Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Master in Computing

Master Thesis

Real-time Realistic Rain Rendering

Student: Carles Creus López
Director: Gustavo Ariel Patow

Date: June 23, 2010

Abstract

Artistic outdoor filming and rendering need to choose specific weather conditions in order to properly trigger the audience reaction; for instance, rain, one of the most common conditions, is usually employed to transmit a sense of unrest. Synthetic methods to recreate weather are an important avenue to simplify and cheapen filming, but simulations are a challenging problem due to the variety of different phenomena that need to be computed. Rain alone involves raindrops, splashes on the ground, fog, clouds, lightnings, etc. We propose a new rain rendering algorithm that uses and extends present state of the art approaches in this field. The scope of our method is to achieve real-time renders of rain streaks and splashes on the ground, while considering complex illumination effects and allowing an artistic direction for the drops placement.

Our algorithm takes as input an artist-defined rain distribution and density, and then creates particles in the scene following these indications. No restrictions are imposed on the dimensions of the rain area, thus direct rendering approaches could rapidly overwhelm current computational capabilities with huge particle amounts. To solve this situation, we propose techniques that, in rendering time, adaptively sample the particles generated in order to only select the ones in the regions that really need to be simulated and rendered.

Particle simulation is executed entirely in the graphics hardware. The algorithm proceeds by placing the particles in their updated coordinates. It then checks whether a particle is falling as a rain streak, it has reached the ground and it is a splash or, finally, if it should be discarded because it has entered a solid object of the scene. Different rendering techniques are used for each case. Complex illumination parameters are computed for rain streaks to select textures matching them. These textures are generated in a preprocess step and realistically simulate light when interacting with the optical properties of the water drops.

Contents

List of Figures

Chapter 1

Introduction

One important aspect of artistic outdoor scene filming is the need to choose specific weather conditions in order to properly trigger the audience reaction. Depending on which mood the artist desires to transmit, different weather conditions must be used. For instance, sunny days are typically associated with alert and cheerful states, while cloudy and rainy conditions are felt more oppressive and may enhance a sense of unrest. Filming scenes with specific conditions may be laborious and expensive; furthermore, it is subject to the ungovernable atmospheric weather. For this reasons, synthetic methods to simulate weather are an important avenue to simplify the task and have already been widely used to produce visually appealing computerized graphics in films. These simulation techniques can be broadly classified into two groups: real-time and off-line. The former group is used in interactive applications like video-games and virtual reality while the latter produces more realistic results and is the one used in filming, where processing can take long times. In this document we will focus on rendering in real-time one of the most common weather conditions: rain.

Rain simulation is a challenging problem due to the variety of different phenomena that need to be computed, all of them with complex physical evolutions. These phenomena include raindrops, splashes on the ground, fog, clouds, lightnings, etc. The amount of small details and particles that need to be simulated rapidly overwhelm current computational capabilities. This forces real-time applications to limit the phenomena simulated and use approximations of their visualizations, while still striving to achieve realistic results.

1.1 Objectives

The essential goal of this project is to propose and explore a new approach for real-time rain rendering. One of our self-imposed constraints is to achieve realistic simulations comparable to state of the art implementations in this field. In order to do so, we plan to build our system around the work done in [Garg and Nayar, 2006]. In it, the authors have developed a database of rain streak textures capturing the drop's complex illumination effects. Realistic rendering employing these textures can be done in the manner of [Tariq, 2007], except that we seek greater simulation accuracy by using the complete database and its lighting parameters.

Raindrops will be animated using a particle system. We propose an approach different from the ones used this far that is both simpler and faster to animate, yet allowing us to perform various kinds of adaptive sampling of the regions that need to be simulated and rendered. Finally, interaction of the raindrops with the whole scene is also considered. This avoids drops penetrating geometry and it will also provoke splashes when collisions are detected, a phenomenon that gives good visual clues to identify rainy weather.

1.2 Document overview

The remaining of this document is structured as follows. First, in Chapter 2 we describe the various phenomena that comprise rain, giving tips on what simulation of each of them would imply. In Chapter 3 we detail the previous work done in this field, remarking some of the results that we use in our approach and how we extend them. Chapter 4 is devoted to our rain simulation proposal. We first introduce a brief summary of the whole algorithm and then we extend each of its parts, giving full detail of the steps that our approach performs. In Chapter 5 we show final results of the algorithm, along performance of our reference implementation. We also comment how it compares to the state of the art and list possible future work modifications to our current proposal. Appendix A lists a few studies related to raindrop's physics.

Chapter 2

Rain phenomena

Rain as a whole consists of a plethora of different phenomena, most of them related to water, how it moves and how it affects lighting. Since water in rain is present in a wide variety of forms, these phenomena require radically different simulation algorithms in order to adapt to each condition. Below we give a summary of what rain simulation implies and the difficulties it encounters, hinting some of the principal physical properties known about it.

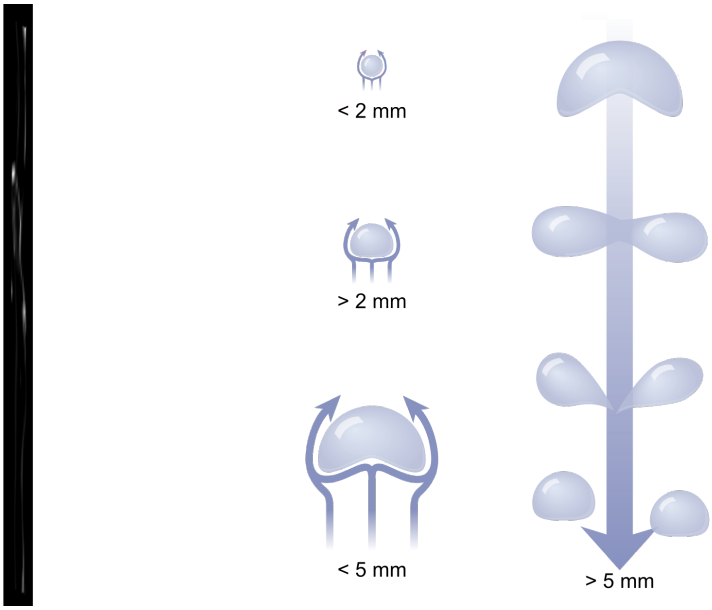
2.1 Rain streaks

Water drops falling are the most identifiable phenomenon in rain. When cloud's water vapor is condensed, it forms drops that, when they reach enough mass, fall to the ground. On the ground level, drops have already reached terminal velocity and, given a small area, their distribution can be considered random and uniform.

As shown in Figure 2.1(b), a drop's shape is elliptic. For small drops it is almost spheric, but as they get bigger aerodynamic forces deform their shape. Moreover, drops cannot get bigger than a specific threshold because their own motion through the air splits them. See Appendix A for a brief explanation of the main known physical properties of raindrops, along the classical references on this topic. The air also induces complex oscillations on the drops, further distorting their shape and creating complex optic effects. Pure raindrops are colorless but their reflections and refractions can be perceived with proper lighting conditions. Although these are the raindrops' intrinsic properties, human perception takes a more important role in the way they are seen: since images in the eye have some persistence and raindrops move relatively fast, falling raindrops are perceived as streaks (see Figure 2.1(a)). In a camera this phenomenon is called motion blur and is caused by its integration time, or exposure time.

