Fachhochschule Salzburg University of Applied Sciences

Physical driving behavior of vehicles on different ground surfaces and the implementation in Unity3D

BACHELORARBEIT 1

Studierende/Studierender Alexander Cerny, 0910601006

BetreuerIn Mag. Thomas Wagner

Salzburg, am 08. Jänner 2012

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe. Weiters versichere ich hiermit, dass ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission weder im In- noch im Ausland vorgelegt und auch nicht veröffentlicht.

.....	.....
Datum	Unterschrift

Zusammenfassung

Vor- und Zuname:	*Alexander CERNY*
Institution:	*FH Salzburg*
Studiengang:	*Bachelor MultiMediaTechnology*
Titel der Bachelorarbeit:	*Physical driving behavior of vehicles on different ground surfaces and the implementation in Unity3D*
BegutachterIn:	*Mag. Thomas Wagner*

Diese Arbeit beschäftigt sich damit, welche physikalischen Kräfte auf das Fahrverhalten eines Fahrzeuges wirken. Außerdem wird erklärt, wie man diese Kräfte berechnen kann und wie diese am Fahrzeug angewandt werden. Das Dokument beschreibt, wie verschiedene Bodenbeschaffenheiten das physikalische Verhalten eines fahrenden Autos beeinflussen können. Es gibt eine kurze Einführung in Unity3D, in der die benötigten physikalischen Komponenten der Game Engine vorgestellt werden. Der Implementationsteil der Bachelorarbeit beschäftigt sich damit, wie man realistische Werte eines Autos für eine Computersimulation bestimmen kann und wie man die physikalischen Prinzipien vom Fahrverhalten in Unity3D integrieren kann. Es wird die Implementation eines realistischen Fahrverhaltens für einen Buggy in Unity3D beschrieben. Außerdem existiert eine Software Architektur für das vorgestellte Konzept, das dafür entwickelt wurde, dass weitere Automodelle oder Motortypen hinzugefügt werden können.

Schlüsselwörter: Fahrphysik, Fahrverhalten, Unity3D, Bodenverhältnisse, Bodenbeschaffenheit, Physik Engine, Game Engine, Implementierung, Software Architektur

Abstract

This paper shows which physical forces influence the driving behavior of a vehicle. Further it explains how you can determine these forces and apply it to the rigid body of the car. The thesis considers that different ground surfaces can affect the driving behavior. This document contains a short introduction to Unity3D which explains some useful components of the physic engine. The implementation part covers how you can find realistic values for your desired car model and how you can implement the theoretical equations in a car simulation. It explains how you can create a realistic driving behavior for a buggy in the Unity3D game engine. The paper contains a software architecture that was developed for this physical concept with the requirement that you can easy add different car models and engine types.

Key words: vehicle physics, vehicle dynamics, driving behavior, Unity3D, road conditions, ground surfaces, ground physics, physic engine, game engine, implementation, software architecture

Contents

List of Abbreviations

abs	Anti-lock breaking system
cm	center of mass
km/h	Kilometers per hour
m/s	Meters per second
Nm	Newton meter
rpm	revolutions per minute
tcs	Traction control system

1. Introduction

1.1. Interest of research and relevance

I had the task to implement a physical correct vehicle control system in my internship. The car control system should react to different ground surfaces as asphalt, mud or ice to create different driving behaviors in different levels.

Currently there are not many free algorithms for efficient vehicle control systems available. Therefore, I decided to introduce myself to the topic. This bachelor thesis describes the theoretical aspects of this topic and how they can be applied in the Unity3D game engine. I have been working with the Unity3D engine for over a year and I'm very experienced in it. Also during my internship I worked with Unity3D. Furthermore, Unity3D has grown to a big market share in the business of game engines.

1.2. Research question

My bachelor thesis should answer the following question:
How to implement in Unity3D the physical driving behavior of a vehicle and how do different ground surfaces affect this behavior?

The goal of this paper is to gain a basic understanding in vehicle dynamics to develop a concept that manages the driving behavior of a buggy on different surfaces. This concept should be applied to the Unity3D game engine and represents a realistic driving behavior. Further, I developed a software architecture for this concept at the end of the thesis. It should be easy to expand the concept by adding new vehicles with a different driving behavior.

1.3. Requirements

During my internship, I was developing a browser game. The game was a multiplayer shooter where the players control tanks and fight against each other. I got the task to develop a concept for physical correct driving cars. The first car to implement in the game was a buggy. I should consider that it has to be possible to add more cars later with a different driving behavior.

The goal of this thesis is to explain how the basic physics of a driving car work. The driving behavior of the vehicles should be realistic and it is allowed to create a challenging control

system. The reason for a more difficult control of the car is that the buggy is able to drive faster than the tanks. The buggy should be difficult to handle at high speeds for balancing purposes. The users are familiar with challenging controls because the game provides a helicopter with a tricky control system.

As I explained the physical aspects of the driving behavior, I demonstrate how we can find realistic properties for an imaginary buggy. I try to explain how we can implement the buggy in Unity3D considering the realistic figures of the buggy and the physical knowledge of the driving behavior.

The last requirement is that the driving behavior differs from map to map. The maps of the game have different settings. For example there is a snow map, a city map and off road maps. Therefore, the buggy should react to different ground surfaces.

1.4. Structure of the thesis

This thesis describes the basic physics of the driving behavior of a vehicle. Chapter 2 explains the theoretical part of the vehicle dynamics. It explains how the car accelerates and is able to drive around a corner.

The goal is to explain how we can implement these physical behaviors in the Unity3D engine. Chapter 3 is a short introduction to the game engine and describes which components are used for the implementation.

Chapter 4 deals with the implementation of a specific buggy in Unity3D. Therefore, we have to research realistic properties for the car to provide a correct driving behavior.

The last section is the conclusion where we critically evaluate the concept that we develop in this thesis.

2. Vehicle Physics

The physics section of the paper describes how a vehicle is able to accelerate, (Palmer 2005, 214-221) which forces decelerate the vehicle (Palmer 2005, 213-214) and how low speed (Palmer 2005, 237-238) and high speed turns work (Monster 2003). It introduces the basic physic forces, which influence the cars behavior, and explains how the dynamic weight

transfer affects the phenomena of skidding and drifting. (Monster 2003) The traction budget defines the maximal acceleration a wheel can stand before it starts to slip. (Beckman 1991, 21-25) Furthermore, there are different ground surfaces presented and explained how they influence the adhesion of the wheels. (Ming 1997, 31-35)

If the further text or the calculations refer to the local coordinate system of the vehicle, it corresponds to this orientation, which is shown in the picture below. In the vehicle's local coordinate system the x-axis is adjusted along the car's forward direction, the y-axis points to the left side of the vehicle and the z-axis' direction is upwards.

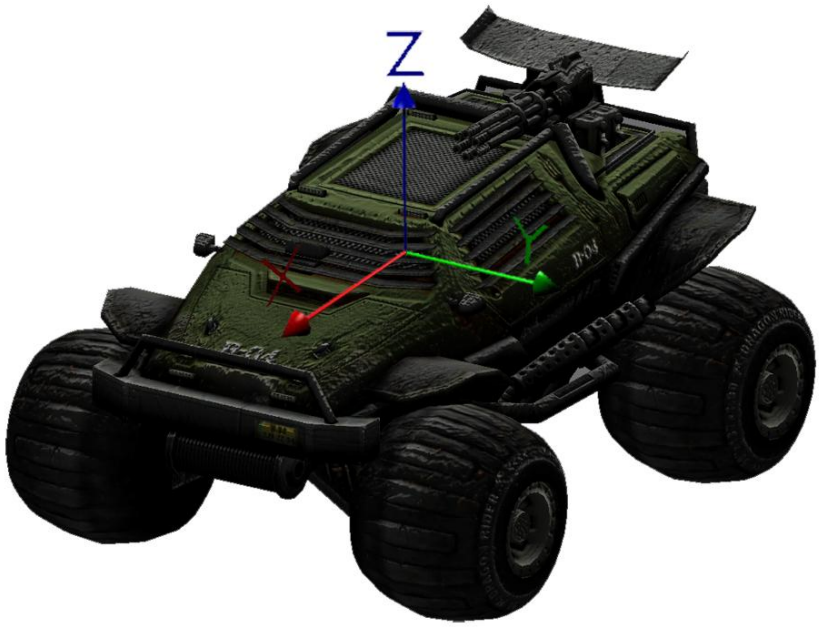

Figure 1: Local coordinate system of the buggy

To simplify the vehicle physics we have to handle the longitudinal and lateral forces separately. First of all we are looking at the straight line physics of a car, which means the acceleration and deceleration of the vehicle. (Monster 2003) Expressed in the vehicle's local coordinate system we handle the forces along the x-axis also known as longitudinal forces. When the car is able accelerate and decelerate we start concentrating on the turning of the automobile and the effects of slipping and drifting. The lateral forces occur during a turn.

2.1. Straight line physics

Many different forces act on a moving vehicle. The figure below shows the straight line forces for an accelerating car. There exist two different types of forces. The resistance forces try to slow down the automobile. The forces of gravity, rolling resistance and aerodynamic drag are resistance forces. The engine produces the traction force, which accelerates the car.

