

Target light – The light to activate.

Shared light – The light shared with another *VehicleLight*. An example of a shared light is a single light shared by two headlights. When one headlight breaks, the shared light will be turned off and the target light of the unbroken light will be turned on.

Sharer – The *VehicleLight* that the shared light is shared with.

On material – The material for the light mesh when it's turned on.

FollowAI

This script is for making vehicles either follow objects or drive to waypoints. If the object that the vehicle is following becomes hidden, the vehicle will drive to the last point it saw it. The script must be on the vehicle's root object.

Public Variables

Target – The object being followed. If the target is a waypoint, it will automatically drive to the next one once it's close enough. If this is changed at runtime call InitializeTarget().

Follow distance – Distance to follow behind a non-waypoint object.

Speed – Percentage of top speed to drive at.

Target velocity – The vehicle will not drive faster than this speed in meters per second. A value of -1 will disable it. It is affected by the previous speed variable.

View block mask – Layermask identifying objects that can block the view of the target.

Stop time reverse – If a vehicle gets stuck this is the time in seconds the vehicle will remain stuck before attempting to reverse.

Reverse attempt time – Time in seconds the vehicle will drive in reverse.

Reset reverse count – Number of times the vehicle will attempt reversing before resetting. Set to -1 to disable.

Roll reset time – Time in seconds the vehicle will remain rolled over before resetting. Set to -1 to disable.

Public Functions

InitializeTarget() – Initializes some variables related to the target. Call this if the target is changed.

VehicleWaypoint

This script is used on waypoints for vehicles with the FollowAI script attached to follow.

Public Variables

Next point – The next waypoint.

Radius – The radius of the waypoint. Once a vehicle comes within this distance, it will move on to the next waypoint.

Speed – Percentage of vehicles' top speed they will drive at when approaching the waypoint.

CameraControl

This script moves and rotates the camera as it follows vehicles.

Public Variables

Target – The vehicle to follow.

Height – The height above the vehicle the camera sits at.

Distance – Distance from the vehicle where the camera sits.

Stay flat – Keeps the camera flat so that its local y-axis always points up.

Cast mask – Layermask for objects that are checked for when they obstruct the view of the vehicle. The camera will move in front of them and avoid clipping into geometry.

Public Functions

Initialize() – Initializes certain variables. This is useful for instances where the target of the script is not set at the very start of the scene. Call this when setting the target.

SetInput(x, y) – Sets the rotational input of the camera, similar to a Vector2.

BasicCameraInput

This script sets the *CameraControl*'s rotational input based on axes in the input manager.

Public Variables

X input axis – The x-input axis, for looking left and right.

Y input axis – The y-input axis, for looking forward and back.

VehicleMenu

This script controls the menu in the demo.

Public Variables

Cam – The main camera.

Spawn point – Where vehicles are spawned.

Spawn Rot – The forward direction of spawned vehicles.

Vehicles – The array of vehicles to be spawned.

Other variables are related to the UI.

MobileInput

This is a basic for getting input from UI buttons attached to the Canvas object in the mobile scene. It also locks the screen orientation.

Public Variables

Screen rot – Orientation to lock the screen at.

Public Functions

SetAccel(float) – Set the mobile accel input.

SetBrake(float) – Set the mobile brake input.

SetSteer(float) – Set the mobile steer input.

SetEbrake(float) – Set the mobile ebrake input.

SetBoost(Boolean) – Set the mobile boost input.

MobileInputGet

This script is meant to be attached to vehicles to get mobile input.

Public Variables

Steer factor – Multiplier for accelerometer steer input.

Flip factor – Multiplier for accelerometer flip input.

Use accelerometer – If true, steering input will be overridden with accelerometer input. Otherwise it will use the steering input fetched from *MobileInput*.

Delta factor – Multiplier for rate of change of input, added to final input.

PerformanceStats

Basic script for displaying the frame rate in the mobile demo.

Public Variables

FPS text – The text that shows the FPS.

Public Functions

Restart() – Same function as quick restarting in *GlobalControl*.

GizmosExtra

This script contains extra functions for drawing gizmos.

Static Functions

DrawWireCylinder(position, direction, radius, height) – Draws a wire cylinder much like DrawWireSphere() and DrawWireCube(). Position is the position of the cylinder, direction is the direction the cylinder extends along, or the normals of the caps, radius is the cylinder's radius, and height is the length of the cylinder.

F

This script contains extra math functions.

Static Functions

MaxAbs(float) – Returns the number with the greatest absolute value.

GetTopmostParentComponent<Component>(Transform) – Gets the topmost parent of the transform with the specified component. The return type is a *Component* and must be casted to the desired type.

MeshUtil

This script contains functions for modifying meshes.

Static Functions

CalculateMeshTangents(Mesh) – Calculates the tangents for the mesh, requires the normals to already be calculated.

VehicleBalance

This script is meant for balancing vehicles such as motorcycles. This script is not completely stable, but I decided to include it anyways in case some people want to experiment with it. It is used on the motorcycle prefab.

Public Variables

Lean factor – How far the vehicle can lean in each direction.

Lean smoothness – Smoothness of leaning.

Lean roll curve – Adjusts the roll amount based on the speed of the vehicle.
X-axis = speed, y-axis = roll amount.

Lean pitch curve – Adjusts the pitch amount based on the speed of the vehicle.
X-axis = speed, y-axis = pitch amount.

Lean yaw curve – Adjusts the yaw amount based on the speed of the vehicle.
X-axis = speed, y-axis = yaw amount.

Endo speed threshold – Vehicles will not be able to perform endos (forward wheelies) above this speed.

Pitch exponent – Exponent for pitch input.

Slide lean factor – Adjusts how much the vehicle leans when sliding sideways.

Steer assist curve – How much to assist steering based on speed. X-axis = speed, y-axis = assist amount.

Tire Shader

The included tire shader has a vertex function that deforms the tire mesh based on a Deform Map and a Deform Normal.

The Deform Map is a grayscale texture defining how much each vertex should move. White = full deformation, black = no deformation. The texture should be wrapped such that the tire does not deform where it connects to the rim.

The Deform Normal is a vector determining the direction of deformation. The magnitude controls how much it deforms. The w-component of the vector is not used.

The *Tires Bumped* shader supports extra textures for occlusion and normal mapping, as well as a slider for the smoothness of the material. These properties behave like the standard shader.

The tire geometry itself should roughly represent a cylinder. Any grooves should be in the normal map, as grooves modeled into the mesh itself will not deform properly. This is because tire deformation is based on a normal vector, not a plane, and vertices are displaced based on their normals.