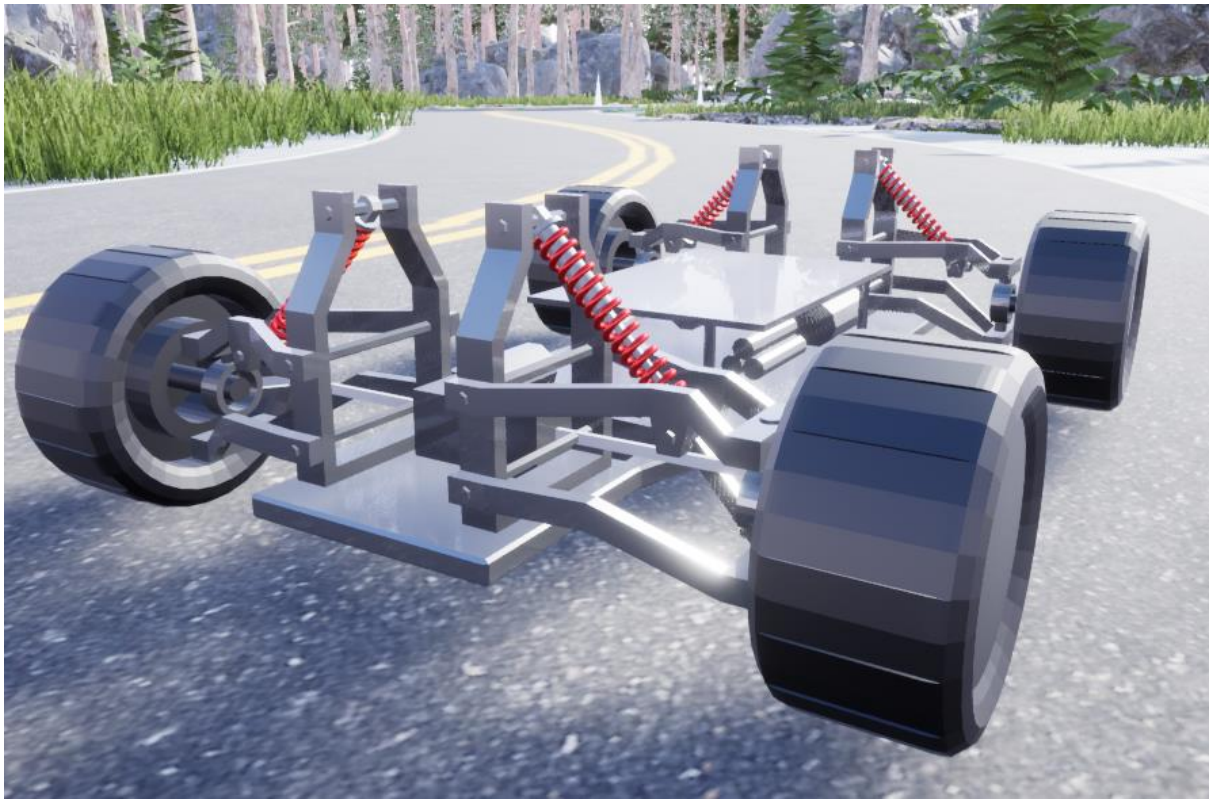
HOCHSCHULE HEILBRONN

Projektdokumentation



Visualisierung von Fahrsituationen eines RC-Autos im fahrdynamischen Grenzbereich

Bearbeiter:	Yahulan Gunasegaran	(188678)
	Daniel Körschner	(206267)
	Pascal Keller	(207475)
	Alexander Schmehlik	(208458)
Betreuer:	Prof. Dr.-Ing. Ansgar Meroth	
Fach:	Mastervorlesung Computergrafik & Multimedia	
Zeitraum:	Wintersemester 2020/21	

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

Im Rahmen der Vorlesung „Computergrafik und Multimedia“ sollen in einer Entwicklungsumgebung für Computerspiele statische und dynamische Objekte visualisiert werden. Üblicherweise basiert die Aufgabe dieser Lehrveranstaltung auf einer parallel dazu laufenden bzw. abgeschlossenen Projektarbeit (bspw. Labor-, Seminar- oder Bachelorarbeit). Dadurch entfällt gegebenenfalls die aufwändige Erstellung einer Simulation, da diese Aufgabe bereits durch das Hauptprojekt abgedeckt wurde. Somit kann der Fokus in dieser Vorlesung auf die grafische Veranschaulichung der Simulation gelegt werden. Insbesondere für Demonstrationszwecke bietet die Visualisierung von Simulationen einen großen Mehrwert.

1.1. Inspiration für das Projekt

Das von unserer Gruppe bearbeitete Projekt basiert auf einer fortlaufenden Projektarbeit bei Professor Wild. Im Rahmen dieser Arbeit soll am Beispiel eines umgebauten RC-Offroad-Buggys die Funktionsweise einer Antriebsschlupfregelung demonstriert werden (siehe Abbildung 1). Außerdem wird aktuell daran gearbeitet, das Verhalten des Fahrzeugs im fahrdynamischen Grenzbereich durch eine integrierte Fahrdynamikregelung zu stabilisieren. Die Hinterräder des Fahrzeugs werden unabhängig voneinander von jeweils einem Elektromotor angetrieben. In dieser Arbeit soll die Mechanik und Physik des RC-Autos modelliert und einige Fahrsituationen im fahrdynamischen Grenzbereich visualisiert werden.

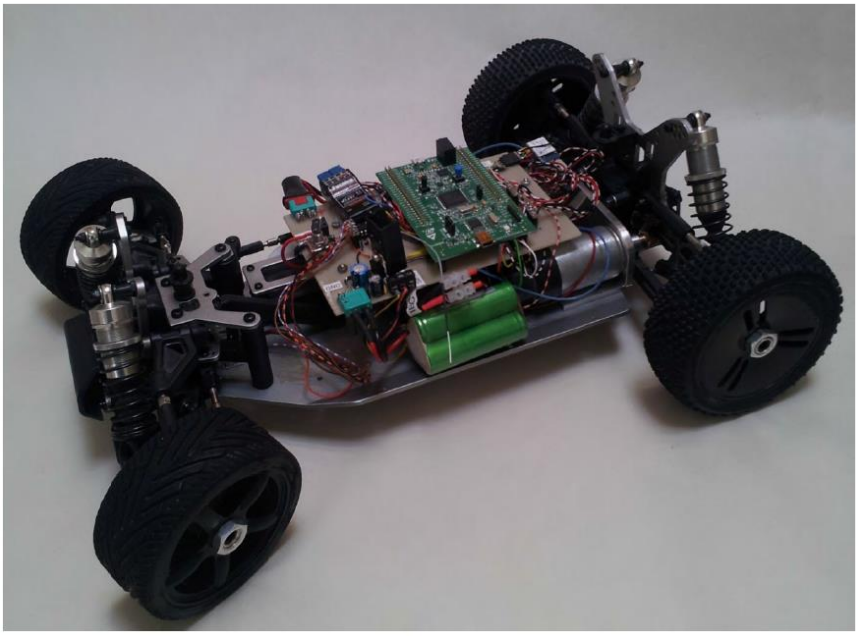


Abbildung 1: Umgebauter Offroad-Buggy des Masterprojekts

1.2. Definition der Ziele

Im Folgenden werden die Ziele definiert, die im Rahmen der in dieser Lehrveranstaltung durchgeführten Projektarbeit angestrebt werden:

❖ Modellierung des RC-Autos

Im ersten Schritt sollen die mechanischen Bauteile des Modellauto modelliert werden. Dies soll mit dem Grafikprogramm Blender umgesetzt werden. Hierbei wird der Fokus vor allem auf die realitätsgetreue Darstellung und Animation der Lenkung und des gefederten Fahrwerks gelegt.

❖ Erstellen einer Fahrstrecke

Mit der Unreal Engine soll eine Fahrstrecke entworfen werden, auf der später das Fahrverhalten des RC-Autos demonstriert werden kann. Es soll außerdem eine stationäre Kreisfahrt umgesetzt werden. Hierfür ist zusätzlich eine kreisförmige Teststrecke notwendig.

❖ Integration eines manuellen und autonomen Fahrmodus

Es sollen insgesamt zwei Fahrmodi implementiert werden. Einerseits soll das RC-Auto wie in einem echten Computerspiel manuell bedienbar sein. Die Steuerung soll mit den gängigen Tasten WASD erfolgen. Außerdem soll ein autonomer Fahrmodus integriert werden, in dem das Fahrzeug selbstständig einem vorgegebenen Kurs folgt.

❖ Erstellen einer detaillierten Landschaft und Vegetation

Zum Abschluss soll die Umgebung der Fahrstrecke durch eine detaillierte Landschaft ergänzt werden. Die Texturen und Objekte (Gräser, Bäume, Steine, etc.) sollen aus Vorlagen aus dem Internet übernommen werden.

2. Modellierung des RC-Autos in Blender

In diesem Kapitel wird die Modellierung und das Rigging des Fahrzeugs in Blender erläutert. Blender ist eine Computersoftware, die eine dreidimensionale Szenenerstellung und Animation ermöglicht [1]. Blender ermöglicht neben der Modellierung auch die Funktionen Rigging, Animation, Simulation, Rendering und Motiontracking [2]. Außerdem können die BLENDER-Dateien in das Dateiformat FBX exportiert werden, was für den Import in die Unreal Engine notwendig ist. Die Abkürzung des Dateiformats FBX steht für das von Autodesk für 3D-Modelle entwickelte *Filmbox*-Format. Der Vorteil dieses Dateiformats ist die Kompatibilität für zahlreiche Programme und das Erhalten der implementierten Funktionen beim Export [3].

2.1. Konstruktion der mechanischen Bauteile

Da es viele Modelle im Internet gibt, die auch teilweise kostenlos downloadbar sind, wird in diesem Abschnitt nur auf die wesentlichen Bestandteile der Modellerstellung eingegangen und die wichtigsten Shortcuts des Programms erläutert. Im Internet gibt es beispielsweise auf Youtube zahlreiche ausführliche Tutorials zu den Funktionen, die Blender bietet. Die wichtigsten Shortcuts sind in Tabelle 1 zusammengefasst.

Tastenkürzel	**Effekt**
Shift-A	Neues Objekt erzeugen
Shift-S	Cursormenü
A	Alles auswählen
Strg-R	Loop-Cuts (erzeugt neue Kanten)
E	Extrudieren
Strg-S	Skalieren
Strg-J	Objekte zusammenfügen
Strg-P	Parenting (zuletzt ausgewähltes Objekt wird zum Parent)
Strg-L	In sich geschlossenes Objekt auswählen (Editiermodus)
Alt-P	Parenting auflösen
Tab	Wechseln zwischen Editier- und Objektmodus
Y	Trennen von Knochen

Tabelle 1: Übersicht der Tastenkombination

Um den Modellierungsaufwand in einem angemessenen Rahmen zu halten, wird das Fahrzeug auf die essenziellen Bestandteile des Fahrwerks reduziert. Hierzu gehören:

- Fahrzeugkörper
- Federung
- Oberer Federungsarm
- Unterer Federungsarm
- Achsschenkelbolzen (Kingpin)
- Lenkachse
- Rad mit Radachse

Die folgende Abbildung 2 zeigt den generellen Aufbau einer Double-Wishbone-Suspension im Vergleich zu unserem Modell.

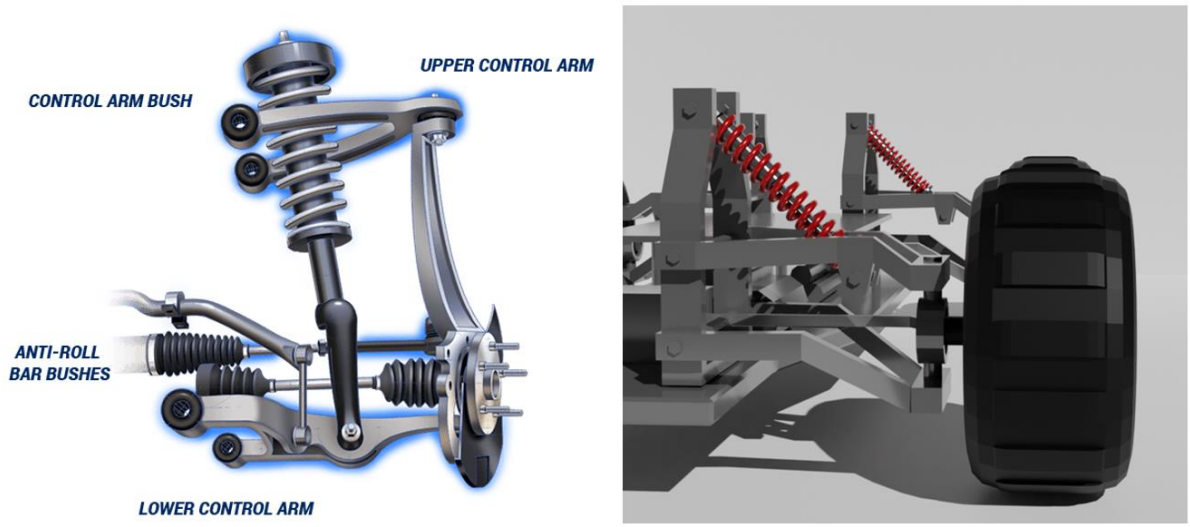


Abbildung 2: Genereller Aufbau einer Double-Wishbone-Suspension im Vergleich zu unserem Modell [4]

Für symmetrische Bauteile bietet es sich an, den *Mirror-Modifier* zu verwenden. Dieser sollte nach dem Abschluss des Modellierungsprozesses im Objekt-Modus angewendet werden.

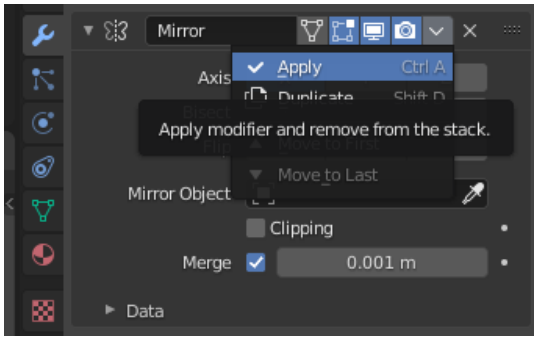


Abbildung 3: Anwenden von Modifieren

2.2. Definition des Zusammenspiels beweglicher Teile

Für das Erstellen eines *Rigs* in Blender müssen sogenannte Knochen eingefügt werden. Dies geschieht im Objektmodus analog zu den *Meshes* mit dem Shortcut *Shift-A.* Hierbei wird *Armature* ausgewählt, wie es in der Abbildung 4 zu sehen ist.

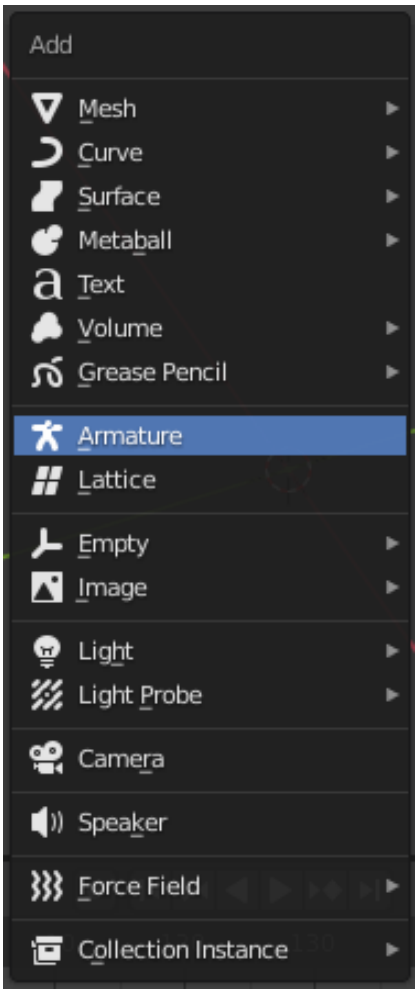


Abbildung 4: Erstellung eines Knochens

Alle weiteren Knochen werden im Editiermodus als Kinder des Root-Knochens erstellt. Der Root-Knochen wird dadurch als Eltern-Knochen definiert und kann die referenzierten Kinder-Knochen manipulieren. Um einen Kinder-Knochen zu erstellen, muss nach der Auswahl des Root-Knochens die Taste *E* zum Extrudieren gedrückt werden. Zusätzlich kann *X*, *Y* oder *Z* betätigt werden, um den Knochen in einer bestimmten Achsausrichtung zu erzeugen. Da der erzeugte Knochen noch mit dem Eltern-Knochen verbunden ist, muss die Verbindung in diesem Fall mittels *Y* getrennt werden (siehe Abbildung 5).