We have to subtract all resistance forces from the produced traction force to get the total longitudinal force of the vehicle. The total resistance force is the sum of the aerodynamic drag, the rolling resistance and the gravity force. We can determine the acceleration of the car with the longitude force.

$$\vec{F}_{longitude} = \vec{F}_{drag} + \vec{F}_{gravity} + \vec{F}_{rr} + \vec{F}_{traction}$$

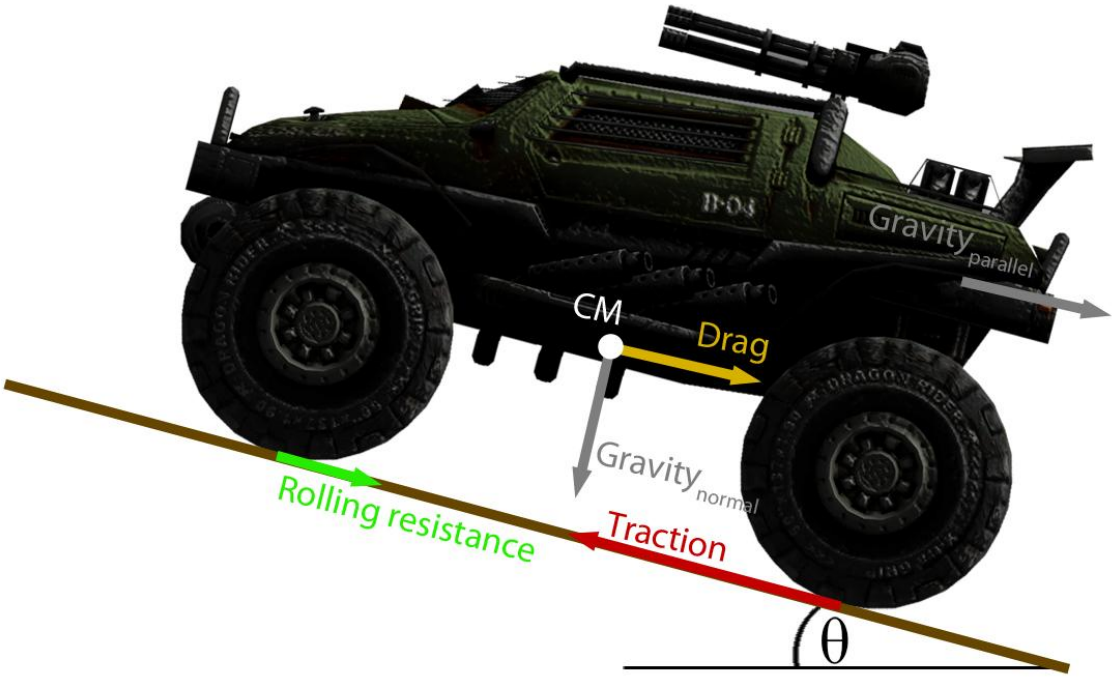

Figure 2: Straight line physic forces

2.1.1. Resistance forces

The resistance forces act on a moving car and try to slow it down. Aerodynamic drag is one of these forces. A driving car is displacing a lot of air. The displacement creates the aerody-

namic drag force and slows down the car. The influence of this force is proportional to the vehicle's velocity squared. That means the faster a car is driving, the more resistant is the aerodynamic drag. The aerodynamic drag force depends also on the surface of the frontal area and the aerodynamic shape of the vehicle. The vector of the aerodynamic drag force points in the opposite direction than the velocity vector of the car. Therefore, we have to multiply the equation by -1.

$$\vec{F}_{drag} = -\left(\frac{1}{2}\right) * \rho * \vec{v} * |\vec{v}| * A * C_{drag}$$

The aerodynamic drag force depends on the mass density of air, ρ, the frontal area of the car, A, and the drag coefficient, C_{drag}. The drag coefficient describes the aerodynamic shape of the vehicle. Streamlined body styles have lower drag coefficients. (Palmer 2005, 213)

"Typical ranges of the drag coefficients for different types of vehicles are 0.29 to 0.4 for sports cars, 0.43 to 0.5 for pickup trucks, 0.6 to 0.9 for tractor-trailers and 0.4 to 0.5 for the average economy car." (Bourg 2002, 168-169)

The last parameter is the car's velocity squared. Here, we are multiplying the velocity vector, $\vec{v}$, times the speed, $|\vec{v}|$. The speed is the magnitude of the velocity vector.

The gravity acts parallel and normal to the surface on the car. The parallel force slows the car down, if it is driving upwards a hill and accelerates it, if the automobile drives down. We have to multiply the parallel force by -1, if the force decelerates the car.

$$F_{gravity} = (\pm 1) * m * g * \sin\theta$$

The normal gravity force, F_{normal}, pulls the car towards the center of earth. This force pulls the car wheels against the ground. According to Newton's third law the ground pushes the four wheels back up. The normal gravity force equals the sum of the ground forces that push the four wheels up. The normal gravity force depends on the car's mass, m multiplied by the acceleration of gravity, g, and the slope angle of the ground, θ. (Palmer 2005, 213)

$$F_{normal} = m * g * \cos\theta$$

The rolling wheels on the ground also decelerate the vehicle. There is friction between the ground surface and the tires. This effect is the rolling resistance and acts on all four wheels.

The rolling resistance force, F_{rr}, equals the normal gravity force, F_{normal}, that act on the car multiplied by the coefficient of rolling resistance, μ_{rr}.

$$F_{rr} = \mu_{rr} * F_{normal} = \mu_{rr} * m * g * \cos\theta$$

The rolling resistance coefficient depends on various parameters. The wheel type, the temperature, the weather and the ground surface influence the coefficient.

The total resistance force is the sum of the aerodynamic drag, the rolling resistance force and the gravity force. It is important to represent all forces as a vector. The vector shows the direction in which the force acts. We will show in the implementation section how we can determine the directions of the different forces.

$$\vec{F}_{resistance} = \vec{F}_{drag} + \vec{F}_{rr} + \vec{F}_{gravity}$$

2.1.2. Traction force

Now we know what decelerates our car. The next step is to find out what accelerates it. The running engine produces a torque. This torque is transferred to the wheels and makes them rotate. The turning rate of the wheels is measured in the wheel torque, τ_{wheel}. The spinning wheels are pushing the ground back. Again, according to Newton's third law the ground is pushing the wheels in the opposite direction. This pushing force is called traction force, $F_{traction}$. To convert the wheel torque to the traction force, we have to divide it by the radius of the tires.

$$F_{traction} = \frac{\tau_{wheel}}{r_{wheel}}$$

2.1.2.1. Engine torque

We have to take a closer look at the wheel torque to calculate the traction force. The wheel torque is transferred from the engine. The engine torque depends on the turnover rate of the engine. The engine turnover rate is measured in rpm (revolutions per minute). The relation between the engine torque and the rpm is not linear. Therefore we need a function for the given engine rpm to determine the delivered engine torque.

$$T_{engine} = f(rpm)$$

To get the equation for $f(rpm)$ we need the torque curve of the vehicle. The torque curve of a car is available from the manufacturer and depends on the engine type. In the following figure you can see an example torque curve from a Porsche Boxter S.

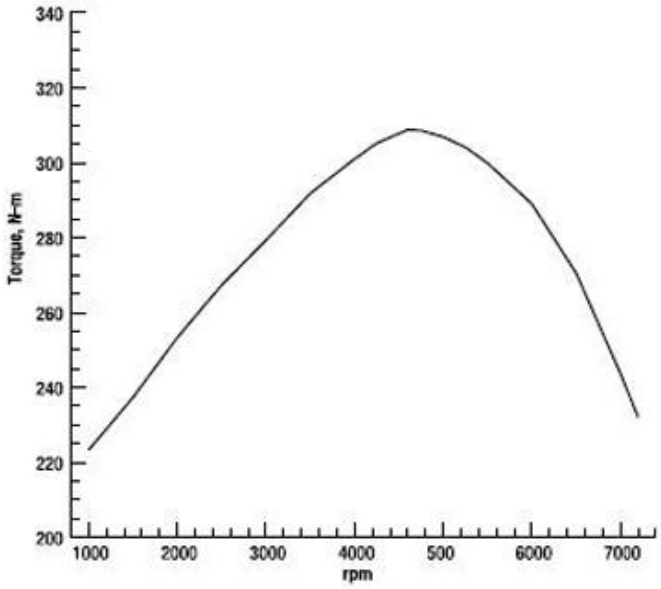

Figure 3: Engine torque curve from a Porsche Boxter S

There are some characteristics for every torque curve. First, the curve has always a peak and the engine delivers the maximal possible torque at this peak. The maximal possible torque relates to a certain amount of rpm. This means, there is a defined rpm value that delivers us the maximal torque for a certain engine. Before the peak, the torque curve increases and afterwards it falls.

The next characteristic of a torque curve is that the engine has a minimum and maximum turnover rate. The engine would stall in reality if the rpm falls below the minimum. We don't let the engine stall in a computer simulation and therefore we set the rpm back up to the minimum for this scenario. If the turnover rate would exceed the maximum, the engine would take damage. The maximum is also called red line rpm.

When we compare these characteristics with the torque curve of the Boxter S, we find out that the engine delivers the maximum torque of 309 Nm at a turnover rate of 4600 rpm. The minimum turnover rate for the engine is 1000 rpm, which delivers a torque of 220 Nm. The redline rpm begins by 7200 rpm. (Palmer 2005, 216)

The values we see in the torque curve are the maximum values, which the engine could deliver. However, the real produced torque depends on the input of the driver or in our case the input of the player. In other words, the throttle position influences the torque. We have to multiply the maximal torque with a number between 0 and 1 referring to the throttle position.

2.1.2.2. Transmission

We are able to calculate the torque that is produced by the engine. However, this torque differs from the one, which is applied to the wheels. The engine torque runs through various sets of shafts and gears. These shafts and gears are called transmission. We take a closer look on two parts of the transmission.

The driver is able to control with the gearshift the current gear. A typical car has five to six forward gears and one reverse gear. These gears are the first part of the transmission. The second part is the differential, which connects the gears to the driving wheel.

The purpose of a transmission is to increase the delivered engine torque. The engine is connected to the transmission, which increases the engine torque and delivers it to the wheels. The car's acceleration would be very slow if the engine would be connected directly to the wheel. (Palmer 2005, 219)

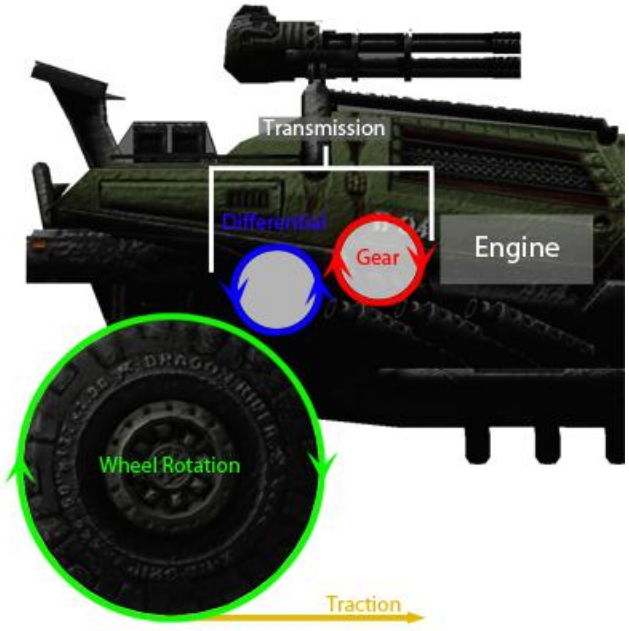

Figure 4: Engine torque transferred via the transmission to the driving wheel

Let's take a look at an example how the gears of the transmission are able to increase the engine torque. The gears have the task to increase the torque at the expanse of the angular velocity. For example we have gear 1 with a diameter of d_1 and gear 2 with a diameter of two times d_1. This means gear 2 has twice the diameter of gear 1. If gear 2 makes 1 revolution, gear 1 makes 2 revolutions in the same time. This means the angular velocity, ω, of gear 1 is twice the angular velocity of gear 2. However, the delivered torque from gear 2 is twice as big as the torque from gear 1. (Palmer 2005, 217-219)

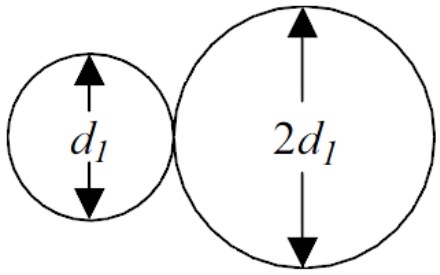

Figure 5: Gear diameters of two gears

$$diameter_{gear\ 1} < diameter_{gear\ 2}$$

$$\omega_{gear\ 1} > \omega_{gear\ 2}$$

$$\tau_{gear\ 1} < \tau_{gear\ 2}$$

The delivered torque is responsible for the acceleration of the car. On the one hand, a gear with a bigger diameter accelerates the car faster. On the other hand, it is the angular velocity decisive for the car's velocity. Therefore is a gear with a small diameter better to deliver a higher velocity. The first forward gear has the biggest diameter and the highest forward gear the smallest diameter. In the first gear the car has the highest acceleration and the lowest top speed. In the highest forward gear the vehicle has the highest top speed and in return the lowest acceleration. The difference of the gears in a car is the diameter ratio between two gears. Every car has three to six forward gears and one reverse gear. For every existing gear there is a gear ratio.