2.2 Clouds

Cloud rendering is one of the most important aspects in order to correctly transmit the mood produced by rain. They are also highly complex systems with simulations that need to take into account many different effects. First of all, they are participating media where light suffers a high degree of scattering and absorption. The density of droplets suspended in the cloud determines the way light is distorted, and variations of this density may produce lighting conditions with a wide difference in coloring and attenuation, as in Figure 2.2. Moreover, simulations are hindered by the fact that light absorption is low, and thus convergence of the algorithms is very slow. Another problem is that the shape of clouds varies depending on their types, some being soft while others like cumulus are sharp. With the latter group, high resolution simulations are needed, increasing even further the computation times. Finally, wind influence must be taken into account to animate the shape of the cloud. These shape changes modify how light is scattered inside it, and thus the illumination simulation must be updated along the animation.



(a) Synthetic rain streak.

(b) Drop shape and maximum size due to aerodynamic forces.

Figure 2.1: The first image depicts a rain streak in front of a black background, taken from the simulation model of [Garg and Nayar, 2006]. Complex lighting effects due to the drop's oscillations are visible as white patterns. The second image shows the raindrop shape when it has different sizes and the approximated maximum size it is capable to achieve before splitting due to aerodynamic forces and surface tension.

Figure 2.2: Thunderstorm clouds over Chaparral, New Mexico.

2.3 Rainbows

When a wave changes between mediums with different propagation speeds, it suffers refraction. Furthermore, the angles at which the waves are refracted depend on their frequencies. These properties cause that, when rain is mild or has just finished, small droplets suspended in the air produce rainbows by refracting the sunlight: light first enters a droplet, reflects in the opposite side and exits the droplet again from the front face, refracting twice in total and causing sunlight's different colors to get separated. A scheme of this phenomenon is shown in Figure 2.3(a). This takes place at the same time in all the droplets, making visible a spectrum of light in the form of an arc (see Figure 2.3(b)), with red on the outer part and violet on the inner one.

(a) Light refracted inside a raindrop.

(b) Rainbow over an Alaskan lake.

Figure 2.3: The first image shows the complex path that light follows when entering raindrops, refracting twice and reflecting one time, separating its various wavelengths. The second image depicts a rainbow caused by this optic effects.

2.4 Lightning

A lightning is an atmospheric discharge of electricity, typically originating in a cloud and hitting the ground. Visually its path seems erratic, branching while it advances, and its color is mainly white but produces deep blue or violet illumination in its surroundings, as in Figure 2.4. Simulation of this phenomenon must produce pseudo-random paths and change the illumination of the hole scene to respond to its varying-intensity flashes. To produce strong flashes that are visually plausible, an important aspect to consider is the computation of the shadows that they cast. Also note that lightnings are produced at the sky but hit the ground, so parts of the scene away from it may receive their illumination almost as a uniform directional light, but the parts of the scene close to the hit point must consider more complex lighting methods. It is also important to highlight that, due to its high altitude, this illumination affects globally the whole scene at the same time, making it more difficult to simulate because a huge amount of data may be needed to be processed for the computation. Most current rendering approaches of lightings only consider distant light approximation to speed up computations.

2.5 Puddles, splashes, coronas and ripples

An immediate consequence of rain is that objects get wet, with water accumulating on the ground creating puddles. These create two optic effects that properly simulated enhance the visualiza-

Figure 2.4: Various real lightning strikes, where its complex paths and global lighting effects are visible.

tion of rain: refraction of the ground under the puddles and reflection of the objects over them. Moreover, since raindrops hit the ground continuously, these effects are distorted by the ripples that the impacts produce. Another consequence of the hits are the splashes they produce. If the ground is still dry or has a thin layer of water, splashes are formed by small droplets and a thin and small corona (see Figure 2.5 as an example). If a puddle is present, the mass of water absorbs the impact and the corona is more noticeable and the splash droplets are not so numerous.

Figure 2.5: A water drop splashes onto the ground. The impact surface has a thin layer of water, thus producing a thin corona and many droplets. Reflection on the ground and refraction through the corona are clearly visible.

2.6 Dripping water

When raindrops fall on non horizontal surfaces they do not accumulate on it but trickle down following the forces they receive, mainly gravity and wind. Movement of the drops is also influenced by properties of the material they lay on, like its impurities and whether it is hydrophilic or hydrophobic. Wetness of the surfaces also modifies the path of the drops by reducing its resistance. Finally, these properties combined with water's surface tension also modify the shape of the drops: the more hydrophobic the surface is, the more spherical the droplets are. Simulation of the drops while they move must take into account that they leave behind part of their mass, creating a trail. Moreover, a drop may split or merge into another drop or trail. Note also that, as simulation proceeds, moving drops either accumulate in a basin of the surface or they fall off of it.

As in the previous phenomena, reflections and refractions must be carefully considered. A relevant case where refraction becomes noticeable is when water drips on glass surfaces. This case is important because visualization of the scene behind the glass can be highly distorted if objects are far away. A simulation of this effect is shown in Figure 2.6.

Figure 2.6: Water dripping on glass as simulated by [Wang et al., 2005].

2.7 Fog and glows

Raindrops reduce visibility for far away objects, producing a haze-like effect. The heavier the rain is, the more pronounced this effect gets. In heavy storms this phenomenon may become like a fog due to the high humidity levels that the air reaches. In rain simulation it is important to mimic these types of atmospheric states because of the obvious effect that they have on lighting. For uniform and static fogs, there are two main aspects to consider. First of all, the accumulation of small water drops in the air absorbs part of the light and thus visibility is reduced, occluding far away objects and lowering the details and depth perception of midrange objects (see Figure 2.7). The other aspect is that fog is a participating media, and thus light is scattered when traversing it. This is specially perceived in the glows around light sources produced by the scattering of the emitted light in their immediate vicinities, as in Figure 2.8.

Figure 2.7: Golden Gate shrouded in dense fog at noon.

Figure 2.8: A scene in the early morning. Notice how the lights in the background are scattered in their surroundings by the mist.

Chapter 3

Previous Work

Many different techniques have been developed in recent years to simulate rain, most of them focusing on a concrete subset of rain phenomena. In this section we explore the most recent ones and give a quick overview of each of them. We first consider two important off-line approaches that are image-based, meaning that they add rain to still images or video. Even though this makes them not suitable for interactive animation, some of their techniques can be applied in this area. Then we focus on real-time rendering algorithms. We start with common simple approaches that are not able to achieve realistic simulations, but that have been widely used due to their low impact on performance. After that, we concentrate on the particle-based simulations. Finally, we detail two singular approaches that have a wider focus than all the previous ones, proposing algorithms that render at the same time many of the phenomena detailed in Chapter 2.

3.1 Off-line

In [Garg and Nayar, 2006] a rain streak simulation model and a rain rendering algorithm are presented. The first part consists of an analysis of a raindrop oscillation model developed in atmospheric sciences in order to infer the parameters that best match the reality. This analysis is done by actually capturing real rain streaks and visually comparing them to the synthetic images produced by their software. To fully capture all the possible illumination conditions, they use a setup as depicted in Figure 3.1, where they proceed by changing camera's θ_{view} and light's θ_{light} and ϕ_{light}. Once the parameters of their procedural simulations are properly tuned to achieve good visual matchings with the real raindrop captures, they produce a database of high quality renders for various values of the illumination parameters. Finally, they propose an algorithm to add rain streaks in still images and animations by blending carefully chosen renders of the database onto them. This approach to simulate realistically illuminated rain is an image-based algorithm that takes as input the image (or video), a coarse depth map of the image (video), camera and lighting parameters and rain configuration. The camera and lights are used to select and shade the renders of the database, the depth map culls the streaks behind solid occluders and the final result is blended onto the input image (video). All this is done as an off-line process with a performance of about 10 seconds per frame on their test machine. An example of their results can be seen in Figure 3.2.

