

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Visual Simulation of Rain

Bc. Giang Chau Nguyenová

Supervisor: doc. Ing. Jiří Bittner, Ph.D.

Field of study: Open Informatics

Subfield: Computer Graphics

May 2021

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Nguyenová** Jméno: **Giang Chau** Osobní číslo: **457753**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**
Studijní program: **Otevřená informatika**
Specializace: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Vizuální simulace deště

Název diplomové práce anglicky:

Visual Simulation of Rain

Pokyny pro vypracování:

Prostudujte stávající metody pro vizuální simulaci deště. Soustředte se na jevy, které je vhodné použít v otevřených scénách používaných pro virtuální dopravní simulátory. V rešerši zmapujte i vizuální kvalitu simulace těchto jevů v současných videohrách (zejména v závodních simulátorech).

Cílem práce je vybrat sadu metod pro realizaci vizuálních jevů souvisejících s deštěm s nastavitelným stupněm intenzity (slabý déšť / intenzivní déšť), které umožní zvýšit vizuální variabilitu prostředí dopravních simulací. Soustředte se na simulaci dopadu deště na vzhled vozovek a jejich okolí (mokrý vozovka, kaluže, vodní tříšť okolo projíždějících vozidel). Cílová implementace zvolených metod bude vytvořena v prostředí Unity s možným využitím rendereru Octane.

Výstupem práce bude zmapování existujících metod a kvality jejich dosažitelných výstupů a implementace vizuální simulace deště a souvisejících jevů. Implementaci otestujte na nejméně třech scénářích s různou intenzitou deště. Vyhodnoťte výpočetní náročnost simulace v závislosti na parametrech implementované metody. Výsledky práce srovnajte s fotografiemi podobných situací a podrobte je jednoduchému kvalitativnímu uživatelskému testu s cílem zmapovat vizuální věrohodnost simulace.

Seznam doporučené literatury:

- [1] Carles Creus and Gustavo A Patow. 2013. Realistic rain rendering in realtime. *Comput Graph* 37, 1–2 (2013), 33–40.
- [2] Yoann Weber, Vincent Jolivet, Guillaume Gilet, and Djamchid Ghazanfarpour. 2015. A multiscale model for rain rendering in real-time. *Comput Graph* 50, (2015), 61–70.
- [3] Y Weber, V Jolivet, G Gilet, K Nanko, and D Ghazanfarpour. 2016. A phenomenological model for throughfall rendering in real-time. *Comput Graph Forum* 35, 4 (2016), 13–23.
- [4] Markéta Rejdová. 2009. Realistické zobrazování deště. Bakalářská práce, ČVUT FEL.
- [5] Aleš Renner. 2016. Rain simulation in large open space scenes. Semestrální projekt ČVUT FEL.
- [6] B. Sun, R. Ramamoorthi, S. G. Narasimhan, and S. K. Nayar. 2005. A practical analytic single scattering model for real-time rendering, *Acm T Graphic*, vol. 24, no. 3, p. 1040.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jiří Bittner, Ph.D., Katedra počítačové grafiky a interakce

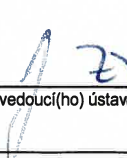
Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **28.07.2020**

Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **19.02.2022**


doc. Ing. Jiří Bittner, Ph.D.
podpis vedoucí(ho) práce


podpis vedoucí(ho) ústavu/katedry


prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání

Podpis studentky

Acknowledgements

I would like to express my gratitude and appreciation to my supervisor doc. Ing. Jiří Bittner, Ph.D. for his consistent support and guidance during the project and later the writing of this thesis.

Furthermore I would like to thank my family and friends for moral encouragement.

Declaration

I declare that this thesis represents my work and that I have listed all the literature used in the bibliography.

Prague, 20 May 2021

Prohlašuji, že jsem předloženou práci vypracovala samostatně, a že jsem uvedla veškerou použitou literaturu.

V Praze, 20. května 2021

Abstract

The project of the thesis focuses on increasing visual diversity in traffic simulators by simulating multiple rain phenomena, which can be seen in the real world. Thorough research of existing methods should help to decide which rain effects can remarkably modify how outdoor scenes appear. The rainy weather in several racing games is then being analyzed and visually compared, which later forms a general idea of the rain simulation design. The work also describes how each rain phenomena is implemented in the Unity game engine with the use of an external plugin OctaneRender for ray-traced outputs.

Keywords: rain simulation, traffic simulator, outdoor environment, Unity game engine, Octane render

Supervisor: doc. Ing. Jiří Bittner, Ph.D.

Abstrakt

Projekt této diplomové práce se zaměřuje na zvýšení vizuální rozmanitosti v dopravních simulátorech, a to simulací různých jevů deště, které lze vidět v reálném světě. Důkladný výzkum stávajících metod by měl pomoci při výběru deštových efektů, které by mohly význačně změnit vzhled venkovních scén. Rešerše také porovnává vzhled deštivého počasí v několika závodních simulátorech, což je později využito při návrhu deštové simulace. V práci je také popsáno jak jsou jednotlivé jevy implementovány v herním engine Unity s využitím přídatného modulu OctaneRender k získání výstupů, které byly vykresleny metodou sledování paprsků.

Klíčová slova: simulace deště, dopravní simulace, venkovní prostředí, herní engine Unity, render Octane

Překlad názvu: Vizuální simulace deště

Contents

1 Introduction	1	5.1.3 Rain	58
1.1 Goals	1	5.2 User Study	59
1.2 Thesis Structure	1	5.2.1 Participant 1	59
2 Analysis	3	5.2.2 Participant 2	59
2.1 Rain in Real World	3	5.2.3 Participant 3	60
2.2 Rain Effects in Racing Games	6	5.3 Performance	60
2.3 Existing Methods of Rain		5.4 Problems and Limitations	61
Simulation	9	6 Conclusion	63
2.3.1 Rainfall	9	6.1 Work Summary	63
2.3.2 Multiple Rain Phenomena	12	6.2 Future Work	64
3 Rain Phenomena	17	A Bibliography	65
3.1 Raindrops	17	B Outputs	71
3.1.1 Size of Raindrop	17	C List of File Attachments	81
3.1.2 Shape of a Raindrop	18	D User Manual	83
3.1.3 Velocity of a Raindrop	19	D.1 Creating a Wet Road Material	83
3.1.4 Appearance of a Raindrop	20	D.2 Creating a PBR Override Wet	
3.2 Splashes and Ripples	21	Road Material	84
3.3 Fog and Glows	22	D.3 Adjusting Car models	85
3.4 Wet Surfaces	24	D.4 Rain Manager	85
4 Visual Simulation of Rain	27		
4.1 Impact of Water on Roads	27		
4.1.1 Wet Roads	27		
4.1.2 Puddles	29		
4.2 Dynamic Simulation of Puddles	30		
4.2.1 Water Surface and Ripples	30		
4.2.2 Splashes and Sprays	31		
4.3 Rainfall	34		
4.3.1 Raindrop Particles	34		
4.3.2 Rendering Rain Streaks	36		
4.3.3 Interaction of Raindrops	37		
4.3.4 Particle Alignment with Moving			
Camera	38		
4.4 Implementation	38		
4.4.1 Appearance of Wet Roads and			
Puddles	39		
4.4.2 Interactive Puddles	41		
4.4.3 Rain Container	43		
4.4.4 Raindrop Splashes and Ripples	45		
4.4.5 Rain Control	47		
4.4.6 Ray-Traced Outputs	47		
4.4.7 System Overview	48		
5 Results	53		
5.1 Visuals	53		
5.1.1 Wet Roads	53		
5.1.2 Driving Cars	56		

Figures

2.1 Bright Streaks and Reflections . . .	4	4.11 Blending Puddle Map	40
2.2 Puddles	4	4.12 Puddle Reflections	41
2.3 Dry and Wet Difference	5	4.13 Wave Texture	42
2.4 Rainfall	5	4.14 Wave Behind Wheel	43
2.5 Driveclub	7	4.15 Car Prefab with Splash Objects	44
2.6 Forza Horizon 2 – Windscreen . . .	8	4.16 Rain Container	45
2.7 Project Cars 2 – Mist	8	4.17 Rain with Moving Camera	46
2.8 Wang and Wade – Double Cone .	9	4.18 Rain Splash Billboard	46
2.9 Garg and Nayar – Rain streaks .	10	4.19 Ripple Normal Maps	47
2.10 Tatarchuk – Rainfall	11	4.20 Rain Control Variables	48
2.11 Tariq – Rain	11	4.21 Puddle Areas	49
2.12 Puig-Centelles et al. – Rain Container	12	4.22 Baked Billboards	49
2.13 Rousseau et al. – Raindrop Refraction	13	4.23 System Overview	51
2.14 Changbo et al. – Vision Persistence	13	5.1 Wet Road (Unity)	54
2.15 Tatarchuk – Splashes	14	5.2 Puddles (Unity)	54
2.16 Creus and Patow – Rain Particles	15	5.3 Puddles (Octane)	55
2.17 Weber et al. – Streak patterns	16	5.4 Road Reflections (Octane)	55
2.18 Weber et al. – Heavy Rain	16	5.5 Car Headlights	56
3.1 Raindrop shape	18	5.6 Waves in Puddles	56
3.2 Raindrop Distortion	19	5.7 Car Splashes	57
3.3 Terminal Velocity	20	5.8 Car Splashes (Back)	57
3.4 Raindrop Appearance	21	5.9 Rainfall	58
3.5 Stationary Raindrop	21	5.10 Rainfall (Octane)	58
3.6 Refraction Radiance	22		
3.7 Raindrops Field of View	22		
3.8 Type of Splash	23		
3.9 Light Scattering Particle	23		
3.10 Glowing Lights	24		
3.11 Water-Air Interface	25		
3.12 Darker When Wet	25		
4.1 Jensen et al. – Two-Layer	28		
4.2 Undulation Data	29		
4.3 Heightfield Representation	30		
4.4 Water Grid	32		
4.5 Splash Mechanisms	33		
4.6 Rain Container Examples	35		
4.7 Garg and Nayar – Raindrop Model	37		
4.8 Rain Inclination	38		
4.9 Wetness Parameter	39		
4.10 Heightmap Defined Puddles . . .	40		

Tables

3.1 Shape Coefficients	18
4.1 Rain Intensity	35
4.2 Distribution of Raindrops	36
4.3 Visual Interpretation of Rain Intensity	36
4.4 Rain Intensity Values	44
5.1 Performance	62



Chapter 1

Introduction

Simulating various weather conditions in outdoor scenes can significantly enhance the realism and diversity of the environment. Of these natural phenomena, rain is probably most frequently seen. However, rain is not defined only by raindrops, the surrounding area after it has stopped raining is also part of the occurrence. Therefore, a believable rain simulation has to consider numerous visual effects that involve complex physical mechanisms. A rainy environment consists of raindrops, puddles, splashes, ripples, fog, light glows, and even rainbows. Unfortunately, the number of little details that need to be simulated exceeds current computational capabilities, and realistically rendering these effects is challenging. This pushes real-time applications to limit the simulation and use approximations instead while still achieving visually convincing results.



1.1 Goals

The fundamental goal of this project is to construct the aforementioned rainy environment for a virtual traffic simulator. Furthermore, suitable rain phenomenon with different intensity levels should be simulated to increase the visual diversity of the traffic conditions. These rain effects will later be used in a project which generates synthetic datasets for machine learning algorithms. Hence, the visual simulation will be implemented using specific tools utilised by the mentioned project to explore the possibilities or identify obstructions along the process.



1.2 Thesis Structure

The thesis is structured as follows: chapter 2 consists of observations of rainy weather in the real world, a comparison of recognised rain phenomena in selected racing games, and finally, a short survey on existing methods for simulating rain. Chapter 3 explains different rain and wet effects, why and how they happen. Chapter 4 presents the visual simulation, the essence of this thesis, and the implementation of rain simulating techniques. Finally, the last chapter 5 shows accomplished results, the performance of practised methods and short qualitative user testing.

Chapter 2

Analysis

In the first section of this chapter, I shortly describe observations made in the real world during rainy weather. I focus mainly on an urban environment, a city during or after a moderate rain with lively traffic. The findings are then compared with the rain effects which occur in some racing games while taking into account the visual quality of the rain. Finally, I illustrate various techniques to simulate rain, many of them focusing on a particular subset of different rain phenomena.

2.1 Rain in Real World

When raining, long stretched reflections of bright light sources can be seen on wet surfaces (see Figure 2.1, left). It is not restricted to wet surfaces, but it is more visible when wet and even more noticeable during the night. The roughness of the surface gives the strength of the reflections. On rougher surfaces, the highlights are hazier and dimmer due to energy conservation.

Everything can be reflected, not only bright light sources. The appearance of these reflections is distorted accordingly by the distance and the viewing angle of the observer. The light angle and the index of refraction also determine the fuzziness of the reflection. Furthermore, reflections in puddles are often dynamically warped by falling raindrops resulting in a blurred streak of the dominant colour of the object [Tat06]. However, smooth wet surfaces can produce a perfect reflection no matter the distance of the light source (see Figure 2.1, right).

When enough water accumulates in a low depth place (holes, cracks or any base of an uneven surface), a puddle might be formed (see Figure 2.2, left). The reflection in the puddle follows Fresnel's law [BWB⁺99]. When looking at a puddle closer to a perpendicular angle, the reflection is low, and the original surface underneath the water layer can be seen [NKON90]. If the surface has a darker albedo and the sky is clear, the reflection is almost like a mirror (see Figure 2.2, right).

Generally, surfaces are diffusely darker when wet and have brighter specularities (see Figure 2.3) due to the smooth air-water interface and the presence of water beneath the surface [JLD99]. However, not all materials become darker. For instance, plastic or metal do not change their albedo colour.



Figure 2.1: The figures show streaks and reflections on a wet surface [Lag12]. The figure on the left shows streaks of reflected headlights of a car. The figure on the right displays more evident reflections, which can be seen on the smooth sidewalk, unlike the reflections on the right side, which are influenced by bumps of the block paving.



Figure 2.2: The left figure shows a small amount of water gathered in crevices and holes of a road. After a while, a puddle is created and reflections can be seen.



Figure 2.3: The main visual cue is that wet surfaces look darker, more specular and exhibit subtle changes in saturation and hue [Lag12].

During heavy rain, atmospheric scattering and attenuation (fog) appear. Light sources in this kind of rain produce a misty glow effect, and objects in the scene have a slight misty rain halo on top of them [Tat06].

Finally, the rainfall itself is perceived as streak particles, but in fact, the effect is caused by motion blurring (see Figure 2.4). These raindrops can cause ripples in areas with enough water depth. The amount of appearing ripples depend on the rain intensity and wind direction. Rain splashes are almost everywhere and are almost invisible in a slight drizzle and even tougher to see during the night.



Figure 2.4: The raindrops falling are not always perpendicular to the ground because of wind.

2.2 Rain Effects in Racing Games

Dynamic weather conditions play a huge role in racing games. The racing track's visuals are weather dependent in real-time, which provides more authenticity to the racing experience.

*Driveclub*¹ (2014) by Evolution Studios Ltd. is a PS4 video game and although it had mixed reviews about its gameplay, the rain visuals overall are quite impressive. The developers released a weather patch that simulates snow and rain particles accurately by the laws of forces and motion. These effects do not only change the visuals but also the handling of the car.

- **Water droplets.** The attention given to rain interaction with the car is remarkable. For example, the rainfall appears to be inclined at a certain angle when sitting in a moving car [Smi14]. The angle changes with the speed of the car – it is easy to notice because these rain particles are lit by the player car's headlamps. Water droplets on the windscreen vary according to the rain intensity and can stream to the sides during the car race (see Figure 2.5, left). Wiper blades also have a harder time dealing with the accumulated water. The technique used here gives the impression of three dimensional water droplets that produce streaks which react to velocity and direction [Lin14].
- **Wet roads.** The road surface is covered with puddles depending on the weather condition. Reflections are screen-space which can be a limitation as only objects visible in the scene are being reflected. This is, of course, not apparent when racing. Puddle sizes vary based on the rain intensity and are located mostly where the road is rutted with wheels (see Figure 2.5, right). Car wheels will also stir up the surface of a puddle and leave a rooster tail behind.
- **General wetness.** The vehicle gets wet accordingly with rain and each drop has individual dynamic lighting depending on the multi-layer shaded paint of the car's body [Smi14]. The game uses a physically-based rendering pipeline so the other elements within the game show realistic wetness as well [Lin14]. Foliage or barriers, most props receive a suitable adjustment in the rainy condition.
- **Gradual change.** While driving around the racecourse, using the randomised weather patterns, the road becomes increasingly reflective as the rain became heavier. Meanwhile, the drying effect depends on a variety of factors - the time of day, whether the sun is shining and even the relative elevation of the track [Fah14].

In the same year, Microsoft Studios published *Forza Horizon 2*² on Xbox (developed by Playground Games, 2014). Because of the open-world gameplay, the rendering budget was unquestionably lower than *Driveclub*'s.

- **Water droplets.** The rain particles receive light but seemingly only from light posts rather than the headlamps of a car, which reduces the realism of the effect,

¹<https://www.playstation.com/en-gb/games/driveclub-ps4/>

²<https://www.forzamotorsport.net/en-us/games/fh2>



Figure 2.5: (Left) A Driveclub screenshot that shows water particles interacting with the window and the wiper blade. The streaks on the window then flow in the proper direction [Lea14]. (Right) A Driveclub screenshot of a wet road. Note the long puddle areas along the road direction [Sab14].

especially at night [Lin14]. The raindrops on the windscreen are produced by transparent textures. These water drops are simplified and less dynamic (yet still visually pleasing), utilised with a water shader that generates somewhat static drops (see Figure 2.6). These streaks are not being altered by the speed or the direction of the car. The wipers do not give the impression of water being pushed away and function in just two modes – on or off.

- **Wet roads.** The surface when wet is covered randomly with patches of puddles, despite actual areas where cars drive mostly. Even though the randomness, it is highly unlikely to identify the difference while racing. Reflections in puddles seem to show less detail and are clipped near the camera [Lin14]. Car wheels spray a stream of water but no noticeable surface stirring traces are left behind in the puddles.
- **General wetness.** The environment around is not given enough attention as in Driveclub’s. Various props are covered with water drops and streaks but most foliage remains identical no matter the weather change. The water beads on the paintwork of a racing car appear flat, missing the volume dynamism of a raindrop.
- **Gradual change.** Dynamic weather allows dampness to build in the atmosphere and the layer of formed clouds can be seen in the sky. When the atmospheric intensity reaches a certain breaking point it starts to rain [Rei14]. Later on, the roads begin to dry in a particular way and over a time scale that one would expect.

Another game worth mentioning is *Project Cars 2*³ (2017) by Slightly Mad Studios. The most noticeable rain impact is the puddles on the road. Supposedly, puddles do not stay in the same place, but fill up and drain across the track realistically [Rod18]. This simulation, however, is handled quite poorly and even light rain leads to an unnatural amount of standing water on the racecourse. This, in turn, causes aquaplaning and hinders the player, making it almost impossible to get past the puddle area even at walking speed. Another intriguing rain detail is the thick stream of raindrops being pushed by powerful gusts during heavy rain (see Figure 2.7), creating yet another believable obstacle to visibility.

³<https://www.projectcarsgame.com/two/>



Figure 2.6: A Forza Horizon 2 screenshot of player's windscreen covered with transparent water drops [Mot16].



Figure 2.7: Strong gusts of wind push and scatter raindrops in short intervals, creating mist which can be seen above the racing car [Thr17].

Each video game has multiple visual cues which show fine rainy weather. The lists above do not cover all of them, it is just pointing out rain phenomena which might be appealing to traffic simulation. Rain in these games can certainly raise the adrenaline in racing competitions. However, too much rainwater may be obstructive and therefore can harm the gameplay. Racing in extreme weather conditions is possible only in games as they are usually cancelled in real life due to safety measurements. Asphalt becomes slippery causing tires to lose grip easier during cornering and cars in front eject a rooster tail of mist over the roadway which makes it tricky to see the next turn. On the other hand, this layer of difficulty is exactly what is needed in realistic traffic simulations.

2.3 Existing Methods of Rain Simulation

Rain simulation is often divided into two parts: the rainfall simulation itself and the proper visualisation of the environmental effects described in section 2.1. Subsection 2.3.1 shortly describes methods to simulate rainfall as such. Subsection 2.3.2 presents techniques, which also include other rain phenomena.

2.3.1 Rainfall

Numerous approaches to rendering rainfall have been studied and developed in the field of computer graphics. Most of them are either based on textured quads or particles [Hem16].

Raindrops as textured quads are made with few primitives which are positioned in front of the observer and rendered as a clear white material. Drop parallax is not possible with this approach unless various textures imitating different depths are used. This approximation of rain has a higher performance which is desirable in real-time applications such as video games or virtual reality. A few calculations are necessary, however, the amount of available rainfall textures is usually small. This limitation does not allow precise control over the rain intensity in real-time and repetitions can be noticed due to finite resolution.

Wang and Wade (2004) [WW04] mapped four artist-generated textures on a double cone centred around the camera to enhance the simplistic approach. The cone ensured that the rain would fall correctly if looked in or against the direction of the rain. They tilt the cone to adjust for camera movement, prolong the textures to mimic motion blur, and scroll the textures to simulate gravity. Figure 2.8 illustrates the double cone with the camera inside.

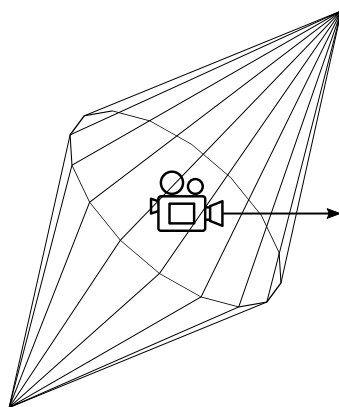


Figure 2.8: A double cone surrounding the camera [WW04].

The double cone system does not exhibit illumination changes due to lighting variations in the environment. A detailed photometric approach for modelling rain streak is described by Garg and Nayar (2006) [GN06]. Authors analyze how oscillations affect the visual appearance of rain streaks and precompute a large database of rain streak textures by raytracing individual images with different parameters (see Figure 2.9). The final appearance depends on many factors – drop oscillation, the drop shape, lighting and viewing directions. Rainfall is then rendered using the database to interpolate the streak

appearance. This method responds to dynamic lighting conditions and camera movement but due to the offline nature of the computations and large memory usage of the database, it is impractical for most interactive applications.