We summarize that the engine produces a torque that that runs through the current gear of a car. The gear influences the torque and is connected to the differential. The differential fulfills the same tasks as the gears. Therefore, it also has a gear ratio that is called the final drive ratio. The differential is connected to the driving wheels and applies the torque to them. The

gear ratios differ from engine type to engine type. The following table shows example gear ratios for a Porsche Boxter S.

Gear	**Gear ratio**
First gear	3.82
Second gear	2.20
Third gear	1.52
Fourth gear	1.22
Fifth gear	1.02
Sixth gear	0.84
Final drive ratio	3.44

Tab. 1: Gear ratios Porsche Boxter S

The transmission influences the torque, which is delivered from the engine and applies it to the driving wheels. It can increase the torque in costs of the angular velocity and vice versa. In other words if we want to increase the car's acceleration we have to decrease the top speed of the vehicle.

When we are aware of the current gear, g_k, the final drive ratio of the car, G, and the current torque which is delivered by the engine, τ_{engine}, we can calculate the wheel torque, τ_{wheel}.

$$\tau_{wheel} = \tau_{engine} * g_k * G$$

"As the wheel turns, the torque of the tire exerts a force on the road. This force is illustrated in the figure below. As the figure shows, the tire exerts a force on the road, and the road pushes back on the tire. This is Newton's third law. Every action produces an equal and opposite reaction." (Conger 2004, 407-408)

Figure 6: Driving wheel exerts the traction force on the road

We have to divide the wheel torque, τ_{wheel}, by the wheel radius, r_{wheel} to get the traction force $F_{traction}$. If we combine this with the last equation we can determine the traction force from the engine torque, τ_{engine}.

$$F_{traction} = \frac{\tau_{wheel}}{r_{wheel}} = \frac{\tau_{engine} * g_k * G}{r_{wheel}}$$

The traction force is the combined force of each driving wheel. We have to divide the traction force by two to get the traction force for each single drive wheel. That is important when we check later the traction budget for each wheel individually.

2.1.2.3. Shifting

As we already heard, the velocity depends on the torque that the engine produces and the gear ratios. We can calculate the top speed a car can drive in a certain gear. Therefore, we need to know the red line turnover rate of the engine. We have to shift in a higher gear to exceed the maximum velocity of the current gear. Therefore, we have a higher top speed in a higher gear. For example, the Boxter S has a maximum turnover rate of 7200 rpm. In the table below are the top speed values for each gear.

$$top\ speed = \frac{r_{wheel} * 2\pi * rpm}{60 * g_k * G} * 3.6$$

Gear	Top speed
First gear	65.8 km/h
Second gear	114.3 km/h
Third gear	165.4 km/h
Fourth gear	206.0 km/h
Fifth gear	246.4 km/h
Sixth gear	299.3 km/h

Tab. 2: Top speed for each gear for the Boxter S

These top speed values are not realistic, because we haven't subtracted the resistance forces. If we have shifted in a higher or lower gear the engine's rpm change. We consider that the velocity of the car before and after the shifting process stays constant. When we shift in a higher gear the rpm fall, when we shift in a lower gear the rpm rise. To calculate the new rpm value we have to multiply the old turnover rate with the new gear ratio, g_{new}, divided by the old ratio, g_{old}. (Palmer 2005, 220)

$$rpm_{new} = rpm_{old} * \frac{g_{new}}{g_{old}}$$

If we want to implement an automatic shifting system, we have to watch the current engine rpm and the current car velocity. (Palmer 2005, 220) The simplest solution is to determine a maximum and minimum value. If the current rpm is falling below the minimum you have to shift down, if it goes beyond the maximum you have to shift up.

2.1.3. Velocity

Considering the engine's turnover rate we can calculate the angular velocity of the wheel, ω_{wheel}. We have to divide the engine's rpm by the current gear, g_k, the final drive ratio, G, and 60, to convert the rpm to revolutions per second.

$$\omega_{wheel} = \frac{2\pi * rpm}{60 * g_k * G}$$

Based on the wheel's angular velocity we can calculate the car's velocity by multiplying with the radius of the wheel, r_{wheel}. This equation only works if there is no slip between the ground and the wheel. This equation doesn't work for the driving wheels because there is always a little slip when they are applying a force on the ground. (Monster 2003)

$$v_{car} = \omega_{wheel} * r_{wheel} = \frac{r_{wheel} * 2\pi * rpm}{60 * g_k * G}$$

We measure the result of this calculation in m/s. If we want to convert this to km/h we have to multiply by 3.6.

2.1.4. Acceleration

We can calculate the car's acceleration at any given time considering the longitude force. We get the following equation when we combine all calculations.

$$\vec{F}_{longitude} = \vec{F}_{traction} + \vec{F}_{drag} + \vec{F}_{rr} + \vec{F}_{gravity}$$

$$F_{longitude} = \frac{\tau_{engine} * g_k * G}{r_{wheel}} + \mu_{drag} * \vec{v} * speed + \mu_{rr} * m * g * \cos\theta + m * g * \sin\theta$$

The acceleration is equal the forces divided by the vehicle's mass, m.

$$\overrightarrow{acceleration} = \frac{\tau_{engine} * g_k * G}{r_{wheel} * m} + \frac{\mu_{drag} * \vec{v} * speed}{m} + \mu_{rr} * g * \cos\theta + g * \sin\theta$$

2.1.5. Breaking

When the driver breaks, the breaking acceleration takes effect to the opposite direction where the car is driving. To determine this breaking acceleration we need the breaking distance data of the car. (Palmer 2005, 226)

$$a_{breaking} = -\frac{v^2}{2x}$$

For example, a Porsche Boxter S takes 34 m to stop a velocity of 26.8 m/s. (Palmer 2005, 226)

$$a_{breaking} = -\frac{26.8^2}{2 * 34} = -10.5\ m/s^2$$

To convert the breaking acceleration to the breaking force $F_{breaking}$ we have to multiply it by the mass of the vehicle.

$$F_{breaking} = a_{breaking} * m$$

When we want to simulate the breaking force more accurate, we apply the force for each wheel separately. Therefore, we do not multiply by the car's mass but by the current load, which is on the wheel. The dynamic weight transfer determines the current load on the wheel. If we simulate a hand break, we only apply the force to the rear wheels.

2.1.6. Dynamic weight transfer

When the car is accelerating or breaking the dynamic weight transfer occurs. This effect pushes the central mass of the vehicle back when the rider steps on the throttle. The dynamic weight transfer is important when it comes to the maximum traction force a tire can stand before it starts to slip. The friction limit for a wheel is proportional to the load on it and the dynamic weight transfer can change this load. (Monster 2003)

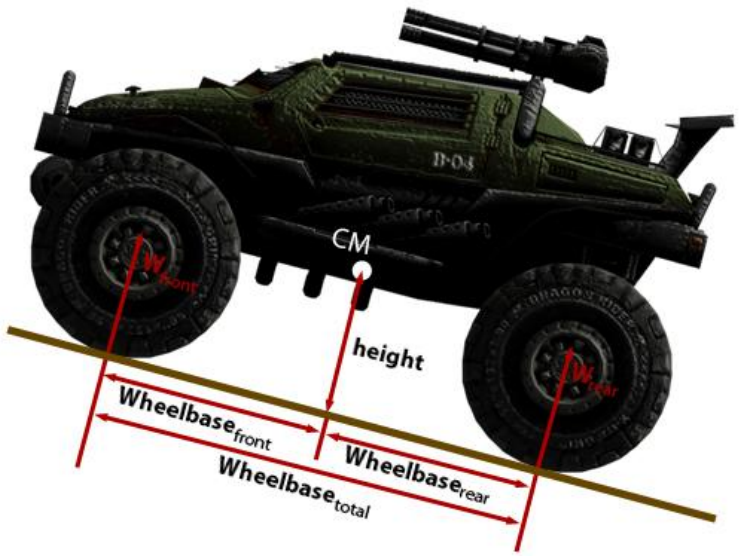

Figure 7: Wheelbase of a car

$$W_{front} = \frac{wheelbase_{rear}}{wheelbase_{total}} * m * g - \left(\frac{height}{wheelbase_{total}}\right) * m * a$$

$$W_{rear} = \frac{wheelbase_{front}}{wheelbase_{total}} * m * g + \left(\frac{height}{wheelbase_{total}}\right) * m * a$$

In this equation m stands for the mass of the car, g for the acceleration of the gravity and a for the acceleration or deceleration of the vehicle. If the car is breaking a is negative. $wheelbase_{total}$ is the distance between the two axles. $wheelbase_{rear}$ is the length between

the rear axle and the center of mass and $wheelbase_{front}$ the difference between the front axle and the center of mass. (Monster 2003)

2.1.7. Wheel traction budget

Our previous examples assumed that there is no slipping between the wheels and the surface. During accelerating, breaking or high speed, turning tires can start sliding and slipping. Each wheel has a maximum force. The wheels start to slide if the acting forces on the tire exceed this maximum. This force depends on the weight applied to the tire. The maximum traction force equals the normal force exerted on the wheel multiplied by the adhesion coefficient, $\mu_{adhesion}$, which depends on the tire type and the road conditions.