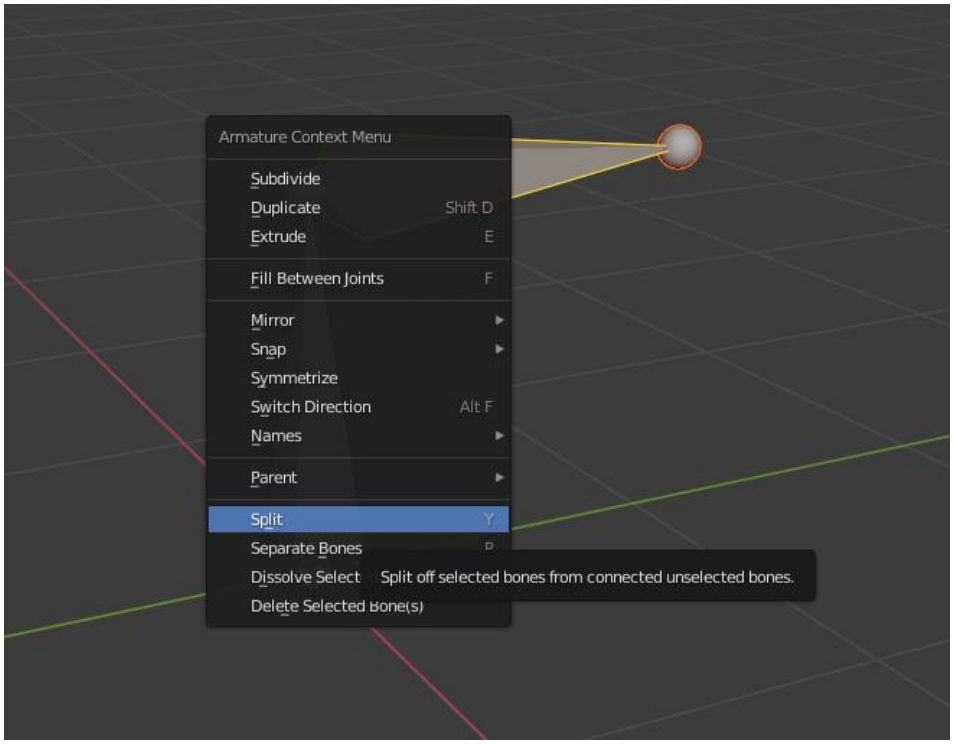


Abbildung 5: Trennen von Knochen

Der erstellte Knochen kann dann durch den Shortcut *Shift-D* beliebig oft dupliziert werden. Im Modell des RC-Autos wurden für jeden Reifen bis zu acht Kinder-Knochen hinzugefügt. Einige dieser Knochen besitzen weitere Kinder-Knochen. In der nachfolgenden Abbildung 6 ist die Struktur des Vorderrads zu sehen.

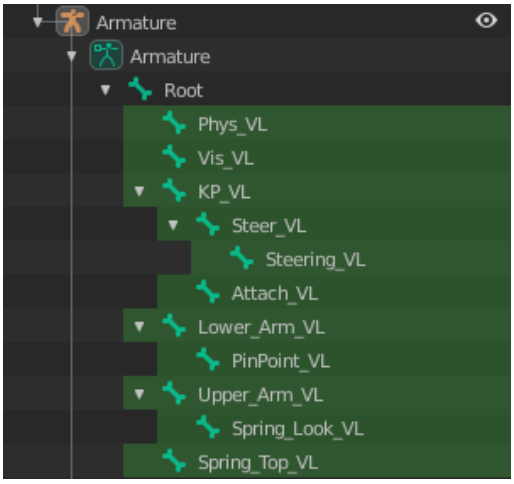


Abbildung 6: Knochenstruktur vom vorderen linken Rad

Die Knochen werden im Editormodus als Minimalgerüst in den *Meshes* platziert. Um Animationsprobleme zu verhindern, sollten die Anfangs- und Endpunkte (*Heads* and *Tails*) so genau wie möglich positioniert werden. In der nachfolgenden Abbildung 7 kann man erkennen, wie unter Zuhilfenahme eines Cursors eine präzise Platzierung erfolgt.

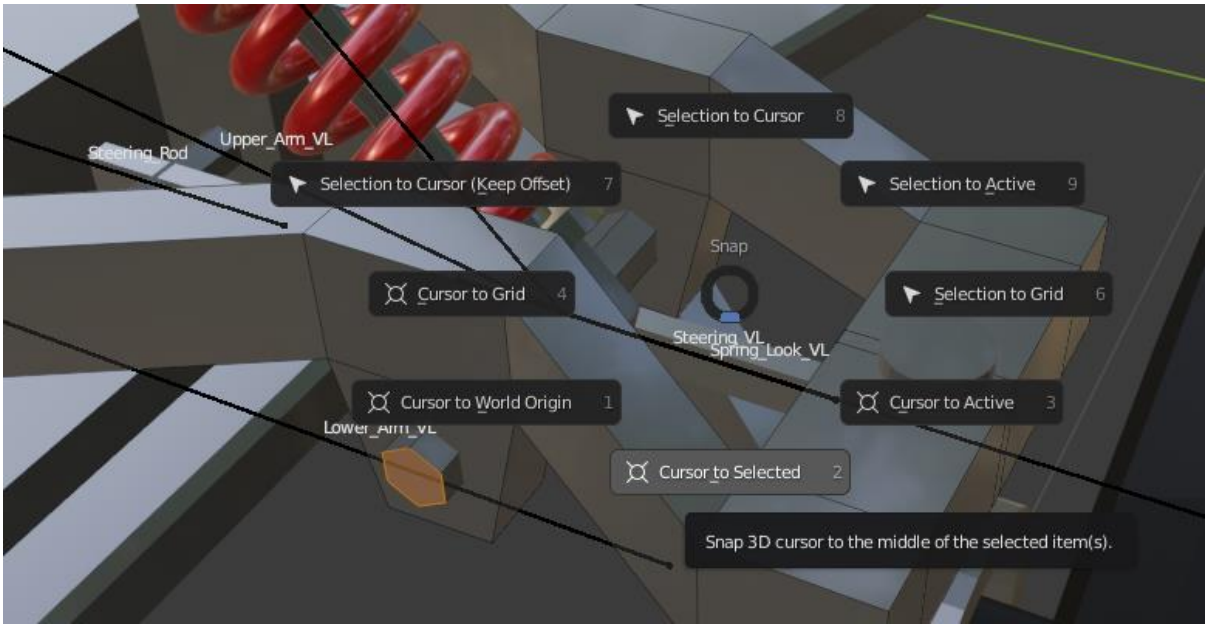


Abbildung 7: Platzierung des 3D-Cursors durch Auswahl von Objektflächen und Kanten

Durch die Auswahl von *Vertices* im Editiermodus und der Option *Cursor to Selected* im Radialmenü, dass mittels *Shift-S* aufgerufen werden kann, wird der Cursor im Zentrum der Auswahl platziert. Anschließend kann der Anfangs- bzw. Endpunkt des Knochens ausgewählt werden und ebenfalls über das Radialmenü mit der Auswahl *Selection to Cursor* platziert werden (siehe Abbildung 8).

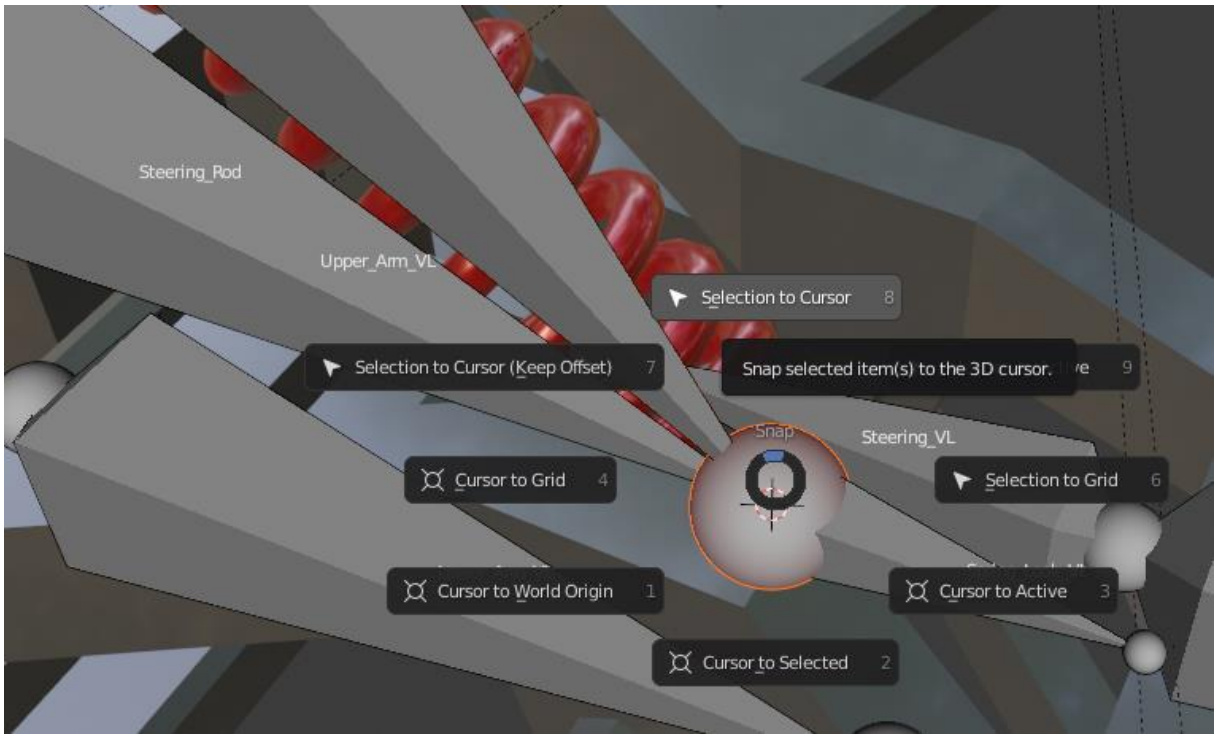


Abbildung 8: Platzierung von Knochen mittels Cursors

Bevor man die Knochen auf diese Art und Weise platziert, sollten diese jedoch vorab von Hand so genau wie möglich vorpositioniert werden, da es ansonsten zu ungewollten Rotationen kommen kann. Dies wird allerdings erst dann auffällig, wenn man versucht, das Model zu animieren. In der folgenden Abbildung 9 und Abbildung 10 ist der Aufbau der in diesem Beispiel als *Sticks* dargestellten Knochen zu erkennen.

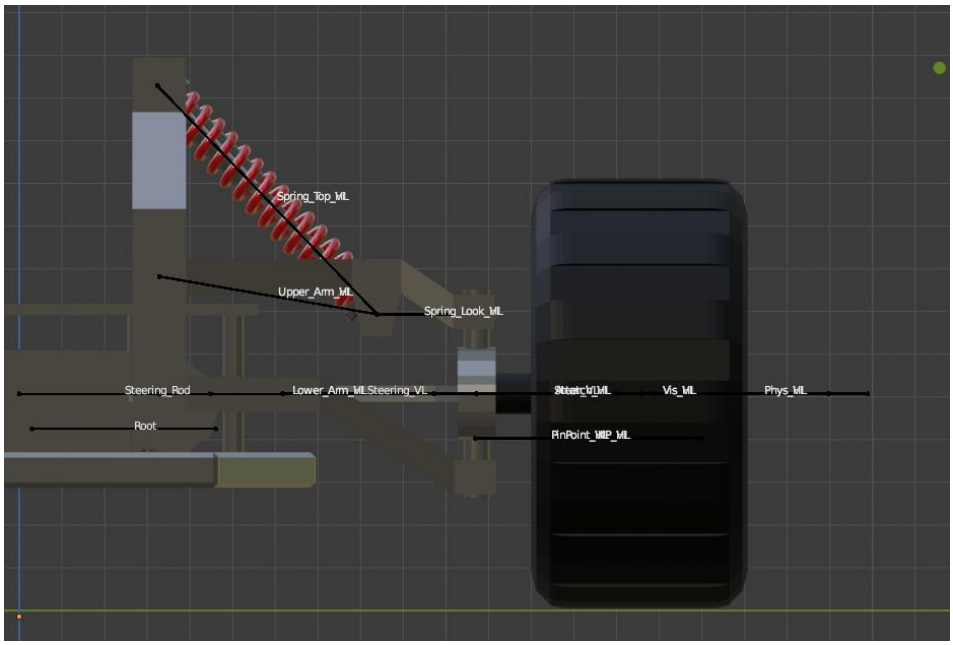


Abbildung 9: Vorderansicht der Knochenpositionen

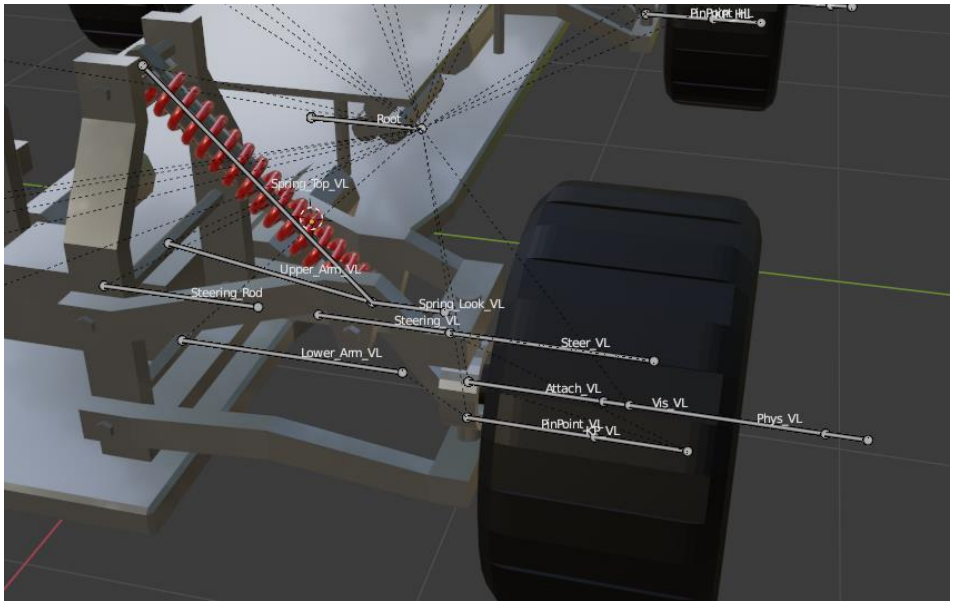


Abbildung 10: Perspektivische Ansicht der Knochenpositionen

Neben den Positionen ist auch die Ausrichtung der Knochenachsen wichtig. Alle Achsen der Root-Kinder *Attach*, *Phys*, *Vis* und *Steering* müssen fluchten. In Abbildung 11 sind die Koordinatensysteme aller Knochen der gesamten Vorderachse dargestellt.

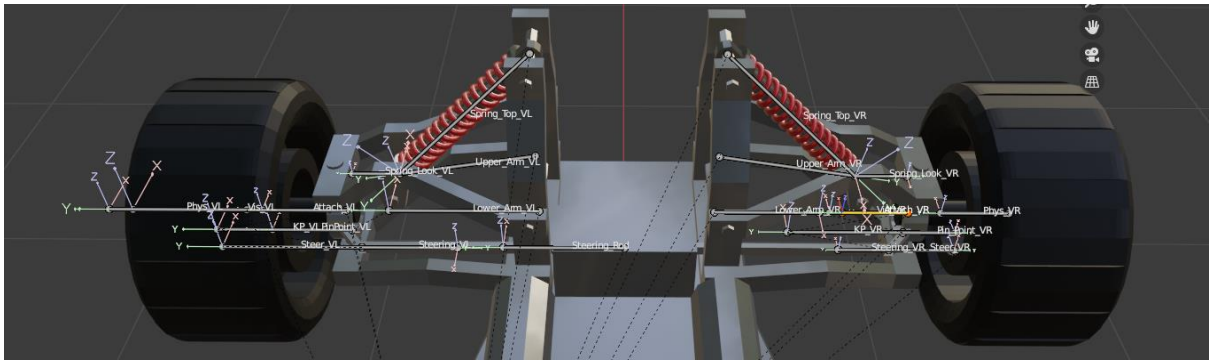


Abbildung 11: Koordinatensysteme der Knochen der Vorderachse

Auf der Achse des Rads befindet sich neben dem Knochen *Vis_VL*, der mit dem eigentlichen Rad-Mesh verbunden ist, auch der Knochen *Phys_VL* und *Attach_VL*. Ähnliches gilt für die Knochen *KP_VL, PinPoint_VL*, *Steer_VL* und *Steering_VL*. Im Folgenden Kapitel wird auf die Bedeutung der Knochen eingegangen, deren Position nicht direkt mit einem Mesh verknüpft ist.

