# Rendering Wet Roads in Driving Simulations

Giang Chau Nguyenová[*]
*Supervised by: Jiří Bittner*[†]

Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University
Prague / Czech Republic

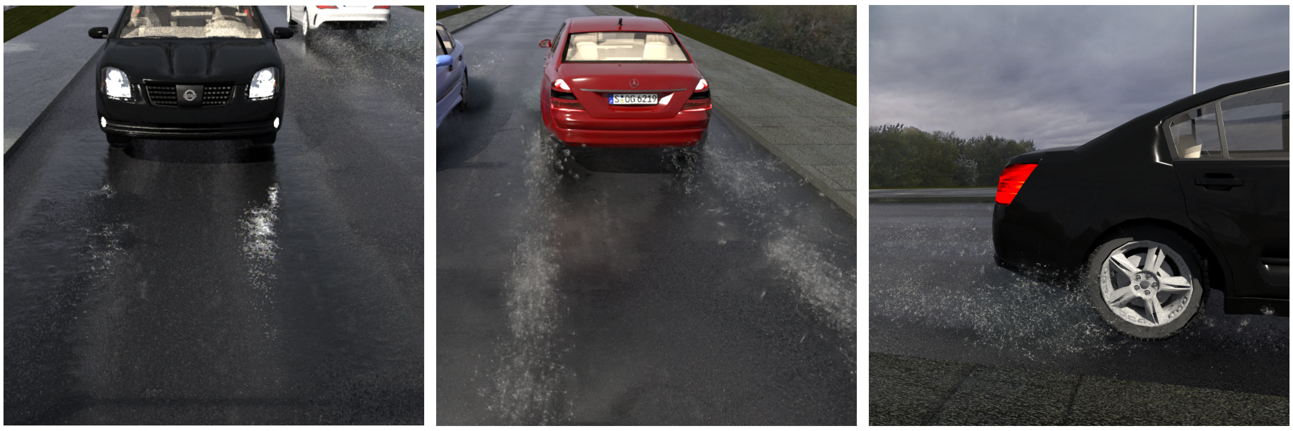


Figure 1: Images of the wet road simulation acquired via ray-tracing. The figure on the left displays a wet road flooded with shallow puddles. The car's headlights reflect on the puddle surface. Splashes of water projected from the tires can be seen in the central figure. The last figure shows a close-up of the splash.

## Abstract

Rain and its impact can significantly enhance the realism and credibility of outdoor scene rendering. It further attributes to the environment's diversity by breaking the repeating patterns of a synthetic world. We describe our approach to visual simulation of the rain phenomena in driving simulations. We focus on varying intensity of the rain effects (e.g. drizzle or heavy rain) seen on roads with driving cars. The simulation includes adjustable wetness of the roads with puddle regions. When a car drives through a puddle, splashes are generated according to an assessment model. The puddles also react to contact with the tires, creating tiny wavy movements based on the wave equation. The simulation is implemented in a game engine, and we propose a way to render high-quality ray-traced outputs using an external ray tracer.

**Keywords:** wet roads, rain simulation, traffic simulator, outdoor environment, game engine, Octane renderer

[*]gcngia@gmail.com
[†]bittner@fel.cvut.cz

## 1 Introduction

Simulating various weather conditions can enhance the environment's diversity, especially for synthetic outdoor scenes made for driving simulations. Of these natural phenomena, rain is probably most frequently seen. Rain is not defined only by raindrops. The surrounding area after it has stopped raining is also part of the occurrence. A believable rain simulation has to take into consideration numerous visual effects that involve complex physical mechanisms. A rainy environment consists of raindrops, puddles, splashes, ripples, fog, light glows, and even rainbows. The number of minor details that need to be simulated exceeds current computational capabilities and rendering these effects is challenging. For real-time applications, it is necessary to limit the simulation and use approximations instead.