In [Garg et al., 2007] they analyze another rain phenomenon: splashing of water drops. They proceed analogously to their previous paper. First of all, they use a real setup to capture a wide variety of splashes in order to be able to empirically infer a mathematical model. As their analysis show, this model must depend on the inclination of the surface relative to the raindrop direction, the material of this surface and the size and velocity of the droplet (see Figure 3.3). The rendering algorithm is also image-based and takes as input an image (or video), a coarse depth map of the image (video), a partition of the image (video) in different materials and the lighting and camera configurations. The rain is then generated as in [Garg et al., 2007], but now they also check

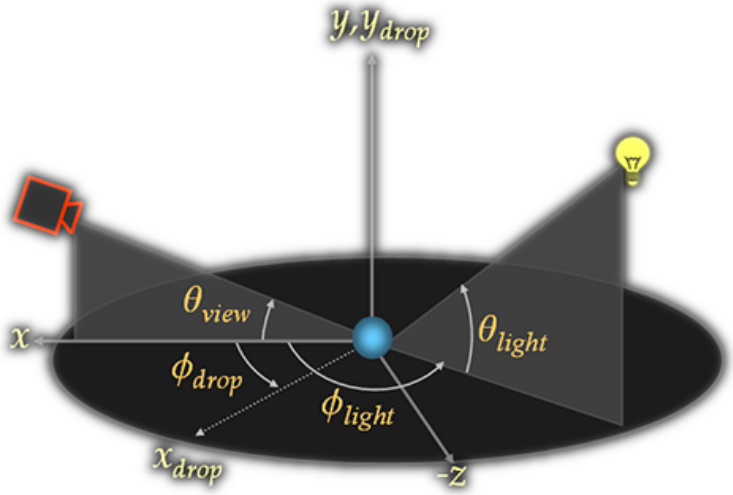


Figure 3.1: Garg and Nayar raindrop model parameters as depicted in [Garg and Nayar, 2006]. The raindrop is located at the origin of coordinates and falls in the opposite direction of the Y axis, the θ and ϕ are the elevation and azimuthal angles for the camera and the light. The x_{drop} is the natural orientation of the raindrop.

Figure 3.2: One of the images shown in [Garg and Nayar, 2006] where rain has been added with their algorithm. A coarse depth map is used to avoid far away rain streaks in front of the traffic light.

collisions in order to add splashes using the probability distributions of their model.

Figure 3.3: Simulated splash distribution in [Garg et al., 2007] for different materials (rusted iron and plastic, as shown on the top right corner of each image) and varying surface inclinations.

The papers [Garg and Nayar, 2006; Garg et al., 2007] represent the most comprehensive analysis of two of the main rain phenomena, i.e. rain streaks and splashes. With thorough studies of the water behavior and its lighting under various conditions, they infer mathematical models that closely match both phenomena. Coupling this result with off-line image-based rendering algorithms they achieve realistic rain renders. These algorithms are CPU-based and thus their performance does not allow interactive framerates. Moreover, their main goal is to add rain in film sequences so an artist must provide the depth map that their approach requires, making it a cumbersome method for animated scenes where the depth map changes each frame.

3.2 Real-time

In this section we focus on approaches similar to our proposal, i.e. the ones whose simulations are performed in real-time. First, simple algorithms are presented in order to give an idea of the most basic rain algorithms, even though they do not actually produce realistic renderings. Then particle-based simulations are detailed to give an overview of the state of the art. In the last part, we explain two special approaches that have been designed to render more rain phenomena than the previous ones.

3.2.1 Simplistic approaches

Traditionally, real-time rain rendering techniques are approximations to the real phenomena. They have a high performance and are mainly used in demanding applications like video-games and virtual reality, where rain simulation time is constrained by the whole logic engine, other physics computation and scene rendering. When little time is available to simulate and render the rain, one of the typical approaches consists of just rendering a few line primitives in front of the observer, using a translucent white material. This method is not able to produce realistic rain streaks and usually it is only used for mild rain. Another solution whose computational cost is independent from the amount of raindrops, and thus is able to simulate heavy rain in the same fraction of time as mild rain, is to use a precomputed texture of rain streaks and blend it onto the scene, scrolling it down at each frame to simulate the drops' fall. Drop parallax is not possible with this approach unless various textures simulating different depths are used. In [Wang and Wade, 2004] a refinement of this technique is presented. Instead of a textured quad in front of the camera, two cones that encompass the observer are used (see Figure 3.4). Four artist-generated rain (or snow) textures are used for the cones. The cones are tilted to adjust for camera movement, the textures are elongated to simulate motion blur and also scrolled to mimic the rain fall. This scroll is done at different speeds for the 4 textures in order to simulate streaks' depth with parallax.

3.2.2 Particle systems

In [Wang et al., 2006] the optical properties of spheric raindrops are analyzed. Even though real raindrops are not spheric, this simplification allows them to derive a closed formula for their algorithm. In a preprocessing step, they compress the environment map and the transfer function of their model using spherical harmonics. To render the raindrops they use real video analysis in

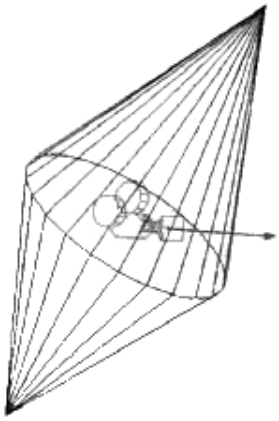


Figure 3.4: Double cone used in [Wang and Wade, 2004] to simulate rain and snow.

order to extract real rain streaks that are later used as textures for the particle system. These textures are shaded using their model, transformed to match the scene's camera and blended to the scene's image. They also add to the output renders homogeneous fog, light glows approximated by Gaussian blurring and rain splashes designed by an artist. The raindrops and the splashes are uniformly distributed in the scene by their particle system. A real image with their rain added can be seen in Figure 3.5. This proposal works in real-time at high frame rates, but it also has some drawbacks: requires real rain video to extract the streak textures, splashes are artist-generated and the preprocessing step forces that the scene where the rain is to be added must remain static.

Figure 3.5: Resulting composite image of the algorithm presented in [Wang et al., 2006].

Raindrop refraction is used in [Rousseau et al., 2006] as the main defining factor of raindrop coloring. They start by analyzing the optic properties of the raindrop and conclude that reflections are limited to its silhouette and thus can be neglected without reducing image quality. In order to model refraction in real time, at a precomputation step they generate a texture mask that determines the direction of the refracted viewing vector for a quasi spherical raindrop. This vector is later used to index a texture storing a wide field of view render of the background. The particle system is stored in a texture where each texel represents a particle's position, information that is

updated per frame. During rendering time the particles are expanded to quad billboards and their color is computed using the mask: either the quad's texel lies outside the drop and the background is directly outputted, or the mask's refraction is used to perturb the direction of view and compute which part of the background is visible. Figure 3.6 summarizes the idea. This allows the rendering of clear refracting raindrops. To add motion blur they change the particles to vertical streaks and render a few drops along this streak, blending them together to simulate the camera's integration time. Finally, they also consider light interaction with the raindrops by modifying their color. The resulting algorithm achieves good visual results when all the objects that can be refracted through the raindrop are far behind the rain (in the background), closest objects are not considered. The authors also mention that raindrop collisions with the ground could be computed, but they do not comment it further.