$$\mu_{adhesion} = \mu_{tire\ adhesion} * \ \mu_{road\ adhesion}$$
$$F_{max} = F_{normal} * \mu_{adhesion} = \mu_{adhesion} * m * g * \cos\theta$$

The traction budget doesn't depend on the mass, therefore we have to look at the acceleration of the tire. We need to divide F_{max} by the mass of the tire to get the acceleration.

$$a_{max} = \frac{F_{max}}{m_{wheel}}$$

We see the maximum acceleration, which the wheel can stand without sliding as the so called traction budget. The car is not only moving in a straight line. If the car turns, it also uses the same traction budget as for acceleration or deceleration. We have to combine the straight line acceleration and the cornering acceleration by the Pythagorean formula. (Beckman 1991, 23)

$$a_{total} = \sqrt{{a_{straightline}}^2 + {a_{cornering}}^2}$$

All we have to do now is to compare the total acceleration that is applied to the tire to the traction budget. If the acceleration stays within the traction budget, we just apply the forces. If they exceed the budget, the wheels start to spin and can't apply more acceleration than a_{max}. Therefore, we must scale the straight line and cornering acceleration by the $ratio$ between a_{max} and a_{total}.

$$ratio = \frac{a_{max}}{a_{total}}$$

$$a_{new\ straightline} = ratio * a_{straightline}$$

$$a_{new\ cornering} = ratio * a_{cornering}$$

Before we can apply the accelerations to the wheel, we have to convert them back to forces by multiplying them with the mass on the tire.

$$F_{new\ straightline} = a_{new\ straightline} * m_{wheel}$$

$$F_{new\ cornering} = a_{new\ cornering} * m_{wheel}$$

We can visualize the traction budget with the circle of traction. The positive y coordinates show the acceleration and the negative the breaking acceleration of the car. The positive x coordinates visualize the right turn acceleration on the tire and the negative values express the left turn acceleration. The circle demonstrates the maximal acceleration a tire can stand before it starts to slip. The example vector shows a car, which is accelerating and turning to the right side at the same time. (Beckman 1991, 24)

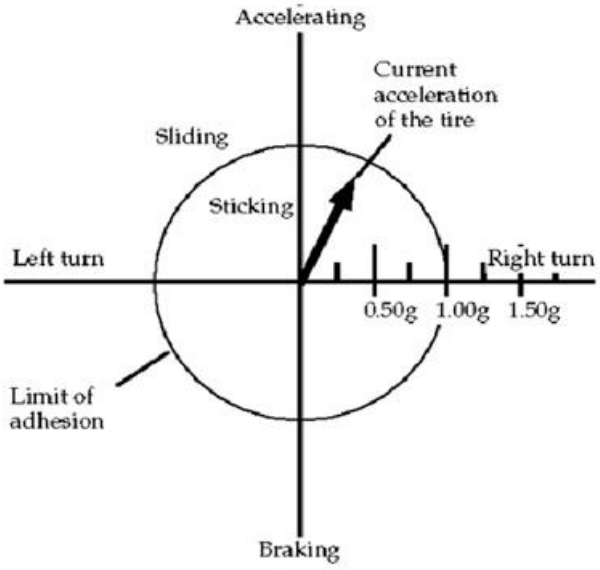

Figure 8: Traction budget of a wheel

2.2. Turning

There are two different cases of turning. The first is when the car performs a turn at low speed and the other is when the vehicle drives around a corner with high speed. The turning forces are also termed as lateral forces. The lateral force points along the y-axis in the local coordinate system of the car. (Palmer 2005, 237)

2.2.1. Low speed turn

A low speed turn is easier to calculate than a high-speed maneuver, because we can ignore the centripetal acceleration and the wheels roll without slipping. A car will drive in a circle with the radius r_{turn}, when it moves with a constant speed and an constant steering angle. (Palmer 2005, 237)

In following example we assume that the car makes a right turn with the steering angle δ. The turn circle's center is the intersection of the two lines drawn perpendicular to the front and rear wheels. We have to draw these two lines from the inner wheels. That means when the car makes a right turn we take the right wheels and by a left turn the left ones. We determine the circle's radius, r_{turn}, on which the car is driving by trigonometric relations. The $wheelbase_{total}$ is the distance between the rear and the front axle. The wheelbase and the two perpendicular drawn lines form a right triangle. With the sine law and three known parameters we can calculate the radius r_{turn}. The parameters are the 90° angle between the line of the rear axle and the wheelbase, the steering angle δ and the side length of the triangle $wheelbase_{total}$.

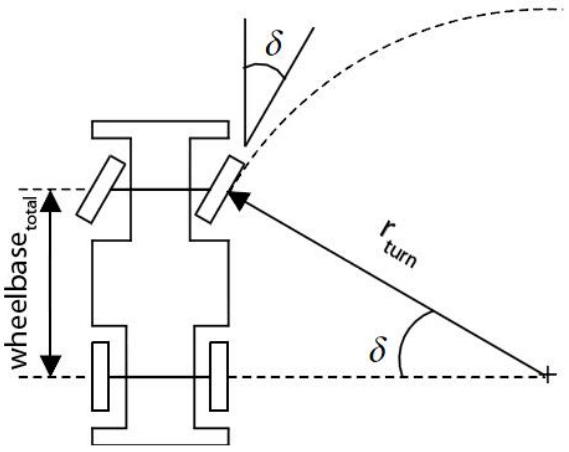

Figure 9: Slow speed turn

$$\frac{r_{turn}}{\sin(90)} = \frac{wheelbase_{total}}{\sin(\delta)}$$

$$\sin(90) = 1$$

$$r_{turn} = \frac{wheelbase_{total}}{\sin(\delta)}$$

If we divide the car's speed by the turning radius, we get the angular velocity of the turning car. The angular velocity expresses how far the car drives along the turning circle per second. The following equation is only valid, if the car turns without sliding.

$$\omega_{turn} = \frac{v}{r_{turn}} = \frac{\mathrm{v} * \sin(\delta)}{\mathrm{l}}$$

2.2.2. High speed turn

The low speed turn calculations are only valid when you park in a parking lot. Turning maneuvers with a higher velocity are high-speed turns. Effects of a high-speed turn can be understeer, oversteer and sliding. To allow these effects we have to calculate every force for each wheel individually.

During turning the tires develop lateral forces also known as cornering forces. The cornering force of each wheel depends on the weight on the tire and the wheel slip angle α.

2.2.2.1. Wheel slip angle

The wheel slip angle is the angle between the tire's heading and the car's velocity vector. We have to convert the velocity to the local coordinate system of the tire to calculate the angle. The wheel slip angle has three parameters. The first is the side slip angle of the car, β. The others are the angular velocity of the car, ω_{car} and for the front wheels the steering angle, δ.

A car is not always moving in the directions of its heading. The angle between the car's orientation and the velocity vector is called car side slip angle, β. We have to convert the velocity vector, $\vec{v}$, from the world coordinate system to the car's local coordinate system to calculate the side slip angle. Now the x-coordinate of the vector is heading to the car's forward direction and the y-coordinate is pointing to the side of the automobile. The velocity vector and the forward vector of the car form a right triangle. The x and y coordinate of the velocity vector form the two legs of the triangle. We can determine the side slip angle β by taking the arctangent of v_y divided by v_x.

Figure 10: Car side slip angle

$$\beta = \arctan\left(\frac{v_y}{v_x}\right)$$

The angular velocity of the vehicle expresses how fast the car is turning around its own center of mass. The rotation rate, ω_{car}, is measured in radians per second. We have to multiply the angular velocity of the car by the rotation radius to convert it to a force. The rotation radius is for the front wheels, the distance from the vehicle's center of mass and the front axle, $wheelbase_{front}$. The distance from the center of mass to the rear axle, $wheelbase_{rear}$, is the rotation radius for the rear tires.

$$F_{lateral} = \omega_{car} * wheelbase_{front}$$
$$F_{lateral} = -\omega_{car} * wheelbase_{rear}$$

The steering angle δ is the wheel's orientation relative to the heading of the car. Only the front wheels have a steering angle. The rear tires are always in line with the vehicle's orientation. If the car drives backwards, we have to subtract the steering angle, otherwise we add it to the wheel slip angle. (Monster 2003)

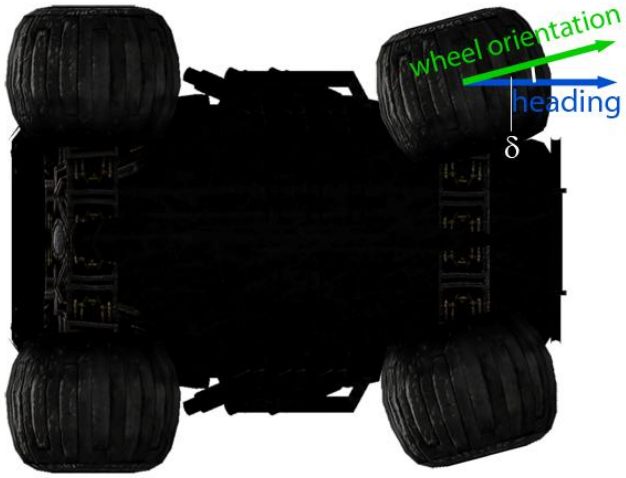

Figure 11: Steering angle

We combine these three parameters to determine an equation for the slip angle of the wheel.

$$\alpha_{front} = \left(\frac{v_y + \omega_{car} * wheelbase_{front}}{\|v_x\|}\right) - \delta * sign(v_x)$$

$$\alpha_{rear} = \left(\frac{v_y - \omega_{car} * wheelbase_{rear}}{\|v_x\|}\right)$$

The wheel's slip angle relates to the tire adhesion coefficient. The adhesion coefficient and the load on the tire determine the cornering force. We have to look up the tire adhesion coefficient, $\mu_{adhension}$, in the wheel slip angle curve. The function of the wheel slip angle curve, $f(\alpha)$, delivers the adhesion coefficient.

$$\mu_{tire} = f(\alpha)$$

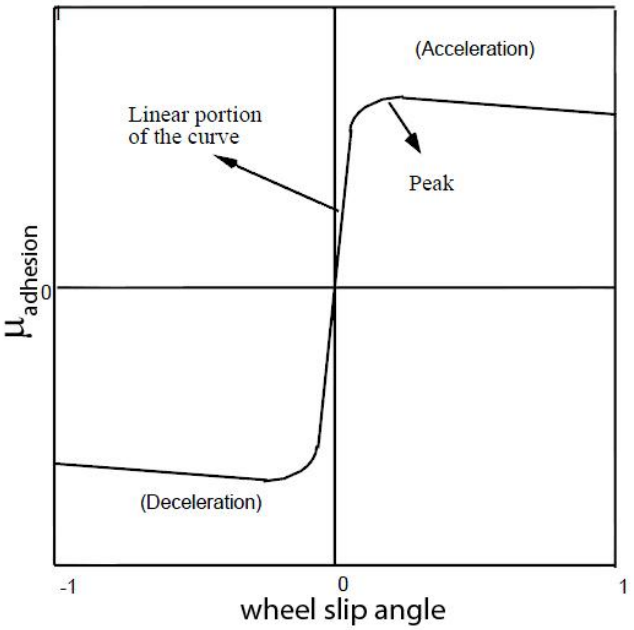

Figure 12: Wheel slip angle curve

The road adhesion coefficient scales this curve.