We present and combine a set of rain-related phenomena that mainly affect the roads and pavements to create a realistic wet environment in driving simulations. The implementation is made in a game engine with the use of offline ray-tracing methods.

## 2 Related work

Numerous proposals for simulating rain and its effects have been made and developed. The techniques can be classified into either rainfall simulation or visualisation of the rain's impact on the environment. Many introduced works focus on combining both to achieve more comprehensive situations.

Tatarchuk [13] used multiple layers of textures to create the illusion of countless raindrops in the Toyshop Demo. The demo also incorporates raindrop splashes, ripples in puddles, lighting and even water droplet animation on a glass surface. In Chango et al. [2], a light source in the scene determines the raindrop's appearance. Countless tiny raindrops in the air form fog, and when the density of rain decreases, a rainbow is made. Multiple rendering passes are performed to render the wet ground, including reflection, refraction, environment mapping, and ripples in the puddle. Creus and Patow [3] proposed a rain rendering algorithm that creates particles in the scene using an artist-defined storm distribution. Their method utilises rain particles and a texture atlas (using Garg's and Nayar's database [5]). Phenomena such as fog, halos, and light glow are added before the rendering of streaks and splashes.

We chose to combine several rain-related phenomena mentioned in those papers above. Our work focuses on the rendering of road surfaces under wet conditions explicitly for driving simulators discussed by Nakamae et al. [9]. The influence of water on various materials is drawn from the work of Jensen et al. [7]. They combine a reflection model for surface water with subsurface scattering to replicate the liquid on the surface and inside the material.

Some other works concentrate on the precipitation itself; for example, Rousseau et al. [12] reproduce the optical properties of a raindrop and its interaction with light. Optimisation of repeatedly computed particles has been studied by Puig-Centelles et al. [11]. The volume in which the rain simulation occurs provides a continuous transition between rainy and non-rainy areas and even supports some level of detail. Weber et al. [14] focus on each rain streak's shape and size. Later they aimed at rendering complex interactions between trees and rain in real-time [15]. Although we do not present any simulation method for rainfall, these articles expose the potential future approach to enhance the rainy environment visually.

## 3 Impact of Water on Roads

The wet environment noticeably influences the appearance of an asphalt road. This rough and porous material looks different due to the water layer on top of the surface and the absorbed wetness beneath. The presence of water on the surface increases specularity and decreases the albedo.

### 3.1 Wet Roads

The leading cause for the darkening is the internal reflection at the boundary of air and water [7]. To correctly simulate the thin water layer behaviour, we would need a layered BRDF – layers for the water and the original road material. The model used by Jensen et al. [7] takes into account the interaction of light with both the air-liquid interface and the liquid-material interface (fig. 2). This method could be somewhat costly, especially if we wish for a dynamic transition between the road's dry and wet state. For a driving simulator, we used a simple set of BRDF parameters capturing both dry and wet surfaces to imitate these effects visually.

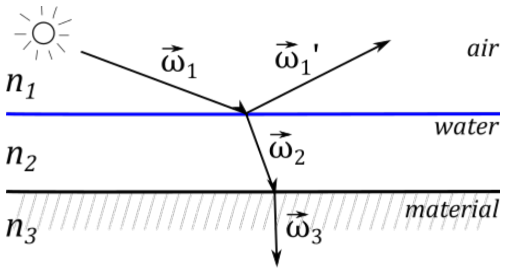


Figure 2: Jensen et al. [7] use a two-layer reflection model to simulate a thin liquid film on top of a material. The amount of light transmitted through each layer can be computed using Fresnel's equation.

As noted in Lekner and Dorf [8], the darkening effect by wetness is more significant if the albedo is low. When the absorption is strong (with rough dark porous materials), a larger fraction of the light is consumed on the first contact with the surface. They described a connection between wet and dry albedo based solely on the index of refraction of the water and the road material. However, it would be helpful to have the scaling factors for diffuse and specularity 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.