Das Verknüpfen der Meshes mit den Knochen kann durch zwei Methoden erfolgen. Mit der *Weight-Paint*-Methode werden alle Bereiche „angemalt“, die dem Knochen folgen sollen. Dies eignet sich vor allem, wenn man statt mehreren kleinen Meshes nur ein großes *Mesh* hat. Die zweite Methode, die auch in dem Projekt verwendet wurde, ist das *Parenting* der Meshes mit den Knochen mit leeren Gruppen. Hierfür werden im Objektmodus zuerst alle *Meshes* ausgewählt und anschließend mit einem Shift-Klick das Bones-Objekt. Danach kann mit *Strg-P* im *Parenting*-Menü die Option *With Empty Groups* ausgewählt werden, wie es in Abbildung 12 zu sehen ist.

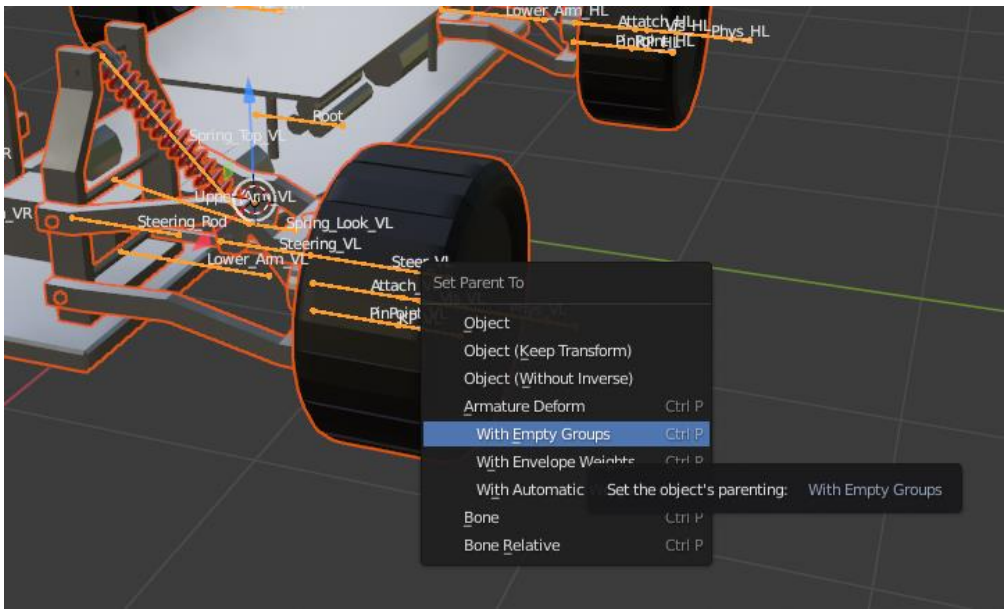


Abbildung 12: Parenting von Knochen und Meshes

Anschließend muss das Objekt ausgewählt werden, das mit einem Knochen verbunden werden soll. Im Editiermodus kann man dann mit der Taste *A* alle Vertices auswählen und in den *Object Data Properties* mit dem Button *Assign* den zugehörigen Knochen zuweisen [5]. In der folgenden Abbildung 13 ist dieses Vorgehen am Beispiel des vorderen linken Rads veranschaulicht.

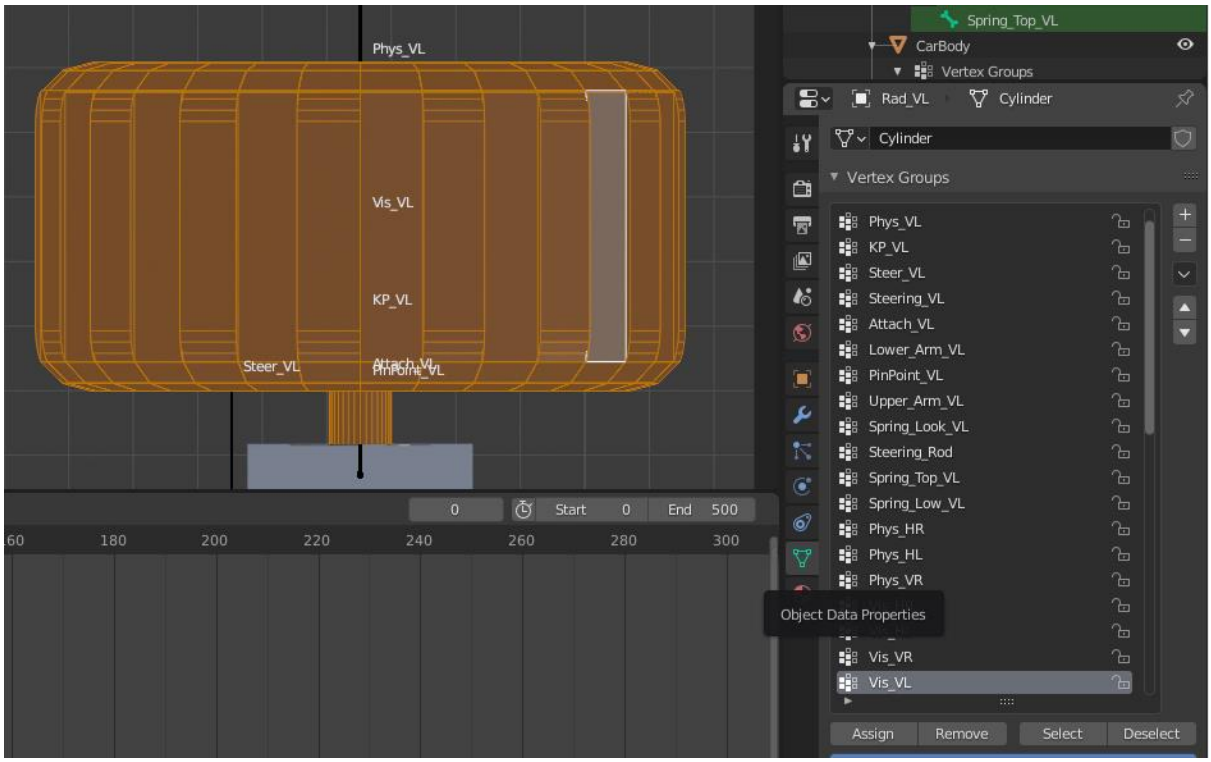


Abbildung 13: Verknüpfung eines Knochens mit dem Mesh über Vertex Groups

2.3. Animation von Bewegungen

Der Animationsablauf in Blender geschieht im Wesentlichen nach dem in Abbildung 14 dargestellten Schema.

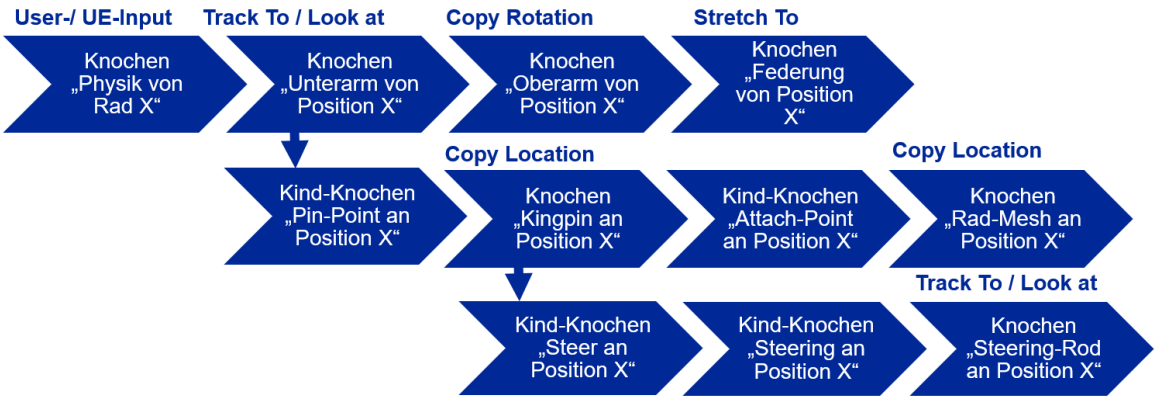


Abbildung 14: Schematischer Animationsablauf in Blender

Hierfür werden im Pose-Mode, der bei Auswahl der Knochen im Dropdown-Menü auswählbar ist, einigen Knochen *Track To*, *Copy Rotation* oder *Stretch To Constraints* zugewiesen. Für das Modell des RC-Autos bedeutet dies, dass wir die Animation der Federung über *Input* an dem Physik-Knochen auslösen. Dieser bewegt sich auf der Z-Achse hoch-/runter und der Unterarm-Knochen rotiert, da dieser über *Track To* auf den Physik-Knochen schaut. In Abbildung 15 ist der *Constraint* des Oberarm-Knochens dargestellt, der diese Rotierung mittels *Copy Rotation* kopiert.

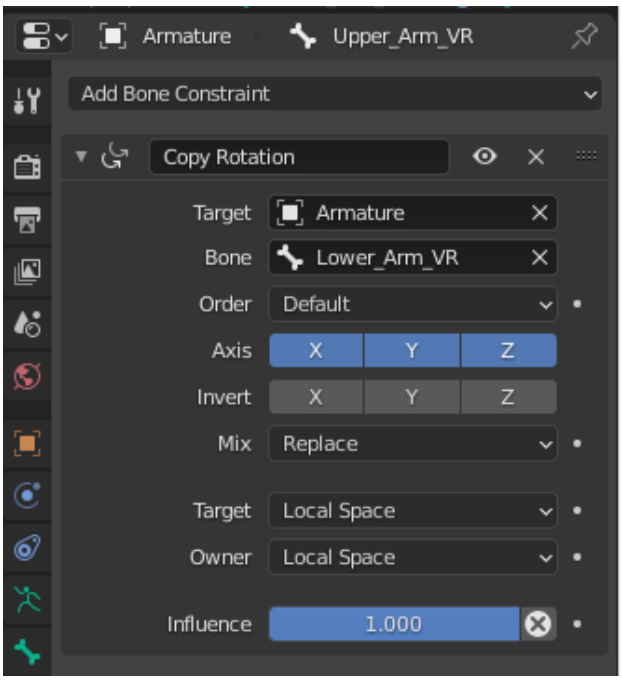


Abbildung 15: Knochen-Constraint des Oberarmknochens

Der Federknochen wird durch *Stretch To* auf das Ende des Oberarmknochens gedehnt oder gestaucht. Für Unreal wurde bei der Federung ein zusätzlicher Knochen am Ende des Oberarmknochens eingefügt, da es in Unreal nicht die Möglichkeit gibt, auf den Endpunkt des Knochens, statt dem Anfangspunkt zu schauen.

Die Rotation des Unterarms bewirkt gleichzeitig die Bewegung seines Kindes (*Pin-Point*), dessen Position, aber nicht dessen Ausrichtung, vom *KP*-Knochen kopiert wird. Durch den KP-Knochen wird wiederum dessen Kind bewegt (*Attach-Point*) und der *Vis*-Knochen, der mit dem Rad-*Mesh* verbunden ist, kann dessen Position kopieren. Dies ermöglicht eine drehungsfreie und bogenförmige Dämpfungsanimation.

Für die Berücksichtigung der Lenkung muss der *KP*-Knochen zusätzlich die Rotation des Physikknochens um die Z-Achse kopieren. Hierdurch wird der Kind-Knochen *Steer* verschoben. Dessen Kinder-Knochen *Steering*, der teilweise mit dem *Mesh* der Lenkachse verbunden ist, wird mitbewegt und schaut mittels *Trach To* auf den Knochen *Steering_Rod*.

Prinzipiell ist das Zuordnen der Funktionen in Blender nicht unbedingt notwendig, da diese nicht nach Unreal exportiert werden können. Auf diese Art und Weise kann man jedoch frühzeitig Probleme identifizieren, die durch eine falsche Orientierung der Knochen entstehen können.

Außerdem kann man nach dem Zuordnen einen *Render* erzeugen, in dem die Bewegungen demonstriert werden. Hierfür nutzt man die sogenannten *Keyframes*. Hierbei handelt es sich um Momentaufnahmen der Positionen einzelner Knochen. Im Modell des RC-Autos ist der Knochen, der alles bewegt, der Physik-Knochen des jeweiligen Rades. Die Aufnahme von *Keyframes* wird durch den *Record*-Button im unteren Timeline-Editor gestartet (siehe Abbildung 16).

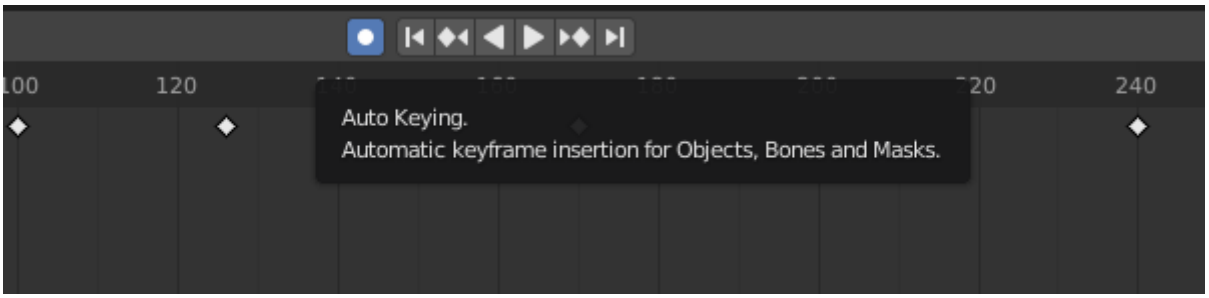


Abbildung 16: Aufnahme von Keyframes

Möchte man nun beispielsweise das vordere linke Rad federn lassen, setzt man zuerst einen *Keyframe*, bei dem der Physik-Knochen bei Z = 0 ist. Anschließend geht man 25 Frames in der Timeline weiter und setzt einen zweiten *Keyframe*, bei dem der Z-Wert = 5 ist. Dadurch bewegt sich der Physik-Knochen bei einer Bildrate von 25 FPS innerhalb einer Sekunde von Z = 0 auf Z = 5. Nachdem alle Keyframes erstellt wurden, kann eine Render-Engine gewählt werden und mittels *Strg-F12* das Rendern der Animation gestartet werden.

2.4. Export zum Dateiformat FBX

Bevor die BLENDER-Datei in das Dateiformat FBX exportiert wird, sollten alle Transformationen angewendet und das Modell falls nötig skaliert werden. Um alle Transformationen anzuwenden wird zuerst im Objektmodus das gesamte Modell ausgewählt. Anschließend muss über den Shortcut *Strg-A* das *Apply*-Menü aufgerufen und die Schaltfläche *All Transforms* betätigt werden (siehe Abbildung 17).

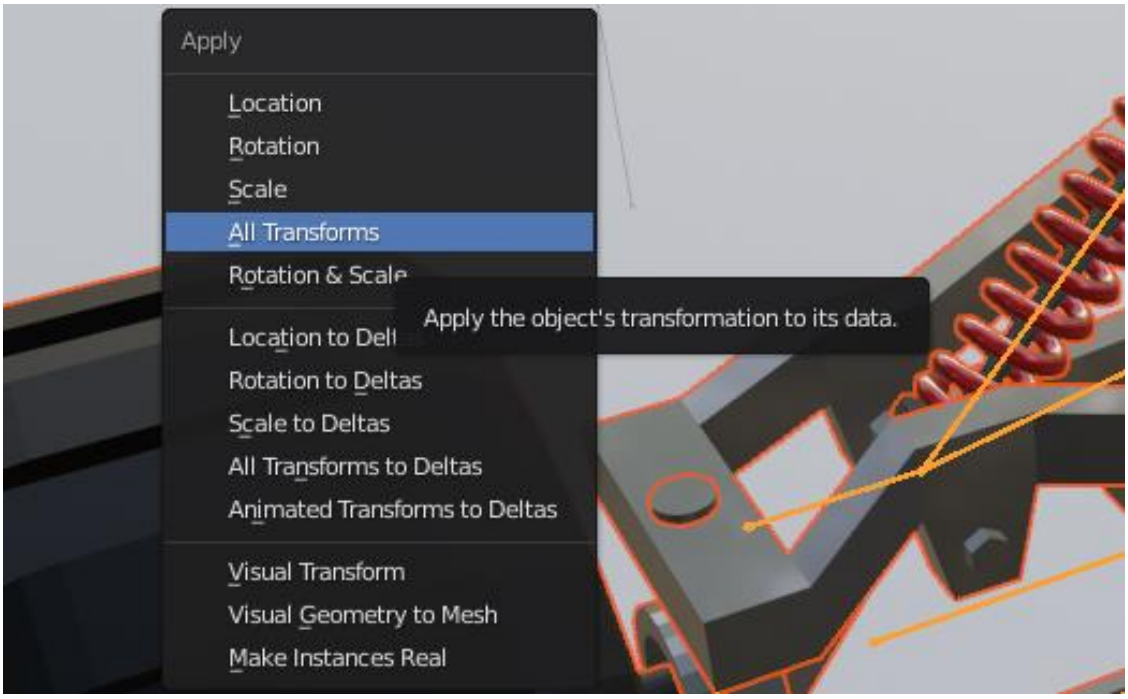


Abbildung 17: Anwenden der Transformationen

Während das Objekt markiert ist, kann dieses über *File → Export → FBX* in das Dateiformat FBX exportiert werden. Im Exportierungsmenü sollten folgende Einstellungen vorgenommen werden.