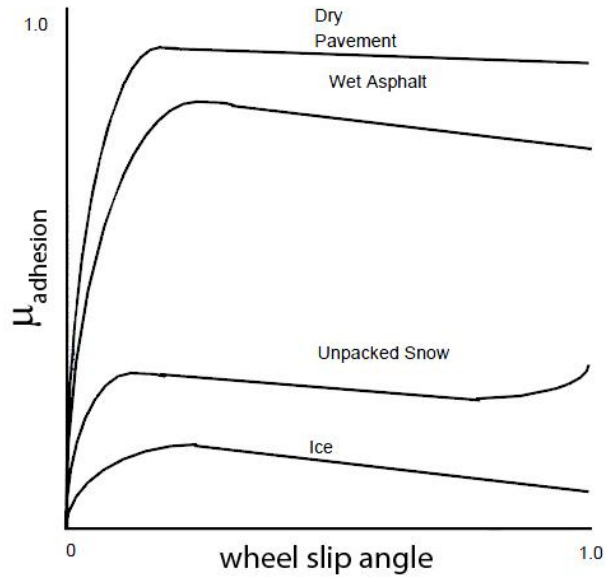

Figure 13: Wheel slip angle curve scaled by road adhesion coefficient

We have to replace this curve with a similar one that has a simple equation to simulate the curve for our game. We can replace it with the following equation. (Ming 1997, 33)

$$\mu_{adhesion} = f(\alpha) * \mu_{road} = \frac{2 * \mu_{max} * \alpha_{max} * \alpha}{\alpha_{max}^2 + \alpha^2} * \mu_{road}$$

This coefficient multiplied by the load on the tire results in the cornering force, $F_{cornering}$. The load on the tire depends on the dynamic weight transfer.

$$F_{cornering} = \mu_{adhension} * F_{load}$$

If we apply the cornering forces of all wheels, they create a torque that is turning the car around its center of mass. We have to check the traction budget of the tire before we can apply the cornering force.

2.3. Different physical surfaces

As we heard in a previous section of the thesis, the tires have a maximum force they can stand, before they start to slip. The amount of this force depends on the load on the wheel and the adhesion coefficient. We can calculate the current load with the dynamic weight transfer. We need the wheel slip angle curve to determine the adhesion coefficient of the tire. We already know this curve from the steering behavior. The produced force decreases after the wheels start to slip. The adhesion coefficient curve describes this loss. The shape of the curve is constant for different surfaces, but the scale is different.

As we heard, different road conditions deliver several coefficients. For example has an icy road a lower coefficient than dry asphalt. The following table shows average peak values for various surfaces.

Surface	Average peak
Dry asphalt, dry concrete	0.8-0.9
Wet asphalt	0.5-0.6
Wet concrete	0.8
Dry off road	0.68
Wet off road	0.55
Gravel	0.6
Ice	0.1
Snow hard packed	0.2

Tab. 3: Adhesion coefficient of different ground surfaces

The wheel slip angle curve delivers the tire adhesion coefficient. A normal tire has a peak adhesion coefficient of about 1.0 and a racecar tire of about 1.5. We have to multiply the current surface coefficient by the tire's coefficient, which we look up with the wheel slip angle function.

$$\mu_{adhesion} = f(\alpha) * \mu_{road}$$

2.4. Summary

Many forces are acting on a driving car. On the one hand, the resistance forces aerodynamic drag, gravity and rolling resistance decrease the velocity of the car. On the other hand, produces the engine a torque, which accelerates the vehicle. The produced torque runs through the transmission and rotates the driving wheels. The driving wheels push against the road and develop the traction force. The traction force accelerates the car. The breaks produce a negative traction force that stops the automobile.

High-speed turning can cause the effects of under steering or over steering. During a turning maneuver, the wheels produce cornering forces. Those forces rotate the car and change the heading and direction of it.

If the traction force and the cornering forces exceed the traction budget of the wheel, the tires start to spin. Different ground surfaces and the dynamic weight transfer influence the adhesion of the tires.

3. Unity3D

Unity3D is a game engine with an integrated physics engine. We are able to program with the C# programming language. We try to implement the physical driving behavior of a buggy in this engine. We do not program exactly C# in Unity3D. Unity Technologies developed a modified version of C# called C# Script. As C# Script doesn't offer a constructor for a class, we have to use the Awake() function instead. The Awake() function is called, when an instance of the class is initialized. Another important function is the Update() function. This is the game loop and is executed every frame. Unity3D delivers plenty components that we can use for our driving behavior besides the modifications of C#.

3.1. Components

We have to calculate a lot of forces that we apply to the car. The car itself is a rigid body from the physical side of view. Unity3D offers a class Rigidbody that possesses a lot of properties and components. One of them is a center of mass. In addition, there are already functions to apply forces to the center of mass or to get the current velocity or acceleration of the rigid body. There is a way to determine the position where we want to add a specific force. This will automatically create a torque to the center of mass. The Rigidbody class offers a lot features we can use for our implementation.

When we create a collider with the shape of the buggy, the physic engine handles the collision. If the car collides with an obstacle, we don't have to consider the physical calculations of the responding forces.

The wheel collider is a special type of a collider and is developed especially for car simulations. The wheel collider simulates the collisions for the attached wheel. Furthermore, it has several assistant functions, which can for example return the current angular velocity of the wheel or the tire's revolutions per minute. In addition, the wheel collider is able to simulate the suspension for the tire. We can also check for every frame the collision point between the wheel and the ground. This is important to get the current surfaces on which the wheel is rolling.

There is already a class to determine the physical properties of an underground. We only have to set up the correct values for the different surface types and attach them to a game object.

4. Implementation of the buggy

The goal is to create a physical simulation for a fictional buggy. The picture below shows the buggy. I have to find realistic figures and properties for the car to create a realistic driving behavior. The challenge is to determine these properties for an imaginary vehicle.

We know all necessary physical equations to calculate the behavior of any car. The next step is to set up specific values for the buggy. According to the fact that the buggy doesn't exist in reality, we have to find existing cars which are similar to the buggy. When we compare the properties of similar vehicles, we are able to estimate realistic figures for the fictional buggy.

We have to set up plenty of properties for the buggy to simulate the driving behavior. I chose the Volvo XC90 3.2 AWD model as reference car for the buggy, because the car body has similar proportions and it has a powerful engine. We have to calculate in every frame the forces, which influence the automobile. Therefore, we put the following calculations in the Update() function of the car script.

Figure 14: Volvo X90 and the buggy

4.1. Software architecture

The goal was to develop a software architecture, which keeps the possibility to add new cars. I considered that another feature of the concept could be to create a car type and an engine type individually. Thereby, we could use the different car types and engine types as construction sets. The concept is for a browser game, which has an ingame shop system that offers upgrades for your vehicles and weapons. Therefore, it is possible with this concept to buy one car and upgrade it with different powerful motors.

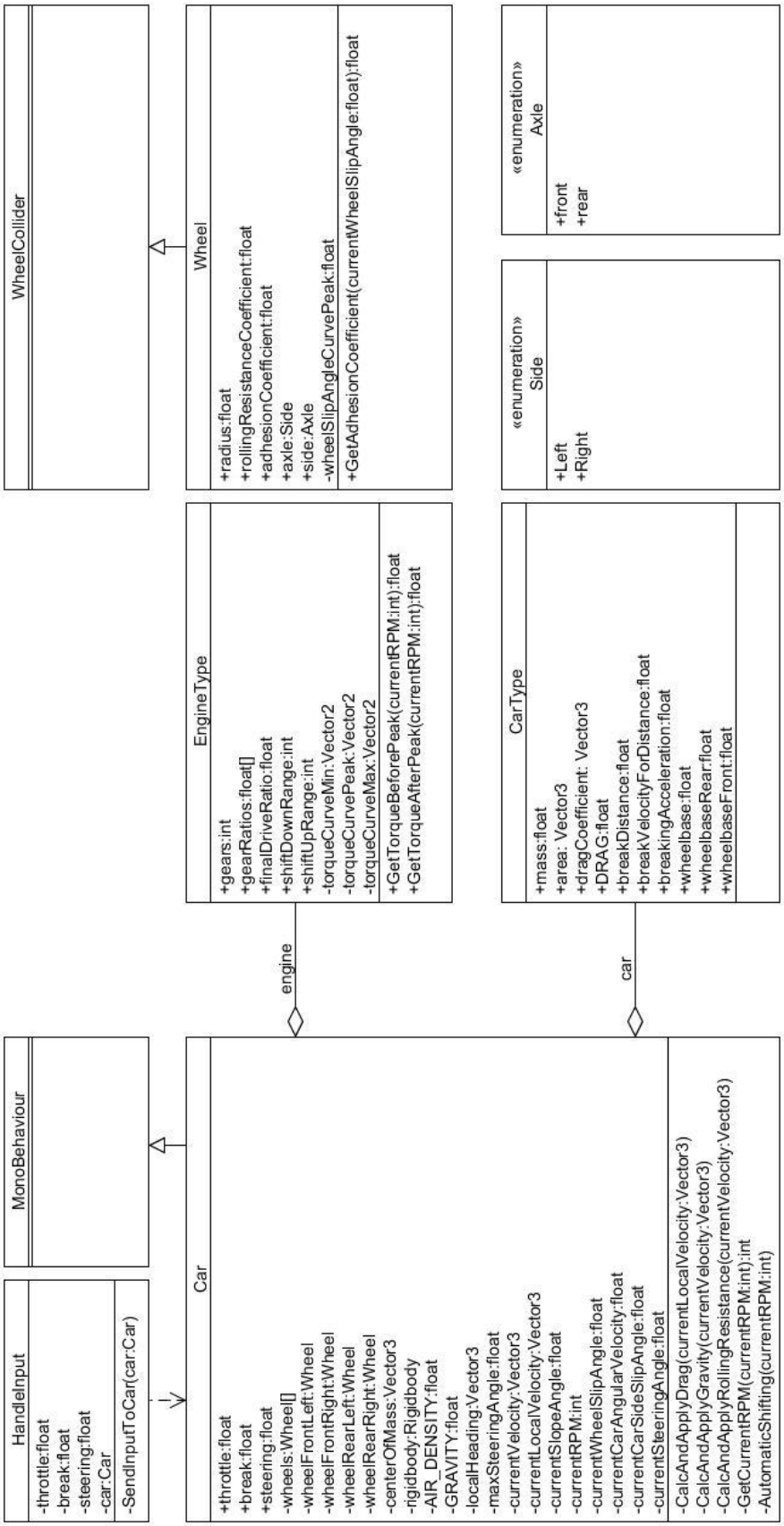

Figure 15: Software architecture

4.2. Aerodynamic drag

The aerodynamic drag force slows down the car in the opposite direction of the velocity vector. Therefore, we need the buggy's current velocity vector, $\vec{v}$. The buggy is a rigid body and we get the current velocity by `currentVelocity = rigidbody.velocity`. We have to estimate the front area of the buggy to calculate the drag force. We can also store the side and top area in a vector, $\vec{A}$. The result will be more accurate, because the car has another drag resistance if it drifts sideways.