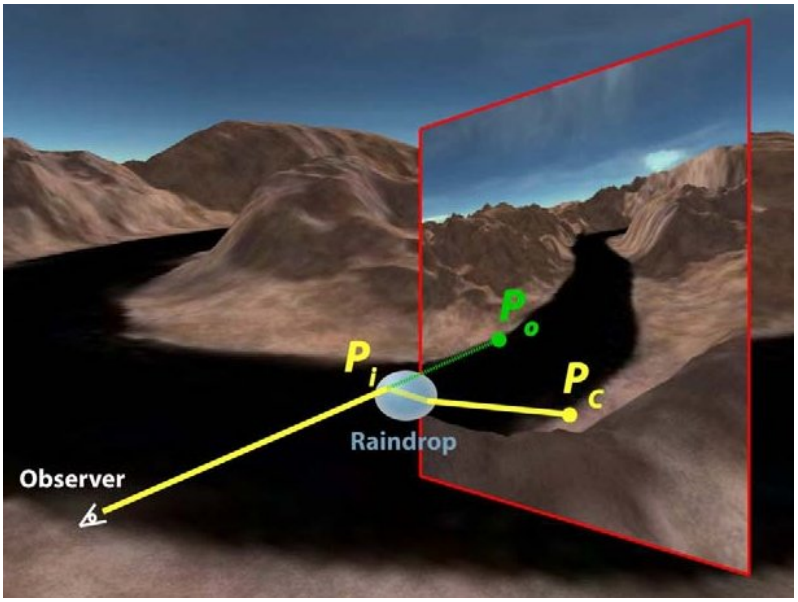


Figure 3.6: The drop refraction approach of [Rousseau et al., 2006]. The red quad is the wide field of view render of the background, P_0 the texel seen from the original view direction, P_c the texel seen after using the refraction perturbation of that specific raindrop and view direction.

In [Tariq, 2007] rain is rendered in real-time as quads textured with a simplified version of the image database presented in [Garg and Nayar, 2006]. By doing so, they achieve realistically looking raindrops at the cost of texture lookups. The animation is done by using the graphics pipeline to update the raindrop positions, then GPU state is changed and the actual render proceeds. This render creates a quad for each particle and computes its lighting parameters needed to use the simplified database. The final color is produced by blending together the 4 textures of the database that are closest to these lighting parameters. To enhance the visualization, they also use fog to add glows around light sources and change the reflectance of the surfaces (see Figure 3.7).

The work done in [Puig-Centelles et al., 2009] is a new rain rendering algorithm based on a particle system. Its setting is a simplified raining area defined by an ellipse and a rain container where actual particles are simulated. This container is a semi-cylindrical volume whose size is big enough to allow the observer to rotate and move a certain amount without needing to compute new particles to populate the container. This is an important difference to the previous methods of [Tariq, 2007] and [Rousseau et al., 2006], where recomputation is needed when the camera changes. Figure 3.8 depicts this container as seen from above it. In the figure, another important property of the algorithm can be seen: the density of the particles far away from the observer is lower than that of the particles close to it. This adaptive scheme, along with a technique

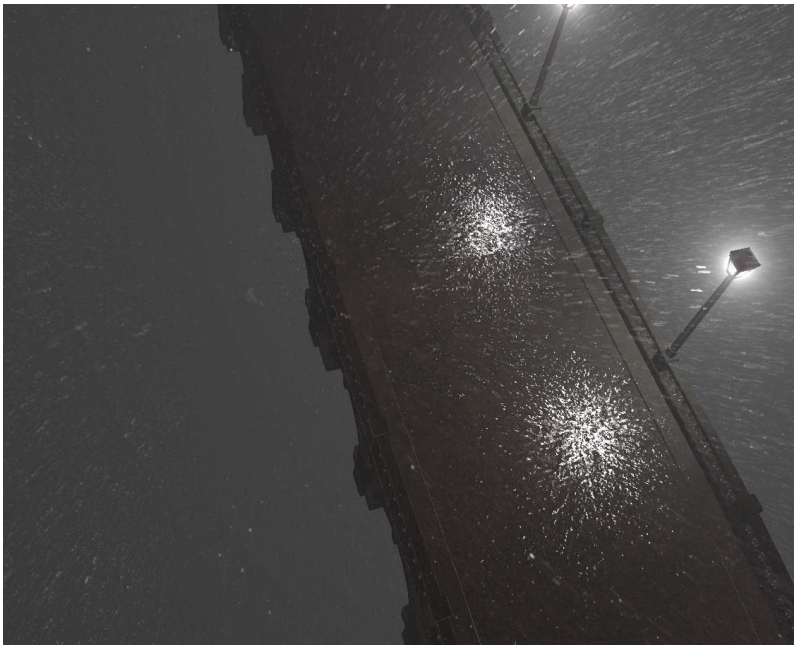


Figure 3.7: Rain render of [Tariq, 2007] where rain streaks area clearly visible, along with light glows and wet surfaces.

that changes the particles' size depending on the distance, allows to simulate heavy rain with less particles than the previous methods. They also handle movements from the rainy area to outside of it by having a transition area surrounding the ellipse where the rain is computed. All the method is GPU-based and consists of two main steps: the first execution of the render pipeline just updates the particle positions, a second render actually updates the frame buffer by blending the particle colors. These particles are expanded to quads in a geometry shader, with a semitransparent material that tries to mimic the rain streaks.

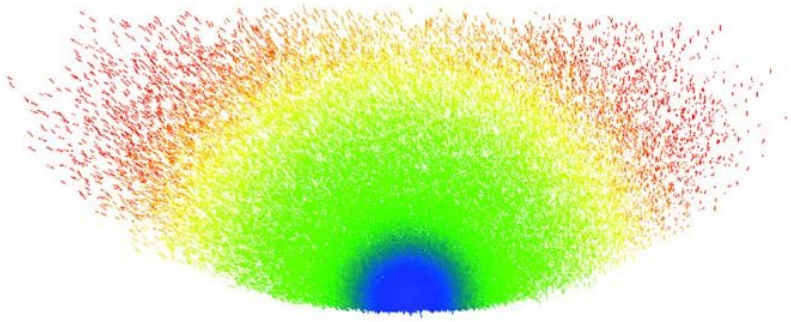


Figure 3.8: Bird's eye view of the semi-cylindrical area where rain is simulated, as shown in [Puig-Centelles et al., 2009]. In blue are shown the particles closest to the observer, in red the most distant ones. It can also be perceived how the density of the particles is adapted to this distance.

3.2.3 Multiple phenomena

Some of the state of the art rain rendering techniques include a combination of a wide variety of rain phenomena in order to achieve more complete simulations. In this category most of the

solutions are off-line due to their high computational costs. Nonetheless they are widely used in cinema special effects thanks to the fact that rendering times in cinema production are not constrained by the necessity of real-time frame rates. Examples include the work in [Borshukov, 2005] or the use of [Herman, 2001] by Pixar®. If physically correct simulation is not needed, in recent years there have been proposed some approximations that run in real-time and still produce visually pleasing results.

In [Changbo et al., 2008] they propose a set of methods to simulate various rain effects. The most basic effect is the simulation of the actual raindrops. Their approach consists of treating the raindrops as microfacets and precompute the lighting for them, storing the result in a spherical texture. With this step they proceed to simulate the camera's integration time to produce a rain streak. This is done by taking the 3 main oscillation types for raindrops and animate them along the streak in order to disturb its thickness. They also develop a new method for the participating media, namely the fog produced by tinny raindrops in the air. There are two different kinds of light that are scattered through the media: skylight's and lamp's. The former is precomputed in a lookup table that depends on the height of the drop from the ground. For the latter they separate the light source single scattering formula into two parts, one is precomputed and another is executed per drop, allowing a real-time computation thanks to this split. The participating media also generates rainbows when the density of the rain decreases. They take into account how the rain changes the properties of the ground, i.e. it is rendered differently depending on whether it is a dry area, wet area or a puddle. For dry areas they use the usual diffuse reflections, wet areas have specular reflections and puddles have both specular reflections and refractions. On the ground, splashes and ripples are also computed.