We chose to alter the parameters of a BRDF model as proposed by Nakamae et al. [9]. They use the Cook-Torrance model, and 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. These parameters were empirically calibrated without considering the physical properties of the surface. We readjusted these numbers to our scenes within the driving simulation made in the game engine (with a Torrance-Sparrow model). In particular, for a completely wet road effect, we multiply its dry-state albedo by 0.3 and its smoothness by 2.5. These values were found experimentally by visually evaluating the appearance of the wet and dry roads. The partially wet road is achieved by linearly interpolating the coefficients mentioned above towards one based on the relative wetness of the road (see section 5.1 for more details).

## 3.2 Puddles

Another situation is when rainwater is present on the surface of a material, but instead of getting absorbed, it gathers in low areas creating puddles. Nakamae et al. [9] divide the road surface into four types: dry, wet, drenched, and a puddle region. The wet region is the type we described above. The drenched regions contain some water, but no puddles are formed. The authors constructed a two-layer reflection model for the puddle region to simulate a thin layer of water covering the road. Regions are defined using several height data and height thresholds for the classification. This data allows the creation of puddles with different depth in each region, taking into account the locality and scale of the road surface's undulations.

## 4 Dynamic Simulation

A static wet surface covered with puddles and cars driving through without any interaction does not provide a realistic impression. 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. In our approach, we split these effects 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.

### 4.1 Water Surface and Ripples

To simulate the interactive wave propagation, we use a heightfield to represent the water surface. 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. This is not a significant disadvantage to the shallow puddles occurring on the roadways.

The waves are represented by a wave equation which is based on the propagation principle of mechanical waves. We use a 2D partial differential equation to simulate the liquid surface waves [6]:

$$\nabla^2 f(\mathbf{x},t) = \frac{1}{s^2}\frac{\partial^2 f(\mathbf{x},t)}{\partial t^2} \tag{1}$$

where $s$ is the velocity of the wave spreading across the surface, $\nabla^2$ is the Laplacian operator, $f$ is the vertical derivation function with parameters $\mathbf{x}$, a spatial vector (in this case a two-component vector) and $t$ as time. The discretization with finite-difference methods can be performed using a 2D map $z_{i,j} = f(\mathbf{x})$, with $i,j \in [0, N-1]$, and $N$ describing the width of our square grid (see figure 3). With the central difference in space and time, the equation is approximated to (eq. 2):

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

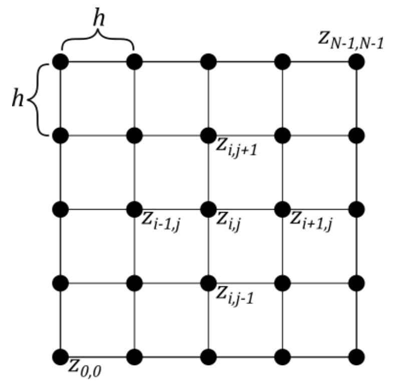


Figure 3: A heightfield with N points along each side used to approximate the water surface.

with $h$ being the size of a single step in the grid and $t$ denoting a relative frame number. Hence, 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 will grow exponentially [6]. 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 adhering to some terrain features.

### 4.2 Splash and Spray

To add splashes into our simulation, we use an assessment tool predicting splash potential for numerous road types, and rainfall by Flintsch et al. [4]. This method primarily provides useful information for supporting highway design. They started by developing a model for water film depth (WFD) on a surface based on its drainage properties and rain intensity. From this, they established a model for estimating the amount of water that is going to be projected by the wheel, given the WFD model, road properties, vehicle speed, and other factors.

The generation of water splashes and sprays is a complex process, and it depends upon several independent situational variables. Usually, splashes and sprays are two separate processes characterized by the droplet's size (splashes consist of drops larger than 1 mm and sprays are formed by droplets smaller than 0.5 mm in diameter). Nevertheless, they are often referred together for simplicity because it is challenging to monitor and measure them individually (labelled plainly as splash). A splash can be deconstructed into four mechanisms: bow waves, side waves, tread pickup, and capillary adhesion (Weir et al. [16]). The bow and side wave consists of relatively large drops while tread pickup and the capillary drips are shattered into spray (see fig. 4). The amount of water thrown also depends on the vehicle speed. Previous research (Flintsch et al. [4], Pilkington [10]) 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.