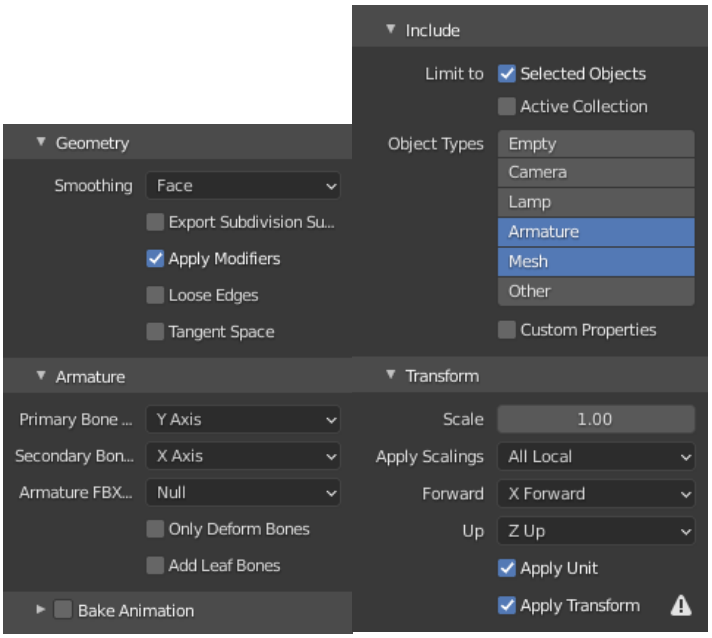


Abbildung 18: Einstellungen zur Exportierung in Blender

Hierbei ist vor allem darauf zu achten, dass die *Forward*- und *Up*-Achsen richtig eingestellt sind. Außerdem wurde festgestellt, dass es zu Problemen führen kann, wenn die Funktion *Bake Animation* aktiviert ist. Daher sollte diese Option deaktiviert werden.

3. Implementierung der Fahrphysik in Unreal

In diesem Kapitel wird näher auf die Implementierung der Fahrphysik in Unreal eingegangen. Insbesondere geht es hierbei um die Einbindung der Objekte aus Blender in Unreal und welche Steuerungsmöglichkeiten in diesem Projekt konkret entwickelt worden sind. Für die Erstellung wurde die Anleitung der *Unreal Engine Docs* und darauf basierende Videos verwendet [6].

3.1. Importieren der FBX-Datei

Bei der FBX-Datei ist auf die richtige Skalierung des Modells zu achten. Blender verwendet als Basis die Einheit Zentimeter, Unreal die Einheit Meter. Die Skalierung kann auch nachträglich geändert werden. Komfortabler ist es allerdings, die Skalierung direkt beim Import vorzunehmen. Die FBX-Datei importiert das Mesh, das Skelett des Modells und die Physics-Assets. Die *Physic-Assets* müssen noch wie im folgenden Kapitel beschrieben angepasst werden.

3.2. Animation der beweglichen Bauteile

Damit das Modell mit der restlichen Simulationsumgebung interagieren kann, müssen vorerst die sogenannten *Physic-Assets* definiert werden. Diese werden für spätere Kollisionsberechnungen benötigt. Die Umsetzung ist in Abbildung 19 zu sehen.

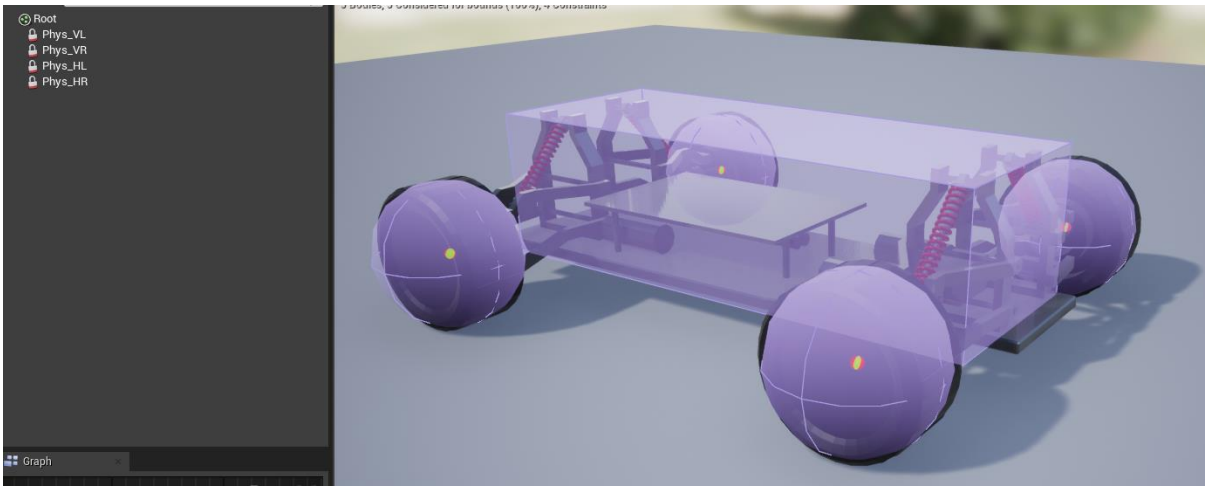


Abbildung 19: Umsetzung der Kollisionsboxen für das RC-Auto

Der Root-Knochen benötigt eine Box und die physischen Radknochen werden als Kugeln dargestellt. Bei den Rädern ist darauf zu achten, dass in den Einstellungen unter *Physics-type* die Option *Kinematic* ausgewählt ist, damit die Fahrwerksberechnung der Engine korrekt funktioniert.

3.2.1. Animation des Fahrwerks

Im *Animation Graph* wird die Animation des Fahrwerks definiert. Der *Animation Graph* muss das Skelett der FXB-Datei referenzieren. Der Zusammenhang der Knochen wird wie in Abbildung 14 dargestellt aufgebaut. Im Unterschied zu Blender ist der *Animation Graph* in der Unreal Engine sequenziell aufgebaut. Daher müssen die Schritte der Animation nacheinander eingeben werden. Für das Modellieren von Fahrzeugen mit Rädern existiert der *Wheel-Handler.* Der *Animation Graph* gibt vor, wie sich die visualisierten Räder bewegen sollen. Ein Ausschnitt ist in Abbildung 20 dargestellt. Der *Wheel-Handler* ist Teil der *Wheeled-Vehicle*-Klasse, die im folgenden Kapitel beschrieben wird.

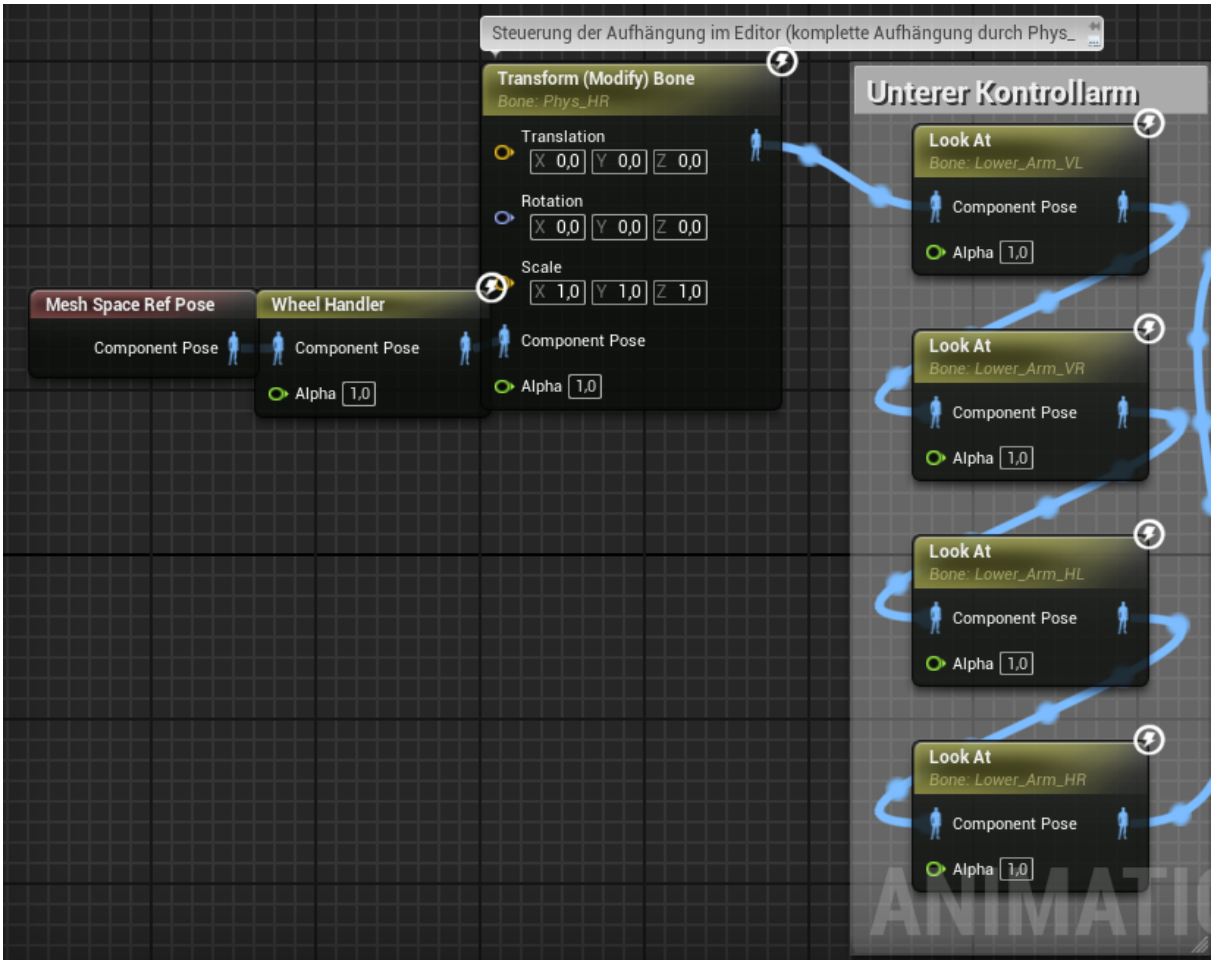


Abbildung 20: Ausschnitt des Animation Graph für das RC-Auto

Mit Hilfe des „*Transform (Modify) Bone*“-Blocks kann das Fahrwerk innerhalb des *Animation-Graph*-Editors getestet werden. Bei korrekter Implementierung der Fahrwerksanimation sollte die Bewegung des *Phys_HR*-Knochens das gesamte Fahrwerk hinten rechts steuern. Erwähnenswert ist, dass zuerst alle Knochen eines Typs verknüpft werden müssen, bevor weitere Verbindungen erstellt werden können. Vor der Implementierung der Radaufnahme müssen daher alle unteren Kontrollarme des Fahrwerks definiert werden.

3.2.2. Animation der Räder

In der Unreal Engine existiert die *Wheeled-Vehicle* Klasse, welche ein Grundgerüst für die Erstellung von mehrrädrigen Fahrzeugen zur Verfügung stellt. Für das Projekt wurde eine Instanz dieser Klasse für das RC-Auto erstellt. Im nächsten Schritt müssen das Fahrzeug-*Mesh* und der *Animation-Graph* der Klasse zugewiesen werden (Abbildung 21).

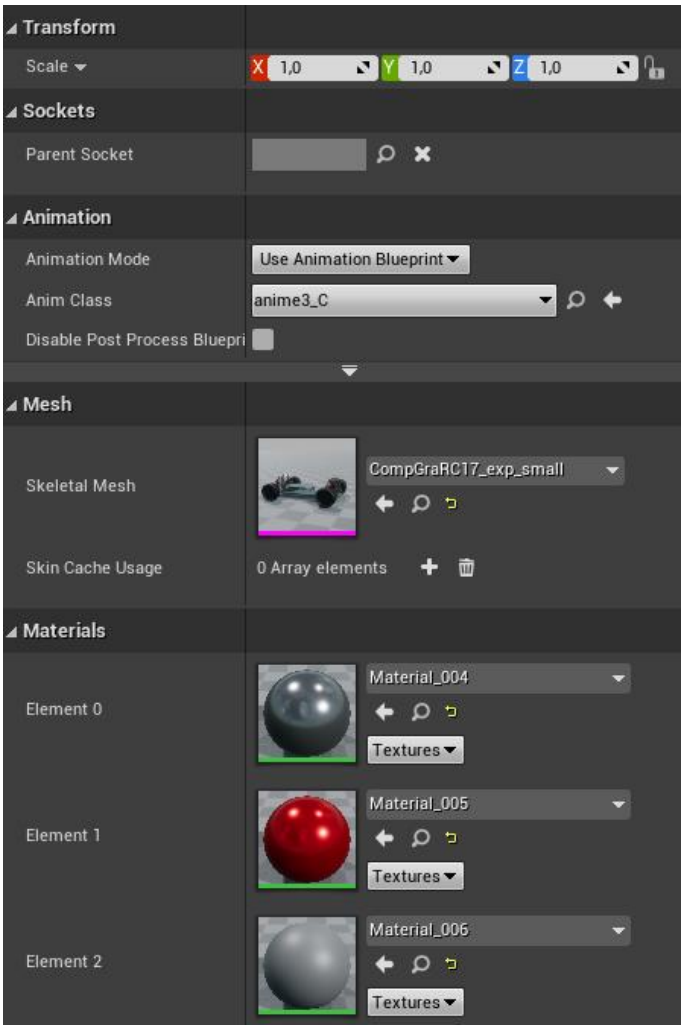


Abbildung 21: Zuweisung des Meshs und des AnimationGraph für das RC-Auto

Im nächsten Schritt werden die Räder des Fahrzeugs erstellt. Hierbei handelt es sich um zwei Instanzen der „*Vehicle Wheel*"-Klasse, welche für die Physikberechnung des Modells benötigt wird. In der Klasse können verschiedene Parameter des Fahrwerks vorgegeben werden. Hierzu gehören beispielsweise die Radumfänge, Feder- und Dämpferkonstanten und die Reibungskoeffizienten der Reifen auf dem Untergrund (Data Asset „TireConfig" notwendig). Es wird sowohl für die Vorder- als auch Hinterräder jeweils eine Instanz benötigt. Dies liegt in erster Linie an den unterschiedlichen Lenkwinkeln der Vorder- und Hinterachse. Außerdem können durch die unterschiedliche Parametrisierung die dynamischen Fahreigenschaften des Fahrzeugs verändert werden.

Die erstellten Räder *Hinterräder* und *Vorderräder* müssen in der *Wheeled-Vehicle*-Klasse unter V*ehicle Movement* auf die gewünschten Knochen bezogen werden. Außerdem muss der Offset der Räder zum Root-Knochen vorgegeben werden, sonst legt die Unreal Engine die Hitboxen für die Berechnung direkt auf den Root-Knochen. Die Umsetzung ist in Abbildung 22 zu sehen.