The ATI® demo documented in [Tatarchuk, 2006] is a set of techniques that approximate the rain phenomena without trying to use physically correct simulations. Figure 3.9 shows an example of its results. Rain streaks are animated with a particle system and rendered using an image-based algorithm that composes the final image with a single pass. Dripping raindrops are set up by an artist and rendered as billboards using the same shading algorithm as normal rain streaks. Splashes are uncorrelated to the actual drop collisions, being instead randomly started on the ground. Their animation is done using billboards textured with a real video of a milk drop splashing on the ground. To avoid repeating effects, the billboards are randomly perturbed by changing their size and transparency and flipping their texture. Their lighting is brighter when the source is located behind the splashes. In heavy rain situations, objects receive a huge amount of drop collisions, each of them creating splashes of small particles that surround the object. This produces an halo like effect along the object's silhouette. To simulate this phenomenon a technique similar to the ones used to render fur is used: series of semi-transparent shells are rendered encompassing the object. Ripples on water surfaces are simulated on a tinny texture that is tiled over the whole scene, changing the bump-mapping of the objects. To avoid repetitions, different scale and rotation factors are used for the objects. In the demo, water droplets trickling down on glass surface are also simulated using the force of gravity, static friction, surface wetness and affinity, wipers and mass lose and merge. The rendering includes reflection of the objects in front of the surface and refraction of the scene behind it. This part of the scene, the one behind the glass, receives shadows casted by the water and an approximation of caustic highlights (see Figure 3.10). The technique used for this last phenomenon builds upon the algorithm of [Kaneda et al., 1999], but it is simplified and adapted to high performance GPU evaluation and rendering. The ATI® demo also includes a certain amount of predefined lightnings whose casted shadows are precomputed and stored in textures. When a lightning strikes, its illumination affects the whole scene casting the precomputed shadows and modifying the shading of all the rain phenomena by making them less opaque. A participating media is taken into account by attenuating light depending on the distance to the observer and rendering glows at the scene's light sources. Finally reflections due to wet materials are computed using impostors that create stretched reflection towards the observer, distorted so that only dominant colors are made distinguishable. This plethora of phenomena comes at the cost of 300 unique shaders to render just the rain system.

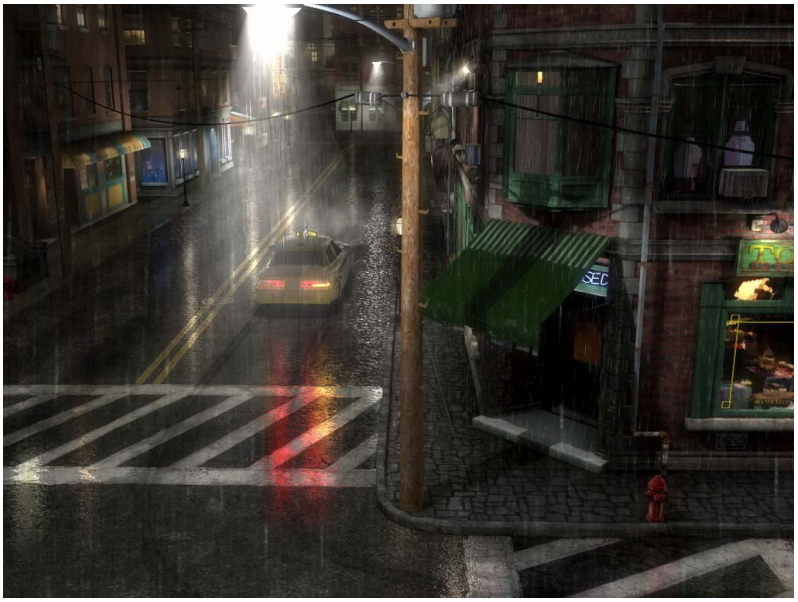


Figure 3.9: Various of the rain effects of [Tatarchuk, 2006] can be seen in the render, most notably the rain streaks and reflections on wet surfaces.

Figure 3.10: Detail of water droplets trickling down on glass in [Tatarchuk, 2006]. Caustic-like effects can be seen on the shadows projected on the toys.

3.3 Summary

In Table 3.1 a brief summary of the previously explained algorithms is presented. They are grouped as in the preceding sections, i.e. the off-line algorithms, the simple approach, particle systems and multiple rain phenomena methods. For each of them its main properties are highlighted;

columns from left to right: which rendering primitive is used for the simulation of raindrops, whether it is a real-time algorithm, use of the GPU to accelerate the rendering or the simulations, rendering of raindrops, raindrop particles removal when the ground is hit or when the raindrops are behind solid occluders, wind, rainbow and lightning simulations, reflection and refraction of the water (including wet surfaces or raindrops), participating media that modifies the illumination, simulation of splashes on the ground, adaptive schemes for Level-Of-Detail optimizations, and simulation of ripples and dripping water.

As can be seen in the table, none of the methods simulate at the same time all the phenomena. [Tatarchuk, 2006] is the most comprehensive one, but at the cost of a highly complex implementation and limiting camera movement. [Changbo et al., 2008] is also extensive, but few details are given in the paper, so its evaluation is hardly possible. Even though [Garg and Nayar, 2006; Garg et al., 2007] are included in the table, they are not real-time algorithms and so they are not the focus of our proposal. The simplistic approach in [Wang and Wade, 2004] only simulates the effect of raindrops, which is the only phenomenon common to all the presented algorithms. Their method is out of our scope since they do not use realistic rendering in order to speed up their system. [Wang et al., 2006] is able to achieve good visual results, but requires as input real rain video in order to extract the streak textures. None of the remaining works in [Rousseau et al., 2006; Puig-Centelles et al., 2009; Tariq, 2007] consider collision of the raindrops with the ground. Interaction of the rain with the scene is one of our important objectives, since it avoids that rain penetrates geometry. In [Puig-Centelles et al., 2009] they do not consider realistic streaks either, instead they use uniform transparent materials. Finally, let us note that the work done in [Tariq, 2007] is similar to our approach, but, apart from the missing collisions, it also uses a simplified illumination computation.

	Animation	Real-time	GPU	Raindrops	Ground collision	Moving camera	Wind	Rainbow	Lightning	Reflection (on ground or on raindrops)	Refraction (in puddles or in raindrops)	Participating media (fog, light glows)	Splashes (raindrop animated splashes on the ground)	LOD (texture mipmaping, adaptive particle simulation)	Ripples (waters surfaces with animated ripples)	Dripping (rain drops dripping on glass surfaces)
[Garg and Nayar, 2006]	particle	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
[Garg et al., 2007]	particle	✗	✗	✓	✓	✓	✗	✗	✗	✗	✓	✗	✓	✓	✗	✗
[Wang and Wade, 2004]	texture	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[Wang et al., 2006]	particle	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✗	✓	✗
[Rousseau et al., 2006]	particle	✓	✓	✓	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
[Tariq, 2007]	particle	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓	✗	✗
[Puig-Centelles et al., 2009]	particle	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
[Changbo et al., 2008]	particle	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗
[Tatarchuk, 2006]	particle / texture	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓

Table 3.1: Summary of the essential properties of the presented algorithms. See Section 3.3 for an explanation of each column and classification of the algorithms.