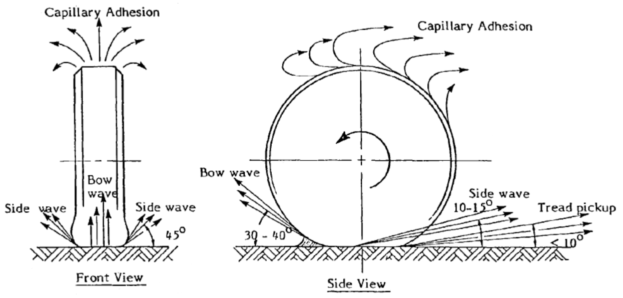


Figure 4: Primary splash and spray generation mechanism taken from Weir et al. [16]. Water passing into the tire's tread pattern can be projected directly behind the wheel (tread pickup) or lingers in as a thin capillary film. Bow and side waves are more notable as the wheel drives into a deeper puddle.

The model by Flintsch et al. [4] determines the maximum amount of water ($M_w$) available for splash and spray according to the equation 3:

$$M_w = v \cdot b \cdot \mathrm{WFD} \cdot \rho_w \tag{3}$$

where $v$ is the car's velocity (m/s), $b$ is tire width (m), WFD is water film depth (m), and $\rho_w$ is water density. They further calculate the amount of water for each mechanism separately (for example if the water from capillary adhesion remains, it will be used for tread pickup). We believe that this level of detail is not crucial for our visual simulation, mainly because we would need to control additional variables. One such variable is the aerodynamics of a car or the tire geometry; the splash behaves differently for a worn-out tire as opposed to new tires with a clear tread pattern.

## 5 Implementation

The implementation is done in the Unity game engine. The state of the road can be changed dynamically with a few parameters defining the amount of water present. Some adjustments had been made in our approach to ray-trace viable images in Octane Render.

### 5.1 Appearance of Wet Roads and Puddles

We believe that the physically-based BRDF of Unity is derived from the Disney work [1] and based on the Torrance-Sparrow microfacet model. We control the wet condition of a road by a variable named *wetness*, a uniform variable of Unity's surface shader. The variable is the alpha for the linear interpolation between the two values of coefficients modifying the albedo and smoothness, as shown below:

```
albedo *= lerp(1.0, 0.25, wetness)
smoothness = min(1.0, smoothness *
 lerp(1.0, 2.5, wetness))
```

It defines the drenching strength caused by rain, allowing the control of progressive damping or drying of the road.

We further used the approach of Nakamae et al. [9] to create the puddle areas on the road. We omitted the drenched region in our approach and instead interpolate the current model (dry or wet road) with the puddle regions 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. Depending on the heightmap form, the water might fill smaller gaps and cracks in the road. A large greyscale texture represents the placement of puddles itself called a *puddle map* to break the repetition of puddle patterns on a tiled road (which occurs if only the height information is used). This puddle map texture is locally blended with the heightmap, so the puddles reflect the road's uneven shape (in figure 5, we demonstrate such a blending method). We control the amount of water in the puddle areas 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. The amount of water regulates the puddle's intensity and could be considered the intermediate condition between wet and puddle region. It is possible to have another water level variable to adjust the flooding or drying rate for small holes or cracks in contrast to the 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. Note that the puddles are still altogether merely a flat surface. These puddles remain featureless and straight like a mirror surface. We added ambient waves by mixing two normal maps that are shifted and scaled in time using the Unity game time variable. Lastly, planar reflections are incorporated to create an impression of real-time water surface using a render texture. The texture is distorted together with the animated bump maps making these reflections moving with the waves.