θ_{view}	110°						90°						70°					
θ_{light}	50°		90°		130°		50°		90°		130°		50°		90°		130°	
ϕ_{light}	130°	10°	70°	30°	10°	150°	30°	10°	110°	50°	170°	30°	170°	90°	110°	50°	130°	30°
Real Images of Rain Streaks																		
Rendered Rain Streaks																		

Figure 2.9: The top row shows actual images of streaks captured under many lighting and viewing directions. The bottom row shows rain streaks rendered using Garg’s and Nayar’s appearance model [GN06].

Tatarchuk (2006) [Tat06] renders multiple layers of textured full-screen quads in front of the camera to create the illusion of countless raindrops in the town environment demo (Figure 2.10). The rainfall is convincing though it is impossible to locate a single drop in the scene, therefore the rain interaction with the environment is nonexistent. Tatarchuk dealt with this obstacle by using collider objects that moved within the world independently from the rain and rendered a splash when hitting an object.

Particle systems simulate each raindrop as a unique entity. Every drop in the scene can be traced and the splash animation can be placed exactly where the raindrop hit an object. This requires complex mechanisms to manage only visible raindrops. Indeed, some of those raindrops may not be visible after projection, depending on their size and distance from the observer. Complexity also lies in simulating a large number of particles with a constant distribution over space.

Tariq (2007) [Tar07] presents a particle approach for animating rain streaks that works completely on the GPU. After the position of particles is updated (animated), each particle is assigned a quad and lighting properties are calculated to choose a suitable raindrop texture, which is then finally rendered. Tariq uses a simplified rain streak image database presented by Garg and Nayar (2006) [GN06] which has been discussed above. The final colour is produced by blending 4 textures of different raindrops under different viewpoint and lighting directions. An accurate analytical fog model is also added to create glows around light sources and change the reflectance of the surfaces. The final result can be seen in Figure 2.11.

Puig-Centelles et al. (2009) [PCRC09] focuses on the optimization of computing new particles repeatedly. The volume (defined by an ellipse and a rain container, see Figure 2.12) in which the rain is simulated is large enough to let the observer rotate

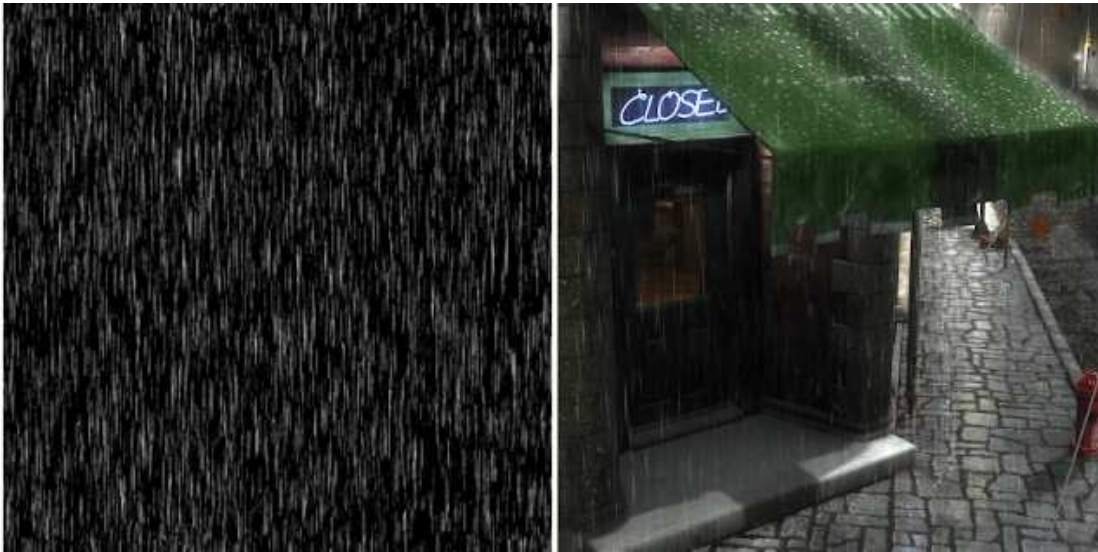


Figure 2.10: The rainfall texture applied for a composite rain effect in the scene (left) and the rendering of rainfall and raindrops with splashes (right) [Tat06].



Figure 2.11: Tariq's rain render where rain streaks are most visible near the lamp posts, along with their light glows [Tar07].

the camera without altering the position of the rain container. They also handle the transition between rainy and non-rainy areas at the perimeter of the ellipse. Additionally, the level of detail is applied: the density of the particles depends on the distance to the observer therefore fewer particles are being processed further from the camera. The rain particles will later be expanded to quads which are rendered as uniform transparent materials to imitate rain streaks.

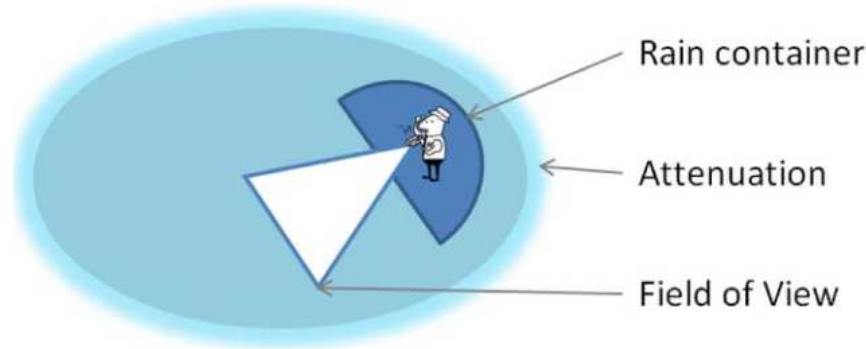


Figure 2.12: The raining area is simulated as an ellipse with an attenuation zone on its border. The zone is used for progressively decreasing the number of rendered raindrops. The rain container encloses the whole set of raindrops included in the particle system [PCRC09].

Another notable method which renders raindrops is presented by Rousseau et al. (2006) [RJG06]. Optical properties of a raindrop are being simulated together with the interaction of light. Their analysis concluded that a raindrop reflection is restricted to its silhouette and can be neglected without any penalty to the visual quality. To model refraction inside a raindrop in real-time, a texture mask is precomputed to determine the direction of the refracted viewing vector. The vector is then used to index another texture, which captures the background of the scene. The final colour is computed using the background scene texture, which is distorted in the raindrop accordingly by the viewing vector (Figure 2.13 shows the general idea). They also take into account the retinal persistence, where almost spherical raindrops appear as streaks, by changing the shape into a vertical line and render a few drops along this line, blending them. Finally, they also consider light interaction with the raindrops by modifying their colour. Without the increase in computational cost, they manage to get almost ray-traced looking raindrops, as long as objects being refracted are far behind the rain (in the background).

■ 2.3.2 Multiple Rain Phenomena

Some of the rendering techniques include a combination of a wide variety of rain phenomena to achieve more comprehensive simulations. Following methods either integrate these effects to correspond with the intensity of the simulated rainfall or simply add several rain phenomena to create a believable rainy scene.

In Changbo et al. (2008) [CWZ⁺08] various rain effects are being simulated. The raindrop's appearance is determined either by natural light (skylight) or a lamp (during the night). The skylight intensity depends only on the height and can be precomputed and stored in a look-up table. For the lamplight, raindrops are being treated as microfacets,

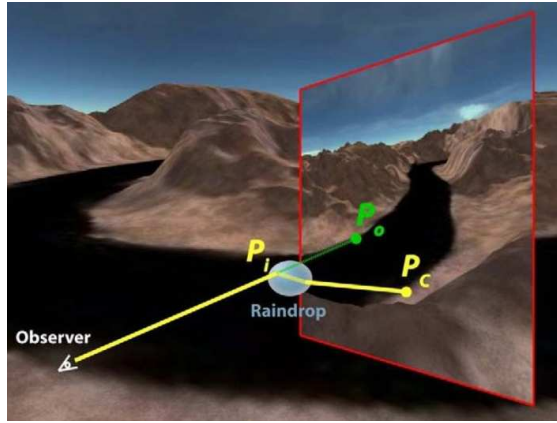


Figure 2.13: The red quad is the wide field of view render of the background, P_i is a pixel of a raindrop, P_o is a texel seen in an original viewing direction and P_c is a texel which gives the final colour after using the refraction perturbation of that specific raindrop [RJG06].

and the emergent light intensity from the lamp is precalculated (for different view directions) and stored in a spherical texture. These precomputations later allow them to determine the light intensity of a raindrop in real-time. The final rain streak visibility is not uniform along its track, they take into account the time lag of vision persistence (more detail in Figure 2.14). The thickness of rain streaks is randomized using three shapes of raindrops to achieve oscillation effect caused by the aerodynamic forces and the surface tension acting on the drop. Fog is formed by countless tiny raindrops in the air according to their spatial distribution. When the density of rain decreases, a rainbow is made. Multiple rendering passes are made to render the wet ground including reflection, refraction, environment mapping, and the mirror images of lamp and ripples in the puddle.

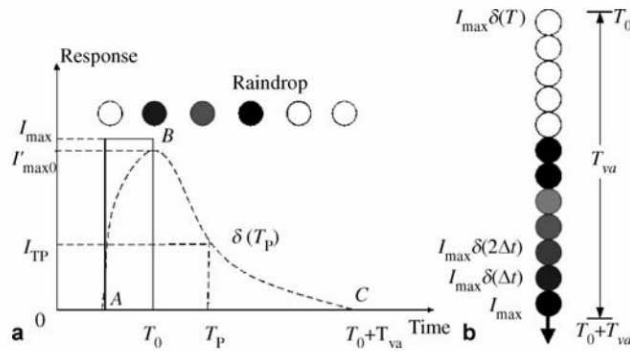


Figure 2.14: (a) The rectangle indicates the intensity of the incoming light signal while the dashed line shows the actual intensity perceived by eyes. The light stimulation to the eye begins at point A. From A to B is the process of energy accumulation, vision brightness reaches the maximum at B. After the light signal ends, it takes some time to lose its energy on the retina (to C). (b) A raindrop stimulates the retina of our eyes to form an image of a rain streak, which takes about 60–200 ms to fade away. The right figure shows the intensity calculation of raindrops along the rain streak track [CWZ⁺08].

Tatarchuk (2006) [Tat06], besides using the textured quads for making rainfall, also presented other sets of techniques that approximate the rain phenomena without trying to use physically correct simulations. For example, splashes are uncorrelated to the actual raindrop collisions and are made by using textured billboards of a milk drop instead (the result can be seen in Figure 2.15). Those billboards are modified in size and transparency to avoid repetition. Ripples in puddles are simulated with a tiled texture that changes bump-mapping of the object. Similarly to splashes, different scale and rotation factors are used to reduce repetitive patterns. The scene is somewhat limited in camera movement but the set of methods is extremely comprehensive. This hybrid rain system requires around 300 distinctive shaders in total to get a convincing rainy scene, including fog, light glows, halos, and lightning.

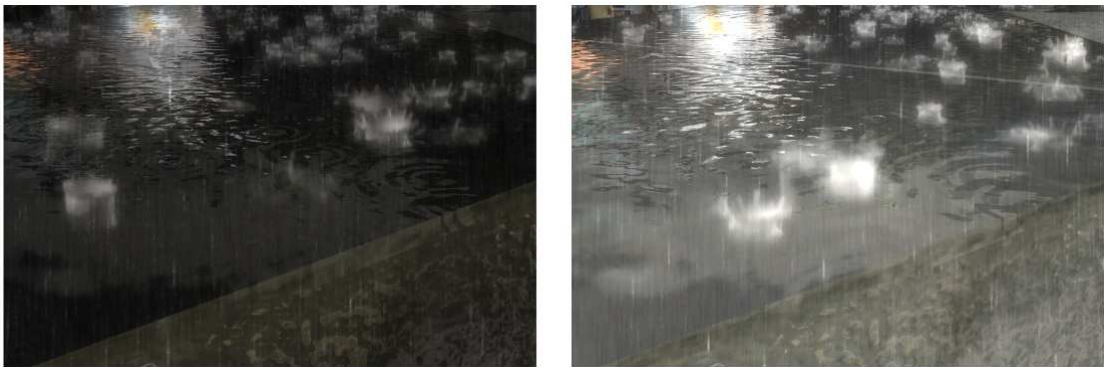


Figure 2.15: Regular raindrop splashes render (left) and raindrop splashes during a lightning flash (right) [Tat06].

Creus and Patow (2013) [CP13] proposed a rain rendering algorithm that creates particles in the scene using an artist-defined storm distribution (specified as a 2D rain intensity map). The dimension or shape of the rain area is not restricted. The only requirement is a well-defined depth map that can be used to compute collisions or cull the particles that are occluded. Their method is split into three stages: in preprocess, rain particles and texture atlas (using Garg’s and Nayar’s database [GN06]) are generated. In the next step, phenomena such as fog, halos, and light glow are added before the rendering of streaks and splashes. The particle simulation itself happens in the last stage. A *hit height* is defined for a particle as the maximum distance it can fall before colliding with an object in the scene. A particle is rendered as a raindrop if it is above the hit height or as a splash during the short period when it goes below (a fixed animation of 21 frames). After the animation, the particle is not rendered and is reused after it spawns at the top again. Figure 2.16 presents a scheme for this behaviour. The final result of heavy rain is well optimized and requires roughly 780 000 particles for a good perception. It cannot, however, process volumetric data nor treat scenes with semi-transparent geometry because the rain is blended once the scene has been fully rendered.

A multiscale rain model was introduced by Weber et al. (2015) [WJGG15] which simulates the progressive loss of visibility induced by the atmospheric phenomenon. Their research shows that the attenuation coefficient must be correlated with the rain intensity and the distribution of raindrops. Using an uncorrelated fog model leads to a non-physically realistic rendering.

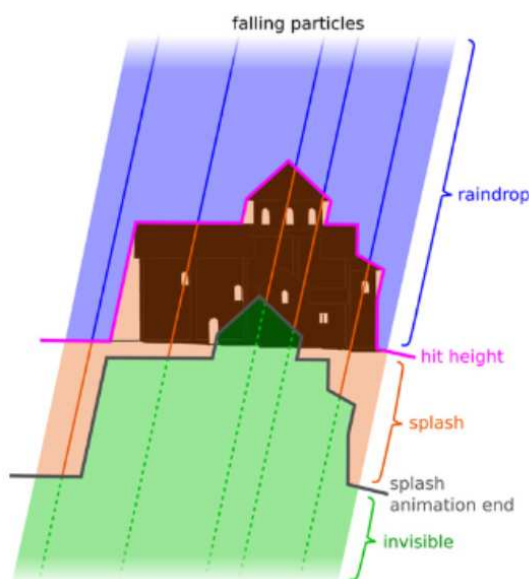


Figure 2.16: States of a particle: a raindrop, a splash after it reaches the hit height, and an invisible state (after which it will be spawned at the top, repeating the process). The length of the orange line is the same for every particle because of the splash animation being identical [CP13].

Their physically-based rain simulation space is decomposed into two areas. The first region, Visible Streaks Region (VSR), deals with the distribution of pixel-sized rain streaks meaning that each raindrop is individually visible. Beyond this region is the Scattering Rain Phenomenon (SRP) which simulates further raindrops that would require a subpixel precision. This phenomenon is continuous in the whole scene from the camera to the furthest visible object in the scene. The rain streak appearance in VSR is based on a frequency model, considering that the colour of a rainfall texture is a mix of different overlapping projected streaks. This controllable rainfall texture is defined by convolution of rain streaks kernels distributed along a simple regular grid. To save memory space, they express rain streaks kernels as a sum of cosine waves, extracted from real patterns in a real rain video using Fourier theory. Figure 2.17 shows a reproduced streak signal. In SRP region the raindrops interact with light which results in fog appearance. They suggest using a simple single scattering model to simulate the attenuation of visibility in a rainy atmosphere to avoid too many computations. They do so according to the same distribution of raindrops in VSR.

A visual result of heavy rain can be seen in Figure 2.18. Due to the similar textured quad behaviour of the rainfall texture, the simulation is not coherent with lateral movements of the camera. Furthermore, streaks cannot be generated and animated in the same way if the observer looks up or down.

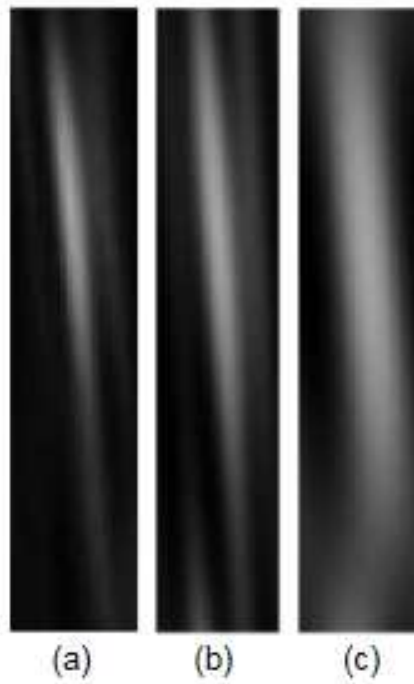


Figure 2.17: The theory states that all frequencies should be used to exactly reproduce the input signal (a). Nevertheless Weber et al. show that using 6 frequencies (b) is sufficient to reproduce a visually similar rain streak. Any lower number of frequencies deteriorates the original signal, as shown in (c) – using only 3 frequencies [WJGG15].



Figure 2.18: Heavy rain and attenuation rendering of Weber et al. [WJGG15].

Chapter 3

Rain Phenomena

This chapter is dedicated to why or how some of the rain phenomena witnessed in real life occur. The sections below describe the main characteristics of several rain effects. The last section 3.4 is dedicated to wet surfaces, more specifically the wet road, as rendering those under various conditions can be quite useful for the study of safe driving in traffic simulations.

3.1 Raindrops

Rain happens when damp air rises and later cools down. The precipitation in the sky will eventually condense and fall to the ground as rain streaks. The type of rain (either a drizzle or a heavy shower) depends on the mechanism which causes the air to rise, forming its distinct cloud and other properties. This section primarily describes the properties of raindrops in different rainfall types and intensities.

3.1.1 Size of Raindrop

Every kind of rainfall will have raindrops of different size. The size of a raindrop cannot exceed a specific threshold because they tend to split due to their motion through the air. Generally, small raindrops are more common, but as the intensity of rain increases, the number of large drops also slightly grows. Occasionally, a few large drops are accompanied by a thick spray of small droplets when there is strong wind slightly above the ground [Kel45].

A typical empirical distribution used for raindrop size is the Marshall-Palmer distribution (1948) [MP48]. The general relation of the drop size distribution can be written as (eq. 3.1):

$$N(D, R) = N_0 e^{-\Lambda D} \quad (3.1)$$

with $N_0 = 8000 \text{ m}^{-3} \text{ mm}^{-1}$ for any intensity of rainfall, $\Lambda = 4.1R^{-0.21} \text{ mm}^{-1}$, R being rate of rainfall given in mm hr^{-1} and D as the radius of the drop in millimetres. $N(D)$ expresses the number of raindrops per unit volume of space (raindrop density) that contains sizes within the interval $(D, D + \delta D)$, for $\delta > 0$.

3.1.2 Shape of a Raindrop

A raindrop will assume an almost spherical shape due to surface tension. As the drop falls through the air, aerodynamic pressure develops at the bottom (see Figure 3.1 a). The weight determines how flattened the shape of the raindrop will be. In conclusion, small raindrops up to 0.5 mm retain an almost spherical form, while the bigger ones become ellipsoidal [Spi48].

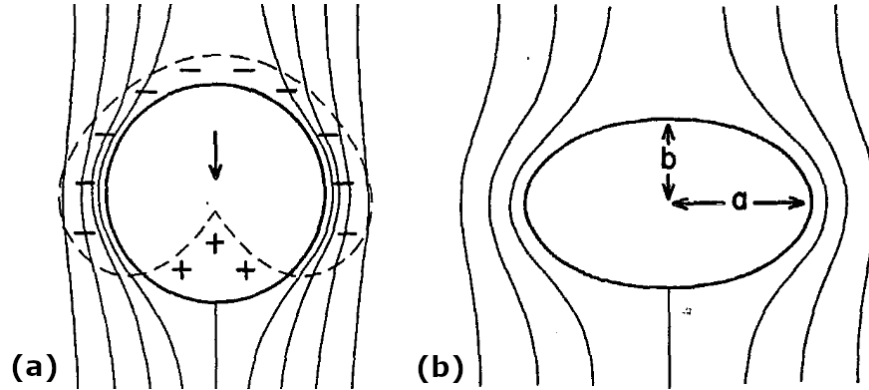


Figure 3.1: (a) Pressure distribution around a spherical shape, positive acts towards the centre, negative outwards. (b) A simplified schematic deformation of a falling water drop described by Spilhaus [Spi48].

Beard and Chuang (1987) [BC87] presented a composite model of drop shape based on the sum of weighted cosines. They distorted a regular sphere by using equation 3.2:

$$r(\theta) = a \left[1 + \sum_{n=0}^{10} C_n \cos(n\theta) \right] \quad (3.2)$$

where a is the radius of the undistorted sphere, θ is the polar elevation point (with $\theta = 0^\circ$ pointing down vertically, see Figure 3.2) and C_n as shape coefficient (from a cosine series fit to the model results, see examples in Table 3.1).

d (mm)	Shape coefficients [$C_n \times 10^4$] for										
	n = 0	1	2	3	4	5	6	7	8	9	10
2.0	-131	-120	-376	-96	-4	15	5	0	-2	0	1
2.5	-201	-172	-567	-137	3	29	8	-2	-4	0	1
3.0	-282	-230	-779	-175	21	46	11	-6	-7	0	3
3.5	-369	-285	-998	-207	48	68	13	-13	-10	0	5
4.0	-458	-335	-1211	-227	83	89	12	-21	-13	1	8
4.5	-549	-377	-1421	-240	126	110	9	-31	-16	4	11
5.0	-644	-416	-1629	-246	176	131	2	-44	-18	9	14
5.5	-742	-454	-1837	-244	234	150	-7	-58	-19	15	19
6.0	-840	-480	-2034	-237	297	166	-21	-72	-19	24	23

Table 3.1: Shape coefficients for cosine distortion fit (eq. 3.2) for drop radii between 2.0 and 6.0 mm taken from Beard and Chuang [BC87].