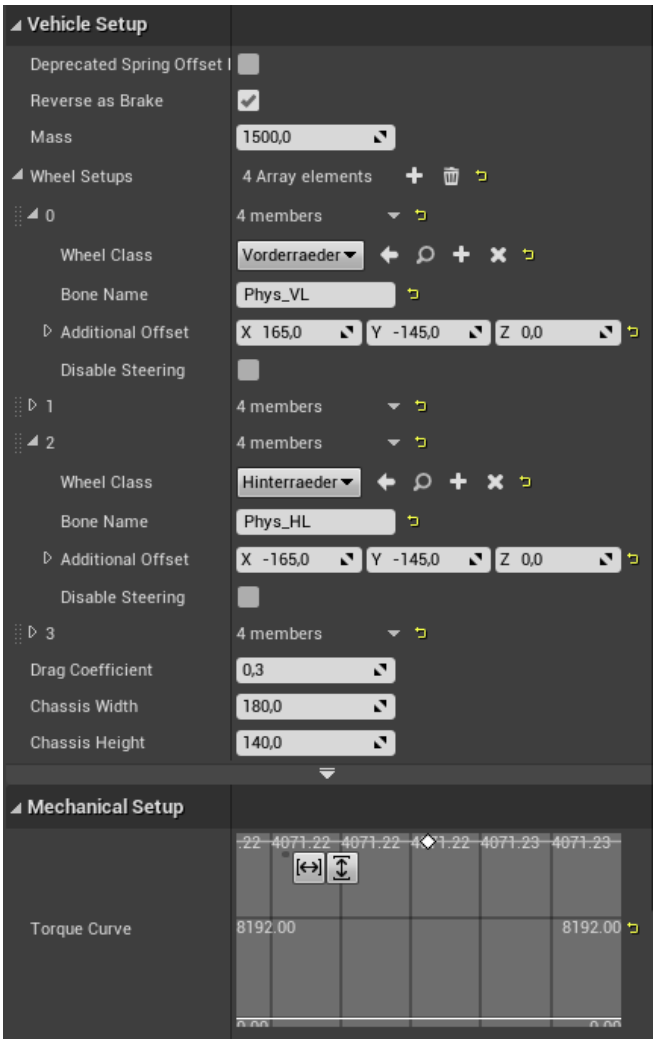


Abbildung 22: Vehicle Movement Optionen der Wheeled-Vehicle Klasse

In den Optionen von *Vehicle Movement* sind auch die Parameter des Antriebsstrangs des RC-Autos untergebracht. Zu den Wichtigsten gehören die Drehmomentkurve des Motors sowie die maximale Drehzahl, mit der die Höchstgeschwindigkeit variiert werden kann. Im *Differential-Setup* kann eingestellt werden, ob das Fahrzeug einen Front-, Heck-, oder Allradantrieb besitzt und wie das Drehmoment auf die Vorder- und Hinterachse bzw. linken und rechten Seite aufgeteilt wird. In unserem Beispiel verwendet das RC-Auto einen Heckantrieb mit einem offenen Differentialgetriebe (*Open Rear Drive*). Mithilfe des Transmission-Setups kann ein Getriebe simuliert werden. Hierfür kann die Übersetzung des Getriebes sowie die Schaltzeiten und die Wahl eines manuellen oder automatischen Getriebes vorgegeben werden.

3.3. Implementierung der Fahrtmodi

3.3.1. Autonomes Fahren

Das autonome Fahren wird mithilfe eines über die Fahrbahn gelegten Splines realisiert. Hierfür wurde die *Blueprint*-Klasse *Splinefollow* angelegt und mit den notwendigen Funktionen ausgestattet. Zu Beginn der Simulation spawnt die Klasse ein Fahrzeug des Typs RC-Auto und setzt dieses als Referenz für die autonome Fahrt (Abbildung 23). Das ist notwendig, da die Unreal Engine manuell gespawnte Fahrzeuge nicht mit der *SplineFollow*-Klasse in Verbindung setzt.

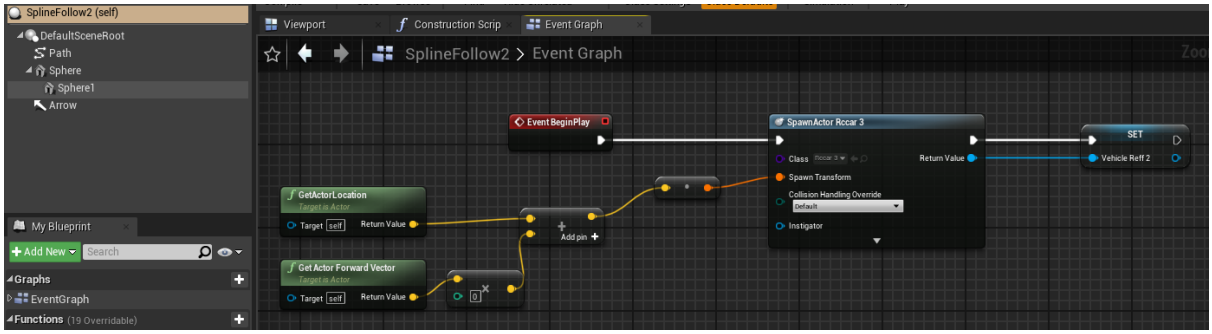


Abbildung 23: Bestandteile der Klasse SplineFollow

Regelung der Lenkung

Für das autonome Fahren wurden in der *Blueprint*-Klasse des RC-Autos jeweils eine Variable für die Lenkung und Beschleunigung angelegt. Für die Umsetzung der automatischen Lenkung berechnet die *SplineFollow*-Klasse die kürzeste Distanz zwischen dem Fahrzeug und dem in der Map erstellten Spline (Abbildung 24).

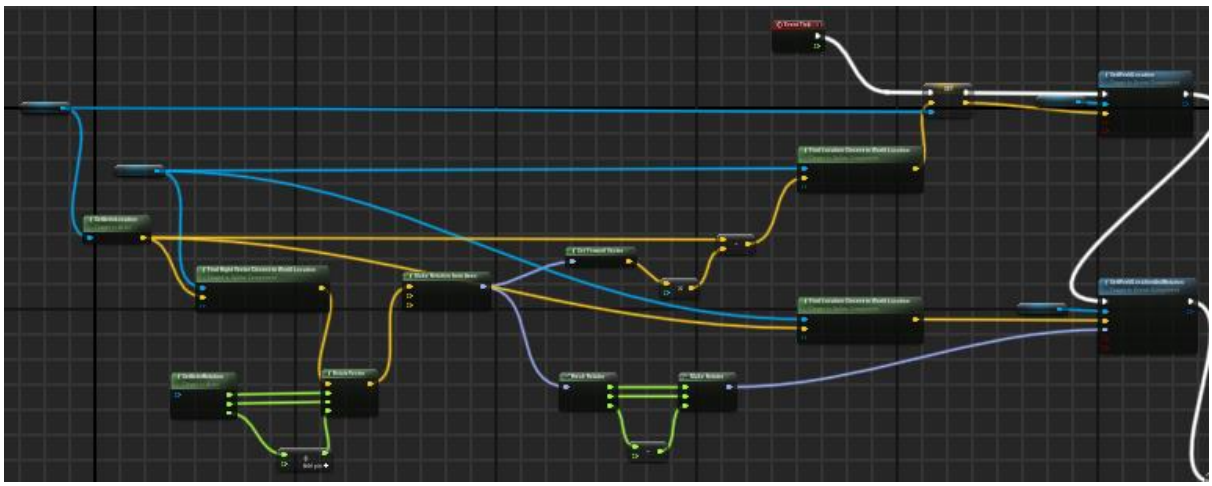


Abbildung 24: Berechnung des erforderlichen Lenkwinkels für das autonome Fahren

Aus dieser berechneten Distanz wird ein Wert für die Lenkung des Fahrzeugs berechnet, welche mit der *Steering*-Variable an die *Wheeled-Vehicle*-Klasse des RC-Autos übergeben wird. (Abbildung 25)

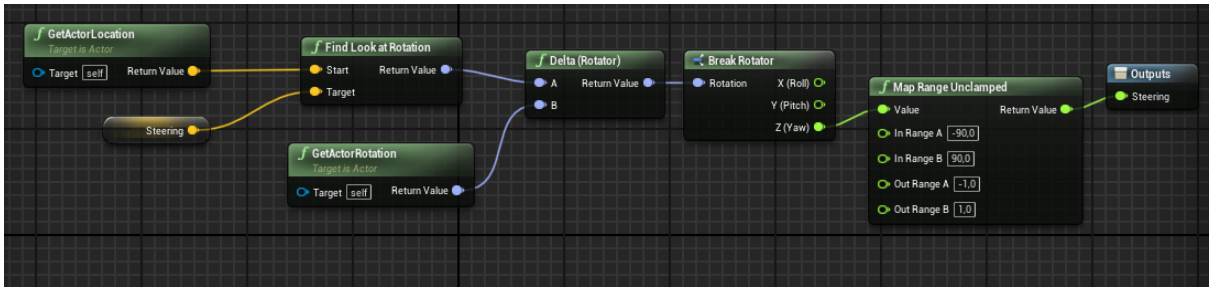


Abbildung 25: „Steering Calc"-Macro des RC-Auto

Mit dem Makro *Steering Calc* wird der Lenkwinkel, der in der *Steering*-Variable gespeichert ist, auf einen Wertebereich zwischen -1 und 1 normiert, um von einem *Set Steering Input* - Block verwendet werden zu können. Die Umsetzung der autonomen Lenkung wurde anhand eines Tutorials durchgeführt. [7].

Regelung der Geschwindigkeit

Die Geschwindigkeit wird ebenfalls im *Eventgraph* des *SplineFollow* berechnet (Abbildung 26).

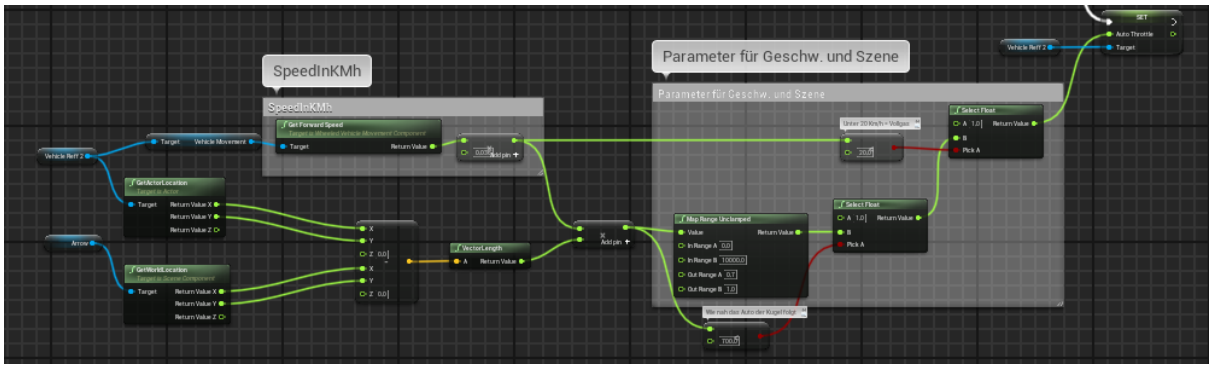


Abbildung 26: Berechnung der Beschleunigung u. Geschwindigkeit für das aut. Fahren

Hierfür wird der Abstand des Fahrzeugs zu einem auf dem Spline liegenden Pfeil berechnet. Zudem wird die momentane Geschwindigkeit des Fahrzeugs ausgewertet und mithilfe eines Vergleichs mit vorgegebenen Geschwindigkeitsparametern wird die Variable *Autothrottle* angepasst, welche an das RC-Auto übergeben wird. Die durch *SplineFollow* berechneten Werte für Lenkung und Beschleunigung werden dann an die *Set...Input*-Blöcke übergeben.Die Berechnung der Geschwindigkeit wurde mithilfe von [8] umgesetzt.

3.3.2. Manuelle Steuerung

Die Steuerung des Fahrzeugs wird im *Event Graph* der *Wheeled-Vehicle*-Klasse implementiert (Abbildung 27). Die Übergabe des Linkwinkels und die Position des Gaspedals werden über die *Set...Input*-Blöcke an den Simulator übergeben. „InputAxis MoveRight“ liest die *A* und *D* Tasten der Tastatur aus und liefert entweder 1 oder -1, je nach gedrückter Taste. In gleicher Weise funktioniert *InputAxis MoveForward* mit den Tasten W und S. Bei gedrückter Taste *L* kann das Fahrzeug für eine stationäre Kreisfahrt einen festen Lenkwinkel fahren und eine langsamere Beschleunigung aktivieren, als mit *InputAxis MoveForward* möglich wäre. Die Kreisfahrt wird solange durchgeführt, bis die *L*-Taste losgelassen wird. Die Tasten *M* und *F* werden verwendet, um zwischen manuellem und autonomem Fahren umzuschalten.

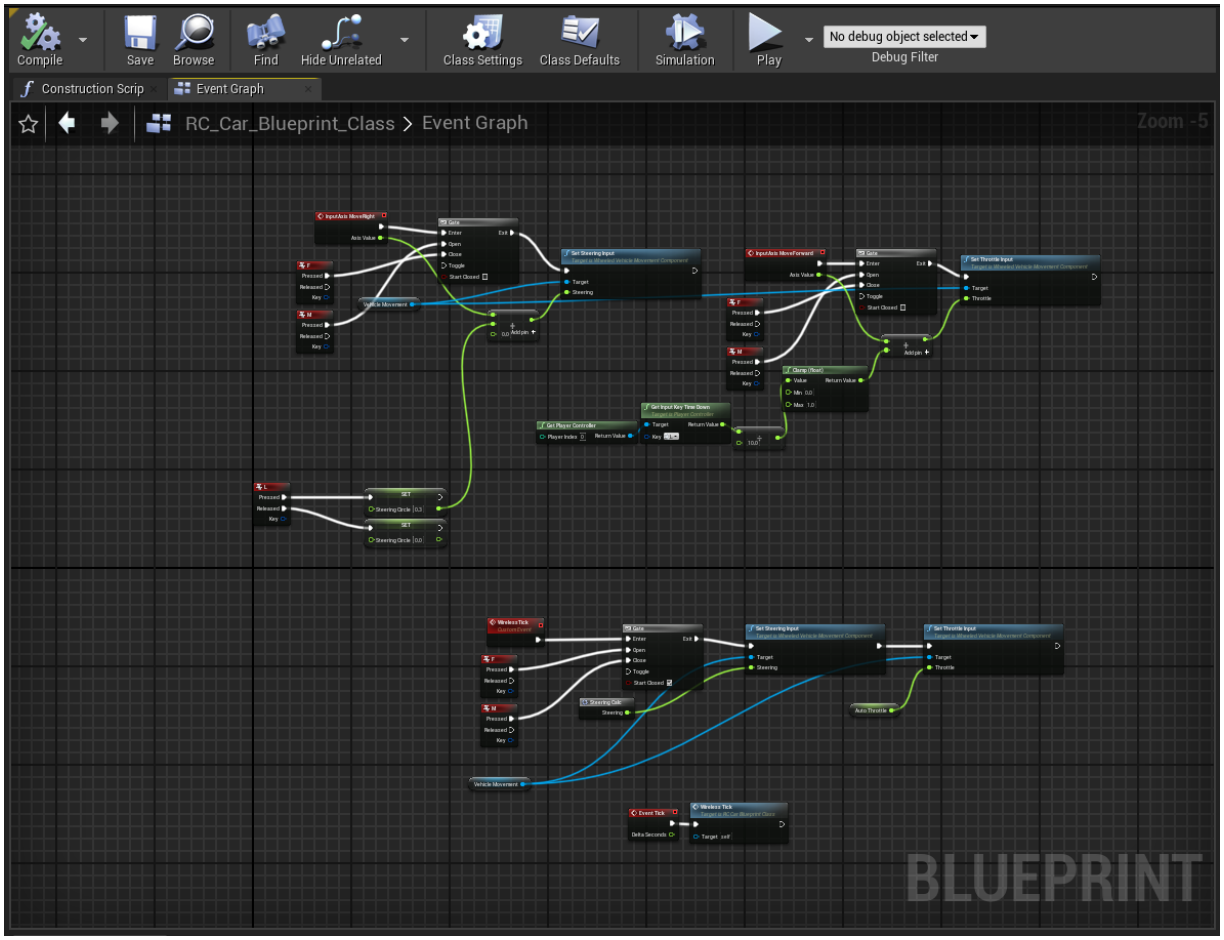


Abbildung 27: Stationäre Kreisfahrt und manuelle Steuerung des RC-Auto

4. Modellierung der Fahrstrecke

In diesem Kapitel wird auf die Erstellung einer Simulationsumgebung näher eingegangen, in dem sich unser Fahrzeug bewegen wird. Da in der Technik mit Simulationsumgebung die Verwendung von Tools gemeint ist, wird im Folgenden die Simulationsumgebung als *Map* bezeichnet, wodurch Missverständnisse vermieden werden sollen.