### 5.2 Interactive Puddles

The wave effect simulated in the 2D array is created in the engine using a custom render texture. The calculation itself is handled in a fragment shader with a double-buffered texture. This texture-based viewpoint efficiently gives us the bounding conditions implicitly (the grid's edges are still). The last frame is stored in the red channel, while the frame before last is stored in the green channel. This

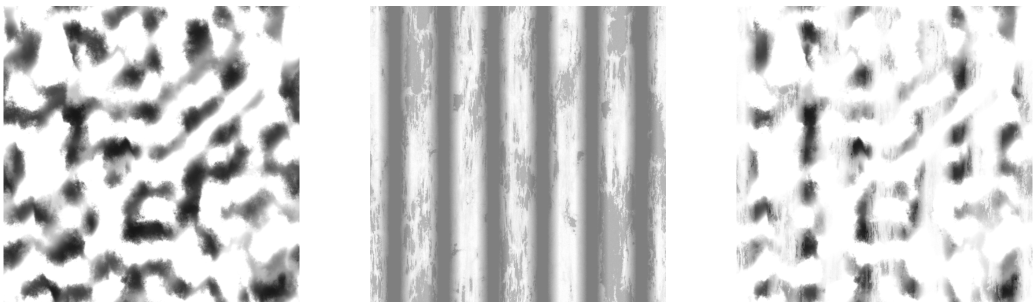


Figure 5: On the left is a puddle map generated from a Perlin noise, a figure in the middle displays the road's height texture (multiple lanes with rutted tire marks). On the right is the result of the screen blending of those two textures. Notice how the puddles slightly aligned to the lower areas of the heightmap.

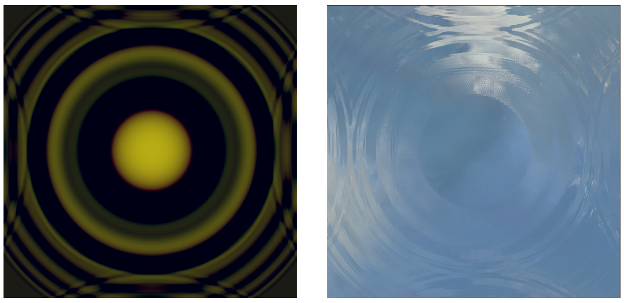


Figure 6: (left) A texture storing two frames of data (in R and G channel) regarding the water surface's height. This texture is used to calculate the height of the wave for the next frame. (right) The displaced water using a normal map generated from such heightfield.

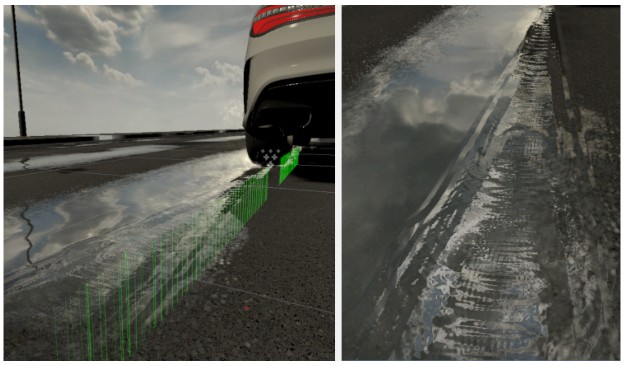


Figure 7: The green lines depict ray casts done at the contact points between the tire and the road. At each point, we disturb the water surface and start a wave at the calculated UV coordinates of the heightfield.

manner of storing the heights begets a black and yellow coloured texture shown in figure 6. Finally, this heightfield is passed as a wave texture to the custom shader for the roads, from which we generate a normal map and blend it with the already existing ambient waves.

To generate the actual wave at the wheel-puddle contact position, we detect where the wheel collides with the road and use ray cast to determine the heightfield texture's UV coordinates. We then set the height value to 1 at this contact point 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 7.