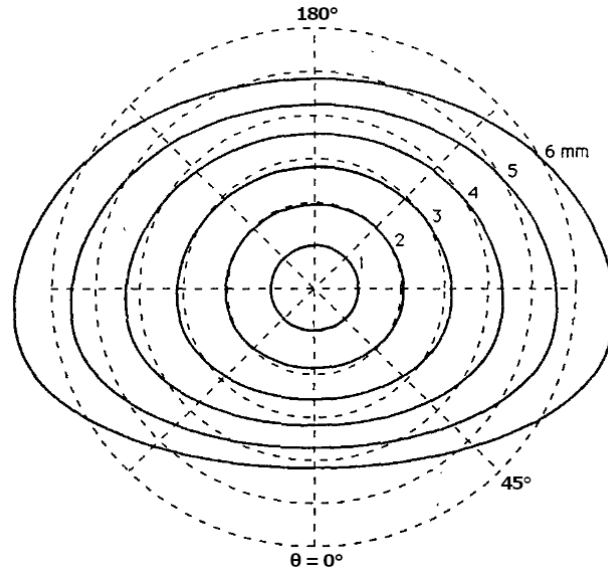


Figure 3.2: Drop shapes for diameters $d = 1-6$ mm. Dashed circles divided into 45-degree sectors show the original sphere for comparison [BC87].

The human eye, however, does not see the spherical or ellipsoidal drop. The retina receives images in continuous time and hence observes an elongated rain streak instead (as already mentioned in subsection 2.3.2).

■ 3.1.3 Velocity of a Raindrop

It is generally accepted that the falling speed of a raindrop is associated with its radius. The speed decreases steadily as gravity and friction forces compensate. Near the ground level, the velocity of a particular drop becomes constant, and the drop reaches the terminal velocity. Larger ellipsoidal drops will naturally have a higher terminal velocity than smaller spherically shaped drops [RJG06].

Gunn and Kinzer (1949) [GK49] did an empirical study on the terminal velocities of falling raindrops for different drop sizes. The terminal velocity for a raindrop of radius r can be therefore described as a function of its size (see the measurements in Figure 3.3). The general formula for the terminal velocity of a fluid is:

$$v = \frac{2}{9} \frac{r^2(\rho - \sigma)g}{\eta} \quad (3.3)$$

given r is the radius of a raindrop (m), ρ is the density of water, σ is the density of air, g is the acceleration due to gravity, and η is the coefficient of viscosity of air. Gunn and Kinzer's measurements under laboratory conditions and fits to their data are still considered the standard against which measurements using more modern optical instruments in natural rain are compared.

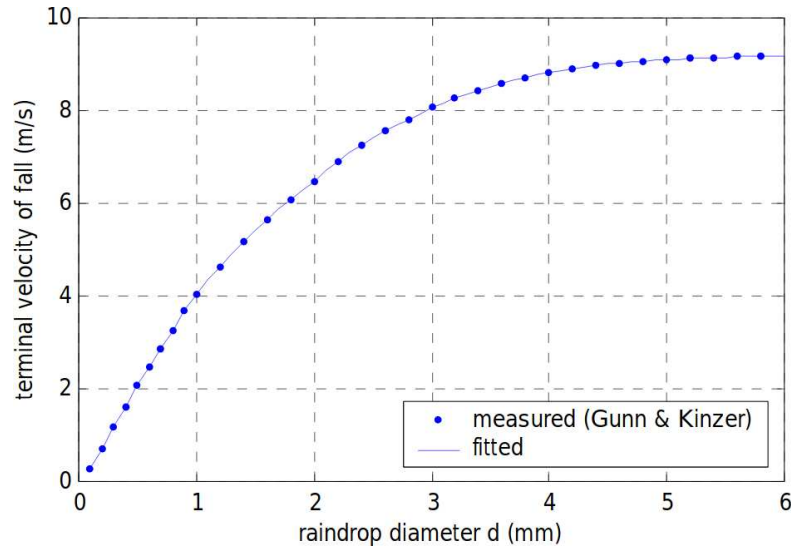


Figure 3.3: A raindrop’s terminal velocity is reached when the force of gravity and the air resistance (drag) becomes equal. Here are measurements by Gunn and Kinzer and a fitted curve. Image courtesy of Blocken and Carmeliet [BC02].

3.1.4 Appearance of a Raindrop

Pure raindrops are semi-transparent, but with strong lighting conditions, they reflect and refract the surroundings. While falling, the air also induces oscillations upon the drop, further distorting the shape and optical effects perceived [CP13].

Garg and Nayar (2007) [GN07] state that a drop’s appearance is a complex mapping of environmental radiance. Figure 3.4 shows the geometry of refraction through and reflection from the drop. The radiance $L(\hat{n})$ of point B is given by the sum of the radiances of refracted ray L_r , of specularly reflected ray L_s and of internally reflected rays L_p . That means $L(\hat{n}) = L_r(\hat{n}) + L_s(\hat{n}) + L_p(\hat{n})$, where \hat{n} is the surface normal at point B . The viewing direction of the camera \hat{v} is omitted since it can be parameterized in terms of \hat{n} . The equation can be rewritten considering that the radiances depend on the environmental radiance L_e , as follows:

$$L(\hat{n}) = RL_e(\hat{r}) + SL_e(\hat{s}) + PL_e(\hat{p}) \quad (3.4)$$

where R , S and P are radiance transfer functions which are the fractions of incident environmental radiance that reaches the camera after reflection, refraction and internal reflection.

Garg and Nayar derived exact expressions for these transfer functions and the geometric mapping from ray directions \hat{r} , \hat{s} and \hat{p} to the normal \hat{n} . Afterwards, they found out that refraction has the most significant contribution to the drop’s radiance – in fact, a raindrop transmits 94% of the incident radiance towards the camera (see Figure 3.5). The radiances due to reflection and internal reflection are notable at the periphery of the drop (see the plot in Figure 3.6). The field of view of a raindrop is approximately 165° , which is similar to a fish-eye lens (see Figure 3.7). Moreover, the light that is refracted is attenuated by 6%. Specular and internal reflection further adds to the drop’s brightness; therefore, a drop tends to be brighter than the portion of the scene it occludes.

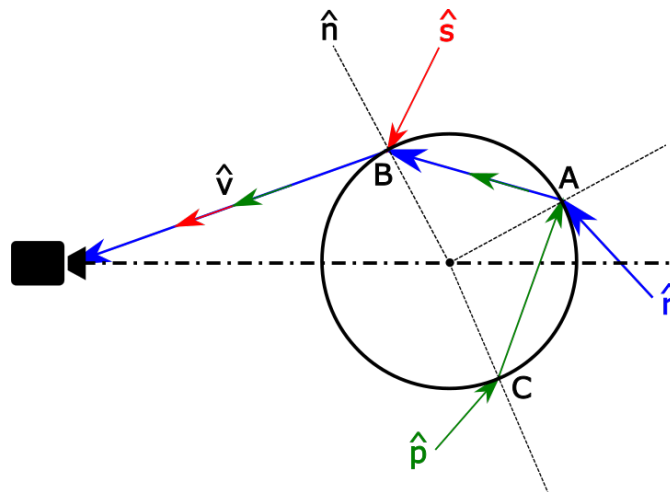


Figure 3.4: The figure shows how rays interact with a raindrop (here as a schematic sphere). Light rays from directions \hat{r} , \hat{s} and \hat{p} reach the camera via refraction, specular reflection and internal reflection of the drop, respectively [GN07].

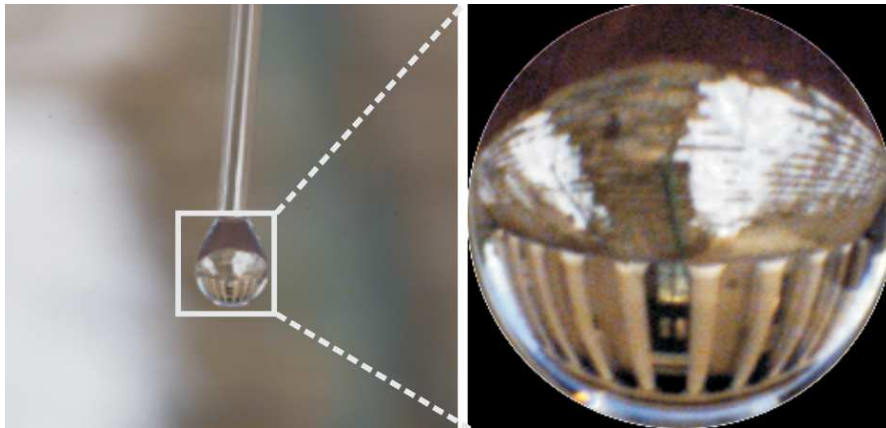


Figure 3.5: Image of a stationary raindrop that geometrically maps the environment [GN07].

3.2 Splashes and Ripples

Rain includes multiple visual effects (as observed previously), which can be rather challenging to replicate in a complex synthetic scene. Rain phenomena interact with scene objects, be it cars splashing water on the road or rainwater falling through the thick foliage of trees. Raindrops hit the ground repeatedly, distorting the flat water surface of a puddle. Circular waves (ripples) appear on the water layer on top of the world geometry accordingly. There is no ripple effect if the area has only a thin layer of water.

When a drop hits the surface, the impact deforms its shape, and some of the water ejects radially outwards, usually in the form of small droplets. This effect is called a splash, and it can occur in two possible ways (as shown in Figure 3.8): *corona splash*, where a thin crown-shaped water wall extends upward before breaking into smaller drops, and a *prompt splash*, where droplets are emitted directly from the base without the crown

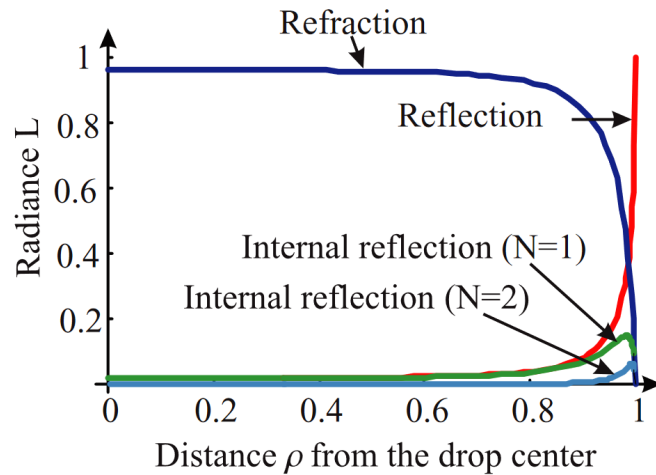


Figure 3.6: The plot shows raindrop radiance due to refraction, reflection and internal reflection as a function of distance ρ (here $L_e = 1$). Specular and internal reflection radiances are relevant near the edge [GN07].

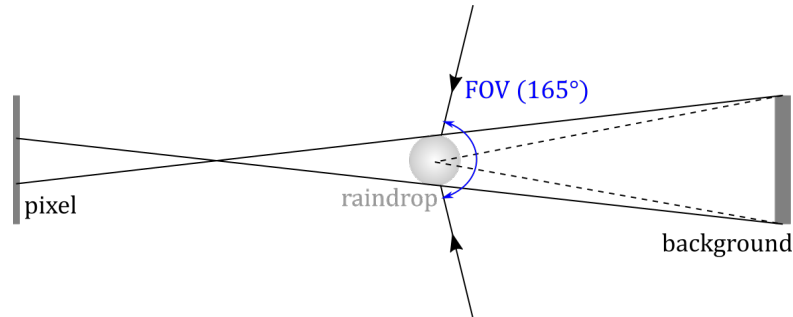


Figure 3.7: The solid angle of the background is minimal to the entire field of view of the raindrop. Therefore, despite the raindrop being transparent, its colour does not depend on the background [GN07].

formation [GKN07]. Typically, corona splashes are noticeable when there is a water layer present. The mass of water absorbs the impact creating a rounded crown shape.

The dynamics of a splash depends on many factors – the properties of a surface (roughness, rigidity, etc.), the properties of a falling drop (size, velocity, etc.) and the angle of impact [GKN07]. The splashing process is inherently random, and each splash may result in a unique set of droplets. Fortunately, differences in droplet sizes are hard to perceive from a drop splash. The height and radius of the crown also vary between surface materials. Nevertheless, it appears just for a short duration before collapsing back onto the surface, so it is barely distinguishable.

■ 3.3 Fog and Glows

Many small particles are floating in the air (known as aerosol), be it foliage exudation, combustion products, volcano ashes or sea salt. When rain occurs, the added moisture in the air raises the humidity over time, and these small particles act as nuclei for tiny water

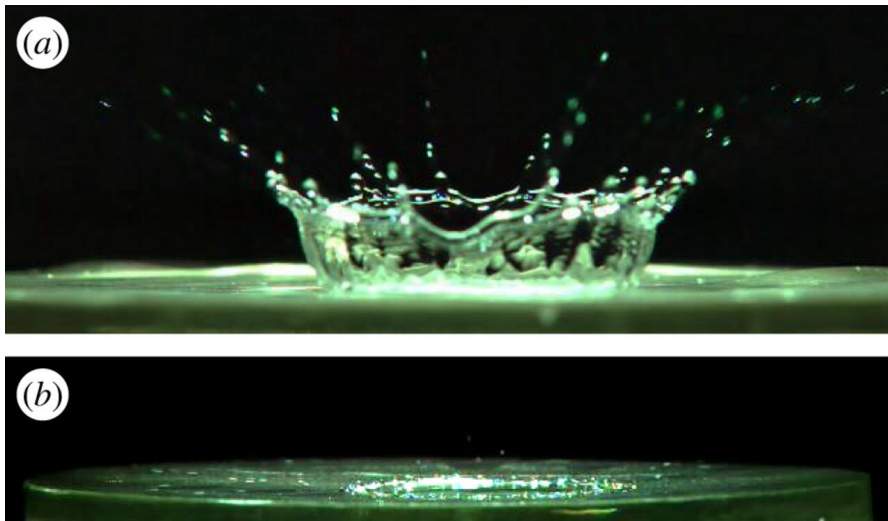


Figure 3.8: Impact of a water drop on an epoxy resin surface, creating a corona splash on the wet surface (a) and a prompt splash on the dry surface (b) [KG16].

droplets. The droplets contribute to the overall haze-like effect, producing a distinctive grey hue. As the humidity further increases, a haze may turn into a fog, which is thicker and further reduces the visibility on the ground level. The transition is gradual, and the intermediate state is called mist [NN99].

The light properties, such as intensity or colour, are unavoidably affected by the before-mentioned particles in the atmosphere. The most pertinent interaction between light and atmosphere is called atmospheric scattering and leads to complex visual effects. How a particle scatters incident light varies dramatically with its size (as seen in Figure 3.9). The particle re-radiates incident energy and therefore behaves like a point source of light. Of course, multiple scattering occurs, and a particle could be exposed to both incident and the scattered light [NN99].

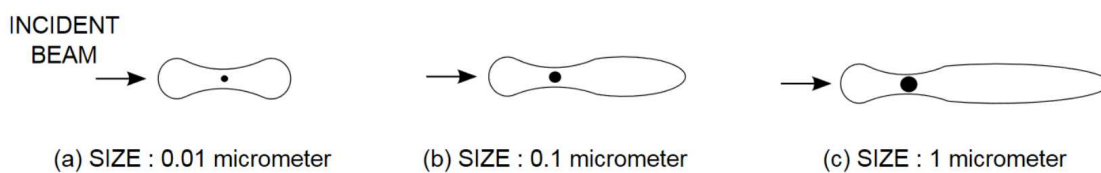


Figure 3.9: A particle in the path of an incident light beam – the exact scattering is related to the ration of the particle size to the wavelength of the light [WJGG15].

The first mechanism due to scattering absorbs part of a light beam as it travels through the atmosphere and is sure to affect visibility, occluding far away objects and lowering the details of the objects in the mid-range to the viewer [CP13]. Another aspect, called airlight, causes the atmosphere to behave like a source of light. The scattering influences environmental lights (including the sun and skylight) in the scene and increases its radiance. This event is mainly perceived as an elongated glow around the light source [SRNN05]. A light glowing example can be observed in Figure 3.10.



Figure 3.10: Light disperses in every direction in a misty night, making the source of the point lights smoother and glow further from its centre [eui19].

3.4 Wet Surfaces

The observations made in section 2.1 generally state that surfaces become darker when wet. Nevertheless, the influence of water on various objects is more complex, and it depends on the properties of the used material. In the following part, wet surfaces mostly refer to a rough or porous material with a higher percentage of air voids within the structure, such as concrete or asphalt.

Two optical properties influence the appearance of materials when wet: water covering the material and a concentration of water beneath the surface [JLD99]. The presence of water on the surface causes the surface to have brighter specularity and make it diffusely darker. The leading cause for this darkening is the internal reflection at the boundary of air and water. Some light reflected from the surface will be kept inside and reflect again due to the water-air interface (see Figure 3.11). Total internal reflection happens when a ray transits from a thicker to less dense medium at a wider angle called the critical angle. The smallest angle of incidence that yields total reflection for the water-air interface is $\Theta_c = \arcsin(\frac{n_2}{n_1})$, which for $n_2 \approx 1$ as refractive index of air and $n_1 \approx 1.33$ for water is approximately 48.75° .

Concerning water that flows into the rough material's pores, the region earlier filled with air, the scattering properties of the material changes. The difference in index of refraction causes the ray of light to be less refracted and reduces the average scattering angle. Hence, the scattered light diverges less from the previous ray, making it more directional in the forward direction, penetrating deeper into the material (see Figure 3.12). The residual amount of light that leaves the material causes reduction in the reflectivity of the material [JLD99].

A different situation is when water is present on the surface of a material, gathering in low areas creating puddles (water film). The presence and placement of the puddles are

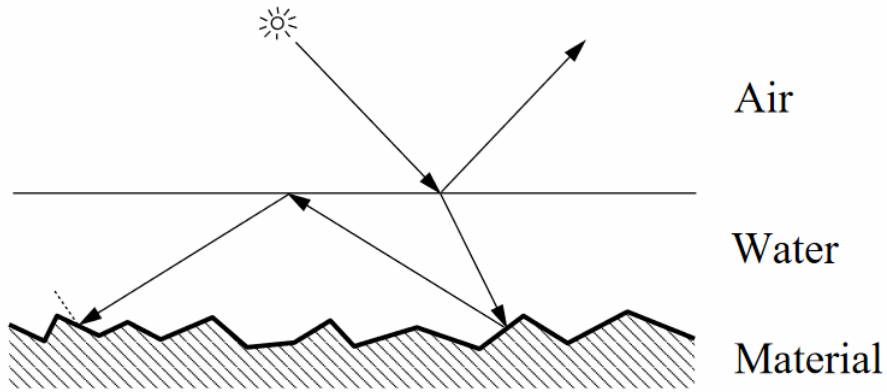


Figure 3.11: A layer of water above the surface reflects less light due to the internal reflection at the water-air interface [JLD99].

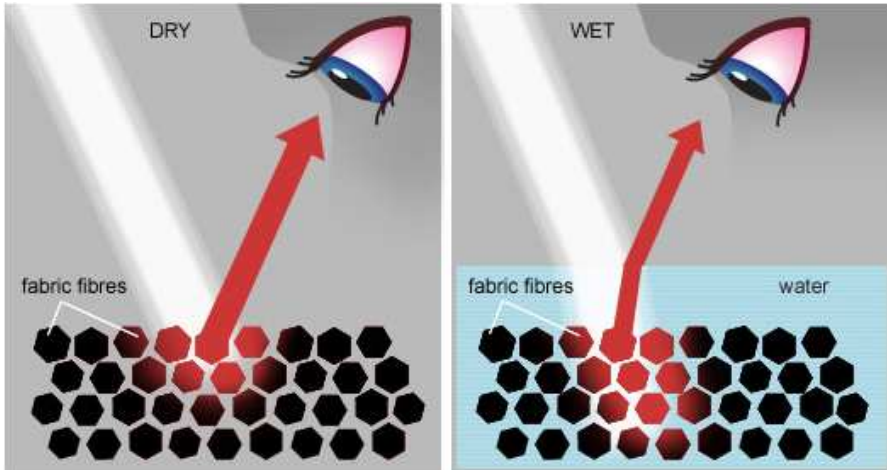


Figure 3.12: On the left, light travelling between air and another medium (here a porous fabric) scatters in all directions. When wet, the scattering occurs in a more forward direction, and more light is absorbed, thus reducing the amount of light that goes back to the eyes [Hob10].

affected by the local precipitation (rainfall intensity and duration) and surface properties, such as geometry, texture, material and permeability (rainwater penetrating through the porous surface) [LL19]. There are many equations and models dedicated to the depth of puddles, both empirical and analytical, which attempt to predict hydrodynamics. Detailed impacts of intervening factors on the water film depth are discussed by Luo and Li [LL19] using their analytical model based on the dynamic simulation.

Chapter 4

Visual Simulation of Rain

This chapter considers several approaches to achieve a good-looking simulation of a wet environment utilising various rain phenomena described in chapter 3.

4.1 Impact of Water on Roads

The rainwater remarkably alters the appearance of an asphalt road, whether it is the darkening of the albedo or the presence of puddles. The following subsections discuss the approach to create these rain effects on the roads.

4.1.1 Wet Roads

The leading cause for the darkening is the internal reflection at the boundary of air and water, as mentioned in previous section 3.4. To correctly simulate the thin water layer behaviour, we would need a layered BRDF – layers for the water and the original road material. Jensen et al. [JLD99] introduced a two-layer surface reflection model that considers the interaction of light with both the air-liquid interface and the liquid-material interface (see Figure 4.1).