Eine Map lässt sich über das *Landscape*-Tool in Unreal erstellen, welches wie in der Abbildung 28 auszuwählen ist. Um mit einer eigenen Landschaft zu beginnen, empfiehlt es sich ein neues *Level* zu generieren, auf der eine individuelle Map erstellt werden kann.

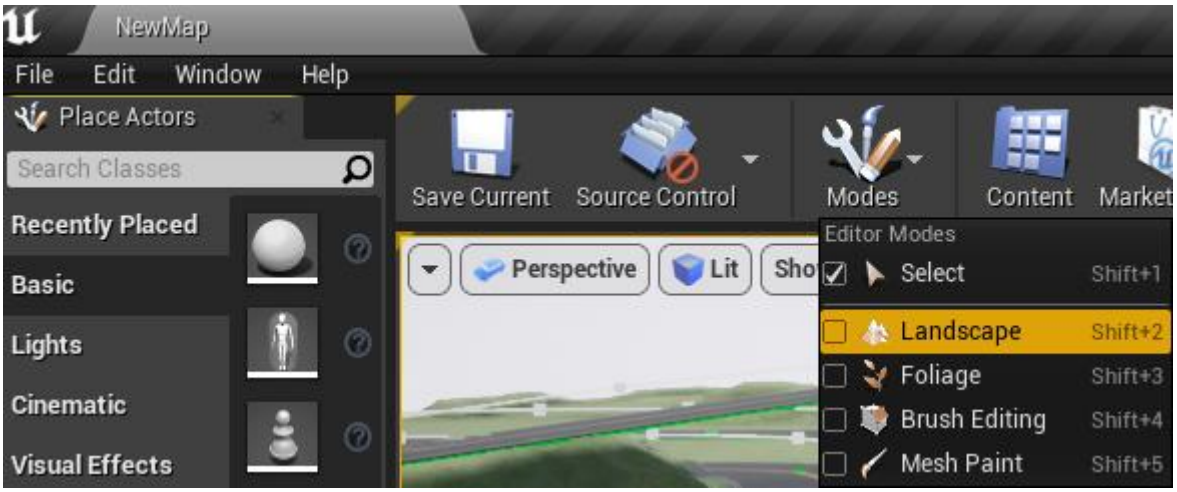


Abbildung 28: Landscape-Tool

4.1. Einführung zu Textures, Materials und Static Meshes

In diesem Unterkapitel wird kurz erläutert, worauf geachtet werden muss bzw. welche Eigenschaften angepasst werden müssen, um Landschaften oder Objekte zu visualisieren. Hierzu wird im Folgenden zunächst auf die theoretischen Grundlagen der *Textures*, *Materials* und *Static Meshes* in Unreal eingegangen.

Eine *Texture* ist eine einzelne Datei, ein statisches 2D-Bild. In der Regel handelt es sich um eine konturlose, spiegelnde oder eine normale *map*-Datei, die in *Photoshop* oder *Gimp* als *tga*-, *tiff*-, *bmp*- oder *png*-Datei erstellt werden kann. Dabei kann es sich um manipulierte Fotos, handgemalte Texturen oder Ähnliches handeln. Diese müssen nach Unreal importiert und als Teil eines *Materials* verwendet werden. [9]

Materials bestehen aus verschiedenen Texturen, die miteinander kombiniert werden. Sie umfassen verschiedene Texturen und Materialausdrücke, die ein Netzwerk von Knoten erzeugen. Letztendlich sind *Materials* das, was im Spiel gerendert zu sehen ist. [10]

Ein *Static Mesh* ist eine aus einer Reihe von Polygonen bestehenden Geometrie, die im Videospeicher zwischengespeichert und von der Grafikkarte gerendert werden kann. Dadurch kann sie effizienter gerendert werden, wodurch ein *Static Mesh* viel komplexer aufgebaut werden kann als andere Geometrietypen. Sie sind also die Grundeinheit, die zum Erstellen der Weltgeometrie für die generierten Levels in der Unreal Engine verwendet werden. Dabei handelt es sich um 3D-Modelle, die in externen Modellierungsanwendungen, wie Blender erstellt wurden, die über den *Content Browser* (siehe Abbildung 29) in den Unreal-Editor importiert, in Paketen gespeichert und dann auf verschiedene Weise zur Erstellung renderbarer Elemente verwendet werden. [11]

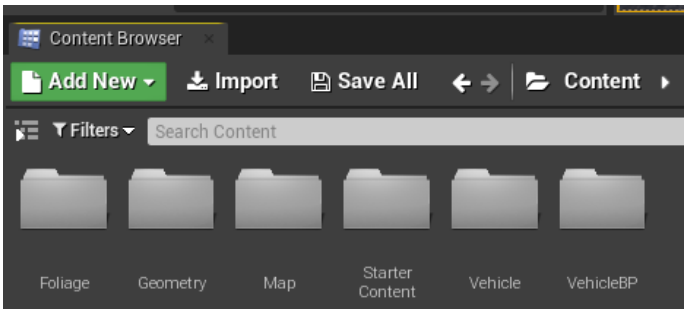


Abbildung 29: Content Browser in Unreal

Um bspw. den Rauheitsgrad oder einen Glanzeffekt eines Materials anzupassen, kann dies in den *Blueprint*-Einstellungen des jeweiligen Materials konfiguriert werden. Hierzu muss lediglich eine Variable erstellt und mit dem entsprechenden Effekt verbunden werden. Als Beispiel ist in der Abbildung 30 eine Variable mit dem Wert 2.0 für die Rauheit vergeben.

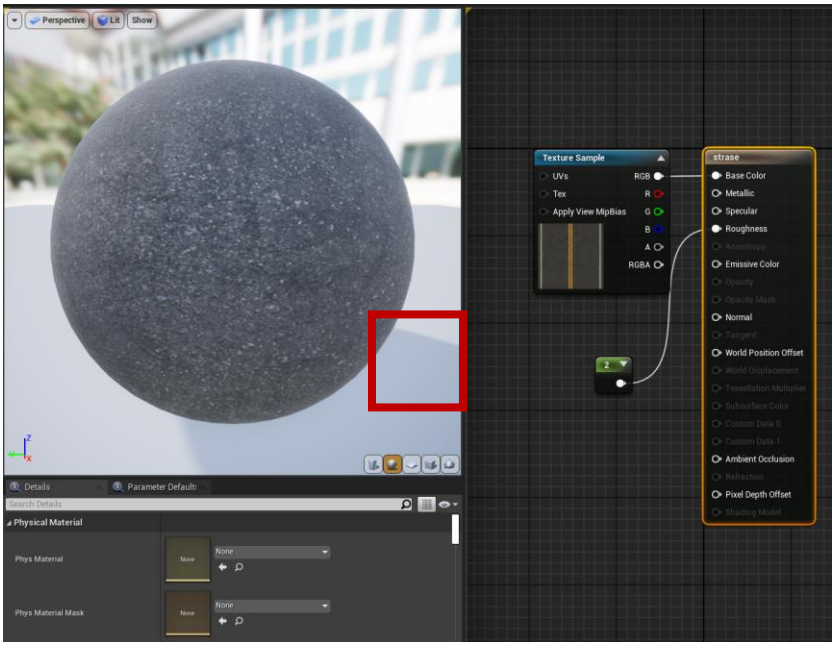


Abbildung 30: Anpassung des Rauheitsgrad

4.2. Erstellen der Straße

Im *Landscape*-Tool kann im *Manage*-Bereich die Funktion *Splines* ausgewählt werden. Diese ist nicht mit dem zuvor genannten *Spline-Follow* zu verwechseln, da dieser sich hauptsächlich nur auf die Map bezieht und keine Eigenschaften eines *Actors* mit sich bringt.

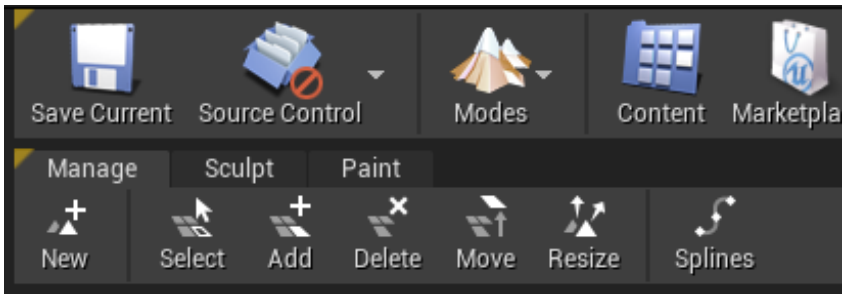


Abbildung 31: Manage-Bereich im Landscape-Tool

Ist die Funktion *Splines* ausgewählt, kann über den Shortcut *STRG* + *linke Maustaste* ein Spline-Punkt gesetzt werden, der in Abbildung 32 mit einem Berg-Symbol in Unreal dargestellt wird. Durch Halten der *STRG*-Taste und Setzen weiterer Spline-Punkte mit der linken Maustaste an beliebigen Stellen, kann ein großer Spline erstellt werden, der aus mehreren einzelnen Segmenten besteht.

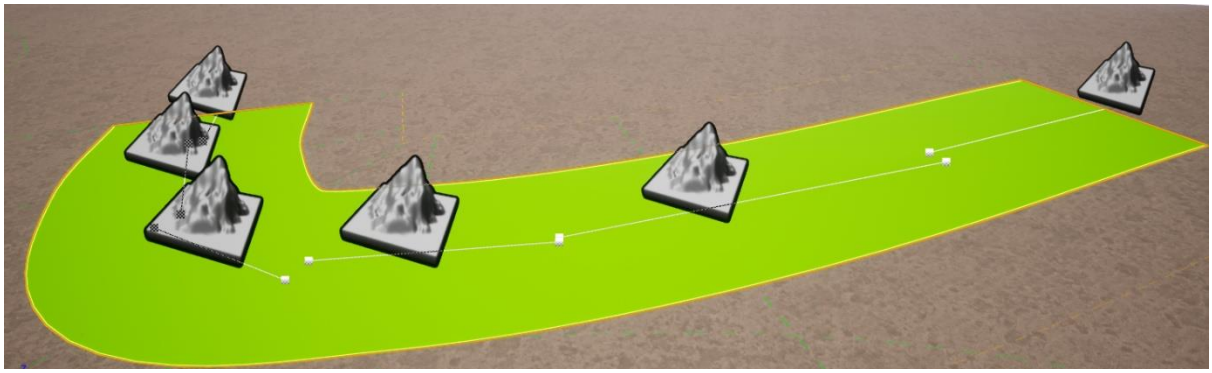


Abbildung 32: Erstellung von Splines in Maps

Um die erstellten Spline-Segmente als eine Straße zu visualisieren, ist ein Mesh mit den dazugehörigen *Materials* und *Textures* notwendig. Hierzu wurde zur Hilfe ein fertiges Kit genommen, das bereits über ein fertiges *Mesh* samt *Material* und *Textures* verfügt. In diesem Hilfe-Kit wurde das Straßensegment mit Blender erstellt und ein fertiges Straßenmuster von *Quixel Megascans* verwendet. In Abbildung 33 ist das für unser Projekt verwendete Straßensegment zu sehen. [12]

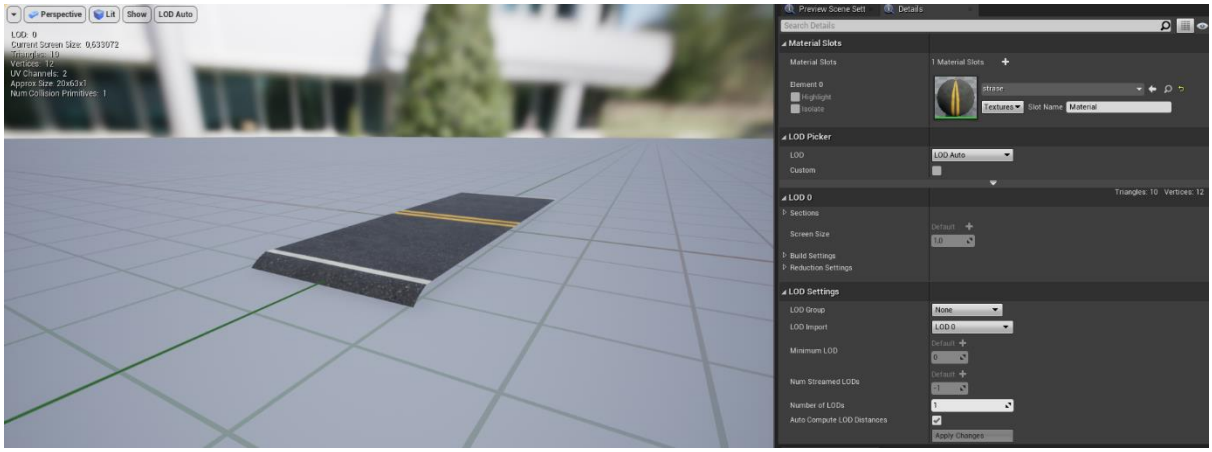


Abbildung 33: Mesh für die Straße

4.3. Hügel, Hindernisse und Bodenschwellen

Da in diesem Projekt der Fokus unter anderem auf die Federeigenschaften des RC-Autos gelegt wird, wurde die Map um weitere Bereiche ergänzt. Wie in Abbildung 34 zu sehen ist, wurde ein Bereich aus mehreren Bodenschwellen erstellt, das aus Zylindern erstellt worden ist. Fährt das Fahrzeug über diesen Bereich, ist visuell die Eigenschaften der Federmechanik zu erkennen.

Abbildung 34: Bodenschwellen

Auf Abbildung 35 ist zu sehen, dass weitere hohe Kegel in die Teststrecke integriert wurden, sodass ein Slalom gefahren werden kann.

Abbildung 35: Slalom

In Abbildung 36 ist eine Hügellandschaft dargestellt, die für den manuellen Fahrtmodus erstellt wurde. Durch das unebene Terrain wird die Federung des Fahrwerks stark beansprucht. Dadurch kann die Animation der beweglichen Teile gut demonstriert werden.

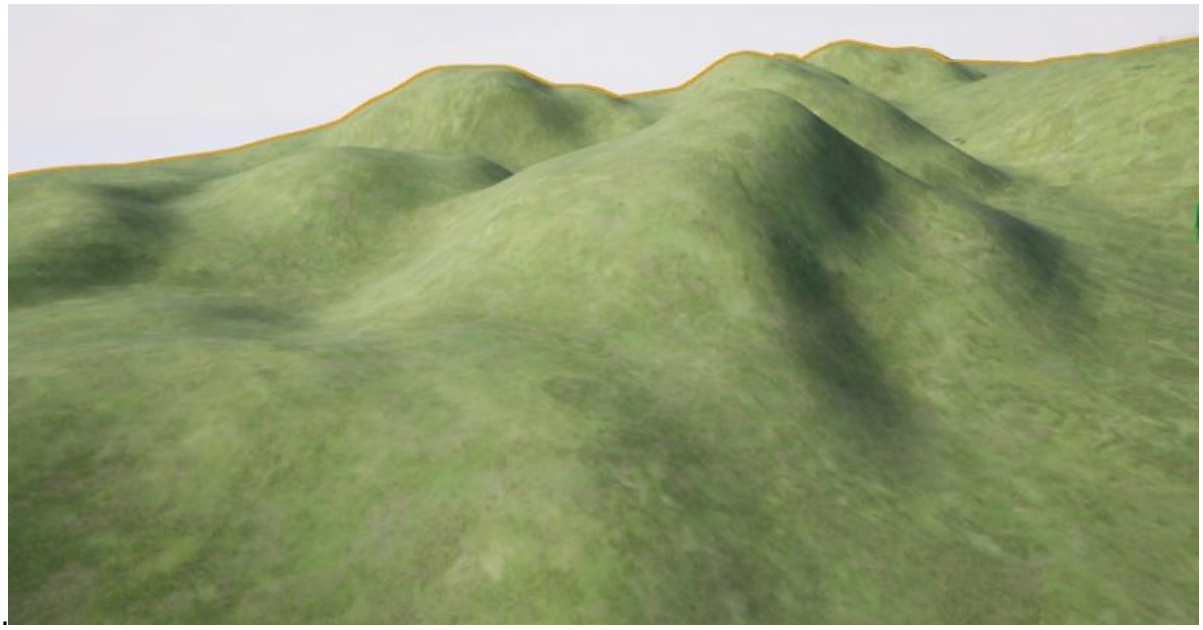


Abbildung 36: Hügellandschaft

In der folgenden Abbildung 37 ist die fertige Teststrecke dargestellt. Die Strecke kann für zukünftige Arbeiten individuell erweitert werden. Im weiteren Verlauf dieses Projekts wird die Landschaft noch durch eine Vegetation ergänzt. Der aktuelle Stand wird jedoch in einer extra Datei gespeichert, falls in nachfolgenden Projekten auf die Vegetation verzichtet werden möchte.