Chapter 4

Rain rendering

In this chapter we detail all the aspects of our rain rendering method. First of all, we give a brief overview of the whole approach, mentioning its scope and enumerating the main steps it performs for the simulation. Then we proceed by describing in greater detail the conceptual model of our rain simulation system, defining all its related terms. In the following sections, we extend on the computations performed in order to generate all the data used by the algorithm and, finally, we comment on the real-time simulation and the rendering pipeline.

4.1 Algorithm overview

The main objective of our method is to achieve a real-time rain rendering algorithm that uses an artist-generated particle distribution, takes into consideration complex illumination effects on the raindrops and, finally, considers interaction of the drops with the scene in order to avoid having rain inside solid objects and to produce splashes where collisions are detected.

Our rain is added into a scene as the final rendering step, that is: once a scene has been rendered, the rain streaks and splashes are blended to the color buffer. The only requirement of the scene is that it provides a well-defined depth buffer that can be used to cull the particles that are occluded. This implies that the current approach cannot add rain to volumetric data, nor treat scenes with semi-transparent geometry.

The proposed method is divided into two different parts, one executed once as a preprocess and the other one being the real-time simulation and rendering.

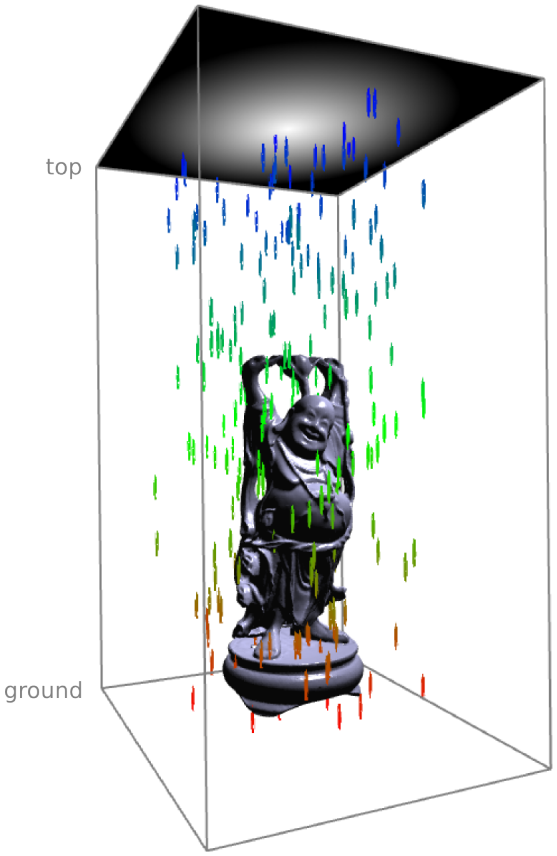
The preprocess is used to generate all the rain data that is latter needed for the real-time simulation. This data is composed by the rain particles and an atlas. The particles are used during rendering time to animate the falling raindrops and create the splashes when they hit the ground. Its placement in the scene follows an artist-generated distribution, specified as an image whose greyscale values define the particle density. The generated atlas stores precomputed renderings of rain streak simulations where complex drop oscillations and lighting effects are taken into account to shade them. This atlas is always the same for any rain, and thus only needs to be created once.

As already hinted, the real-time simulation starts by rendering the scene. After that, the particles generated in the preprocess are scheduled for simulation depending on various selection criteria, namely its proximity to the observer and whether they are potentially visible or not. Particles are transformed in order to set them in their proper places, with respect to their fall, and sent to the GPU. In the programmable pipeline of the graphics hardware, we compute collisions of the particles with the scene and render them as billboards that represent rain streaks (if no collision is found) or splashes (when a collision has happened). The texture for a rain streak billboard is taken from the pre-generated atlas by selecting its subimage that best fits the illumination parameters of that particular drop. The final result is blended onto the scene render using transparency.

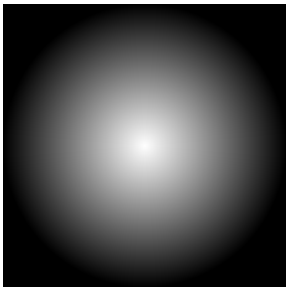
4.2 Rain simulation model

Figure 4.1(a) depicts a small rain simulation volume along with some of the most important factors that govern it. The grey box that bounds the scene is the volume where rain is simulated and it is called the *rain space*. In order to create it, an orthographic camera is used. Its frustum defines the space in a way such that its z-near plane corresponds to the *top* of the space and the z-far to its *ground level*. Raindrops move in the view direction of that camera, originating at the top and going straight to the ground with uniform constant speed. When a drop reaches the ground, it is respawned again at its original position at the top of the rain space. This produces a cyclic movement for each particle, followed continuously during the whole simulation.

The spatial distribution of the drops follows an artist-generated probability density function defined by the means of a *density map*. In Figure 4.1(a) this map is the circular gradient depicted at the top, shown on its own in Figure 4.1(b). In the example, the density map is used to make drops denser at the center and absent at the extremes. This distribution only affects the placement of drops in the rain space's horizontal plane, while vertically they are distributed using a uniform random source that shifts their starting heights.



(a) Example of a rain simulation model, basic scheme.



(b) Density map used in the example.

(c) Depth map of the example, as captured from the top.

Figure 4.1: The image on the left shows a basic scheme of our rain simulation model. The rain space is shown as a grey bounding box, raindrops are colored using codes for their height (blue for top, red for ground level), the density map can be seen at the top as a circular gradient with maximum density at the center. At the insets on the right, the density map is depicted on its own (top image) and the depth map of the scene, as captured from its top, is also shown (bottom image).

The camera used to define the rain space also captures a *depth map* of the scene (see Figure 4.1(c)). This information is used to compute the *hit height*, i.e. the maximum distance each drop can move before colliding with an object or the ground. During simulation, a particle is considered a raindrop while it is above its hit height and a splash during a short period of time when it goes below its hit height. A splash is rendered always at the hit height, ignoring the actual particle position. When the splash animation finishes, the particle is no longer rendered in the scene. In Figure 4.2 a scheme of this behavior is presented. Note that, in any case, simulation of the particle fall is always considered, advancing all the time and respawning normally at the top when it reaches the ground.

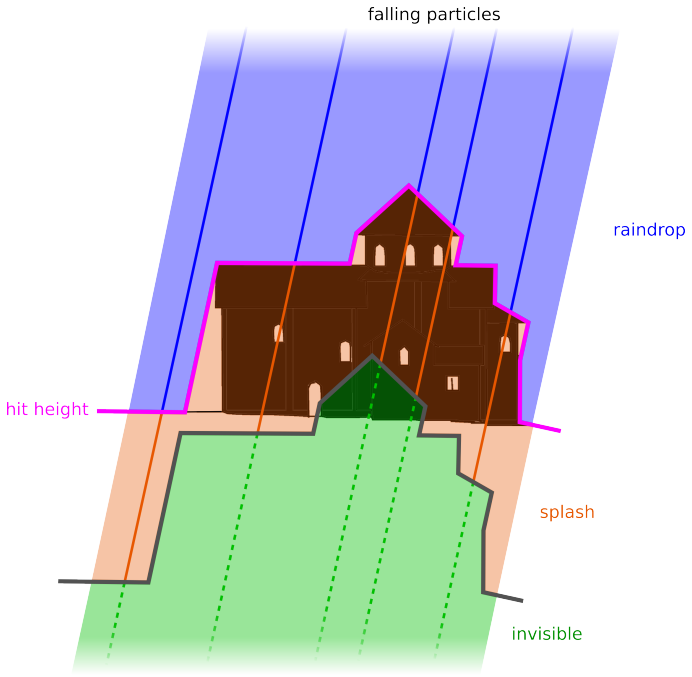


Figure 4.2: Scheme of the various states a particle may be in: raindrop, splash or invisible. Particles in the image are depicted by the slanted straight lines. The blue part corresponds to the time when a particle is rendered as a raindrop. When it arrives to the hit height (pink line along the silhouette of the scene), it changes into a splash animation that stays static in the hit position. Nonetheless, the particle movement is still off-screen simulated. The orange part of the lines corresponds to how much the particles move while being rendered as a splash. Finally, when the animation of the splash finishes, nothing needs to be rendered and thus the moving particle has no contribution in the screen. In the scheme, this last state is depicted as the dotted green parts of the lines. Notice how all the orange parts of the lines have the same length since the splash animation takes the same time for all the particles. Length of the blue and green parts are particle dependent and are influenced by the scene configuration.