Weidlich and Wilkie [WW07] rely on thin layers to assume that all incident and refracted rays meet at a particular point at every layer. The model is evaluated recursively and can be described by an equation (here for two layers, but it can be generalized to an arbitrary number of layers):

$$f_r = f_{r1}(\vec{\omega}_i, \vec{\omega}_o) + T_{12} \cdot f_{r2}(\vec{\omega}_i', \vec{\omega}_o') \cdot a \cdot t \quad (4.1)$$

where T_{12} is Fresnel's transmission coefficient and defines the amount of light transmitted from layer 1 to layer 2 and can be computed as [JLD99]:

$$T_{12} = \left(\frac{n_1}{n_2}\right)^2 (1 - R_{12}(\vec{\omega}_i, n_1, n_2)) \quad (4.2)$$

where R_{12} is the amount of reflected light at the layers 1-2. The coefficient a is the absorption term according to Bouguer-Lambert-Beer law:

$$a = e^{\alpha d \cdot \left(\frac{1}{\vec{\omega}_i'} + \frac{1}{\vec{\omega}_o'}\right)} \quad (4.3)$$

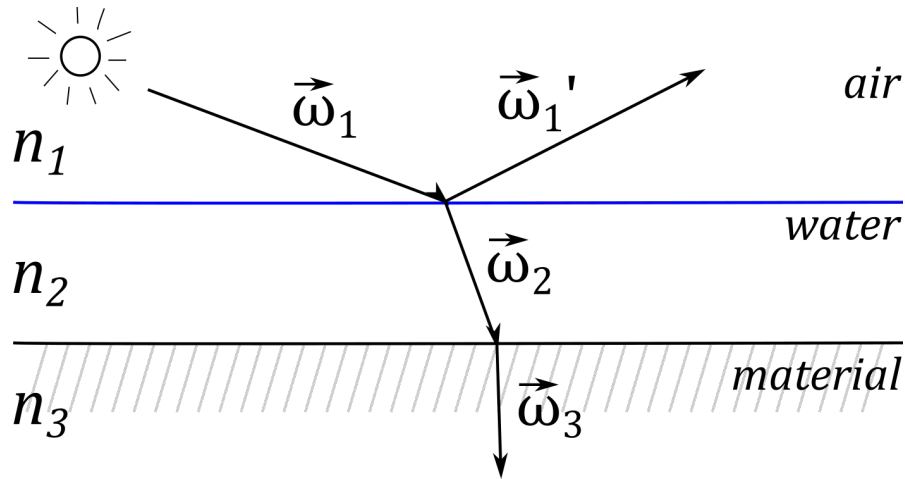


Figure 4.1: Jensen et al. [JLD99] use a two-layer reflection model to simulate a thin liquid film on top of a material. The amount of light transmitted through each layer can be computed using Fresnel’s equation.

α is a material-specific wavelength-dependent absorption coefficient, and d is the thickness of the layer. The term t is the total reflection term, which handles the energy lost during the possible total internal reflection in the layer:

$$t = (1 - G) + T_{21} \cdot G \quad (4.4)$$

where G is the Torrance-Sparrow geometry attenuation term and accounts for the energy which does not leave the medium immediately, for more details, please refer to the paper by Weidlich and Wilkie [WW07].

These two-layer methods could be fairly costly in real-time, especially if a dynamic transition between the road’s dry and wet state is needed. It is enough for a driving simulator to use a simple set of BRDF parameters capturing both dry and wet surfaces to imitate these effects visually.

As noted in Lekner and Dorf [LD88], the darkening effect by wetness is more significant if the albedo is low. They estimated the absorption effect on the reduction of the relative refractive index. When the absorption is strong, a larger fraction of the light is consumed on the first contact with the surface. The connection between wet and dry albedo described so far is still based mainly on the index of refraction of the water and the surface material. It would be helpful to have the scaling factors for diffuse and specular part separated, to shift from dry to slightly wet and to wet roads. Many other approaches consider the surface’s porosity and roughness, or even some sets of real-world measurements are available ([SSR⁺07], [ZV07]).

Nakamae et al. (1990) [NKON90] adjusted the parameters of a BRDF model by Cook-Torrance. The factor by which they attenuate the dry diffuse parameter into a darker wet material is between 0.1 to 0.3. The specular component is assumed to be 5 to 10 times increased instead, simulating the thin layer of water. These parameters were empirically calibrated without considering the physical properties of the surface. Nevertheless, it satisfies the scenes’ requirements within the driving simulation, and therefore the wet effects can be approximated by simply tweaking the dry BRDF parameters.

4.1.2 Puddles

Nakamae et al. [NKON90] further simulated water's appearance on the road, describing a lighting model explicitly aimed at driving simulators. They classified a road surface into four regions to render in rainy conditions:

1. A dry region,
2. a wet region, where the surface is wet, but no water gathers,
3. a drenched region, where water remains but no puddles are formed,
4. and a puddle region.

Procedural bump-mapping defines the properties of the road surface so there can be 'dry regions' in the higher area and 'puddle regions' in the lower area. Both 'wet regions' and 'drenched regions' exist between these two areas. Each group consists of several height data and a threshold of height for the classification (see Figure 4.2). These data allow the creation of puddles with different depth in each region, taking into account the locality and scale of the road surface's undulations.

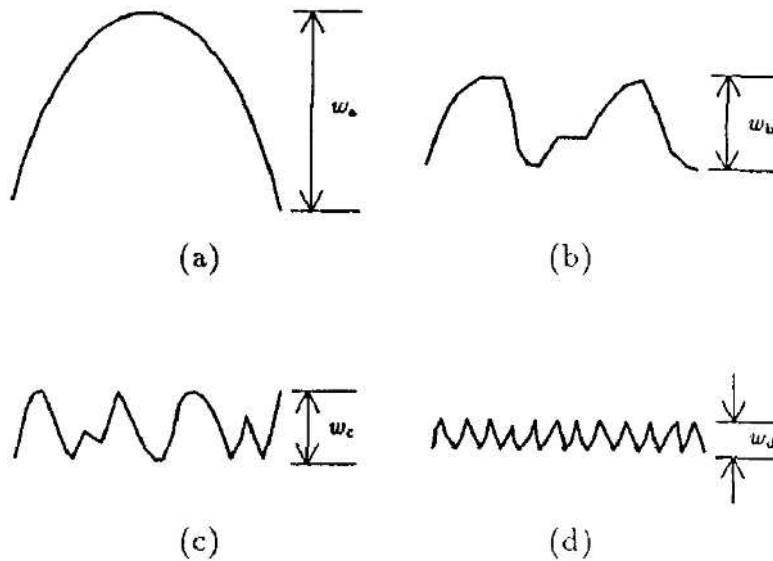


Figure 4.2: Several undulation data classify road conditions into four types taken from Nakamae et al. [NKON90]. Different types of undulation are recursively generated at different levels and then composed together, providing local control of the surface's height.

The paper also proposes reflection models for each of the mentioned regions. They likewise constructed a two-layer reflection model to simulate the puddle region. However, a few conditions must be considered to simplify the calculation, e.g. the puddle surface is entirely flat, and puddles are usually very shallow [NKON90]. The two-layer model can, for example, reproduce the presence of dirt particles in puddles. Nevertheless, the murky water can also be achieved by a single layer of material representing water of darker colour. Therefore, the two-layer reflection model is not used in this project.

4.2 Dynamic Simulation of Puddles

When a car drives through a puddle, the water stirs up, and ripples spread across the otherwise calm surface. If the puddle holds enough water and the car drives at a certain speed, the car wheels break the waves, which results in splashing and spraying. These effects can be split into two separate phenomena to reproduce – ripple propagation on the water surface and the splash and spray formation when the wheel interacts with the puddle area.

4.2.1 Water Surface and Ripples

A height field can approximate the water surface – each cell in a grid represents the water height above an underlying terrain. The surface is then deformed in a vertical direction, creating an illusion of a passing wave. The advantage of this technique is that a shader can calculate the wave appearance, which is well supported on GPU, and therefore it is considerably fast. The drawback might be that there is no information about the water mass (see Figure 4.3). This is not a significant disadvantage to the shallow puddles occurring on the roadways.

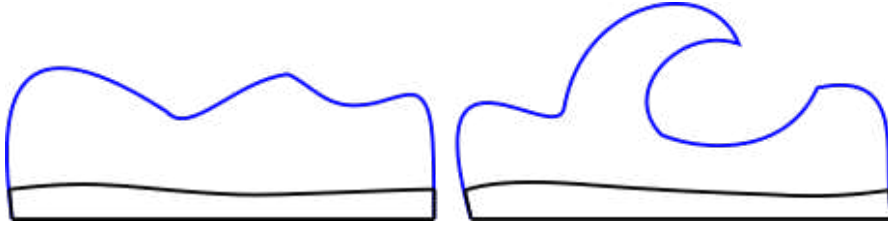


Figure 4.3: The height grid approach with only a vertical velocity field cannot interpret effects such as overturning waves or flows correctly (right-hand figure).

The ambient waves on the water can be described in terms of the distribution of wavelengths and amplitudes. Fast Fourier Transform (FFT) spectral approach by Tessendorf [Tes01] simulates such patch of waves in the ocean. His representation expresses the height of a wave $h(\mathbf{x}, t)$ at the horizontal position $\mathbf{x} = (x, z)$ as:

$$h(\mathbf{x}, t) = \sum_{\mathbf{k}} \tilde{h}(\mathbf{k}, t) e^{i\mathbf{k} \cdot \mathbf{x}} \quad (4.5)$$

where $\tilde{h}(\mathbf{k}, t)$ is the height amplitude function, which determines the structure of the surface, and $\mathbf{k} = (k_x, k_z)$ is a two-dimensional vector with each component storing a wave frequency in a specific direction.

The transformed spectral data (acquired using the inverse FFT) represent the uniformly spaced signals, which can be tiled over a larger region. The periodicity of the generated waves become apparent when viewed from afar while using a small patch size. Some LOD methods can be deployed to readjust the patch size, or a noise function might be used to cover the visible recurrent tiling. Another disadvantage is that it is challenging to add additional interaction across the water surface, such as floating objects. The shape of waves depends on the underlying spectra, making the heightfield's local modification hardly attainable [Mik14].

A wave equation can express the forces on the water formed by the interaction of objects. The equation is based on the propagation principle of mechanical waves. Concerning the surface simulation, a 2D partial differential equation to describe the waves in the vertical position z is applied:

$$\frac{\partial^2 z}{\partial t^2} = s^2 \left(\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} \right) \quad (4.6)$$

where s is the velocity of the wave spreading across the surface and t is a variable of time. The general solution for a square $N \times N$ section of water with homogeneous boundary and zero initial conditions is [Gom00]:

$$z(x, y, t) = \frac{2}{N} \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} A_{mn} \sin\left(\frac{m\pi x}{N}\right) \sin\left(\frac{n\pi y}{N}\right) \cos(c\omega t) \quad (4.7)$$

where

$$\omega = \frac{\pi}{N} \sqrt{(mx)^2 + (ny)^2} \quad (4.8)$$

and coefficients A_{mn} are found by evaluating:

$$A_{mn} = \frac{2}{N} \int_0^N \int_0^N f(x, y) \sin\left(\frac{m\pi x}{N}\right) \sin\left(\frac{n\pi y}{N}\right) dx dy \quad (4.9)$$

where $f(x, y)$ is the initial shape of the water surface. The solution can be approximated by evaluating only the significant terms – mainly the trigonometric functions; however, that would still be inefficient. Hence a numerical solution should be utilised.

The discretization with finite-difference methods can be performed using a 2D map $z_{i,j}$, with $i, j \in [0, N - 1]$, and N describing the width of our squared grid (see Figure 4.4). With the central difference in space and time, the equation is approximated to:

$$\begin{aligned} z_{i,j}^{t+1} = & \frac{s^2 \Delta t^2}{h^2} (z_{i+1,j}^t + z_{i-1,j}^t + z_{i,j+1}^t + z_{i,j-1}^t) \\ & + \left(2 - \frac{4s^2 \Delta t^2}{h^2}\right) z_{i,j}^t - z_{i,j}^{t-1} \end{aligned} \quad (4.10)$$

with h being the size of a single step in the grid and t denoting a relative frame number. Consequently, the computation of $z_{i,j}$ in the current frame needs only the value of its four direct neighbours from the last frame and the value of $z_{i,j}$ from the last two frames. The stability constraints are non-moving boundaries of the grid and $\frac{s^2 \Delta t^2}{h^2} \leq \frac{1}{2}$ otherwise, the heightfield would grow exponentially [Gom00]. The final z value is scaled by a damping coefficient $a < 1$ to decrease the wave's energy. If the damping is not applied, the wave motion will stay indefinitely. This coefficient can be locally adjusted, so the wave behaves more naturally to adhere to some terrain features.

4.2.2 Splashes and Sprays

The generation of water splashes and sprays is a complex process, and it depends upon several independent situational variables. It is a function of vehicle (tire) speed, the water area's depth, tire design, and many additional factors such as the tire's aerodynamics or even the wind.

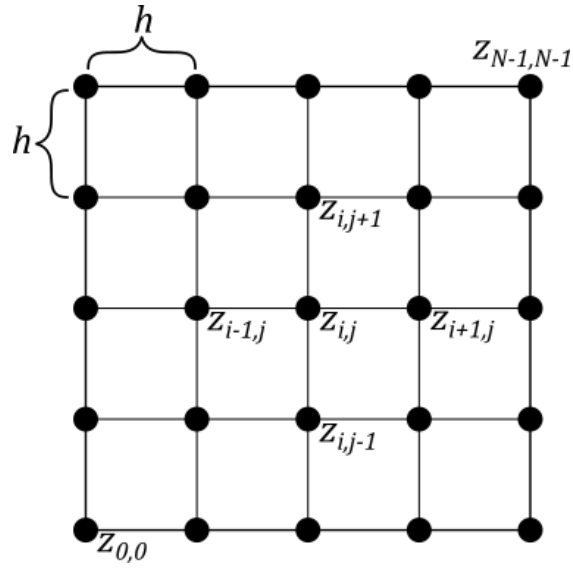


Figure 4.4: A heightfield with N points along each side used to approximate the water surface, which is treated as a tightly stretched membrane that ignores gravity [Gom00].

Usually, splashes and sprays are two separate processes characterised by the droplet's size. Splash is an action of forcing the water out of the tire's path and consists of drops larger than 1 mm in diameter. Sprays are formed by atomised water droplets smaller than 0.5 mm [Pil90]. Nevertheless, they are often referred together for simplicity because it is challenging to monitor and measure them individually (further labelled plainly as splash).

A splash can be deconstructed into four mechanisms: bow waves, side waves, tread pickup, and capillary adhesion (Weir et al. [WSH⁺78]). The bow and side wave consists of relatively large drops while tread pickup and the capillary drips are shattered into spray (see Figure 4.5). Previous research (Flintsch et al. [FWGV12], Pilkington [Pil90]) mention that the minimum speed before a measurable spray is formed is in the range of 48 to 64 km/h. A maximum speed may exist if the car starts to hydroplane instead of making splashes.

There is an assessment tool predicting splash potential for numerous road types and rainfall by Flintsch et al. [FWGV12]. Their method primarily provides valuable information for supporting highway design. They started by developing a water film depth model (WFD) on a surface based on its drainage properties and rain intensity. From this, they established an exposure model for estimating the amount of water that is going to be projected by the wheel (a splash model), given the WFD model, road properties, vehicle speed, and other circumstances. The models which they presented are essentially empirical. The parameters and values were determined using laboratory and field experiments, theoretical developments, and computer simulations. This methodology shows some limitations in term of realistic simulation but can be well trusted as a guide to designing a believable visual component.

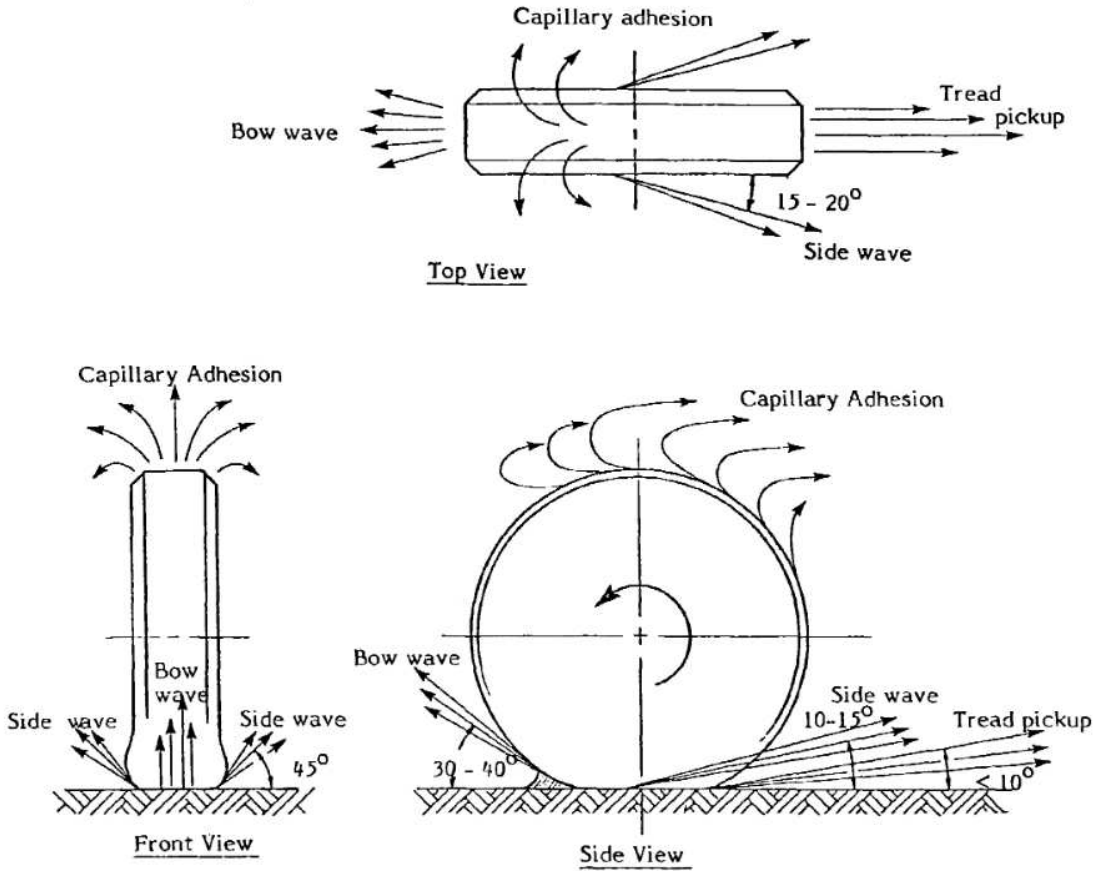


Figure 4.5: Primary splash and spray generation mechanism taken from Weir et al. [WSH⁺78]. Water passing into the tire's tread pattern can be projected directly behind the wheel (tread pickup) or lingers in as a thin capillary film. Bow and side waves are more notable as the wheel drives into a deeper puddle.

The splash model by Flintsch et al. [FWGV12] determines the maximum amount of water (M_w) available for splash and spray according to the equation 4.11:

$$M_w = v \cdot b \cdot \text{WFD} \cdot \rho_w \quad (4.11)$$

where v is the car's velocity (m/s), b is tire width (m), WFD is water film depth (m), and ρ_w is water density. They further calculate the amount of water for each mechanism separately in a certain order until the total amount of available water is exhausted. For example, if the water from capillary adhesion remains, it will be used for tread pickup. The bow and side waves are activated last [FVD14].

A potential limitation of this model could be the empirical WFD model. The authors stated that the water depth model needed improvement, especially for a lower amount of precipitation. WFD is highly associated with local precipitation (rainfall intensity and duration) and pavement properties (geometry, texture, materials and permeability), and many researchers evaluate the impacts of these intervening factors on WFDs based on field data. According to Luo and Li [LL19], the empirical models of WFD lack in the local setting, so the results might be invalid in other regions. The model is also

affected by how the rainfall intensity is defined or just by the storm design overall. A sole constant intensity value cannot reflect a varied precipitating environment. Nevertheless, an accurate WFD model is not so crucial to reproduce a resembling splash effect for the driving simulator. Therefore an analytical model does not necessarily have to be applied.

4.3 Rainfall

A common approach to simulate rain is to build a particle system. Its complexity and comprehensiveness come from all the raindrop effects that are taken into account. Whether the particles are tracked individually or not, it is crucial to include each particle's interaction with the environment. Precise collision detection for the possibly million spawned raindrops is nearly impossible in real-time if no optimisation is incorporated. Besides the particles' position and update, their appearance should correspond to or at least resemble the natural phenomena. The following section outlines which methods can be advantageous to the visual simulation of raindrops in scenarios where the camera attached to a car moves quickly. The particle system has to overcome such constraint of accommodating the new camera states in each frame.

4.3.1 Raindrop Particles

The main drawback of the particle methods is the number of raindrops needed to translate and visualise to give a realistic impression. The management cost of many particles in a vast scene can be reduced by limiting the particles into a smaller container, incorporating some LOD techniques to change the size and location of the particles while preserving the realistic appearance of rain [PCRC09]. The rain container's shape can be of various shape and size, heeding cost-effective performance and a suitable distribution of particles within the container. Creus and Patow [CP13] represented their rain container in a shape of a block, defined by an orthographic camera such that its near plane matches the top of the space and far to its ground level (see Figure 4.6a). Particles then move in the view direction from the top straight to the bottom with a constant speed equal to the droplet's terminal velocity. Puig-Centelles et al., on the other hand, shaped the container in a cylindrical manner with an ellipsoidal base at the camera. The placement allows a better alignment of the particles to the observer's field of view and effectively discards wasteful particles at the back part behind the cylinder (see Figure 4.6b).

The particles inside the container can be altered in order to follow the level of detail pattern. More particles can be rendered near the camera, while further away, the number will decrease according to the observer's distance. Puig-Centelles et al. [PCRC09] additionally expand the particle's size the further it is from the camera. The size retaliates the reduced amount of particles, visually simulating the attenuation due to countless overlapping raindrops.