$$\vec{A} = \begin{pmatrix} front\ area \\ side\ area \\ top\ area \end{pmatrix}$$

We have to compare the buggy to the Volvo X90 to get realistic values. The car bodies of the two vehicles have similar dimensions. Only the wheels of the buggy are bigger. Therefore, we can calculate with the measures of the Volvo.

Figure 16: Dimensions of the Volvo X90 and the buggy

$$\vec{A} = \begin{pmatrix} 2.1m * 1.7m \\ 4.8m * 1.7m \\ 2.1m * 4.8m \end{pmatrix} = \begin{pmatrix} 3.57\ m \\ 8.16\ m \\ 10.8\ m \end{pmatrix}$$

The trick to store different values in a vector works also for the drag coefficient, $\vec{C}_{drag}$. The buggy has a more aerodynamic shape in the forward direction, than in the sideways direction. When we store the values in a vector, we can determine a higher value for the side coefficient of the car.

$$\vec{C}_{drag} = \begin{pmatrix} front\ coefficiency \\ side\ coefficiency \\ top\ coefficiency \end{pmatrix}$$

The Volvo X90 has a drag coefficient of 0.36 in the forward direction. (Carinf 2012) We adopt this value for the buggy and estimate a side and top coefficient of 0.8.

$$\vec{C}_{drag} = \begin{pmatrix} 0.36 \\ 0.8 \\ 0.8 \end{pmatrix}$$

We have to transform the velocity vector from the world space to the local coordinate system of the buggy, because the area vector and the drag coefficient vector are defined in the local space of the buggy.

The function `transform.InverseTransformDirection(currentVelocity)` converts the velocity vector to the local coordinate system of the buggy. The speed is the magnitude of the velocity vector, $|\vec{v}|$, `speed = currentVelocity.magnitude`. The last missing parameter is the mass density of air, ρ. Air has a mass density of about 1.29 kg/m³. Now, we are able to calculate the aerodynamic drag force.

$$\vec{F}_{drag} = -\left(\frac{1}{2}\right) * \rho * \vec{v} * \ |\vec{v}| * \vec{A} * \vec{C}_{drag}$$

The mass density of air, the area vector and the drag coefficient vector are constant. We can set them in the Awake() function of the buggy. We merge these values by multiplying them and store the vector as drag constant, $\vec{\mu}_{drag}$.

```
Vector3 DRAG = -0.5 * areas * dragCoefficient * airDensity;
```

We don’t have to calculate this in every single frame. According to this, the simpler equation per frame is:

$$\vec{F}_{drag} = \vec{\mu}_{drag} * \vec{v} * |\vec{v}|$$

$\vec{\mu}_{drag}$ is negative because the drag force always points in the opposite direction of the velocity vector. The last step is to apply the aerodynamic drag force to the buggy's center of mass. We have to consider that the force vector is in the local coordinate system of the car. Therefore, we use `rigidbody.AddRelativeForce(dragForce)`.

4.3. Gravity

We concentrate on the gravity force that acts parallel to the ground. This force decelerates the buggy if it drives upwards a hill and accelerates the car if it rolls downwards. We have to find out in which direction the car is driving. The angle between an upward vector in world coordinates and the heading vector of the buggy determines in which direction the car is driving. If the angle is greater than 90 degrees, the car is moving downwards. Otherwise, the car drives upwards. We have to transform the three-dimensional vector to a two-dimensional vector, because we calculate the slope of the ground for the straight-line gravity force. Therefore, the x-axis and the z-axis of the buggy's local coordinate system are important. We need the following code to get the heading vector of the buggy and an up vector from world space to transform these vectors to the local space of the car.

```
localUp = transform.InverseTransformDirection(Vector3.up);
localHeading = transform.InverseTransformDirection(transform.forward);
```

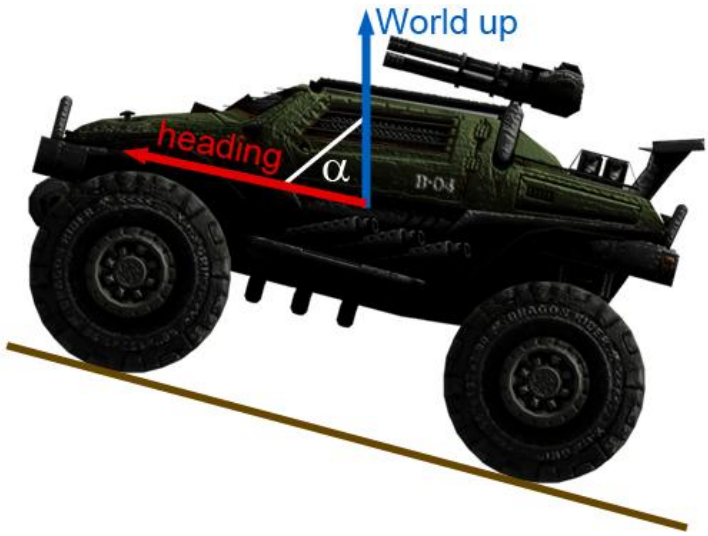

Figure 17: Direction of the gravity force

The next step is to create a two dimensional vector with the x and z coordinates and calculate the inner product of the vectors. The inner product returns the angle, α, between the two vectors. If the angle is greater than 90°, the gravity force acts in the same direction than the velocity. If the angle is smaller or equal 90°, the gravity force acts in the opposite direction of the velocity. The velocity of the buggy determines only the direction of the gravity force, because of that we have to normalize the vector.

$$\alpha > 90, \qquad dir = +1$$
$$\alpha \leq 90, \qquad dir = -1$$

There are still some parameters missing. The mass of the buggy, m, is 2200 kg, similar to the reference Volvo. (Volvocars 2012) The acceleration due to gravity, g, is 9.81 m/s². We have to calculate the inner product of the car's heading vector and the worlds forward vector to get the slope angle of the ground, θ. We convert the vectors to the local coordinate system of the buggy and create two-dimensional vectors with their x and the z-values.

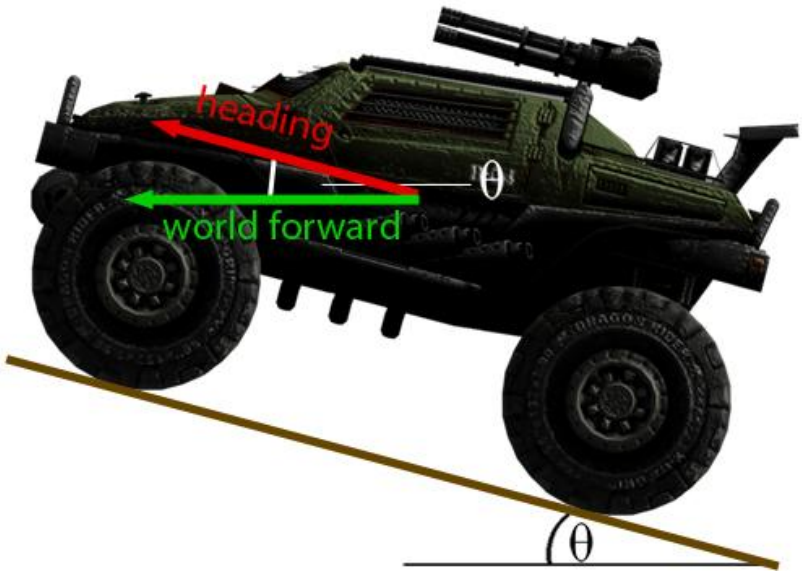

Figure 18: Calculations of the ground slope angle

```
localForward = transform.InverseTransformDirection(Vector3.forward);
localHeading = transform.InverseTransformDirection(transform.forward);
```

$$\vec{F}_{gravity} = \left(dir * \frac{\vec{v}}{|\vec{v}|} \right) * m * g * \sin\theta *$$

The result of this equation is the gravity force vector, which we can apply to the buggy's center of mass. We used for the calculation the velocity in world space. Therefore, we can use `rigidbody.AddForce(gravityForce)` to apply the force.

4.4. Rolling resistance

We can use a lot of values from earlier equations to calculate the rolling resistance force. We know already the velocity, $\vec{v}$, the mass of the buggy, the acceleration due gravity, g, and the slope angle of the ground, θ. The rolling resistance force acts in the opposite direction of the velocity. The last missing piece is the rolling resistance coefficient, μ_{rr}, that depends on the tire type.

"Typical car tires have a rolling resistance coefficient of about 0.015, while truck tires fall within the range of 0.006 to 0.01." (Bourg 2002, 169) We choose for the buggy a rolling resistance coefficient, μ_{rr}, of 0.015.

$$\vec{F}_{rr} = \left(-\frac{\vec{v}}{|\vec{v}|}\right) * \mu_{rr} * m * g * \cos\theta$$

We apply the calculated rolling resistance force to the buggy's rigid body by `rigidbody.AddForce(rollingResistanceForce)`.

4.5. Traction force

The engine produces the traction force. The throttle controls the turnover rate of the engine. The user input handles the throttle value. The throttle value has a range from zero to one. When the player pushes the throttle completely down, the throttle value should increase from zero to one. If the player steps off the throttle, the value is zero. When a user applies a throttle input, we have to increment the current engine rpm. We have to consider that the rpm shouldn't increase linear in the simulation.