Finally, we have observed that the proposed simulation model is general enough to also work with a perspective camera, but some considerations must be taken into account. First of all, each particle's direction becomes different from each other, making the rain look dispersive. Since particles move with the same speed and take the same time to be respawned at the top position, all of them advance the same distance. In an orthographic camera, this implies that each drop reaches the ground, but in a perspective camera only drops near the center of projection reach it, the remaining ones only having time to traverse part of the distance since their direction is not straight to the ground. It is also important to highlight the fact that the depth map is less precise with the parts of the scene near the ground level, so hit heights are computed with more error. These effects are depicted in Figure 4.3. Overall, perspective frusta do not produce visually

appealing results if not defined very carefully, and so we will not consider them further in the remaining of this document.

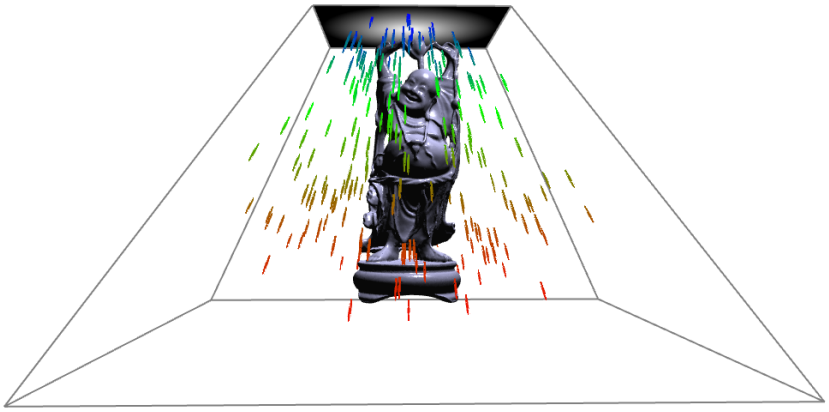


Figure 4.3: Rain space defined with a perspective camera. Notice how directions are different for each drop. In the image, it can also be observed how raindrops near the center of the rain space reach lower positions than the ones at the extremes. This is caused by the fact that all the raindrops traverse the exact same distance, but not all of them move straight to the ground.

4.2.1 Simulation subspaces

Since the rain space may be orders of magnitude bigger than what the observer can see, the simulation model is divided into various subspaces to optimize simulation and rendering times. The objective of this partition is to perform computations only in a small volume surrounding the observer, as shown in Figure 4.4.

To reduce the volume of the rain space, a *local top* and a *local ground* planes are defined (in green in Figure 4.4). Instead of using the rain space's top and ground, the particles originate at the local top and run to the local ground, respawning at the former when they reach the latter. This reduces the length that the particles must run in their cyclic movement, so less particles are needed to achieve the same visual density at the expense of increasing repetitiveness of the rain streaks, but, in all our experiments, we have not noticed any artifact related to this effect. The rain space that lies between these planes is called *local space* and, in general, is a subset of the rain space. During simulation time, these planes are moved up and down to guarantee that the observer is always at an equidistant position from both of them.

To further reduce the extension of the volume where rain is computed, we use a *culling radius* around the observer to ignore the particles outside of it. This radius is presently approximated with a prism oriented in the same direction as rain fall (also shown in red in Figure 4.4).

Both the local space and the culling radius reduce the distance of the furthest particles that can be seen, so using too narrow local spaces or small radii may cause errors in the visualization (see Figure 4.5). Inadequate local spaces cause that particles respawn and disappear too close. With small radii, rain simulation is only performed in the immediate surroundings, so missing rain in the vicinity can be perceived.

4.3 Generation of the rain data

In this section we detail the information that needs to be computed prior to the beginning of the simulation. As mentioned in Section 4.1, we first create an atlas used to texture the raindrops with realistic precomputed lighting patterns. Then we create the particles that represent drops,

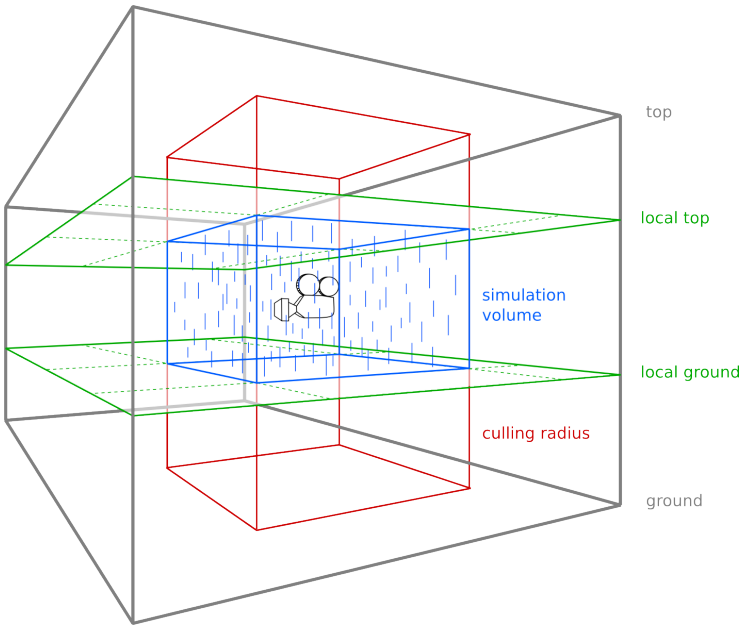


Figure 4.4: The grey box corresponds to the whole rain space. Green planes are the local top and local ground. The red box depicts the approximate culling radius around the camera where simulation should occur. Intersection of the culling box with the subspaces defined by the local planes produces the simulation volume, in the picture shown as the blue box bounding the observer.

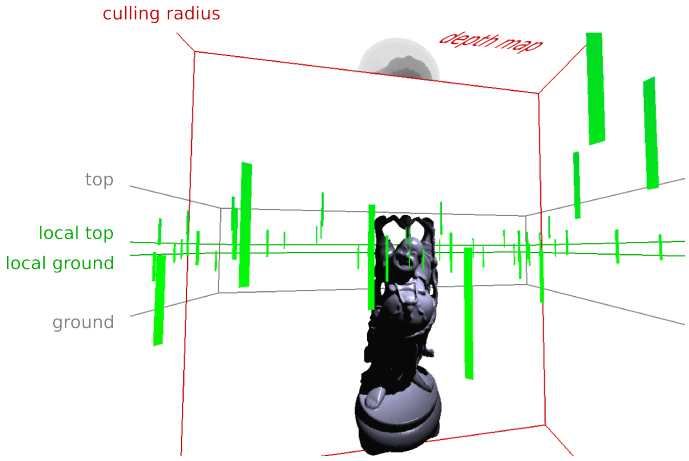


Figure 4.5: A capture of a rain scene as seen from inside its rain space. The grey box that bounds it is visible in the background and its depth map is shown at the top (we only depict the part inside the culling radius, i.e. the red box). In the image, two visualization errors are evident. First, the culling radius is set too close to the observer, causing that only the particles in its immediate vicinity are visible, that is the ones inside the red box. The second one is caused by the narrow local space (shown in green in the background). The local planes define a space so thin, that particles are only simulated in a slice of the screen, disappearing at its bottom and respawning at its top. Wider local space could make this effect invisible by forcing it to take place outside the viewing frustum.

distributing them following an artist-generated density map. Finally, the particles are packed together in various groups, forming simulation units that are treated at once during rendering time.