The relation of the distance with the number of particles follows a linear distribution (distance d increases while the number n_p decreases), defined as $n_p = 1 - d$. Meanwhile, the particle size (s_p) with relation to the distance is quadratic $s_p = d^2$ [PCRC09]. The particle distribution in the container can be purely artistic as well – the rain intensity is not entirely uniform in every part of the scene. Allowing the manually specified density map simplifies the control of rain distribution. Creus and Patow [CP13] consider such a

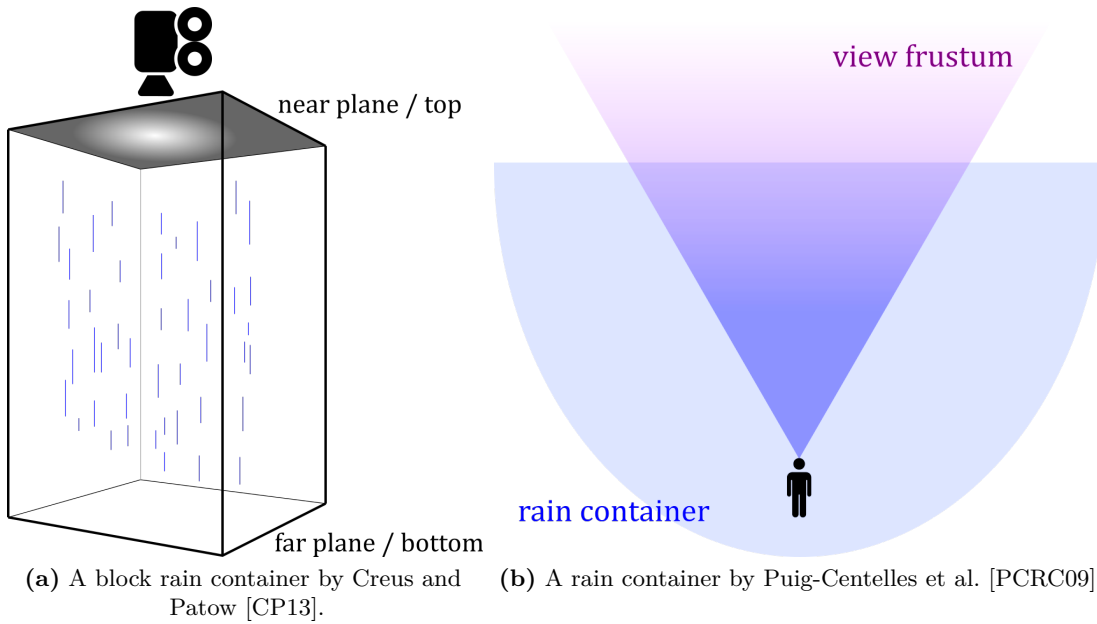


Figure 4.6: The rainfall simulation can be limited to a smaller section near the camera (observer).

map as a grid filled with quantised density values. For each unique value, a packet with a corresponding amount of particles is created.

Regardless of the level of detail, the number of rain particles is ultimately determined by the rain intensity. The intensity is classified according to the rate of precipitation (in mm/h) into:

1. light rain,
2. moderate rain,
3. heavy rain.

The rain intensity is coarsely organised, though a more granular division can be used (drizzle, cloudburst). Refer to Table 4.1 for the specific rate of fall. The precipitation rate is utilised to predict the number of raindrops per unit volume with a specific size interval in eq. 3.1, see Table 4.2. The definition of rain intensity can also be used to manage the visual simulation of the other rain-related phenomena.

	rainfall rate (mm/h)	raindrop sizes (mm)
light	≤ 2.5	0.2 – 2.5
moderate	2.6 – 7.5	0.6 – 3.0
heavy	≥ 7.6	1.5 – 2.5

Table 4.1: The rainfall rate for each intensity of rain and typical rain drop sizes according to Barani Technologies [Bar20] and Kelkar [Kel45].

rainfall rate (mm/h)	raindrop diameter (mm)					
	D = 0.5	1.0	1.5	2.0	2.5	3.0
R = 2 (<i>light</i>)	1360	231	40	7	1	0
5 (<i>moderate</i>)	1854	429	99	23	5	1
10 (<i>heavy</i>)	2260	638	180	50	14	4

Table 4.2: Distribution of raindrops in terms of rainfall intensity (R) and diameter of raindrops (D) calculated using eq. 3.1. Values are expressed in m^{-3} .

The distribution of raindrops or rainfall rate is strenuous to apply in a visual simulation, especially if the performance should be in check. Table 4.3 circumscribes the rain intensity without the aid of instrumental measurements, which can be used as a visual guide [Mon13]:

	individual drops	raindrop splashes	puddles formation
light	easily seen	hardly any	slow
moderate	not easily seen	noticeable	rapid
heavy	not identifiable (rain sheets)	large splashes to a height of few centimetres	very rapid

Table 4.3: A table of visual perception of rain intensity and its impact – word interpretation of measured values [Mon13].

4.3.2 Rendering Rain Streaks

Most often, the rain streaks are approximated with rectangles or ellipses of constant brightness. This approach can only be used when the rendered rain is far from the camera, where the streaks are thin enough to make their brightness distribution irrelevant. The raindrops closer to the camera cover a greater amount of pixels and therefore reveal an elaborated intensity pattern.

As known from subsection 3.1.4, a raindrop contains a distorted wide-angle projection of the environment, albeit at a lower resolution [GN07]. This behaviour can be faked without ray-tracing by re-mapping an environment texture onto the shape on a raindrop, as Rousseau et al. [RJG06] suggest. The approximation made to the physics makes it possible to render streaks in real-time.

Moreover, the average brightness of a raindrop tends to be higher than the background it occludes, and its intensity is not affected by other raindrops. The brightness is further deformed by a motion blur (see Figure 2.14), causing the streak to be more transparent. Garg and Nayar [GN06] rendered a database of rain streak textures by using their

determined oscillation model (see Figure 2.9). The lighting configuration parametrises each texture as seen in Figure 4.7. Three different variables determine a streak: θ_v the camera's elevation angle, θ_l the elevation of the light source, and ϕ_l the azimuthal angle of the light.

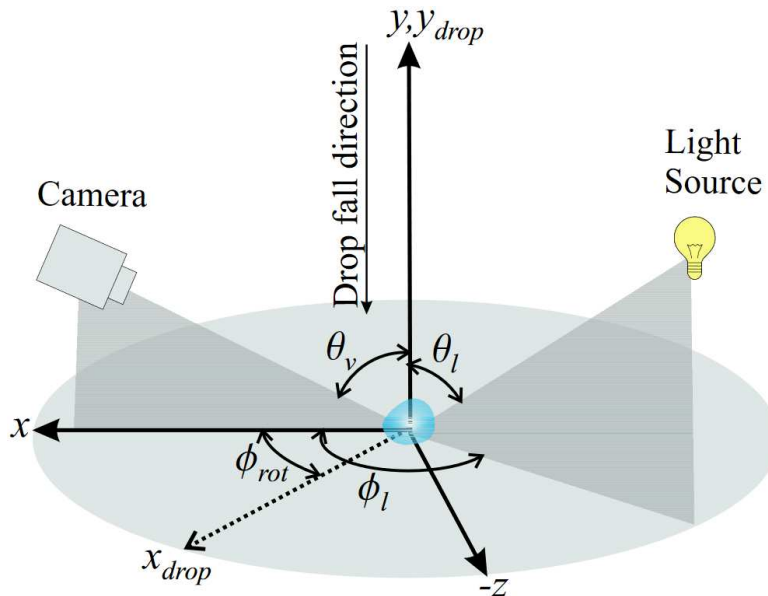


Figure 4.7: The raindrop is at the origin of coordinates and falls in the opposite direction of the y -axis. θ_v is the camera's elevation angle, θ_l is the elevation of the light source, and ϕ_l is the azimuthal angle of the light. The x_{drop} is the orientation of a raindrop toward the camera [GN06].

4.3.3 Interaction of Raindrops

The rain interacts with the environment and would not usually fall through objects in the scene. When a falling drop hits a surface, it causes a tiny splash along with a ripple effect if the drop happens to end up in water (see section 3.2). Tracking each particle to detect its collision assures the raindrop culling at the right time and correct placement of splashes. However, it is viable only for a low amount of particles in real-time applications and should be considered only near the camera. Creus and Patow [CP13] compare the rain streak position projected by an orthographic camera with the scene's depth buffer. The particle is visible as a rain streak if it is closer than the buffer's value. Otherwise, it is either substituted as a splash animation or invisible (after collision). The streaks do not update the depth buffer to avoid higher particles occluding the ones near the bottom.

The splashes do not need to be associated with each raindrop (from the work of Tatarchuk [Tat06] and Lagarde [Lag12]). Seldom can a specific raindrop be tied to a particular water splash. Based on this visual evidence, the two systems – raindrops and splashes – can be simulated separately. Tatarchuk generated random rays starting from the top of the world to the bottom and rendered water splash animation at the origin of a detected collision. The random height position can be calculated from a depth map as well, using an orthographic camera (placed on top and looking down as the one by

Creus and Patow [CP13]). The random position is generated in the world space and later projected to retrieve the height position of normalised device coordinates from the depth map. An inverse projection then recovers the final height of the world coordinate.

Analogous to raindrop particles, the splashes can also be displayed solely near the observer (and integrating related LOD techniques similar to one of the raindrop particles). The appearance of a drop splash is challenging to model, but luckily the details of a splash are not perceptible in typical situations (also discussed in section 3.2). An animated texture sheet of high-quality splash images sequence can be re-used for thousands of particles. Distinctive scaling factors then modulate the size or transparency to overcome the repetitions [Tat06].

4.3.4 Particle Alignment with Moving Camera

The rain generally falls in a vertical direction, almost at a zero angle, if the wind forces are wholly omitted. The camera in the scene is usually attached to a driving car, and the rain container follows in the vicinity of the camera. Because of the car's horizontal movement, the rain appears inclined at a certain angle – the higher the velocity of a car, the greater the inclination angle (see Figure 4.8). For a bystander, the rain appears to fall straight down, but for the camera, the rain looks like it has an opposite horizontal velocity to the car. Due to this, the rear of the car gets hit less by the rain than the front.

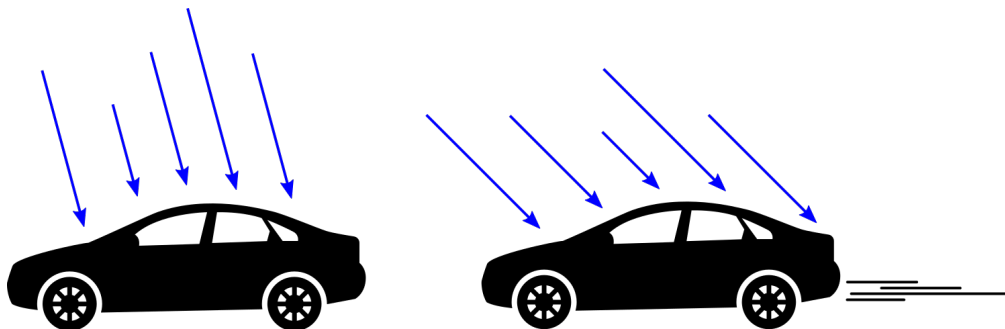


Figure 4.8: If the car is stationary or drives slowly, the rain pours down. As the car accelerates, the raindrops appear to come towards the car at an ever-increasing angle.

4.4 Implementation

The implementation of the aforementioned rain-related effects is executed in the Unity¹ game engine. Some adjustments had been made to the visual simulation in Unity to path-trace viable images with the Octane Render².

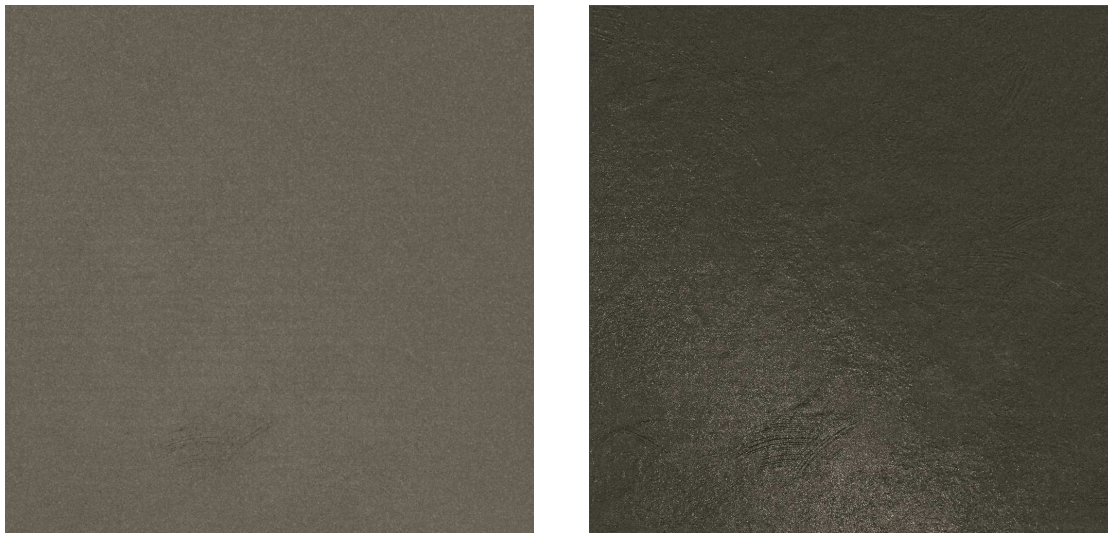
¹<https://unity.com/>

²<https://unity.otoy.com/>

4.4.1 Appearance of Wet Roads and Puddles

The physically-based BRDF of Unity is assumed to be derived from the Disney work [BS12] and based on the Torrance-Sparrow microfacet model. A custom surface shader represents the road material, which allows direct changing of the material attributes. The information regarding how wet the road (wet region) is can be controlled by a uniform variable named *wetness*. It defines the drenching strength caused by rain, allowing the control of progressive damping or drying of the road (see Figure 4.9). The variable is the alpha coefficient for the linear interpolation between the two values of coefficients modifying the albedo and smoothness, as shown in a fragment of code below:

```
albedo = albedo * lerp(1.0, 0.3, wetness);
smoothness = min(1.0, smoothness * lerp(1.0, 2.5, wetness));
```

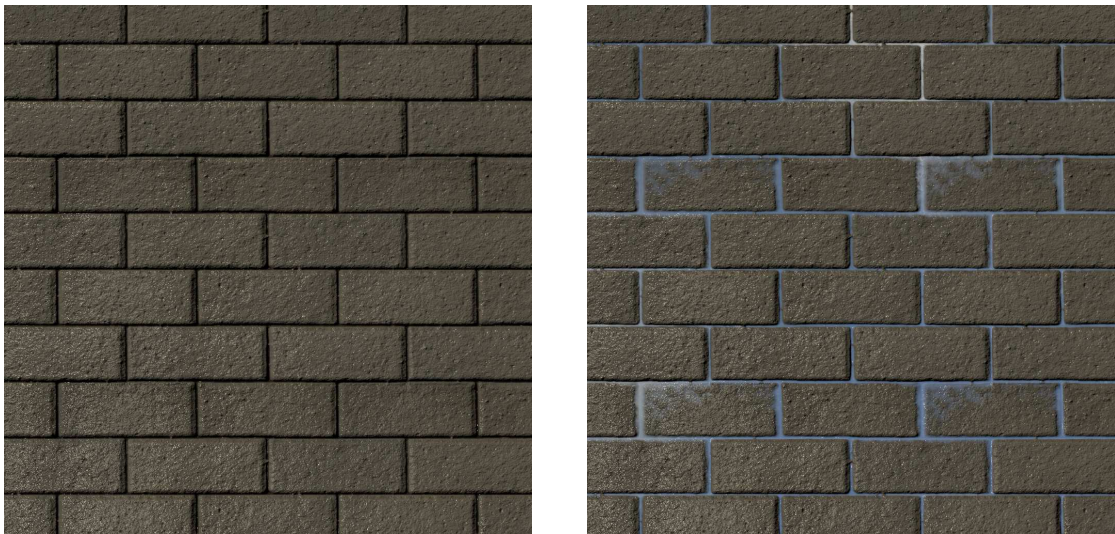


(a) : A dry asphalt road material with minimal cracks.

(b) : A drenched version, notice the darkened albedo and increased specularity.

Figure 4.9: The figure displays a road material with two different values to the *wetness* variable.

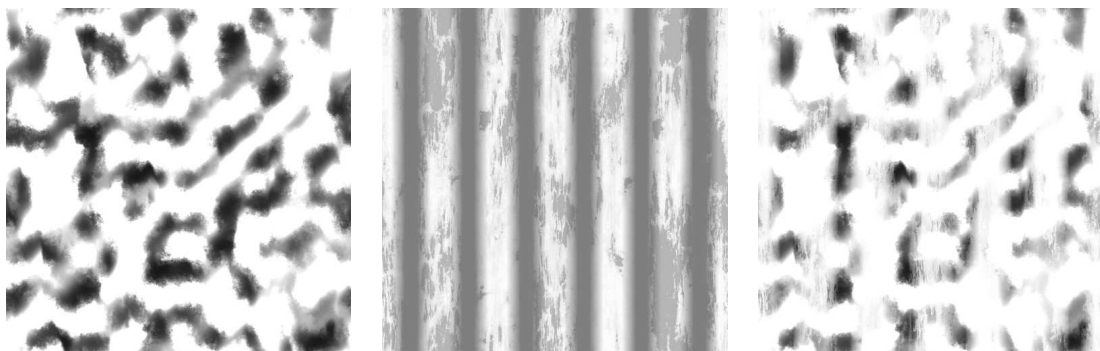
As for puddles on the road, the drenched region (see section 4.1.2) was omitted, and instead, the current model (dry or wet road region) is interpolated with the puddle surface properties to simulate water accumulation in lower areas. A traditional heightmap is used to regulate where a puddle will form in place of elaborate undulation data. Given the characteristics of the heightmap, the water might fill smaller gaps and cracks present on the road. A heightmap texture of the material might work in some cases, where the structure is recurring (see Figure 4.10). However, to break visual repetitions of emerging puddles on a tiled lengthy road, another larger greyscale texture called a *puddle map* is used to represent the random placement of puddles. This puddle map texture is locally blended with the heightmap, so the puddles reflect the road's uneven shape (in Figure 4.11 such a blending method is demonstrated).



(a) : Drenched pavement with no additional water areas

(b) : Wet pavement with water gathered inside the tiny gaps between the bricks

Figure 4.10: A heightmap of a material defines where water concentrates according to the material geometry. Besides the regular gaps, the water also fills other lower areas, highlighting the repeated pattern if the texture is tiled.



(a) : puddle map

(b) : height map

(c) : blended puddle map

Figure 4.11: On the left is a puddle map generated from a Perlin noise, a figure in the middle displays the road's height texture (multiple lanes with rutted tire marks). On the right is the result of the screen blending of those two textures. Notice how the puddles slightly aligned to the lower areas of the heightmap.

The amount of water gathered in the puddle areas is defined by another shader uniform variable *waterLevel*. The variable visually simulates a puddle's depth – if the final blended area is lower than the specified water level, it will be filled with puddle water accordingly:

```
puddle = max(0, (waterLevel - blendedPuddleMap))
albedo = lerp(albedo, albedo * 0.8, puddle);
smoothness = lerp(smoothness, 1, puddle);
```

The amount of water regulates the puddle's intensity and could be considered the intermediate condition between the wet and puddle region. It is possible to have another variable to adjust the flooding or drying rate for small holes or cracks compared to the prominent puddle areas.

The water layer which fills the puddles is smooth, and with the normal vectors facing straight up, the puddle should be highly reflective. The water reflections are imitated using a camera with its near plane set at the same level as the object reflecting the scene (clipping objects underneath). The camera then renders into a render texture. The reflection texture is passed to the surface shader and is connected to the emission parameter, eventually adding to the final colour of the water surface. Note that the puddles are still altogether merely a flat surface. These puddles remain featureless and straight like a mirror surface; hence displacing the surface with ambient waves evokes the impression of a real-time water puddle. The feature is added by mixing two normal maps that are shifted and scaled in time using the Unity game time variable. The exact process happens with the reflection texture, and together with the animated bump maps, these reflections are moving with the waves (see Figure 4.12).



Figure 4.12: Notice the slightly displaced reflections in the right-hand figure.

4.4.2 Interactive Puddles

The pushing wave effect simulated within the 2D array is implemented in the engine using a custom render texture. The calculation is handled in a fragment shader with a

double-buffered texture of size 512x512. This texture-based viewpoint efficiently provides the bounding conditions implicitly (where the grid's edges are staying still). The last frame is stored in the red channel, while the frame before last is stored in the green channel. This manner of storing the heights begets a black and yellow coloured texture shown in Figure 4.13.

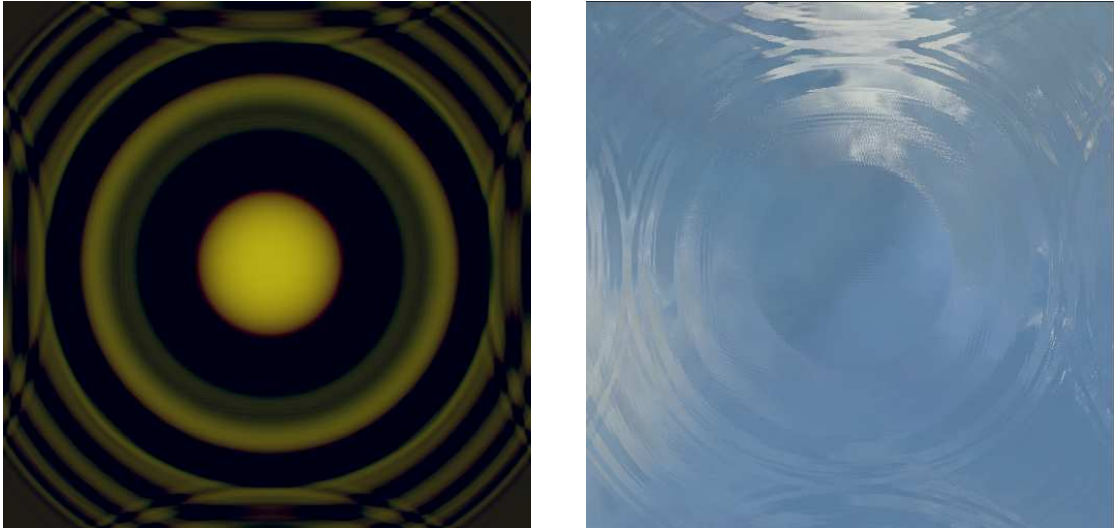


Figure 4.13: (left) A texture storing two frames of data (in R and G channel) regarding the water surface's height. This texture is used to calculate the height of the wave for the next frame. (right) The displaced water using a normal map generated from such a heightfield.