As for splashes, we replaced the empirical WFD model of Flintsch et al. [4] with our straightforward method for forming puddles on the road. We do not deal with the drainage properties. 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) we set. The depth of a puddle is calculated in millimetres with a maximum depth of 10 mm for the shallow puddles.

We apply the findings from Weir et al. [16] to the splash particle system provided by the game engine. We put together threshold values for each of the four splash component based on WFD and vehicle speed. The maximum amount of water (eq. 3) is then used to fine-tune the properties of each splash mechanism described above. We adjust the emission rate, shape, and angle of the water projection 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. We detect puddles 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.

## 5.3 Ray-Traced Outputs

Besides the rasterized version from the engine, we can also obtain path-traced images 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 rasterization can achieve. However, Octane render does not have complete access to the game engine's rasterization pipeline, so we had to make some adjustments to achieve visually pleasing results.

Octane does not recognize the custom shader since only the standard shaders of Unity are supported. Hence, we assembled a new road material as a PBR Override Material shader – the OctaneRender-specific material type. We chose a combination of different materials 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 shader (Open Shading Language) through a scripted graph node. The puddles need to be placed roughly at the same spot, similar to the rasterized outputs, because we calculate the splashes directly from the game engine (see fig. 8). 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.

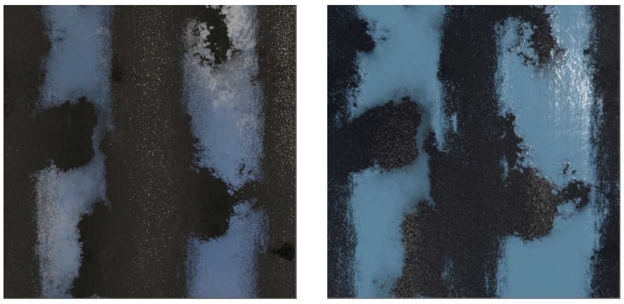


Figure 8: The puddles are mapped roughly the same way for both Unity (left) and Octane (right). We coloured the puddle areas in Octane blue for demonstration purposes (otherwise they are too subtle to notice with weak reflections).

Octane has no support for Unity's particle system either. We dealt with this drawback by baking the particles into a mesh, and we do so every frame. One wheel of a car can emit a maximum of 2000 particles to generate a splash. We pack those particles into four game objects, each with a single baked mesh. The splash particles from Unity are rendered as billboards, so we bake them faced to the render target camera (as seen in figure 9). We opted for the PBR Override materials for the splashes to get more realistic water behaviour.

## 6 Results

We evaluated the wet road simulation in the Unity version 2019.3.12f1 and Octane Render version 2020.1.5.

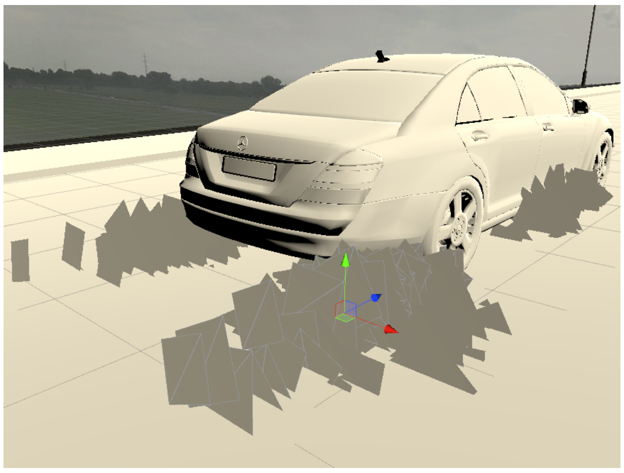


Figure 9: The splash particles are baked as textured billboards aligned to the render target camera. We replaced the standard Unity material for the Octane specific specular material.