Therefore, we determine the ratio between the current rpm and the maximum rpm and multiply it by two. The result is the parameter of a function to get a coefficient. This coefficient handles the increase of the engine's turnover rate. This function delivers a high coefficient at a low turnover rate and a low coefficient for a high turnover rate. (Unity Technologies 2012)

$$ratio_{rpm} = \frac{currentRPM}{maxRPM} * 2$$

$$ratio_{rpm} \leq 1, \qquad \mu_{increment} = f(ratio_{rpm}) = 10 - ratio_{rpm} * 9$$

$$ratio_{rpm} > 1, \qquad \mu_{increment} = f(ratio_{rpm}) = 1.9 - ratio_{rpm} * 0.9$$

If the throttle is pushed down, we have to increment the current rpm by the `RPM_INCREMENT` constant multiplied by the calculated coefficient, $\mu_{increment}$. The value for the `RPM_INCREMENT` constant is 200 revolutions per second for the buggy. When the throttle value is zero, we decrease the current rpm by 300 per second.

We need minimum and maximum rpm values for the automatic shifting. We have to shift up in a higher gear when the current rpm exceed the maximum rpm. Vice versa, we have to shift in a lower gear if the rpm falls under the minimum rpm. We set the minimum rpm to 1650 and the maximum to 6000. We have to calculate a new engine rpm when we shift in a higher or lower gear.

$$rpm_{new} = rpm_{old} * \frac{g_{new}}{g_{old}}$$

We copy the gear ratios from the Volvo X90.

Gear	Gear ratio
First gear	4.15
Second gear	2.37
Third gear	1.56
Fourth gear	1.16
Fifth gear	0.86
Sixth gear	0.67
Final drive ratio	4.06

Tab. 4: Gear ratios Volvo X90

After we updated the current engine rpm, we can look up the delivered torque in the torque curve. Consequently we have to determine a torque curve for the buggy. “For game programming purposes, the torque curve needs to be expressed as a mathematical expression. The easiest way to mathematically model a torque curve is with a series of straight lines.” (Palmer 2005, 222)

The following figure shows the torque curve of the Volvo X90.

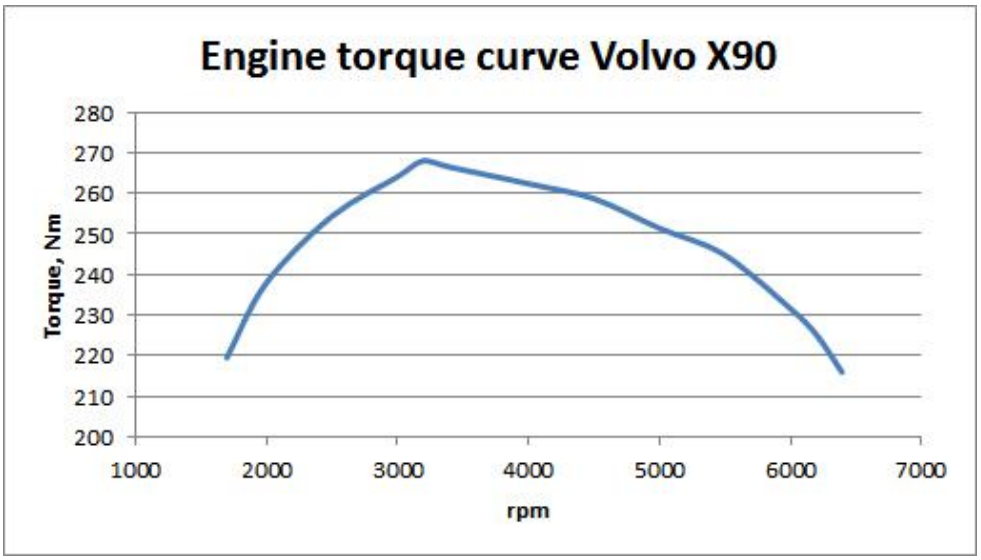

Figure 19: Engine torque curve from the Volvo X90

If we want to model this torque curve with straight lines, we have to pick the start, peak and end value of the curve. The start point is at 220 Nm torque and 1650 rpm. The next point in the curve is the peak value with 268 Nm torque and 3200 rpm. As the last point we choose the redline rpm of 6500 which delivers a torque of 215 Nm. We get as result the torque curve of the buggy modeled with straight lines.

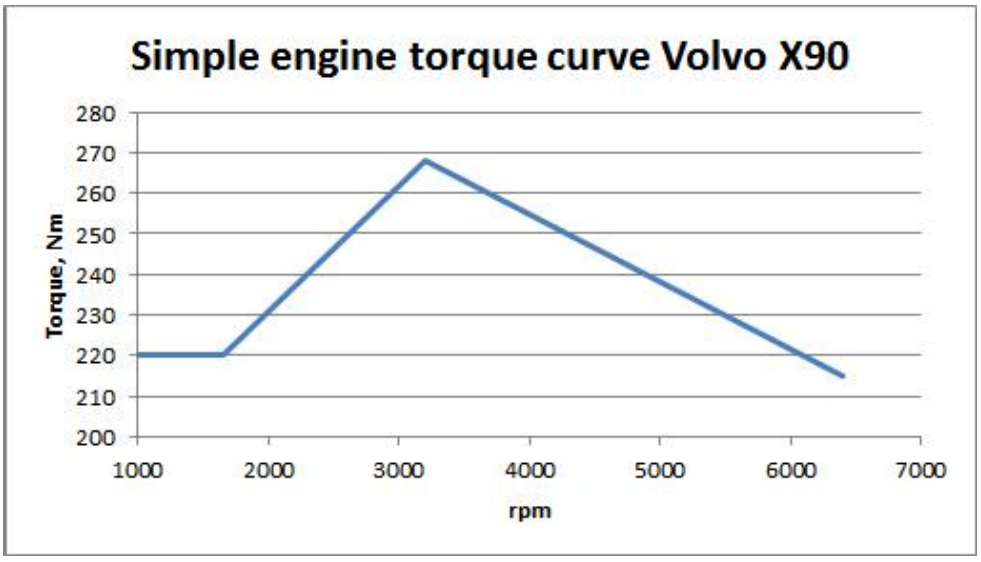

Figure 20: Simpler engine torque curve from the Volvo X90

We have to determine two different functions to get the engine torque for a certain engine rpm. The first function describes the slope before the peak and the second function the slope after the peak. We must estimate the gradient of the two lines to get the following two functions.

$$rpm \leq 3200, \qquad \tau_{engine} = f(rpm) = 0.031 * rpm + 168.85$$
$$rpm > 3200, \qquad \tau_{engine} = f(rpm) = 316 - 0.015 * rpm$$

We take the first function if the current rpm are between 1650 and 3200 and the second function for a turnover rate between 3201 and 6500 rpm. Now, we are able to determine the delivered engine torque for a given rpm. The traction force equals the wheel torque, τ_{wheel}, divided by the wheel radius, r_{wheel}. The engine torque, τ_{engine}, multiplied by the current gear ratio, g_k, the final drive ratio, G, and the throttle position, $throttle$, determine the wheel torque.

$$F_{traction} = \frac{\tau_{wheel}}{r_{wheel}} = \frac{\tau_{engine} * \ g_k * G * throttle}{r_{wheel}}$$

The calculated traction force equals the forces of both driving wheels. We have to check the traction budget particular for each tire before we apply the force. The traction budget also includes the cornering forces. Therefore, we have to wait until we have determined the cornering forces.

4.6. Breaking

The Volvo X90 has a breaking distance of 42.06 m for a velocity of 27.72 m/s. (TheAutoChannel 2012) We can calculate the breaking acceleration with this information.

$$a_{breaking} = -\frac{v^2}{2x} = -\frac{27.72^2}{2 * 42.06} = -9.13\ m/s^2$$

We have to check the traction budget for each tire individually before we apply the breaking force. Again, the traction budget also includes the cornering forces. Therefore, we have to calculate the cornering forces before we can apply the breaking force for a wheel.

We apply the breaking force particular for each wheel. If we want to simulate a handbrake, we apply the breaking force only to the rear wheels. We have to multiply the acceleration by

the mass, which is resting on the wheel, to get the breaking force. Therefore, we have to calculate how much load is pushing down on each single wheel. The amount of load depends on the dynamic weight transfer.

$$F_{breaking} = a_{breaking} * m_{wheel} = -\frac{v^2}{2x} * m$$

4.7. Dynamic weight transfer

If the current wheel is a front tire, we need the following equation.

$$F_{front} = \frac{wheelbase_{rear}}{wheelbase_{total}} * m * g - \left(\frac{height}{wheelbase_{total}}\right) * m * a$$

$wheelbase_{total}$ is the distance between the two axle and $height$ is the distance between the ground and the buggy's center of mass. a is the acceleration of the car which is the traction force divided by the mass of the vehicle.

$$F_{rear} = \frac{wheelbase_{front}}{wheelbase_{total}} * m * g + \left(\frac{height}{wheelbase_{total}}\right) * m * a$$

The calculations return the load forces per axle, because we are calculating for each wheel individually we have to divide the axle load by two. This delivers the load force on the current tire.

$$F_{wheel\ load} = \frac{F_{front}}{2}$$

$$F_{wheel\ load} = \frac{F_{rear}}{2}$$

We get the mass on the current tire if we divide the load force on the wheel by the acceleration due to gravity.

$$m_{wheel} = \frac{F_{wheel\ load}}{g}$$

4.8. Turning

We need the wheel slip angle to determine the cornering force. The wheel slip angle depends on the car's side slip angle, β. To calculate β we need the velocity vector of the car in the vehicle's local coordinate system.

$$\beta = \arctan\left(\frac{v_y}{v_x}\right)$$

```
localVelocity = transform.InverseTransformDirection(currentVelocity);
sideSlipAngle = Mathf.Atan(localVelocity.y/localVelocty.x);
```

The next unknown parameter is the angular velocity of the car, ω_{car}. The `Rigidbody` class handles the angular velocity of the car.

```
angularVelocity = rigidbody.angularVelocity;
```

We also need the distance from the rear, $wheelbase_{rear}$, and front axle, $wheelbase_{front}$, to the car's center of mass to calculate the wheel slip angle. We store the position of the center of mass in the vector `cm` to keep the opportunity to change the position of the center of mass later. The position of the cm influences the driving behavior. The $wheelbase_{rear}$ is the absolute value of the wheel's x-coordinate minus the center of mass' x-coordinate. The same works for the front wheels.

```
wheelbaseRear = Mathf.Abs(wheelRearLeft.x – cm.x);
wheelbaseFront = Mathf.Abs(wheelFrontLeft.x – cm.x);
```

The last parameter of the wheel slip angle is the steering angle of the wheels, δ. The player controls the steering angle. We have to set a maximal possible steering angle. We set a maximum steering angle of 15° for our buggy. The user input increases the current steering angle towards the maximum. The current steering angle decreases back to zero if the player stops pressing the turning button.