Finally, this heightfield is passed as a wave texture to the custom shader for the road material. A normal map is generated from this wave texture and then added to the already existing ambient waves. The normal maps are combined with the Unreal Developer Network (UDN) blending method [Unr12], [CS12]. It is a type of linear blending, except that only the x and y component from the detail map is added:

```
float3 r = normalize(float3(n1.xy + n2.xy, n1.z));
```

The actual wave motion is started at the wheel-puddle contact position after a collision via ray cast is detected. The place of contact is used to determine the heightfield texture's UV coordinates, and the height value is set to 1 with an update zone method available from the Unity engine's custom render texture. The engine's discrete collision detection should be taken into account, so the ripples are connected into a reasonable looking wave motion (for example, linearly interpolating between the two contact points), as seen in Figure 4.14.

For splashes, the empirical WFD model of Flintsch et al. [FWGV12] was replaced with the straightforward method for forming puddles on the road. The drainage properties are not dealt with for simplicity purposes. Therefore, the water film depth can be obtained only from the puddle map blended with the road's heightmap and the water level variable. Of course, this approach only determines where a splash is likely to occur, given the puddles' placement (and the depths) set by the maps. The depth of a puddle is calculated in millimetres with a maximum depth of 10 mm for the shallow puddles.

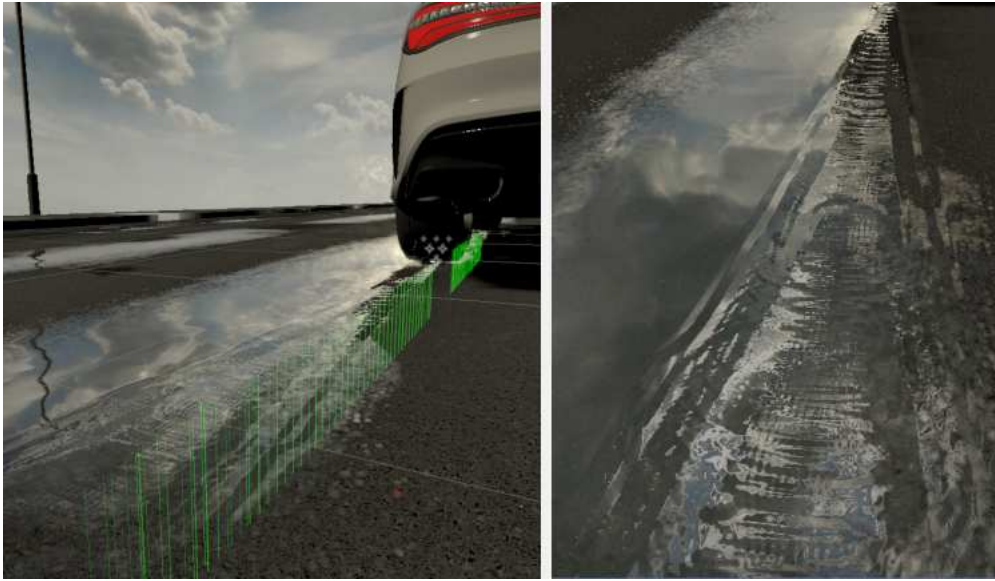


Figure 4.14: The green lines depict ray casts done at the contact points between the tire and the road. At each point, we disturb the water surface and start a wave at the calculated UV coordinates of the heightfield. The collision detection does not operate with the entire area of the tire.

The findings from Weir et al. [WSH⁺78] are applied to the splash particle system provided by the game engine. Threshold values are put together for each of the four splash component based on WFD and vehicle speed. The maximum amount of water (eq. 4.11) is then used to fine-tune the properties of each splash mechanism described above. The emission rate, shape, and angle of the water projection are modified to match the amount of water and the vehicle's actual speed. Each wheel of a car has its *splash* game object assigned and will emit the particles when it drives into a puddle (see Figure 4.15). The puddles are identified with the same method as calculating the wave heightfield's UV coordinates, and this is executed in the same pass. The simulated water surface on the road does not react to the generated splash particles, nor is there any particle-particle interaction.

4.4.3 Rain Container

The raindrops fall in a block-shaped container near the primary camera. The dimensions of the rain container depend on the type of scene and can be adjusted. For example, a rain container in a wide-open motorway environment will be far-reaching. Meanwhile, in cities, the container might be smaller because of occluding buildings that might discard a moderate amount of particles.

The number of active rain particles is determined by the rain intensity (light, moderate, or heavy) and the container's diagonal. The rainfall is simulated by generating a fixed amount of particles (*raindropCount*) which happens randomly during a time window (*rainfallRate*). The magnitude of the diagonal linearly scales the raindrop count variable, so the container is evenly populated with raindrops regardless of its size. Table 4.4



Figure 4.15: Each car wheel has its splash game object, levelled near the bottom of the wheel.

displays the values for each of the three rain intensities. The visual feedback from Table 4.3 aided in fixing those numbers.

	raindrop count	raindrop rate	splash probability
light	9	50	0.5
moderate	43	70	0.33
heavy	115	90	0.25

Table 4.4: Displayed values are for a rain container with dimensions 40x15x80 in Unity length units. The length of the diagonal of the container scales the raindrop count linearly. Raindrop rate is in s^{-1} .

The random positions of raindrop particles are uniformly distributed perpendicular to the camera's forward vector. In the camera's viewing direction, the positions are extracted from a normal distribution and clamped so that the raindrops concentrate in front of the camera. The size of raindrop particles does not adhere to the typical proportions given by the rain intensity (see Table 4.1) but are in relation to the distance from the camera. The slight distinctions in the drop diameters cannot be easily perceived because of the intricate variety of brightness within a rain streak. Hence, raindrops closer to the camera are modelled to be smaller in contrast to the raindrops far in the back in a quadratic manner (see section 4.3.1). Figure 4.16 illustrates the rain container and the particles within.

Particles fall vertically from above, starting at the randomly generated position within the container. A ray is cast downward from this position to detect when the raindrop will hit an object in the scene and turn into a splash. The length of the falling trajectory and

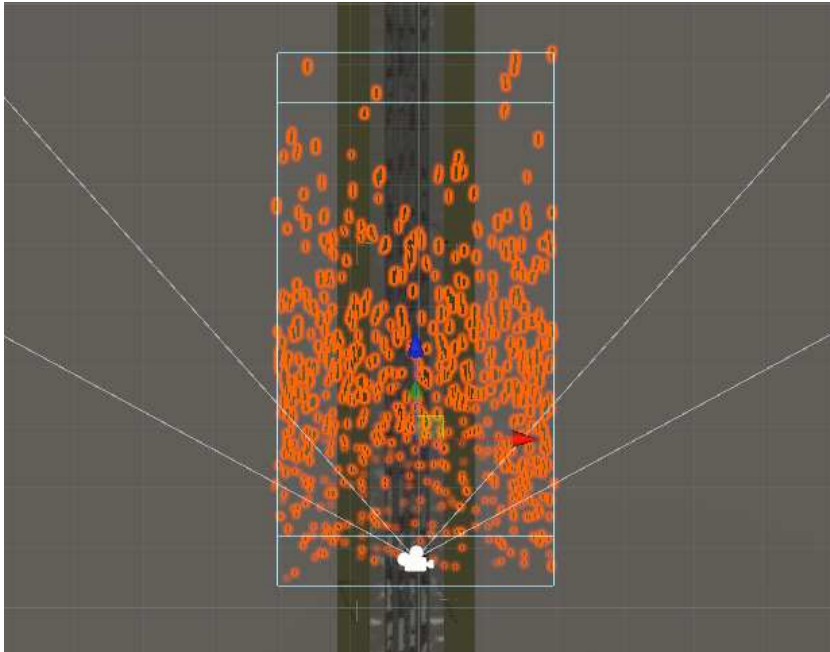


Figure 4.16: The figure shows the rain container from the top view with the position of the camera. Raindrops are outlined in orange colour. The normal distribution is centred at the bottom side of the container and clamped following the *three-sigma rule*.

the velocity is used to determine the time duration of the raindrop particle – this ensures that raindrops do not pass through tangible objects and are adequately destroyed.

The falling trajectory can be deviated to imitate the raindrop alignment discussed in subsection 4.3.4 by setting the velocity over time for the particle system. The velocity is changed in the opposite direction than the camera movement (see Figure 4.17). This extension is applied to all of the particles, which might disturb the level of detail set by the size of raindrops – larger particles will move closer to the camera, causing unrealistically long rain streaks. This matter can be avoided by editing each particle individually at the expense of performance.

■ 4.4.4 Raindrop Splashes and Ripples

Splash and ripple effects are reproduced independently of the raindrop particles without any notable visual discrepancy.

Splashes are rendered on the top surface of an object (respectively, the collider of an object) if nothing is blocking the falling raindrop particle to detect its collision with the object. A splash billboard (shown in Figure 4.18) appears with a certain probability (given by values in Table 4.4) at the randomly generated position (excluding the height), at which the raindrop spawns. The splash simulation likewise happens within a rain container because of those previously attained positions.

The water surface simulation technique (applied for waves) is utilised again to form ripples caused by raindrops. Although the active raindrops do not have distinctive diameters according to the rain intensity, the generated ripples on the puddle surface have various strength given by the frequencies of raindrop sizes (Table 4.2). The diameter



Figure 4.17: The car drives to the right side while rain particles fall from top to bottom, slightly shifted to the left side. The alignment works only for the primary camera and should not be observed from a different camera (for instance, from a static camera nearby).



Figure 4.18: A texture with alpha source from grey scale used to render a splash particle at a random position on the surface of an object in the scene. The particle lasts some time between 100-300 ms.

of a raindrop defines its ripple energy, which prevents visual invariability or regularity. The render texture representing ripples is of a lower scale of 256x256 and tiled to cover the road area. Such a small grid makes it reasonable to simulate countless ripple-like effects while repressing heavy computations (see normal maps in Figure 4.19).

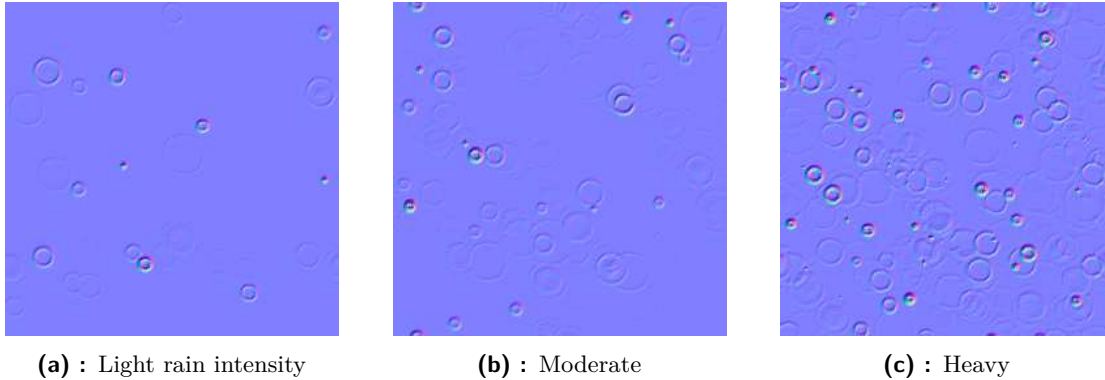


Figure 4.19: Exported normal maps readjust the lighting calculations, resembling ripples caused by fallen raindrops on a water surface. Each map covers an area of 2 by 2 Unity units.

4.4.5 Rain Control

The rain intensity R has four categorical values: 0 (no rain), 1 (light rain), 2 (moderate rain), and 3 (heavy rain). The condition of the wet rainy environment can be described by the specified variables wetness w and water level wl . By adding a time variable t , the wet state will also depend on time. The situation can be modelled by a function that considers the duration of rain intensity and outputs a pair of values between 0 and 1: $f(R, t) \rightarrow (w, wl)$. Figure 4.20 illustrates a simplified relation of the mentioned variables.

4.4.6 Ray-Traced Outputs

Besides the rasterised version from the engine, path-traced images can be obtained with the Octane render plugin for Unity. The render is unbiased, offering photorealistic quality that is by far superior to what any real-time engine based on rasterisation can achieve. However, Octane render does not have complete access to the game engine's rasterisation pipeline, so a few adjustments must be made to achieve visually pleasing results.

Octane does not recognise the custom shader since only the standard shaders of Unity are supported. Hence, a new road material is assembled as a PBR Override Material shader – the OctaneRender-specific material type. A combination of different materials is chosen instead of a single material type. The material is made of a mix of a diffuse road and specular water material using the Octane's node graph tool.

The material has roughly the same properties and is controllable by two variables *wetness* and *waterLevel* specified from Unity. The *wetness* variable adjusts the diffuse colour of the road material (using gamma) and the roughness parameter, which controls the specular highlight's distribution on the road. The *waterLevel* regulates the gamma of the puddle map. The puddle map blending and mapping were executed with an OSL

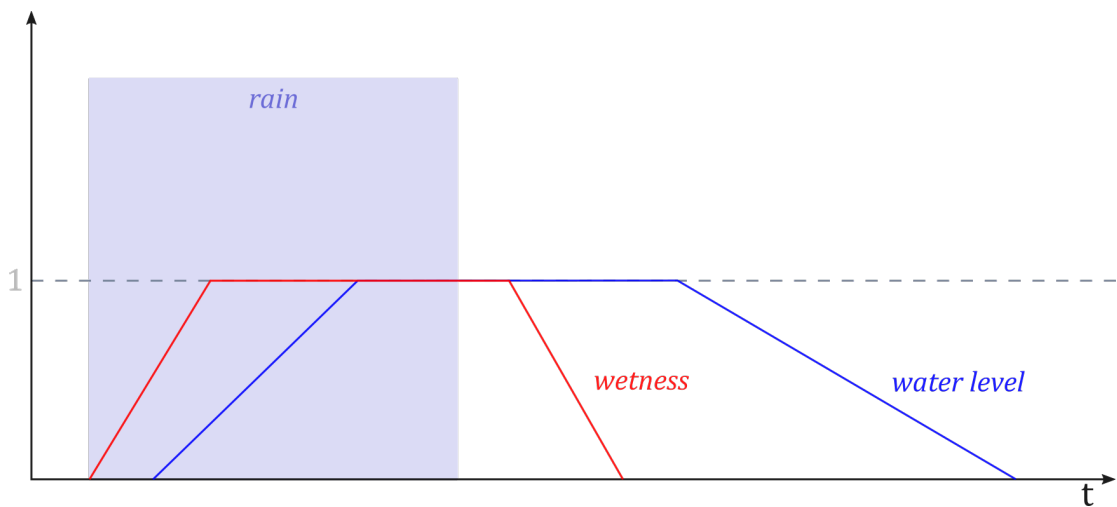


Figure 4.20: When the rain starts, the wetness will increase and cause overall surface damping (wet regions). Halfway, the puddles will be slowly filled as the permeability of the material decreases. The road will be entirely drenched before the water level reaches the maximum. The drying process will not start right after rain but after a short while – the length might depend on variables such as temperature. The accumulated water in puddle areas lasts longer, unlike the thin water layer, which evaporates quickly.

shader (Open Shading Language) through a scripted graph node. The puddles need to be placed roughly at the same spot, similar to the rasterised outputs because the splashes are calculated directly from the game engine (see Figure 4.21). The puddles in Octane do not react to the wheels' contact as they do in the game engine; however, they display the ambient waves on the surface with an animation node to simulate time.

Octane has no support for Unity's particle system either. This drawback is dealt with by baking the particles into a mesh, and it is done so every frame. One wheel of a car can emit a maximum of 2000 particles to generate a splash. Those particles are packed into four game objects, each with a single baked mesh. The splash particles from Unity are rendered as billboards, so they are baked facing the render target camera (as seen in Figure 4.22). Raindrop and rain splash particles are baked in each frame as well. PBR Override materials are opted for these baked particles to get more realistic water behaviour.

■ 4.4.7 System Overview

The road textures have been created using Substance Designer³ and exported to Unity with a Substance in Unity⁴ plugin. The material is a smooth asphalt road with decals such as cracks, dirt stains and dark tire tracks. The Substance Archive file (SBSAR) imported into Unity allows a setting of a seed value. This value defines the placement and the number of cracks and dirt, so slightly altered textures can be generated to increase the variability.

³<https://www.substance3d.com/products/substance-designer/>

⁴<https://assetstore.unity.com/packages/tools/utilities/substance-in-unity-110555>

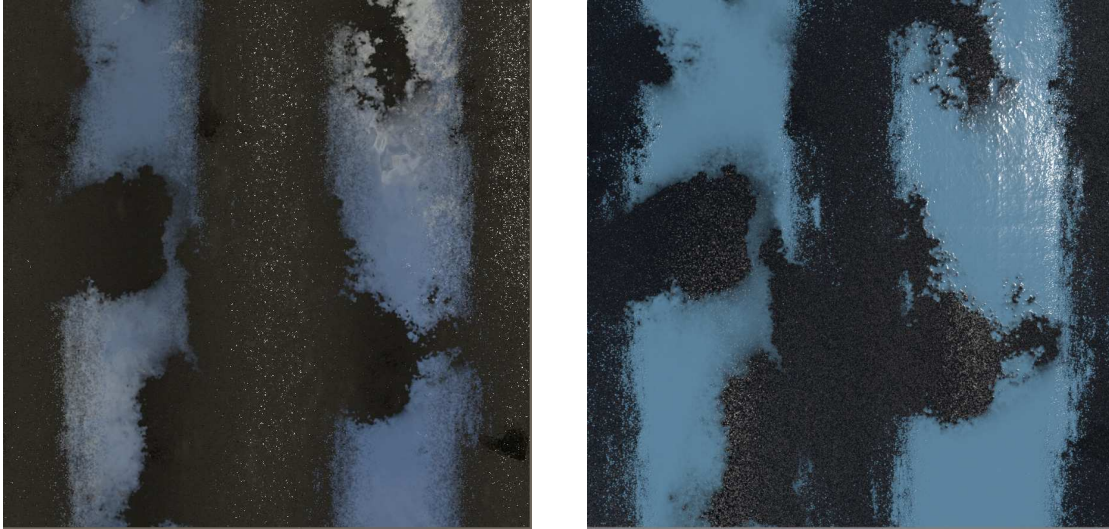


Figure 4.21: The puddles are mapped roughly the same way for both Unity (left) and Octane (right). The puddle areas in Octane are coloured blue for demonstration purposes (otherwise they are too subtle to notice with weak reflections).

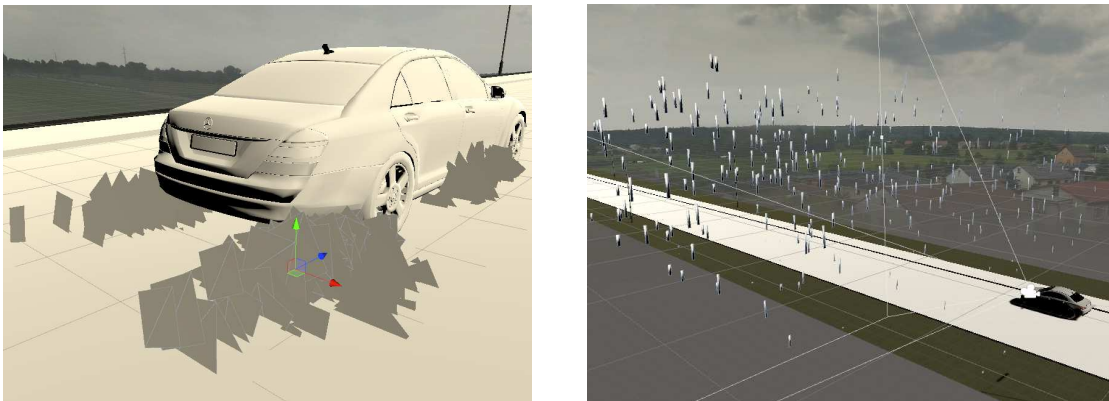


Figure 4.22: The particles are baked as textured billboards aligned to the render target camera. The standard Unity material is replaced for the Octane specific specular material to obtain more genuine water visuals.

The textures are used for both the Unity and Octane material. Unity switches between those two materials according to which type of outputs are desirable.

Before the scene exporting process, Lua scripts are loaded to edit the copy of a scene imported from Unity into the Octane plugin. Because the simulation runs only in the Unity game engine, every setting or property is therefore defined in Unity. If some data is required to be transferred into the Octane's copy of the scene (for example, rainfall intensity, wetness or water level), a dummy game object is created. This object carries encoded values (in the RGB channel), which the loaded Lua scripts find and then edit the Octane's nodegraph according to the extracted values.

The recorder tool available from the plugin then exports the scene into a compact ORBX file, which is then rendered using the standalone version of Octane. See the chart in Figure 4.23, which displays the flow of data more accurately.

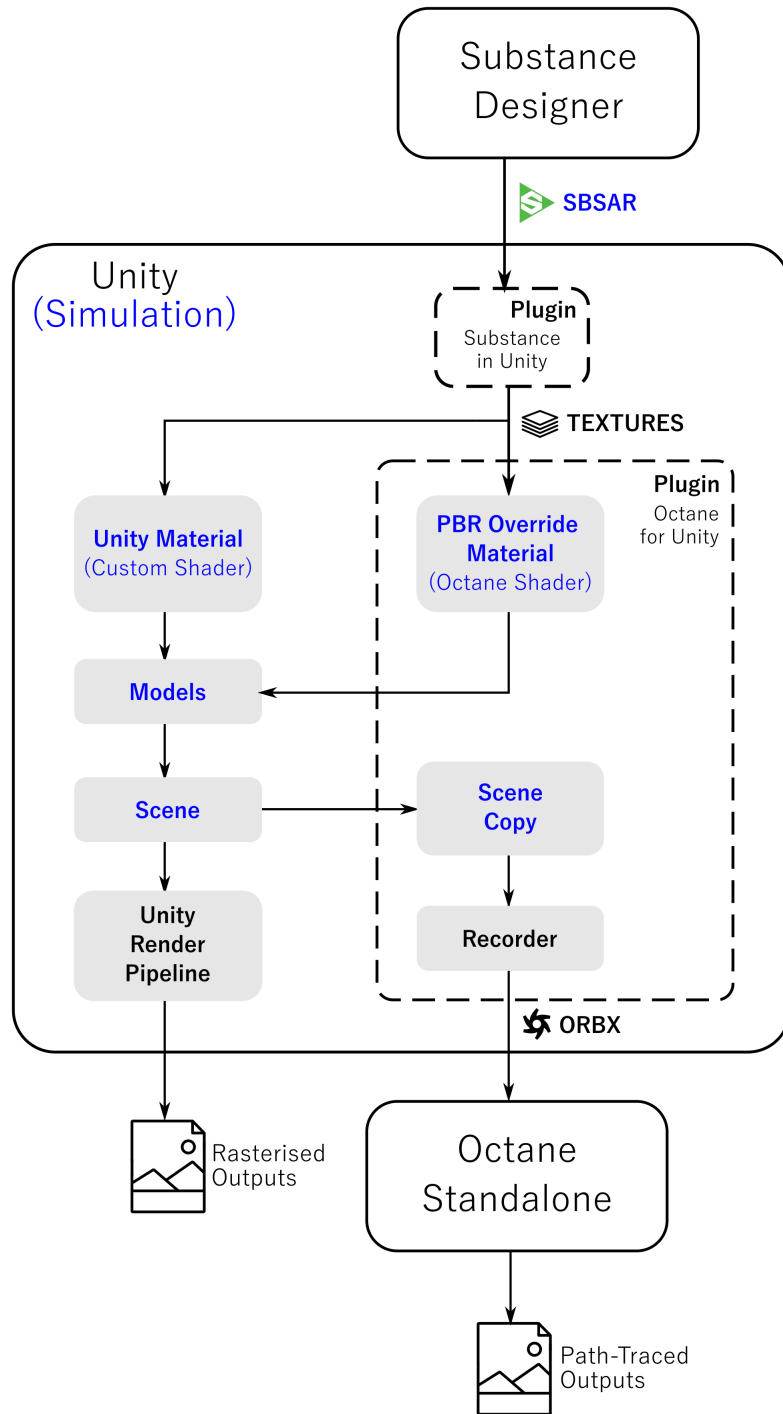


Figure 4.23: A data flow diagram of the system. The blue text highlights the parts which were directly changed or implemented in the visual simulation.