We constructed a simple testing scene in the game engine composed of a long straight road with two lanes, and pavements on each side. There are four cars in total, each driving at a slightly different speed to showcase distinctive splashing amounts.

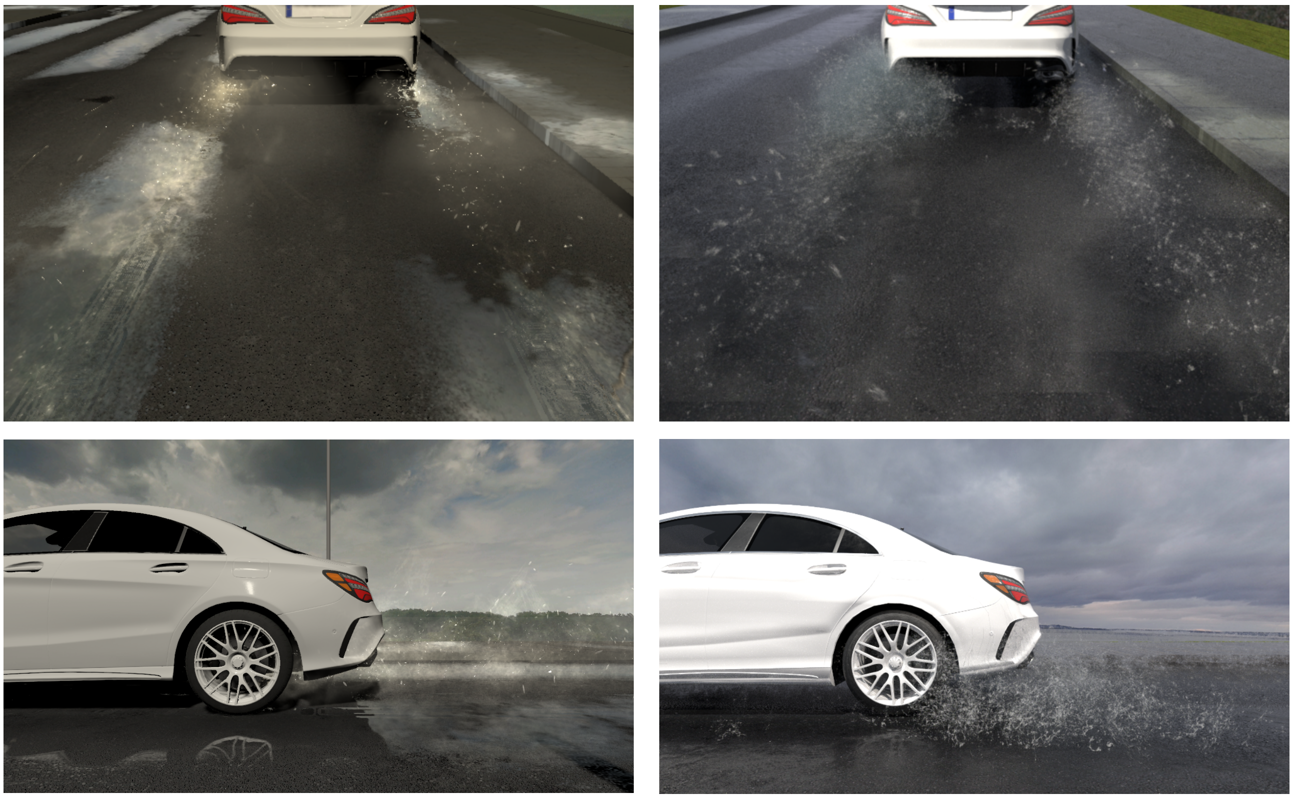
We set the resolution of both rasterized and ray-traced outputs to 1920x1080. For the ray-traced version from Octane, we render using a path-tracing kernel with 350 samples and denoise the image after the frame is finished rendering. The close-up visuals from Unity can be seen on the left while the outputs from Octane are on the right in figure 10.