Abbildung 37: Vollständige Map

5. Das Gameplay

In diesem Kapitel wird darauf eingegangen, wie das Gameplay unseres RC-Cars funktioniert. Außerdem wird erläutert, wie das Wechseln mehrerer Kameraperspektiven und ein Head-up-Display integriert wird.

5.1. Kameraperspektiven

Für dieses Projekt wurden mehrere Kameraperspektiven implementiert, wodurch die Fahrt des RC-Autos aus unterschiedlichen Perspektiven betrachtet werden kann. Insgesamt wurden drei Kameras implementiert. Falls zukünftig weitere Kameraperspektiven gewünscht sind, können diese jederzeit hinzugefügt werden. Wie in der Abbildung 38 zu erkennen ist, müssen hierzu lediglich neue Kameras als *Camera Component* ergänzt und dem Array hinzugefügt werden. Die Kamerakontrolle ist im *Eventgraph* der *Blueprint*-Klasse des RC-Autos als *Camera Control* integriert.

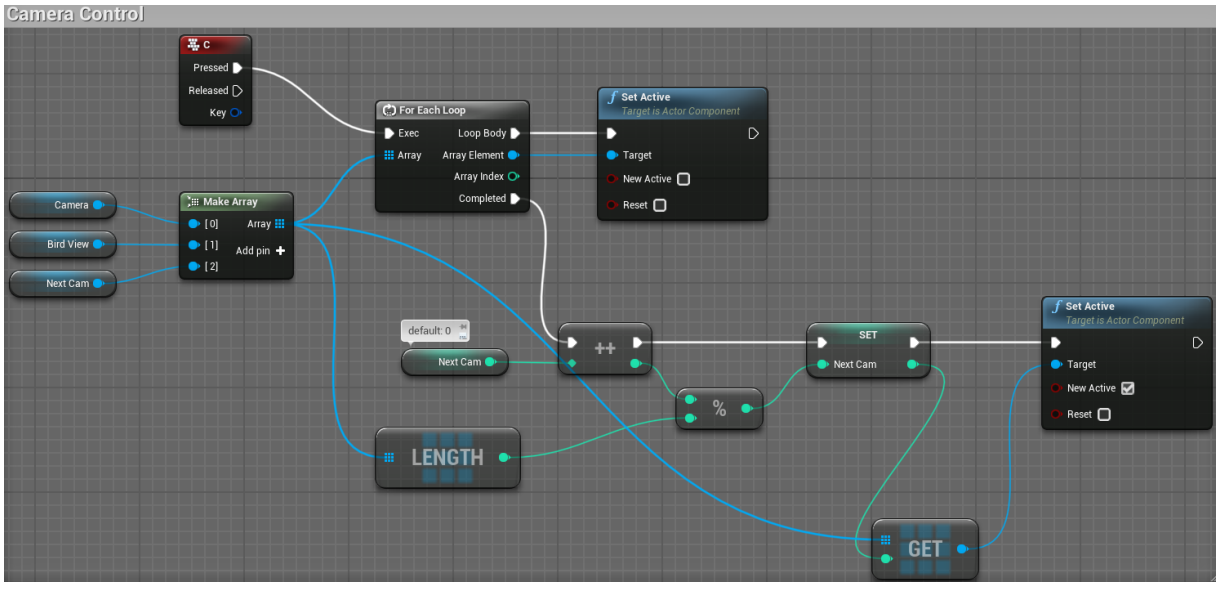


Abbildung 38: Blueprint: Kamerasteuerung

Der Bruchteil des aktuellen Indexes, welcher um Eins erhöht wird, wird durch die Länge des gesamten Arrays geteilt. So wird das Array komplett durchlaufen und immer die aktuelle Zelle bzw. der aktuelle Index ermittelt, wodurch die entsprechende Kamera aufgerufen wird. Beim Drücken der Taste C kann die Kameraperspektive gewechselt werden. Als Standardperspektive wird die Ansicht von hinten gewählt (Abbildung 39). Diese Perspektive wird auch oft in herkömmlichen Computerspielen verwendet.

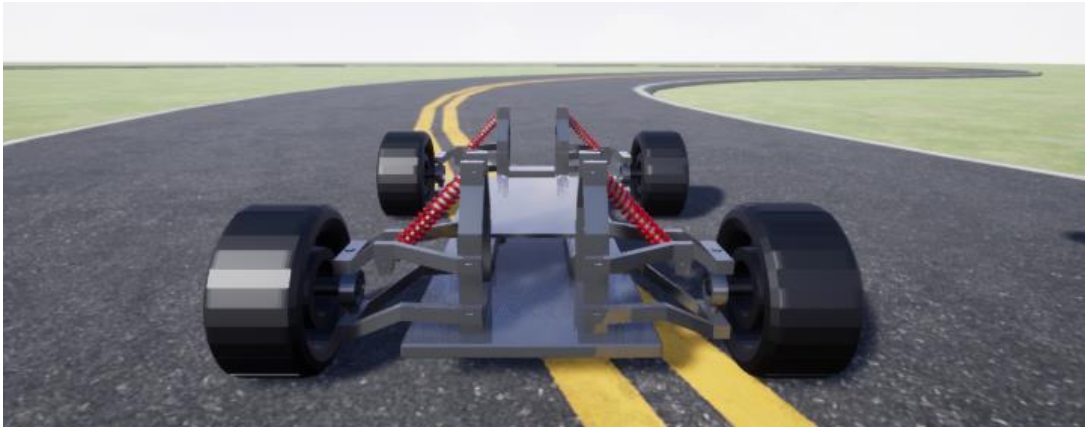


Abbildung 39: Kameraperspektive von hinten

Durch die Kameraansicht aus der Vogelperspektive (Abbildung 40) kann bei dynamischen Kurvenfahrten das Unter- bzw. Übersteuern des Fahrzeugs besser beobachtet werden.

Abbildung 40: Kameraperspektive von oben

Um die Animation des Fahrwerks besser beobachten zu können, wurde eine weitere Kameraperspektive im Bereich der Vorderachse eingefügt (Abbildung 41).

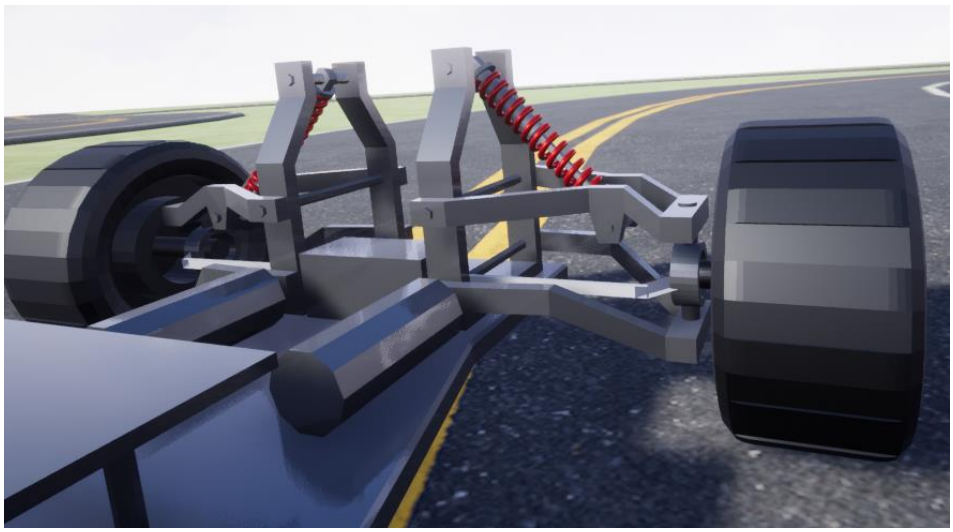


Abbildung 41: Kameraperspektive Detailaufnahme der Federung

5.2. Head-up-Display

Zur Veranschaulichung der wichtigsten Shortcuts und der Geschwindigkeit des Fahrzeugs wurde ein Head-up-Display erstellt. Die Erstellung orientiert sich an einem YouTube-Tutorial, weshalb auf die einzelnen Schritte nicht genauer eingegangen wird [13]. Das fertige Head-up-Display ist in der Abbildung 42 im laufenden Spielbetrieb zu sehen. In der linken unteren Ecke des Bildschirms wird die Geschwindigkeit des Fahrzeugs angezeigt. In der linken oberen Bildschirmecke sind die wichtigsten Shortcuts zum Umschalten verschiedener Fahrmodi abgebildet. Im oben referenzierten Tutorial wird außerdem beschrieben, wie eine bewegliche Tachonadel implementiert werden kann. Darauf wurde in diesem Projekt verzichtet, könnte jedoch mit geringem Mehraufwand jederzeit hinzugefügt werden.

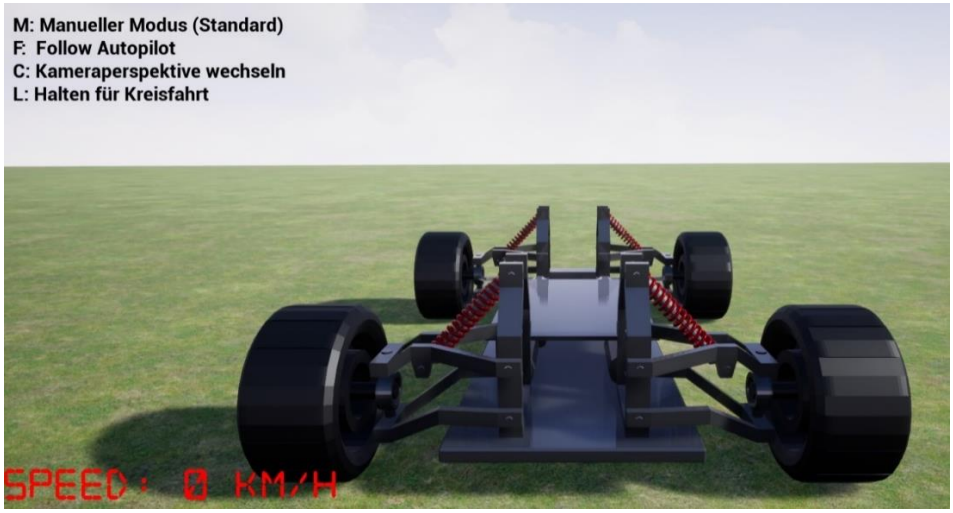


Abbildung 42: Head-up-Display der Spieloberfläche

5.1. Spielsteuerung

In der nachfolgenden Tabelle 2 sind alle Tastaturbefehle beschrieben, mit denen das Fahrzeug bzw. die verschiedenen Fahrmodi eingestellt werden können.

Befehl	**Tastenkürzel**
Beschleunigen	Pfeil-Taste nach oben
Bremsen	Pfeil-Taste nach unten
Rückwarts	Pfeil-Taste nach unten (HALTEN)
Lenkung	Pfeil-Taste nach rechts/ links
Kamera umschalten	C
Fester Einlenkwinkel	L (HALTEN)
Autonomer Fahrmodus	F
Manueller Fahrmodus	M

Tabelle 2: Tastaturbefehle

6. Gestalten der Landschaft und Vegetation

Zum Abschluss soll die Umgebung der Fahrstrecke um eine realistische Landschaft erweitert werden. Umweltobjekte können prinzipiell von Vorlagen aus dem Internet übernommen werden. In diesem Projekt werden Objekte verwendet, die aus der Open-World-Demo „A Boy and His Kite“ stammen. Von offizieller Seite wurde diese Demo anlässlich der Games Developer Conference im Jahr 2015 vorgestellt, um die technischen Möglichkeiten der Unreal Engine 4 zu präsentieren. Die Demo wurde für Benchmark-Zwecke entwickelt und beinhaltet insgesamt 138 Texturen mit einer Auflösung von 2048 x 2048 bis 8192 x 8192 Pixeln, inklusive Meshes mit LODs. Entstanden sind diese in einem Photogrammmetrie-Prozess, den man normalerweise aus dem Bereich der Messtechnik kennt. Die Assets werden dabei auf Basis von Fotos erstellt, die wiederum mit speziellen Kameras aufgenommen wurden. Dadurch entstehen sehr realitätsgetreue Texturen und Objekte.

6.1. Import der Kite Demo

Um die Objekte der Kite-Demo zu unserem Projekt hinzuzufügen, muss die sogenannte „Open World Demo Collection“ heruntergeladen werden. Diese ist im offiziellen Marktplatz der Unreal Engine verfügbar. Der Marktplatz kann über den Epic Games Launcher unter der Rubrik *Unreal Engine* aufgerufen werden. Über die Schaltfläche *Dem Projekt hinzufügen* können die Inhalte automatisch in den entsprechenden Projektorder kopiert werden (Abbildung 43).

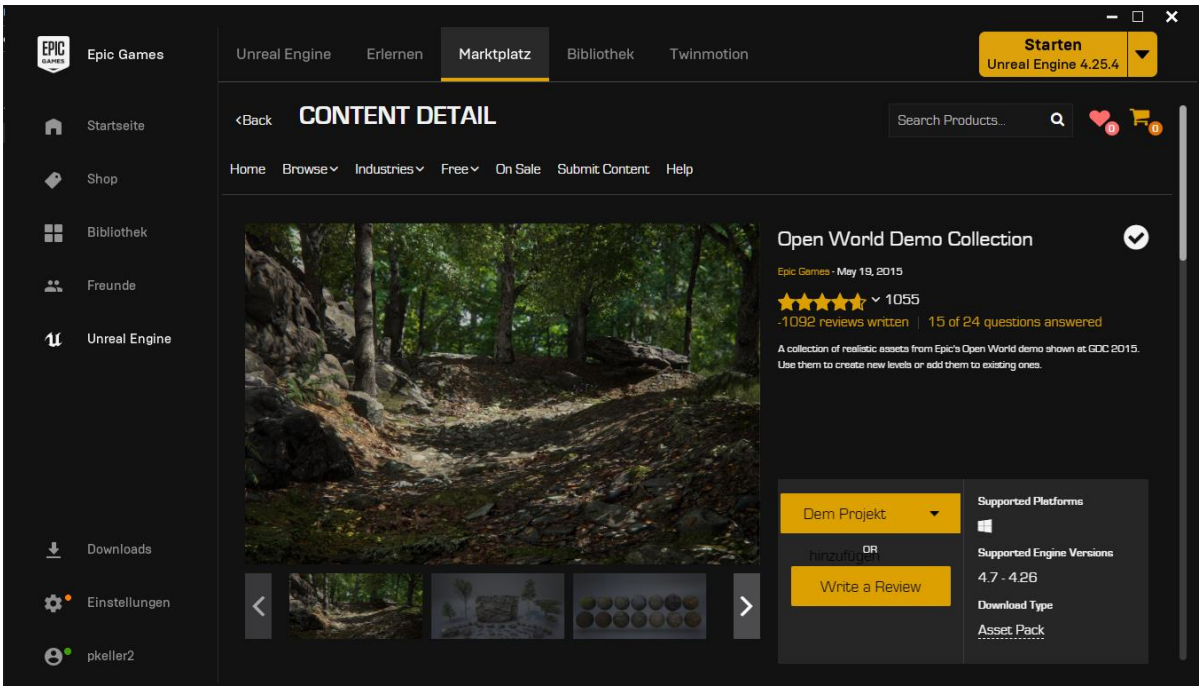


Abbildung 43: Auswahl der Open World Demo Collection im Marktplatz der Unreal Engine

6.2. Hinzufügen von Objekten mit dem Foliage-Tool

Um statische Landschaftsobjekte in die Map einzufügen, wird das Foliage-Tool verwendet. Zur Einarbeitung wurde auf ein kurzes Tutorial in YouTube zurückgegriffen [14]. Über das Foliage-Tool können Objekte mithilfe der Pinselfunktion der Map hinzugefügt werden. Durch konfigurierbare Parameter kann auf die Anzahl, Dichte, Größe und Ausrichtung der erzeugten Objekte Einfluss genommen werden.

Im ersten Schritt muss das Foliage Tool geöffnet werden. Dies geschieht durch die Auswahl der Schaltfläche *Foliage* über den Reiter *Modes* im Unreal Editor (siehe Abbildung 44). Anschließend muss im Content-Browser das entsprechende Objekt ausgewählt und durch Ziehen mit gedrückter Maustaste in den linken oberen Bereich des geöffneten Foliage-Tools gezogen werden. Nun muss das Objekt gewählt auswerden, dass erzeugt werden soll. Eine Mehrfachauswahl ist auch möglich. In diesem Fall werden Objekte unterschiedlichen Typs gleichzeitig erzeugt. Nach der Auswahl erscheinen im unteren Bereich weitere Einstellmöglichkeiten. Dort können diverse Parameter verändert werden. Die für unser Projekt angepassten Einstellungen werden im Folgenden kurz erläutert.