Chapter 5

Results

The chapter presents the obtained rasterised and ray-traced outputs from a constructed testing scene. Simple qualitative testing is performed with participants, and their opinions are noted. Next, the performance of the implementation is evaluated on a slightly modified scene to measure accurate values valid to the applied methods of visual rain simulation. Section 5.4 points out several limitations to the current approach and discusses problems emerging from combining a rasteriser with a path-tracer.

5.1 Visuals

The showcased test scene consists of a one straight asphalt road with two lanes and pavements. There are buildings, vegetation and some props on each side to populate the otherwise half-empty world. The objects surrounding the road emphasise the accumulated puddles and influence the final colour of the raindrops. Five cars on the road drive at a slightly different speed, demonstrating distinctive splashing amounts. At present, the lighting condition is set to produce two kinds of weather situation: sunny and a slight overcast (cloudy). Full images are disclosed in appendix B, both version from Unity and Octane side by side.

5.1.1 Wet Roads

The reference photographs of wet roads are displayed next to the results for visual comparison and help to refer to a few wet-related effects which can be witnessed in the presented outputs (figures 5.1 to 5.4).

To summarise the visuals of wet roads, the reflections in Unity are more specular, and the simulated water layer adequately expresses the porous type of the material. The imitated reflections seen in puddles are clear, which corresponds to the puddles seen during clear days. Still, the puddles might end up unrealistically luminous if the intensity of the reflection texture is not scaled proportionately.

The glossy reflections from Octane simulate the drenched regions efficiently. The thin water layer on top is composed of the specular material, which authentically represents the double layer effect. The puddles might not be as reflective as in Unity, but this could be refined by increasing the rendering samples or using a different normal map for the water surface. Octane outputs illustrate shallow puddles reflecting the contours of the objects in the vicinity.



Figure 5.1: (Unity) (Left) The photo captures a wet road on a sunny day with a thin layer of water. The asphalt is dark but with bright reflecting patches, which are striking further from the camera. The right side is already starting to dry, as it is higher than the rest of the road. (Right) The albedo of the wet road is darker, and the directional light is arranged, so the road reflects the light causing shiny spots (especially when the sun shines fully). The reflections imitate the thin water layer on top of a porous material.



Figure 5.2: (Unity) (Left) The shallow puddles in the photo are accumulated in the lower areas engraved by countless wheels over the year. (Right) The puddles have bright specular reflections and firmly adhere to the height information of the road. Adjusting the emission factor might tone down the brightness intensity. However, apparent reflections during sunny days are ordinary phenomena.



Figure 5.3: (Octane) (Left) The puddles in the photo are stretched along the road direction, close to the border of the tram track. The picture is taken from an angle that shows dark puddle reflections of the surrounding area. (Right) The puddles in Octane are fainter, more matching the murky type of water. The boundaries of the puddle areas are adequately mixed into drenched regions, unlike in Unity. The continuous transition makes the puddles a constituent part of the road material, better simulating shallow puddles.



Figure 5.4: (Octane) (Left) The photo exhibits a weak red reflection of the body of a passing car on a damp road. (Right) The thin water layer transforms the rough material into a smooth surface, proved by weak reflections on the road (not only in the puddles). This feature is most apparent in the path-traced version.

5.1.2 Driving Cars

Cars driving on the wet road demonstrate multiple rain-related phenomena. The following images portray the effects in detail. Except for the simulation of rippling waves caused by wheels driving into the puddles, which works exclusively in Unity, the results are examined in the rasterised and ray-traced version (figures 5.5 to 5.8).

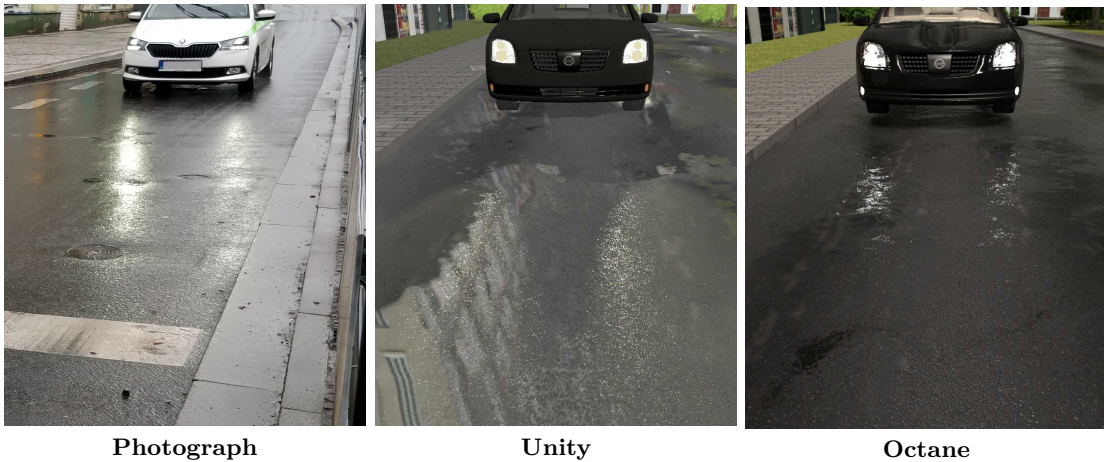


Figure 5.5: The car's headlights flash on the damp road covered with puddles. The light beam highlights the water-covered texture of the coarse asphalt creating shiny spots along the direction of light.



Figure 5.6: (**Unity**) The wheel passing through a puddle pushes the water out of its way to the sides. (**Octane**) The wave simulation does not work in Octane, but plain foam particles are generated when the tire makes contact with the puddle.

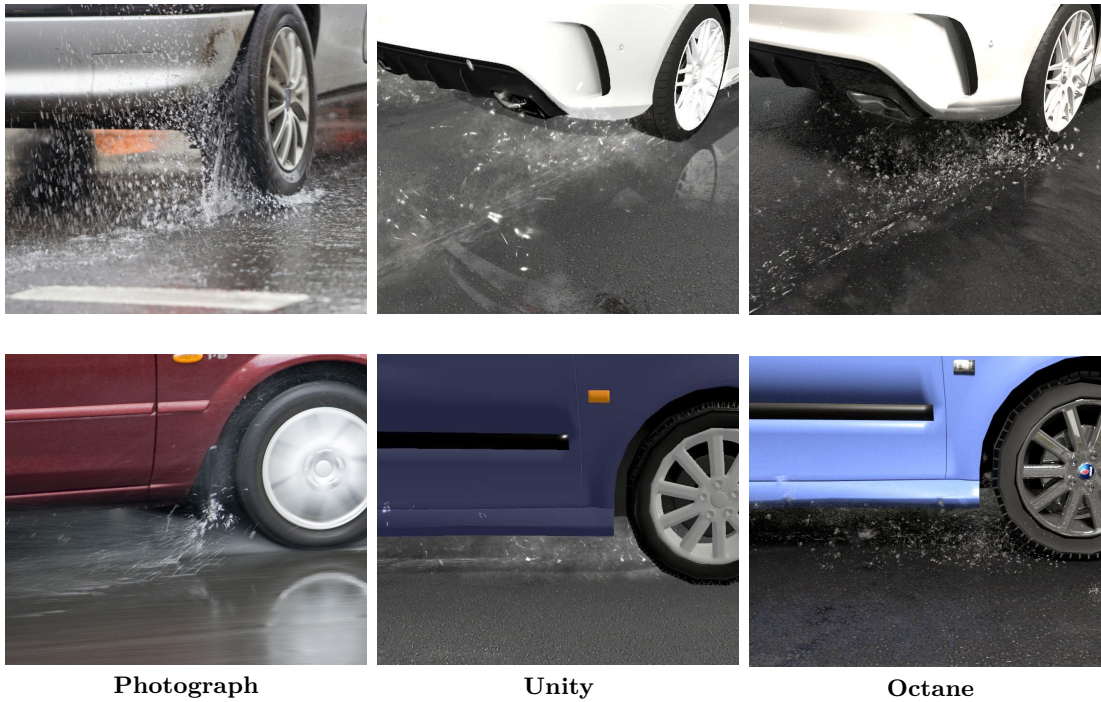


Figure 5.7: The figures show distinct shapes and aspects of splashing coming from under the wheels of a car. Driving into shallow puddles creates subtle spraying to the sides. Photos were taken from [3DC14] and [Cre].



Figure 5.8: (**Unity**) The splashes projected from the bottom of tires are smooth and interact with light in a way that displays the specular glints of the water material. Part of the splash goes sideways (side waves), and the small particles in the air imitate the spraying of tiny drops. (**Octane**) The splashes rendered in Octane are opaque, better resembling the spraying particles behind the wheel.

5.1.3 Rain

The simulated rainfall is compared to a photograph of heavy rain (see Figure 5.9). The light rain in the implementation is barely visible in images but is still included in Figure 5.10 to demonstrate varied rain intensity. The attenuation of visibility correlated to the rainfall and the distribution of raindrops is not considered in the implementation. Moreover, generating more particles induces visual artefacts due to the resolution and will not produce the natural meteorological phenomenon of decreased visibility.



Figure 5.9: Raindrops in Unity are rendered as transparent streaks, which causes them to be a lot brighter. The light source in the scene defines their colour opacity. The attenuation produced by countless rain particles in the air is not implemented in either scene, which lessens the visual quality of heavy rain weather as witnessed in the photograph.

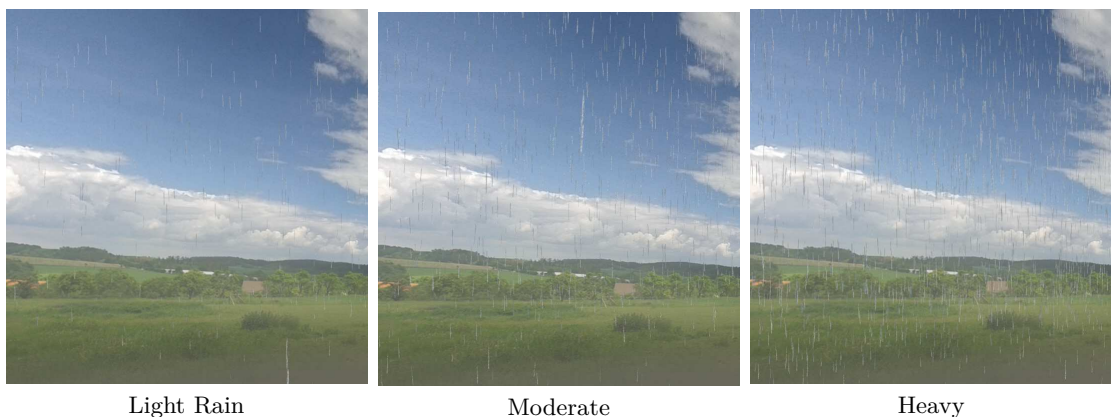


Figure 5.10: (Octane) The raindrops rendered in Octane reflect the surrounding due to the glossy material. In this situation, the environmental map defines the raindrop's brightness intensity.

5.2 User Study

A simple qualitative user study has been carried out to evaluate the visual plausibility of the simulation. The participants have been given several images from the game engine and the path-tracer. They judged the appearance of wet roads, the cars driving in a wet environment, the rain, and reoccurring phenomena in detail. They were asked to answer some questions listed below during the test scenarios:

1. What do you see in the displayed picture?
2. Did something catch your attention?
3. Can you characterise the weather in those pictures?
4. Are you particularly pleased by any presented picture?
5. Do you have any additional observations?

5.2.1 Participant 1

The first participant perceives that puddles and splashes in Unity are much more apparent than in Octane, which could be due to the higher specularity of the water material in Unity. On the other hand, the puddles in Octane are subtle because the reflections in them are weak. The participant believes that it is caused by the difference in the amount of water in puddles rather than the strong imitated reflections made in the rasterised version. As for the rain, the observer noticed the raindrop being quite long and opaque. The raindrops in Octane were told to be dim, especially in the city scene, where most raindrops reflect the vicinity. Due to this difference in the appearance of the raindrops, the participant would say that the rainfall intensity was not the same. The participant also points out that the splashes in Octane are *clear*, that each splash and foam is standing out. Meanwhile, in Unity, the splashes tend to form a soft white cloud around the car tires.

5.2.2 Participant 2

The second participant was pleased by outputs from Octane, presumably because of how puddle regions mixed well with the wet regions (described them as *natural*). However, the participant found the low-quality environment (buildings, vegetation) vexatious compared to the detailed road, especially in the Unity version. The participant mentioned that the puddles in Unity were shinier (and looking a bit *off*), striking as a sunny day after rain. The participant mentions how the rainy scene, especially in Octane *should feel* as heavy rain, but there was probably not enough raindrops. The participant said they could still see the buildings from afar, even though they should be less visible (if it was supposed to rain heavily). Finally, the car splashes in Octane were well received in contrast to the version in Unity, where the *misty* trail behind seemed rather odd.

■ 5.2.3 Participant 3

The third participant saw the difference in puddles between Octane in Unity as, again, the varied amount of accumulated water. The reflections in Unity were *too colourful*, despite the weather implied by the skybox. In either version, the wet road insinuated an environment after heavy rain, especially in the sunny scenes. The participant also noticed the discrepancy between the rain and the skybox. They also felt that the splashes in Unity were *flat* and *white*. However, the ones in Octane were natural since the splash had more *depth* – splashes were reaching the sides of the tire.

■ 5.3 Performance

The wet environment was implemented and evaluated in Unity version 2019.3.12f1 and Octane Render version 2020.1.5. There are five scenes on which the measurements are taken, each with a slightly different configuration (see Table 5.1). The buildings, vegetation and other props were removed to acquire values valid to the simulation alone. The scenes have a thoroughly wet road with accumulated puddles, and cars drive at a speed of 45 km/h. The resolution for both rasterised and ray-traced outputs was set to 1920x1080. The performance was observed on a computer with:

- CPU: Intel XEON E5-2630 v3 @ 2.4GHz,
- GPU: NVIDIA GeForce GTX TITAN Black.

The simulation operating in Unity works in real-time. The specific number of frames per second or the rendering time is written in Table 5.1. On the other hand, the render of a single frame in Octane might take considerably longer as the number of objects increases. It is mainly affected by the number of samples per pixel, resolution and whether adaptive sampling is used. The configuration of the Octane rendering kernel is (the rest are left at default values):

- Type: path-tracing kernel
- Samples: 200 (adaptive sampling at 85)
- Diffuse/specular/scatter depth: 8/24/8
- Ray epsilon: 0.0001
- Filter size: 1.0
- GI clamp: 1.0
- Path termination power: 0.2
- Noise threshold: 0.02

The Octane Render plugin enables only one GPU for calculation. However, the plugin allows exporting the whole scene into an ORBX file, which is possible to render in the Octane standalone version utilising more GPUs. The main bottleneck is the exporting of objects generated during the runtime of Unity into the scene graph of Octane Render, which takes part before the ray-tracing and might take as much time as the rendering itself. It takes an average of 3-5 minutes for the configuration to get one frame rendered (excluding the export); therefore, it is used for offline sequences only.

5.4 Problems and Limitations

There are still a few unresolved problems in the adopted methods of visual simulation. Some limitations were already mentioned in the performance section above. The perplexing part is integrating the rasterisation methods into the ray-tracer without completely changing the simulation already done in the game engine.

The wave simulation cannot be applied effectively for the ray-tracer without adding more workload toward the exporting part into the scene graph inside Octane. There is no clear way to detect and update the correct part of the road where a car drives through a puddle and change the material's normal map. The stirring waves near tires are turned off during ray-tracing, causing visual incongruity.

Another problem is assembling a plausible PBR water material to render the splashes and raindrops in Octane. The present particles are flat baked rectangles without the volume to accurately render complex water interaction. As a result, the splashes caused by cars might seem slightly plastic in some angles of the directional light.

A motivation to find a better way to update the generated particles also arises, so the final ORBX file is compact in terms of filesize. For example, a 300 frames long sequence of a scene with five cars takes up about 0.7 GB, which will swiftly increase with the number of cars approaching a hundred.

	scene 1	scene 2	scene 3	scene 4	scene 5
rain intensity	-	-	light	moderate	heavy
cars in scene	1	5	1	1	1
# of triangles	571.5k	1.5M	569.7k	572.2k	574.8k
# of particles	1575.6	16797.6	2045.4	2732.8	4044.8
# of rain particles	0	0	244.8	1142.6	2795.4
Unity FPS	140.44	72.98	131.16	126.94	110.28
Render time (ms)	7.12	13.78	7.68	7.9	9.08
1 frame (Octane)					
ORBX export time (s)	16.52	57.62	20.35	20.55	21.33
ORBX size (MB)	443	467	446	447	451
Scene size (MB)	1828	1872	1833	1834	1836
ray-tracing time (min)	3:56	3:33	3:43	3:48	4:05
300 frames (Octane)					
ORBX export time (min)	40:31	427:25	42:21	92:21	95:33
ORBX size (MB)	498	710	591	924	1637

Table 5.1: The splash particle count depends on whether the car wheels encounter a puddle area or how fast the cars drive. The rain particles (raindrops and their splashes) are recorded separately from the total number of active particles. The numbers have been averaged over five independent measurements. Note that the exporting time of an ORBX file for one frame is recorded in seconds and for the three hundred frames in minutes.

Chapter 6

Conclusion

I have studied and implemented a few rain-related phenomena using the specified tools. The results resemble effects observed in real life, although the proper physical simulation was simplified in some cases. For example, empirical models were used to form puddles on the road and shape splashes behind car wheels. Besides, the accomplished work reveals the troubles of simultaneously engaging both rasteriser and ray-tracer engines to provide the best outputs. I have demonstrated that it is possible to have somewhat of a smooth workflow between these two engines.

6.1 Work Summary

The initial research aimed to understand multiple rain phenomena seen in real-world events. The analysis emphasised rain effects which could prove helpful to simulate in outdoor scenes for traffic simulators. The findings were then witnessed in several racing games while taking into account the visual quality of the rain. Many effects simulated in those games turned out to be quite compelling to have in the traffic simulator and were included in the design. Existing works for simulating rain and its related effects were discussed, and they further assisted in developing the visual simulation of rain. A set of methods were proposed which could enhance visual heterogeneity by rendering rain effects on roadways.

The main focus was the phenomena seen on the rough surfaces made of asphalt or concrete. First, a changing road surface matching the intensity of rain and controlled by two variables was implemented. Furthermore, the scattered puddles in the streets react dynamically to driving cars, conforming to the assessment model that predicts splash potential for numerous road types and rainfall. Subsequently, precipitation and its impact on the scene were added, significantly influencing the overall synthetic wet environment. The precipitation occurs in three stages of intensity, each with a pre-defined amount of particles fitting the simulation space of the rain container.

A simple test scene provided quick visual feedback to calibrate the anticipated water materials of raindrops and splashes. The scene additionally enabled to promptly evaluate the results obtained either by rasterisation or ray-tracing.

■ 6.2 Future Work

The future objective might involve integrating fog and attenuation of visibility in general. The degraded perceptibility brings forward yet another hindrance for the drivers, which dramatically affects their steering performance. The attenuation model should not overlook the distribution of raindrops and be in correlation with the rain intensity. Also, the configuration of rainfall intensity should be revisited to grasp the continuously changing raindrop amount, including raindrop count and rate.

A notable part of the following actions will likewise address the imperfections emerging from the fundamental problem in bridging the gap between a rasteriser and a ray-tracer. Optimisation and other improvements should be studied concerning the problems mentioned in the earlier chapter.

Appendix A

Bibliography

- [3DC14] 3DCarShows. Johannesburg roads flooded. https://3d-car-shows.com/wp-content/uploads/2014/02/Johannesburg_Roads_Flooded.jpg, 2014. Accessed: 2021-05-02.
- [Bar20] Jan Barani. Rain rate intensity classification, Oct 2020.
- [BC87] Kenneth V. Beard and Catherine Chuang. A New Model for the Equilibrium Shape of Raindrops. *Journal of the Atmospheric Sciences*, 44(11):1509–1524, 06 1987.
- [BC02] Bert Blocken and Jan Carmeliet. Spatial and temporal distribution of driving rain on a low-rise building. *Wind & Structures*, 5:441–462, 10 2002.
- [BS12] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, volume 2012, pages 1–7. vol. 2012, 2012.
- [BWB⁺99] Max Born, Emil Wolf, A. B. Bhatia, P. C. Clemmow, D. Gabor, A. R. Stokes, A. M. Taylor, P. A. Wayman, and W. L. Wilcock. *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*, pages 40–43. Cambridge University Press, 7 edition, 1999.
- [CP13] Carles Creus and Gustavo A. Patow. R4: Realistic rain rendering in realtime. *Computers & Graphics*, 37(1):33 – 40, 2013.
- [Cre] Creativity103.com. A rainy day on the roads. http://cr103.com/collections/Transport/slides/wetcars_DSC1553~0.html. Accessed: 2021-05-02.
- [CS12] Barré-Brisebois Colin and Hill Stephen. Blending in detail. <https://blog.selfshadow.com/publications/blending-in-detail/>, 2012. Accessed: 2021-04-19.
- [CWZ⁺08] Wang Changbo, Zhangye Wang, Xin Zhang, Lei Huang, Zhiliang Yang, and Qunsheng Peng. Real-time modeling and rendering of raining scenes. *The Visual Computer*, 24:605–616, 01 2008.
- [eui19] euioefhigel. Weekly photo-soft. <https://charleschen9307.wordpress.com/2012/11/14/weekly-photo-soft/>, 2019. Accessed: 2020-08-17.