### 6.1 Performance

The performance was observed on a computer with:

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

The simulation running in Unity works in real-time, with an average of 100 fps (9.8 ms). On the other hand, the render of a single frame in Octane might take considerably longer as the number of cars and splashes increases. It is mainly affected by the number of samples per pixel, resolution and whether adaptive sampling is used. The Octane Render plugin enables only one GPU for calculation. The plugin allows exporting the whole scene into an ORBX file, which is possible to render in the Octane standalone version utilizing 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 7-10 minutes for our configuration to get one frame rendered (excluding the export); therefore, we only use it for offline sequences.



(a) Unity (b) Octane

Figure 10: Images on the left show how the simulation looks in the Unity game engine, using only standard particle shaders to render splash. We put the ray-traced version from Octane on the right side for visual comparison.

### 6.2 Problems and limitations

There are still a few unresolved problems in our approach. Some limitations of the methods were already mentioned in the sections above. The tricky part is integrating the rasterization methods into the ray-tracer without completely changing the simulation already done in Unity. We cannot use our wave simulation effectively in the ray-tracer without adding more workload toward the exporting part into the scene graph inside Octane. We are also motivated to find a better way to update the generated particles, so the final ORBX file is compact in terms of filesize (300 frames long sequence of our scene takes up about 0.8 GB). Another problem is assembling a plausible PBR water material to render the splashes in Octane since our particles are flat objects without volume.

## 7 Conclusion

We proposed a set of methods that can enhance visual diversity by rendering rain effects on roadways. We focused only on the phenomena seen on the roads and other rough surfaces made of asphalt or concrete. We designed and implemented a varying road surface according to the intensity of rain, and we regulate the amount with two variables. Furthermore, the puddles on the road react to driving cars dynamically. Still, other wet conditions and details can further be implemented to construct a believable rainy environment. We think that precipitation or fog would benefit the overall visual quality of the artificial rainy environment. We plan to add these into our scene using the tools provided by the Unity game engine and the Octane render plugin. Our focus is also to address the imperfections emerging from the fundamental problem in bridging the gap between a rasterizer and a ray-tracer.

## References

[1] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, volume 2012, pages 1–7. vol. 2012, 2012.

[2] 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, 2008.

[3] Carles Creus and Gustavo A. Patow. R4: Realistic rain rendering in realtime. *Computers & Graphics*, 37(1):33 – 40, 2013.

[4] 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.

[5] Kshitiz Garg and Shree Nayar. Photorealistic rendering of rain streaks. *ACM Trans. Graph.*, 25:996–1002, 2006.

[6] Miguel Gomez. Interactive simulation of water surfaces. In Mark DeLoura, editor, *Game Programming Gems*, pages 187–194. Charles River Media, 2000.

[7] 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. Springer Vienna, 1999.

[8] J. Lekner and M. C. Dorf. Why some things are darker when wet. *Applied optics*, 27 7:1278–80, 1988.

[9] Eihachiro Nakamae, Kazufumi Kaneda, Takashi Okamoto, and Tomoyuki Nishita. A lighting model aiming at drive simulators. *SIGGRAPH Comput. Graph.*, 24(4):395–404, 09 1990.

[10] G. B. Pilkington. Splash and spray. *Surface Characteristics of Roadways: International Research and Technologies*, 1990.

[11] Anna Puig-Centelles, Oscar Ripolles, and Miguel Chover. Creation and control of rain in virtual environments. *The Visual Computer*, 25:1037–1052, 11 2009.

[12] Pierre Rousseau, Vincent Jolivet, and Djamchid Ghazanfarpour. Realistic real-time rain rendering. *Comput. Graph.*, 30:507–518, 2006.

[13] 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.

[14] 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.

[15] Yoann Weber, Vincent Jolivet, Guillaume Gilet, Kazuki Nanko, and Djamchid Ghazanfarpour. A Phenomenological Model for Throughfall Rendering in Real-time. *Computer Graphics Forum*, 2016.

[16] 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.