The last missing parameter is the velocity transformed in the buggy's local coordinate system. We already stored this in the `localVelocity` variable.

$$\alpha_{front} = \left(\frac{localVelocity_y + \omega * wheelbase_{rear}}{\|vlocalVelocity_x\|}\right) - \delta * sign(localVelocity_x)$$

$$\alpha_{rear} = \left(\frac{localVelocity_y - \omega * wheelbase_{front}}{\|localVelocity_x\|}\right)$$

We have determined the wheel slip angle. The cornering force depends on the adhesion coefficient. The function of the wheel slip angle curve returns us the adhesion coefficient of the tire for a certain wheel slip angle. We use the function below to look up the tire's adhesion coefficient of the buggy.

$$\mu_{tire} = f(\alpha) = \frac{2 * \mu_{max} * \alpha_{max} * \alpha}{\alpha_{max}^2 + \alpha^2}$$

We have to set a peak value for the adhesion coefficient. The wheel slip angle curve for the buggy has a maximum adhesion coefficient of 1.2 at a wheel slip angle of 5 degrees.

$$\mu_{tire} = f(\alpha) = \frac{2 * 1.2 * 5 * \alpha}{5^2 + \alpha^2}$$

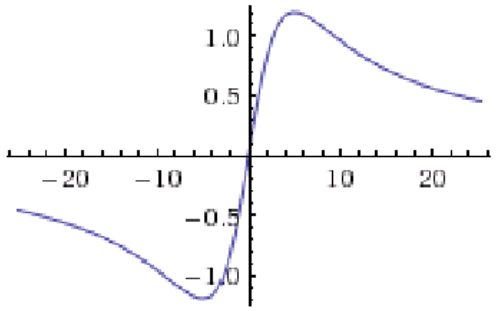

Figure 21: Simpler wheel slip curve

The road surface adhesion coefficient multiplied by the adhesion coefficient of the wheel delivers us the adhesion coefficient for the cornering force. We can check on which physical material the wheel is rolling with the wheel collider of Unity3D. It could be possible that the wheels stand on different surfaces. Because of this, we have to make the following calculations for each wheel individually.

```
WheelHit hit;
currentWheel.GetGroundHit(out hit);
groundAdhesionCoefficient = hit.collider.material.staticFriction;
adhesionCoefficient = groundAdhesionCoefficient * tireAdhesionCoefficient;
```

$$\mu_{adhesion} = \mu_{tire} * \mu_{road}$$

We are able to calculate the adhesion coefficient for a given wheel slip angle. Now, we have to determine how much load is on the current wheel. Therefore, we need to look at the dynamic weight transfer. The load force of the wheel multiplied the adhesion coefficient equals the cornering force.

$$F_{cornering} = \mu_{max} * F_{wheel\ load}$$

We have to check the traction budget before we can apply the cornering force to the current wheel.

4.9. Traction budget

The following calculations are for each wheel individually. We have calculated the traction force, the breaking force and the cornering force. The breaking forces are negative traction forces. The forces can only be applied, if the combined force doesn't exceed the traction budget. We get the maximum acceleration the car can stand before slipping by the following equation. We have already calculated the slope angle of the ground, θ, for the gravity force. $\mu_{adhesion}$ is the combination of the road adhesion coefficient and the tire adhesion coefficient.

$$a_{max} = \mu_{adhesion} * g * \cos\theta$$

The traction force divided by two times the mass delivers us the acceleration for one driving wheel. We calculate the traction force only for the driving wheels. The rear wheels are the drive wheels of the buggy. For the front wheels $a_{traction}$ is zero.

$$a_{traction} = \frac{F_{traction}}{2 * m}$$

The calculated cornering force is already the force of the current wheel. We have to divide the cornering force by the mass on the current tire. How to calculate the mass on the current tire is already described in the dynamic weight transfer topic.

$$a_{cornering} = \frac{F_{cornering}}{m_{wheel}}$$

Now, we check if the combined cornering acceleration and traction acceleration exceed the maximal acceleration of the tire.

$$a_{total} = \sqrt{a_{traction}{}^2 + a_{cornering}{}^2}$$

If they exceed the traction budget, we must scale the straight line and cornering acceleration by the $ratio$ between a_{max} and a_{total}.

$$ratio = \frac{a_{max}}{a_{total}}$$
$$a_{new\ straightline} = ratio * a_{straightline}$$
$$a_{new\ cornering} = ratio * a_{cornering}$$

Before we can apply the accelerations to the wheel, we have to convert them back to forces by multiplying them with the mass on the tire.

$$F_{new\ straightline} = a_{new\ straightline} * m_{wheel}$$
$$F_{new\ cornering} = a_{new\ cornering} * m_{wheel}$$

The last problem is that the forces have no directions. The direction for the traction force is the forward vector of the buggy and the forward vector multiplied by -1 for the breaking force. It is important to add the force at the position where the wheel collides with the ground. Thereby the physic engine automatically calculates and applies a torque to the buggy's center of mass.

This is possible with the function `rigidbody.ApplyForceAtPosition()`.The collision information where the wheel collides with the ground, is stored in the variable `hit`.

```
Vector3 wheelTractionForce = wheelTractionAcceleration * mass/2;
wheelTractionForce *= transform.forward;
rigidbody.ApplyForceAtPosition (hit.point, wheelTractionForce);
```

The cornering force is pointing to the wheel's right or left side depending on the steering direction. The wheel slip angle decides the side in which the car is turning. If the wheel slip angle is greater than zero it is a right turn and we take the right vector of the wheel. Otherwise, it is a left turn and we have to multiply the right vector of the wheel by -1.

$$\vec{F}_{cornering} = a_{cornering} * m_{wheel} * (\pm 1) * \overrightarrow{right}_{wheel}$$

Again, we apply the cornering force at the collision position of the ground and the tire.

```
rigidbody.ApplyForceAtPosition (hit.point, wheelcorneringForce);
```

4.10. Unity3D Components

4.10.1. WheelCollider and suspension

The WheelCollider can simulate the wheel suspension. Therefore, you have to set two properties and attach the mesh of the wheel as a child object of the WheelCollider.

The `WheelCollider.suspensionDistance` determines the maximum expansion distance of the suspension. The suspension of the car is simulated by a spring. The variable `WheelCollider.suspensionSpring` manages the properties of the spring.

`WheelCollider.suspensionSpring.spring` defines the forces of the spring. A higher value makes the spring move faster. `WheelCollider.suspensionSpring.damper` dampens the speed of the spring and makes it move slower.

4.10.2. Physic Material

We can use the physic material component to create different ground surfaces. We have to set up the dynamicFriction and staticFriction variables to the peak value of our surface. The physical materials contain much more settings we could define, but we have not discussed in this thesis. The last thing we have to do is to attach the physical material to the ground collider.

5. Conclusion

I started to explain and describe the basic physic model of a driving. The theoretical part involves which forces influence a moving car and how the front wheels produce cornering force to turn the vehicle. It also explains how too much acceleration can cause spinning tires and how a slippery surface affect the adhesion between the ground and the wheels. I have developed a software architecture for the vehicle dynamics. This architecture provides the possibility to create new vehicles or different engine types. We can add a Ferrari as second car

by importing a Ferrari model and by creating a new `carType` and a new `EngineType`. I described how we can implement the physic calculations and the concept in the Unity3D game engine. The implementation part refers to a fictional buggy, therefore I explained how to find reference figures of existing cars to set up a realistic driving behavior for the imaginary vehicle.

A requirement was to develop a realistic driving behavior for a buggy but there are some features missing. I didn't describe the physics of the suspension, I only explained how we could set it up in Unity3D with the WheelCollider component. Other missing physical features are the abs (anti-lock braking system) and the tcs (traction control system). I didn't go into the abs and tcs features for this concept because of game balancing purposes. The buggy is able to drive much faster than the tanks in the game, because of that it is difficult to hit the car. Therefore, it the driving behavior should be challenging for the player when he drives at high velocity. It is tricky to control an breaking automobile at high speeds without an abs. The same intension was taken for the tcs feature. When the player makes a mistake and stops the car, it is more difficult to accelerate without the tcs. The player has to accelerate slowly the buggy because otherwise the car does a burnout. These are the reasons why I decided that the balancing purposes are more important than an exact realistic driving behavior.

It is possible to create a physically correct fun racer game with this concept. When you want to develop a realistic racing game, you have to expand it by features like abs and tcs. Furthermore, the realistic properties of a car are only the starting basis for your car simulation. You have to run many tests and have to tweak the values of the vehicle until they fit in your desired driving behavior.

List of Figures

List of Tables

References

Beckman, Brian Phd. *The Physics of Racing.* Unpublished articles, 1991.

Bourg, David M.. *Physics for game developers*. Beijing: O'Reilly, 2002.

Carinf. "Volvo XC90 T6 (2002) - technical specifications." carinf.com - car information and technical specifications. http://www.carinf.com/en/4480412435.html (accessed January 18, 2012).

Conger, David. *Physics modeling for game programmers*. Boston, MA: Course Technology PTR, 2004.

Ming, Qian. *Sliding Mode Controller Design for ABS System.* PhD diss., Virginia Polytechnic Institute and State University. 1997.

Monster, Marco. "Car Physics." Car Physics for Games. http://www.asawicki.info/Mirror/Car%20Physics%20for%20Games/Car%20Physics%20for%20Games.html (accessed January 16, 2012).

Palmer, Grant. *Physics for game programmers*. Berkeley, CA: Apress ;, 2005.

Rototest . "Powertrain Performance Graph for Volvo XC90 3.2 AWD." Rototest Research Institute. http://www.rri.se/popup/performancegraphs.php?ChartsID=756 (accessed January 18, 2012).

TheAutoChannel. "2008 Volvo XC90 Overview." The Auto Channel. http://www.theautochannel.com/news/2007/10/09/066255.html (accessed January 18, 2012).

Volvocars. "Volvo XC90." Volvo cars. www.volvocars.com/intl/all-cars/volvo-xc90/details/pages/technical-spec.aspx (accessed January 18, 2012).

Wheels. "2011 Volvo XC90 AWD." Wheels.ca. www.wheels.ca/newVehicles/profile/789439 (accessed January 18, 2012).