- [Fah14] Mike Fahey. Driveclub’s dynamic weather update completely changes the game. <https://www.kotaku.co.uk/2014/12/09/driveclubs-dynamic-weather-update-completely-changes-game>, 2014. Accessed: 2020-05-20.
- [FVD14] G. Flintsch, H. Viner, and A. Dunford. Splash and spray and its impact on drivers. Technical report, International Safer Roads Conference, 2014.
- [FWGV12] Gerardo W. Flintsch, Brian Williams, Ronald Gibbons, and Helen Viner. Assessment of impact of splash and spray on road users: Results of controlled experiment. *Transportation Research Record*, 2306(1):151–160, 2012.
- [GK49] Ross Gunn and Gilbert D. Kinzer. The Terminal Velocity of Fall for Water Droplets in Stagnant Air. *Journal of Meteorology*, 6(4):243–248, 08 1949.
- [GKN07] K. Garg, G. Krishnan, and S.K. Nayar. Material Based Splashing of Water Drops. In *Proceedings of Eurographics Symposium on Rendering*, Jun 2007.
- [GN06] Kshitiz Garg and Shree Nayar. Photorealistic rendering of rain streaks. *ACM Trans. Graph.*, 25:996–1002, 07 2006.
- [GN07] Kshitiz Garg and Shree Nayar. Vision and rain. *International Journal of Computer Vision*, 75:3–27, 07 2007.
- [Gom00] Miguel Gomez. Interactive simulation of water surfaces. In Mark DeLoura, editor, *Game Programming Gems*, pages 187–194. Charles River Media, 2000.
- [Hem16] Nico Hempe. *Bridging the Gap between Rendering and Simulation Frameworks Concepts, Approaches and Applications for Modern Multi-Domain VR Simulation Systems*. Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg, 2016.
- [Hob10] Bernie Hobbs. Darker when wet. <http://www.abc.net.au/science/articles/2010/11/11/3063513.htm>, 2010. Accessed: 2020-08-03.
- [JLD99] Henrik Wann Jensen, Justin Legakis, and Julie Dorsey. Rendering of wet materials. In Dani Lischinski and Greg Ward Larson, editors, *Rendering Techniques’ 99*, pages 273–281, Vienna, 1999. Springer Vienna.
- [Kel45] VN Kelkar. Size of raindrops. In *Proceedings of the Indian Academy of Sciences-Section A*, volume 22, pages 394–399. Springer India, 1945.
- [KG16] Kerstin Koch and Roland Grichnik. Influence of surface structure and chemistry on water droplet splashing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374:03, 08 2016.
- [Lag12] Sébastien Lagarde. Water drop 1 – observe rainy world. <https://seblagarde.wordpress.com/2012/12/10/observe-rainy-world/>, 2012. Accessed: 2020-05-21.
- [LD88] J. Lekner and M. C. Dorf. Why some things are darker when wet. *Applied optics*, 27 7:1278–80, 1988.

- GB3ew_ZdR3G2dVp_PMZ0kQ?iid=UGo9VGf2R1K0EeYy_qbBlg, 2018. Accessed: 2020-06-29.
- [Sab14] John Sabol. Dynamic weather comes to driveclub. <https://www.isrtv.com/playstation-4/dynamic-weather-comes-driveclub/>, 2014. Accessed: 2020-05-20.
- [Smi14] Bill Smith. Driveclub uses real world weather physics, weather effects compared with forza horizon 2. <https://gamingbolt.com/driveclub-uses-real-world-weather-physics-weather-effects-compared-with-forza-horizon-2>, 2014. Accessed: 2020-05-20.
- [Spi48] A. F. Spilhaus. Raindrop size, shape and falling speed. *Journal of Meteorology*, 5(3):108–110, 06 1948.
- [SRNN05] Bo Sun, Ravi Ramamoorthi, Srinivasa G. Narasimhan, and Shree K. Nayar. A practical analytic single scattering model for real time rendering. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, page 1040–1049, New York, NY, USA, 2005. Association for Computing Machinery.
- [SSR⁺07] Bo Sun, Kalyan Sunkavalli, Ravi Ramamoorthi, Peter Belhumeur, and Shree Nayar. Time-varying brdfs. *IEEE transactions on visualization and computer graphics*, 13:595–609, 06 2007.
- [Tar07] Sarah Tariq. Rain. Nvidia whitepaper, NVIDIA Corporation, 2701 San Tomas Expressway Santa Clara, CA 95050, 2007. <http://developer.download.nvidia.com/SDK/10/direct3d/Source/rain/doc/RainSDKWhitePaper.pdf>.
- [Tat06] Natalya Tatarchuk. Artist-directable real-time rain rendering in city environments. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, page 23–64, New York, NY, USA, 2006. Association for Computing Machinery.
- [Tes01] Jerry Tessendorf. Simulating ocean water. *SIG-GRAPH'99 Course Note*, pages 30–35, 01 2001.
- [Thr17] Throneful. Project cars 2 - rain gameplay (pc hd) [1080p60fps]. <https://www.youtube.com/watch?v=qvznlmdwsW4>, 2017. Accessed: 2020-06-29.
- [Unr12] Epic Games, Inc. Unreal Developer Network. Material basics - detail normal map. <https://docs.unrealengine.com/udk/Three/MaterialBasics.html>, 2012. Accessed: 2021-04-19.
- [WJGG15] Yoann Weber, Vincent Jolivet, Guillaume Gilet, and Djamchid Ghazanfarpour. A multiscale model for rain rendering in real-time. *Comput. Graph.*, 50(C):61–70, August 2015.
- [WSH⁺78] David H. Weir, Jay F. Strange, Robert K. Heffley, et al. Reduction of adverse aerodynamic effects of large trucks, volume 1. Technical report, United States. Federal Highway Administration, 1978.

- [WW04] Niniane Wang and Bretton Wade. Rendering falling rain and snow. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, page 14, New York, NY, USA, 2004. Association for Computing Machinery.
- [WW07] Andrea Weidlich and Alexander Wilkie. Arbitrarily layered micro-facet surfaces. In *GRAPHITE 2007*, pages 171–178. ACM, December 2007.
- [ZV07] Hao Zhang and Kenneth Voss. Bidirectional reflectance study on dry, wet, and submerged particulate layers: Effects of pore liquid refractive index and translucent particle concentrations. *Applied optics*, 45:8753–63, 01 2007.



Appendix B

Outputs



Figure B.1: (Unity) Dry road



Figure B.2: (Unity) Wet road



Figure B.3: (Octane) Dry road

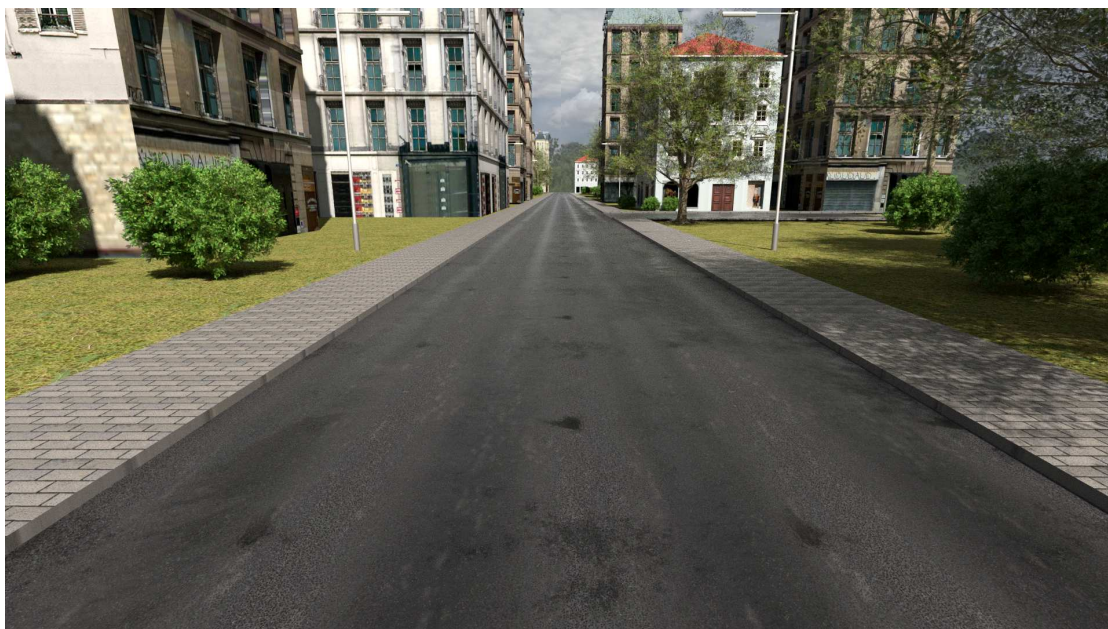


Figure B.4: (Octane) Wet road



Figure B.5: (Unity) Wet road with shallow puddles



Figure B.6: (Unity) Wet road with puddles



Figure B.7: (Octane) Wet road with shallow puddles

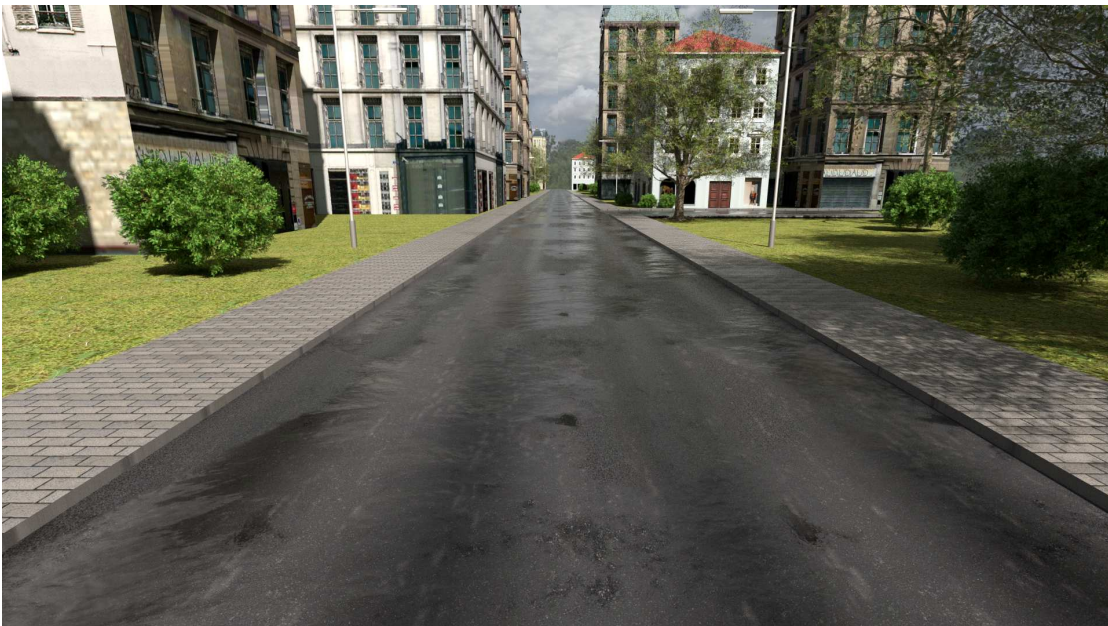


Figure B.8: (Octane) Wet road with puddles



Figure B.9: (Unity) Cars driving on the wet road



Figure B.10: (Unity) Cars driving on the wet road (side)



Figure B.11: (Octane) Cars driving on the wet road



Figure B.12: (Octane) Cars driving on the wet road (side)



Figure B.13: (Unity) Cars driving in rain



Figure B.14: (Octane) Cars driving in rain



Figure B.15: (Unity) Rainfall

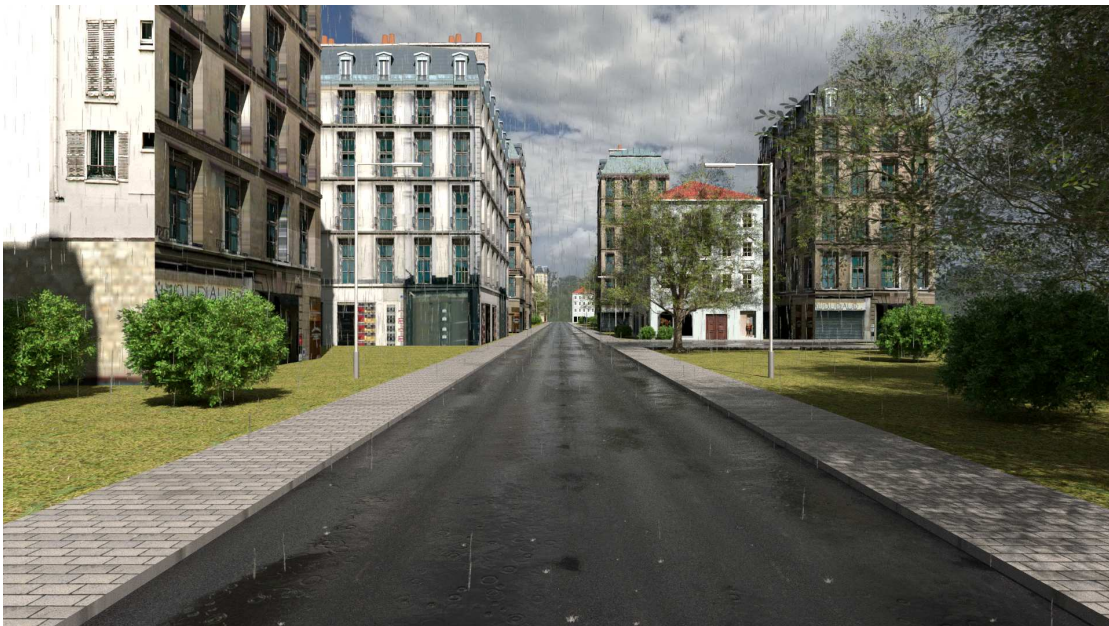


Figure B.16: (Octane) Rainfall

Appendix C

List of File Attachments

The project consists of a written part, both in PDF and the source code of Latex (`pdf`, `latex`). Included are also source files of given projects created in Substance Designer and Unity with the Octane plugin (`src`). In addition, appendices such as rendered images and videos are attached in respective directories (`img`, `vid`). Testing scenarios and other materials are added as well (`usr`).

The structure of the directory is as follows:

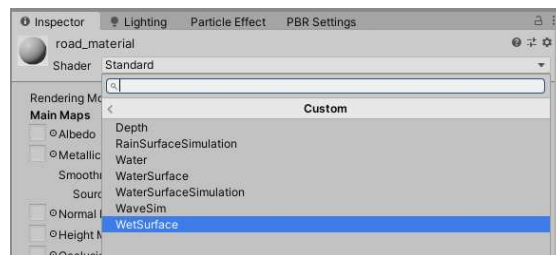
```
.
├── DP_Nguyenova_Giang_Chau_2021.pdf
├── README.txt
├── src/
│   ├── UnityProjects.zip
│   └── SDProjects.zip
├── usr/
│   ├── Scenario.txt
│   ├── TestMaterials/
│   └── Logs/
├── img/
│   ├── Cars/
│   ├── Rain/
│   └── WetRoads/
├── vid/
└── latex/
```


Appendix D

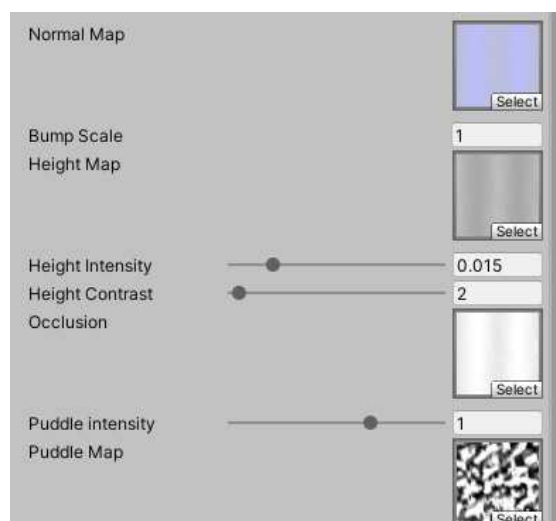
User Manual

D.1 Creating a Wet Road Material

To create a configurable wet road material in Unity, choose the custom shader *Wet Surface* as a shader for a newly created material.

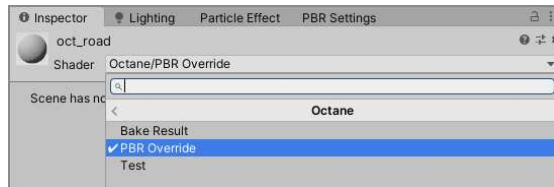


Set references to textures for each property in the custom shader and apply the material on a road mesh.

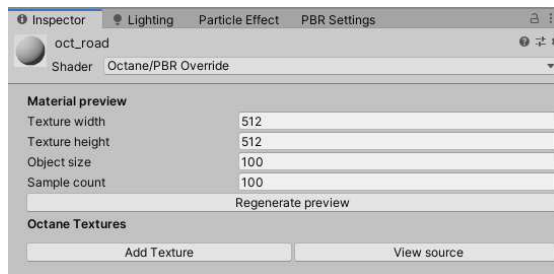


D.2 Creating a PBR Override Wet Road Material

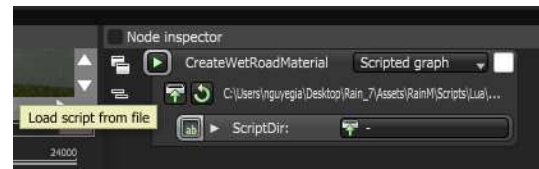
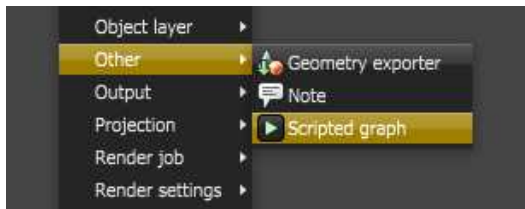
Load OctaneRender plugin into Unity and include a PBR Render Target into the scene. Choose the shader *Octane/PBR Override* as a shader for a newly created material. Apply the yet empty material on a road mesh and render the scene once.



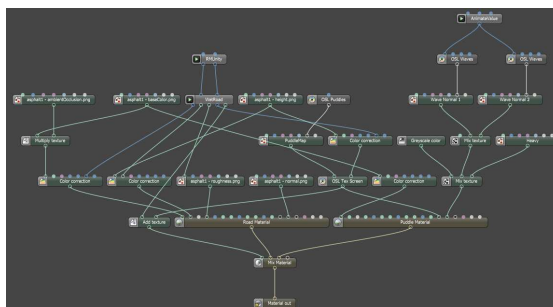
Press the *View source* button at the bottom of the material inspector. A nodegraph editor will open.



Next, create a *Scripted graph* node and from the node inspector load a LUA script from directory: `'../Rain_9/Assets/RainM/Scripts/Lua'` named *CreateWetRoadMaterial.lua*.



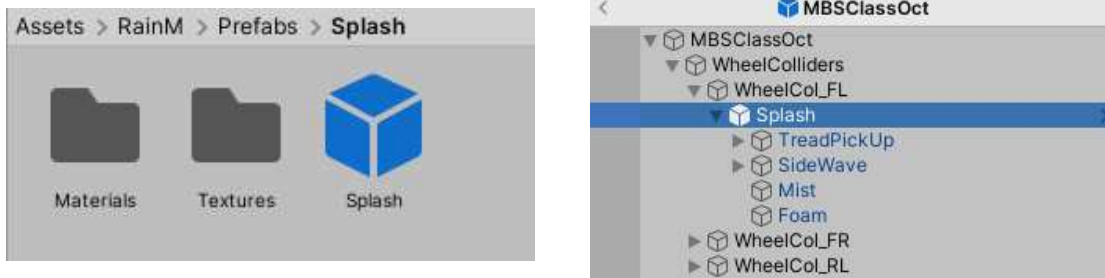
Execute the script, and a node graph for a road material should appear. For certain nodes (RGB Texture), set the textures representing the material. Afterwards, delete the *Scripted graph* node.



D.3 Adjusting Car models

The car wheels need a splash particle system attached to drive on wet roads correctly. Each car model might be slightly different; therefore, the scripts do not set this automatically. The cars themselves should be prefabs, so luckily this operation has to be performed only once.

From the *Prefabs* folder, take the *Splash* game object and nest it inside the *Wheel Collider*.



D.4 Rain Manager

The rain manager controls the weather (the amount of wetness in the scene). Simply drag and drop the rain manager prefab from the *Prefabs* folder somewhere into the scene and reset its transformation.

The following properties of the Rain Manager can be changed:

- **Rain Intensity** The dropdown menu allows 4 types of rainfall: *none*, *light*, *moderate* and *heavy*.
- **Wetness** Float number between 0 and 1 indicating the wetness on the road.
- **Water Level** Float number between 0 and 1 indicating how much are puddles filled.
- **Water Surface Simulation** Enable/disable the water simulation. If turned off, cars will not create ripples and splashes while steering into puddles.
- **Puddle Intensity** Manually controls the reflection intensity (the emission property) of road materials.
- **Wave Displacement** Controls the reflection offset, which simulates the strength of waves on puddle surfaces.
- **Wave Scale** Controls the size of waves.
- **Wave Speed** Controls the direction and speed of moving waves (simulating wind).

To switch between Unity and Octane, manually set the needed references for the component (different materials are used). Otherwise, leave it intact. If OctaneRender is chosen, tick the *Octane Render* box, designating the baking of particles.