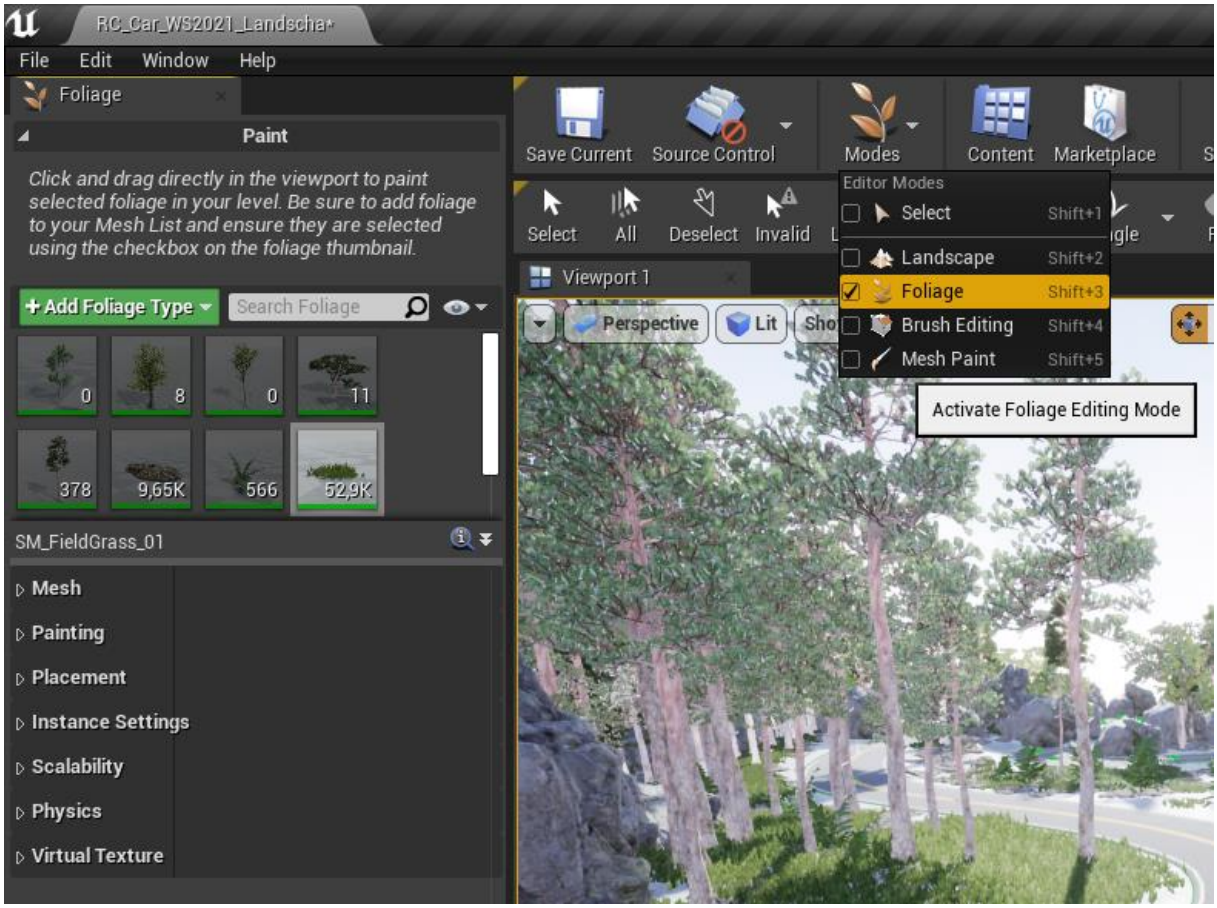


Abbildung 44: Auswahl des Foliage-Tools über die Benutzeroberfläche des Editors der Unreal-Engine

Painting → Density:

Mit der Einstellung *Density* wird die Anzahl der Objekte definiert, die innerhalb eines Pinselbereichs erzeugt werden sollen. In der Abbildung 45 ist am Beispiel der Erzeugung von Bäumen der Einfluss des Parameters dargestellt.

Abbildung 45: Spawnen von Bäumen, links: Density = 1, rechts: Density = 5

Painting → Scale X:

Um ein natürliches Erscheinungsbild der Umgebung zu bewahren, können die Objekte in unterschiedlichen Größen erzeugt werden. Die Skalierung eines Objekts wird über einen Zufallsgenerator definiert, dessen Mindest- und Höchstwert über den Wert *Scale X* eingestellt werden kann. Wenn also beispielsweise die Grenzwerte 1.0 und 10.0 gewählt werden, wird die in Destiny definierte Anzahl an Objekten mit einer zufälligen Skalierung innerhalb der Grenzwerte erzeugt.

Abbildung 46: Erzeugung von Bäumen mit Scale-X-Werten zwischen 0,5 und 2,5

Placement → Align to Normal

Über die Einstellung *Align to Normal* kann festgelegt werden, ob Objekte senkrecht zur Landschaftsoberfläche oder senkrecht zum globalen Koordinatensystem erzeugt werden sollen. In der Abbildung 47 wird die Wirkungsweise der Funktion am Beispiel von Gräsern und Bäumen dargestellt. Bei einer Erzeugung senkrecht zur Fläche würden bspw. die Bäume an Berghängen schief aus dem Boden wachsen. Deshalb sollte die Funktion bei Bäumen grundsätzlich deaktiviert werden. Bei Gräsern und Bodenbedeckungen sollte die Funktion jedoch aktiviert werden, da sonst schwebende Objekte entstehen können.

Abbildung 47: Auswirkungen der Einstellung „Align to Normal", links: Inaktiv, rechts: Aktiv

Instance Settings → Collision Presets

Über diese Einstellung kann festgelegt werden, in welcher Art und Weise die Landschaft mit der Umgebung interagieren soll. In unserem Projekt wurde lediglich für die Bäume, Steine und das Geröll die Einstellung *BlockAll* verwendet. Somit kann das Fahrzeug nicht durch sie hindurchfahren, bzw. muss über das Geröll drüberfahren. Bei Kleinpflanzen und Gräsern wurde dieses Verhalten durch die Einstellung *NoCollision* deaktiviert.

6.3. Bilder der finalen Map

In der nachfolgenden Abbildung 48, Abbildung 49 und Abbildung 50 ist das Ergebnis der Landschaftserstellung unter Zuhilfenahme des Foliage-Tools zu sehen. Die Umgebung wurde im Stil einer Gebirgslandschaft gestaltet. Der Boden ist mit einer Mischung aus Geröll, Gräsern und Farn bedeckt. Außerdem existiert neben den zahlreichen Gebirgszügen auch eine Waldlandschaft. Die Abbildung 51 bietet einen Überblick über die gesamte Map aus der Vogelperspektive.

Abbildung 48: Gebirgszug

Abbildung 49: Waldlandschaft

Abbildung 50: Darstellung der Bodenbedeckung bestehend aus Gras, Geröll und Farn

Abbildung 51: Gesamte Map mit eingefügter Landschaft und Vegetation

7. Zusammenfassung und Ausblick

Insgesamt betrachtet kann man sagen, dass sich durch die Wahl unseres Themas „RC-Auto“ ein breites Spektrum an Umsetzungsmöglichkeiten ergeben hat, was uns einen umfangreichen Einblick in die verschiedenen Facetten der Unreal Engine ermöglicht hat. Sowohl die Modellierung eines dynamisch animierten Fahrzeugs, die Implementierung der Fahrphysik, das Erstellen einer Fahrstrecke und der Erzeugung einer umfangreichen und detaillierten vegetativen Landschaft demonstrierte uns die zahlreichen Möglichkeiten im Bereich der Computergrafik.

In Zukunft könnte das Projekt noch weiter ausgebaut werden. Zum jetzigen Zeitpunkt wird die Fahrphysik im *Blueprint*-System der Unreal Engine umgesetzt. Es wäre theoretisch möglich, ein Simulationsmodell des Fahrzeugs in Simulink zu entwerfen und anschließend die Simulation mit der Unreal Engine zu verknüpfen. Außerdem wäre es auf diese Art und Weise möglich, einen Fahrdynamikregler zu implementieren. Insbesondere bei schnellen Kurvenfahrten übersteuert das Fahrzeug aktuell sehr stark. Durch gezielte Lenk- und radindividuelle Bremseingriffe könnte das Verhalten des Fahrzeugs im fahrdynamischen Grenzbereich stabilisiert werden. In Abbildung 52 ist die Wirkung eines Fahrdynamikreglers in einer schnellen Rechtskurve dargestellt.

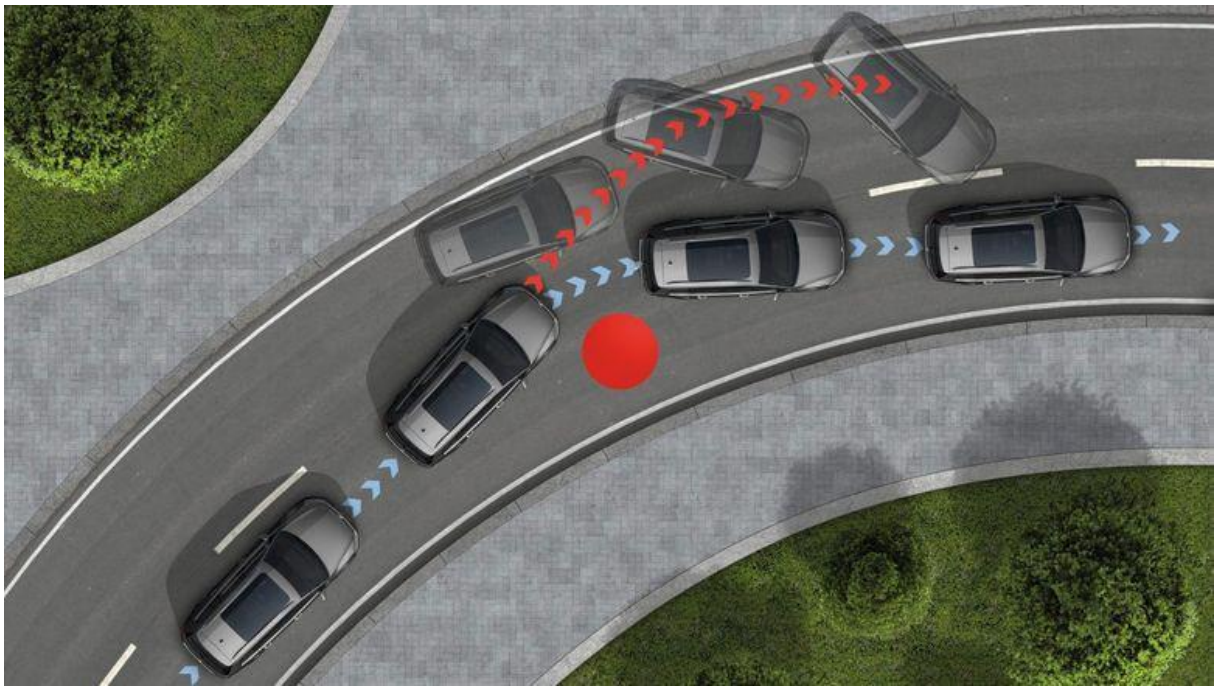


Abbildung 52: Eingriff eines Fahrdynamikreglers beim Übersteuern in einer rasanten Kurvenfahrt, gezielter Bremseingriff vorne links erzeugt Gegenmoment und stabilisiert das Fahrzeug [15]

Prinzipiell wäre es auch möglich, durch Anpassungen der entsprechenden Softwareparameter die Reibungskoeffizienten der Fahrzeugräder zu erhöhen. Selbst in den schärfsten Kurven könnte das Fahrzeug dann bei einer entsprechenden Parametrisierung mit hoher Geschwindigkeit agile Lenkmanöver durchführen. Das entspricht jedoch keinem realistischen Fahrverhalten.

Da das Projekt ohnehin an die laufende Arbeit bei Professor Wild angelehnt ist, könnte deshalb in Zukunft ein Fahrdynamikregler programmiert und dessen Funktionsweise in Kooperation mit Professor Meroth visualisiert werden. Es ist jedoch anzumerken, dass hierzu ein umfassendes Hintergrundwissen bezüglich der Fahrdynamik eines Fahrzeugs vorausgesetzt wird. Deshalb ist das Thema nur bedingt als reines Visualisierungsprojekt für die Lehrveranstaltung „Computergrafik und Multimedia“ geeignet. Optimal wäre es, wenn die Simulation des Fahrzeugs und die Implementierung des Fahrdynamikreglers in einem Parallelprojekt umgesetzt und dann im Fach „Computergrafik & Multimedia“ grafisch veranschaulicht wird.

Außerdem wäre es möglich das Head-up-Display des Fahrzeugs zu erweitern. Neben der Anzeige diverser fahrdynamischer Kenngrößen während der Fahrt könnte auch noch eine Instrumententafel realisiert werden. Dort könnte die Anzeige der Fahrzeuggeschwindigkeit um eine Anzeige der Motordrehzahl ergänzt und eventuell durch eine dynamische Anzeige mit einer Tachonadel erweitert werden. Außerdem könnte der Fahrer beispielsweise über kritische Fahrsituationen benachrichtigt werden (Unter-/Übersteuern, Eingriff des Fahrdynamikregler, etc.). Auf jeden Fall bietet die im Rahmen dieses Projekts entwickelte Basis des RC-Autos und der Teststrecke für folgende Projekte noch zahlreiche Möglichkeiten, kreative Ideen zur Weiterentwicklung einzubringen.

8. Literaturverzeichnis

[1] W. Gelbricht, „3D-Grafiksoftware,“ 21 01 2021. [Online]. Available: https://de.wikipedia.org/wiki/3D-Grafiksoftware.

[2] Blender, „About,“ 21 01 2021. [Online]. Available: https://www.blender.org/.

[3] FileTypes, „Dateiendung FBX,“ 21 01 2021. [Online]. Available: https://www.filetypes.de/extension/fbx.

[4] Moog, „Is double wishbone suspension best for your car?,“ 06 01 2021. [Online]. Available: https://www.moogparts.eu/blog/double-wishbone-suspension.html.

[5] A. Ontuzhan, „How to rig vehicle in Blender 2.8 for UE4,“ 03 12 2020. [Online]. Available: https://continuebreak.com/articles/how-rig-vehicle-blender-28-ue4/.

[6] Xenome, „How to Build a Double Wishbone Suspension Vehicle,“ 27 01 2021. [Online]. Available: https://docs.unrealengine.com/en-US/InteractiveExperiences/Vehicles/DoubleWishboneVehicle/index.html.

[7] M. Filipovic, „Unreal Engine 4 Vehicle Spline Path FollowTutorial,“ 27 01 2021. [Online]. Available: https://youtu.be/FVyUVJra3KI.

[8] M. Filipovic, „Unreal Engine 4 Auto Throttle Tutorial,“ 27 01 2021. [Online]. Available: https://youtu.be/ThuSqb7neZc.

[9] Epic Games, „Textures,“ 04 01 2021. [Online]. Available: https://docs.unrealengine.com/en-US/RenderingAndGraphics/Textures/index.html.

[10] Epic Games, „Materials,“ 04 01 2021. [Online]. Available: https://docs.unrealengine.com/en-US/RenderingAndGraphics/Materials/index.html.

[11] Epic Games, „Static Meshes,“ 04 01 2021. [Online]. Available: https://docs.unrealengine.com/en-US/WorkingWithContent/Types/StaticMeshes/index.html.

[12] Smart Poly, „Landscape Road Spline Tutorial - Unreal Engine 4,“ 30 12 2020. [Online]. Available: https://www.youtube.com/watch?v=8WIWuybAKj4.

[13] Mr. Z., „Unreal Engine 4 - Speedometer Tutorial (Part 2),“ 15 01 2020. [Online]. Available: https://www.youtube.com/watch?v=7ZfEXf61hiI.

[14] Der Sky, „Das Foliage-Tool | Bäume, Gras und mehr spawnen ► Unreal Engine Tutorial (German),“ 02 01 2021. [Online]. Available: https://www.youtube.com/watch?v=YZmRICYB_sU.

[15] Porsche Austria GmbH & Co OG, „ESP,“ 2021 02 15. [Online]. Available: https://www.volkswagen.at/technik-lexikon/esp.