4.3.1 Rain streak atlases

As has already been mentioned in Section 4.1, we use the database[1] generated in [Garg and Nayar, 2006] as textures for the raindrop billboards. This database has in total 15000 individual images for rain streaks. Unfortunately, this huge amount of data cannot be sent simultaneously to the graphics hardware as individual textures. Moreover, since we cannot predict in advance which ones will be needed in a specific frame of an animation, we are unable to select the required texture subset on a per frame basis. Thus, we are forced to pack all of them in an atlas in order to be able to access the whole information at any time.

Each texture of the database is parameterized by its lighting configuration. As shown in Figure 4.6, this configuration is determined by three different variables: θ_{view} elevation angle of the camera, θ_{light} elevation of the light source and ϕ_{light} azimuthal angle of the light source. Only positive values are considered for θ_{view} and ϕ_{light}, with negative ones mapping to their correspondent absolute value. Our atlas is a grid where each of its positions is associated to a concrete combination of values of those three variables, thus uniquely determining which texture of the database should be placed in them. In Figure 4.7, two atlases used by our system are depicted, showing the organization of the grid. The atlas on the left corresponds to the textures where point lighting is considered, the one on the right is for environment lighting. In the latter case, the texture is smaller since only the θ_{view} parameter needs to be taken into account.

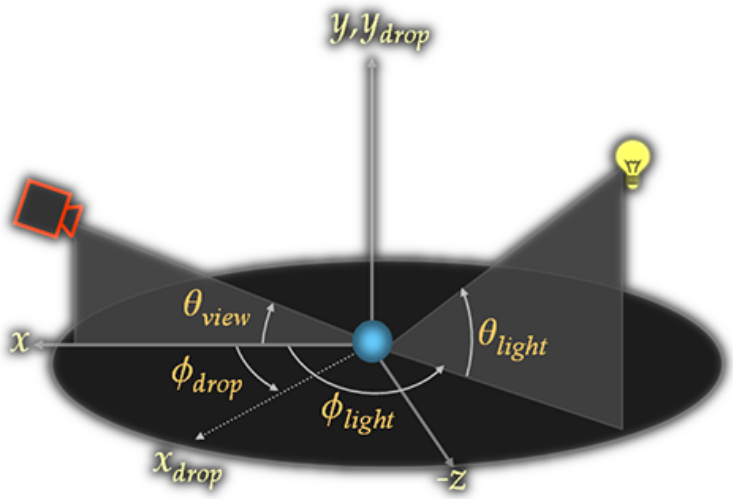


Figure 4.6: Garg and Nayar raindrop model parameters as depicted in [Garg and Nayar, 2006]. The raindrop is located at the origin of coordinates and falls in the opposite direction of the Y axis, the θ and ϕ are the elevation and azimuthal angles for the camera and the light. The x_{drop} is the natural orientation of the raindrop. Only θ_{view}, θ_{light} and ϕ_{light} need to be computed in order to identify textures of the database.

[1] The database can be downloaded from http://www1.cs.columbia.edu/CAVE/databases/rain_streak_db/rain_streak.php

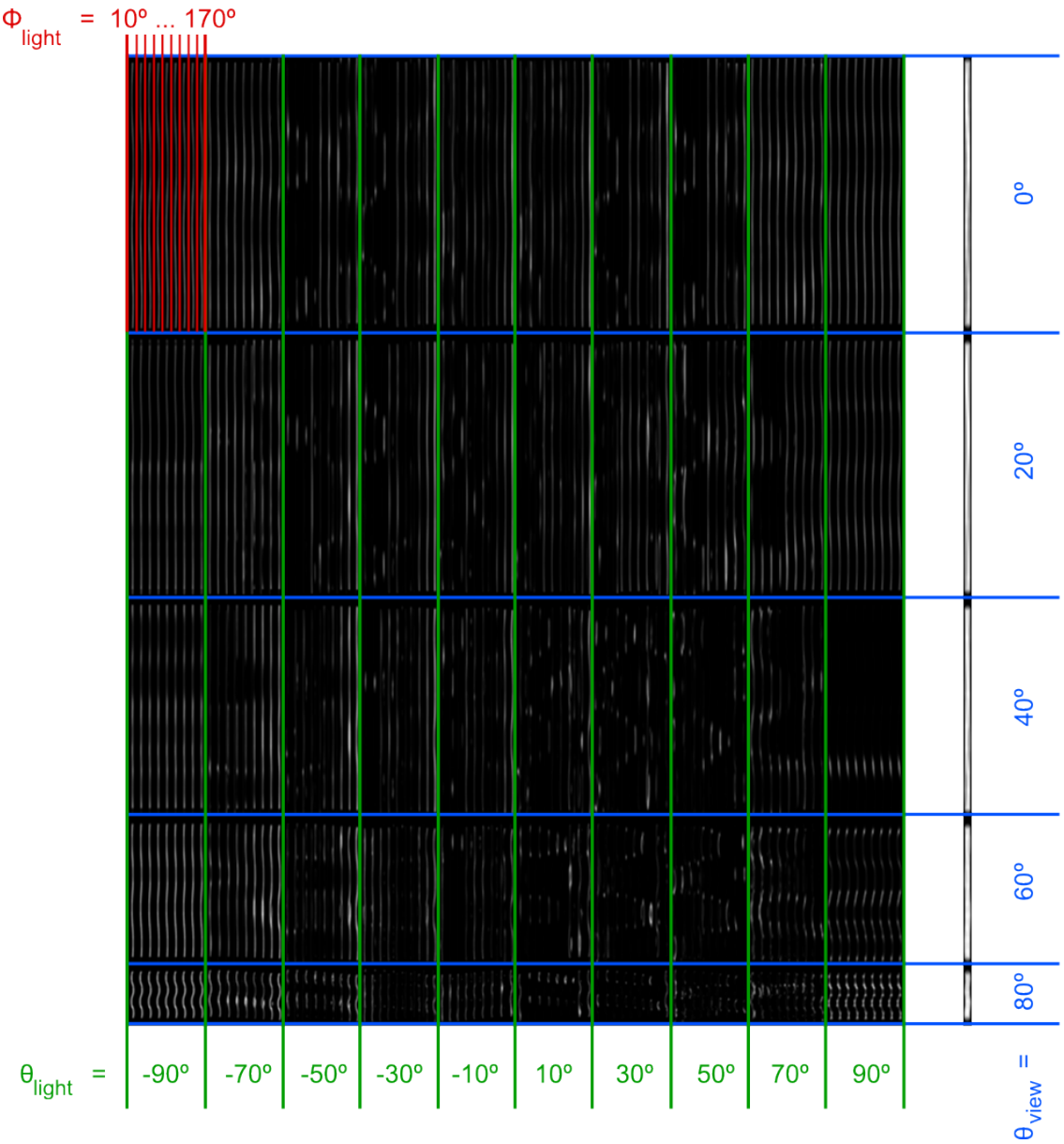
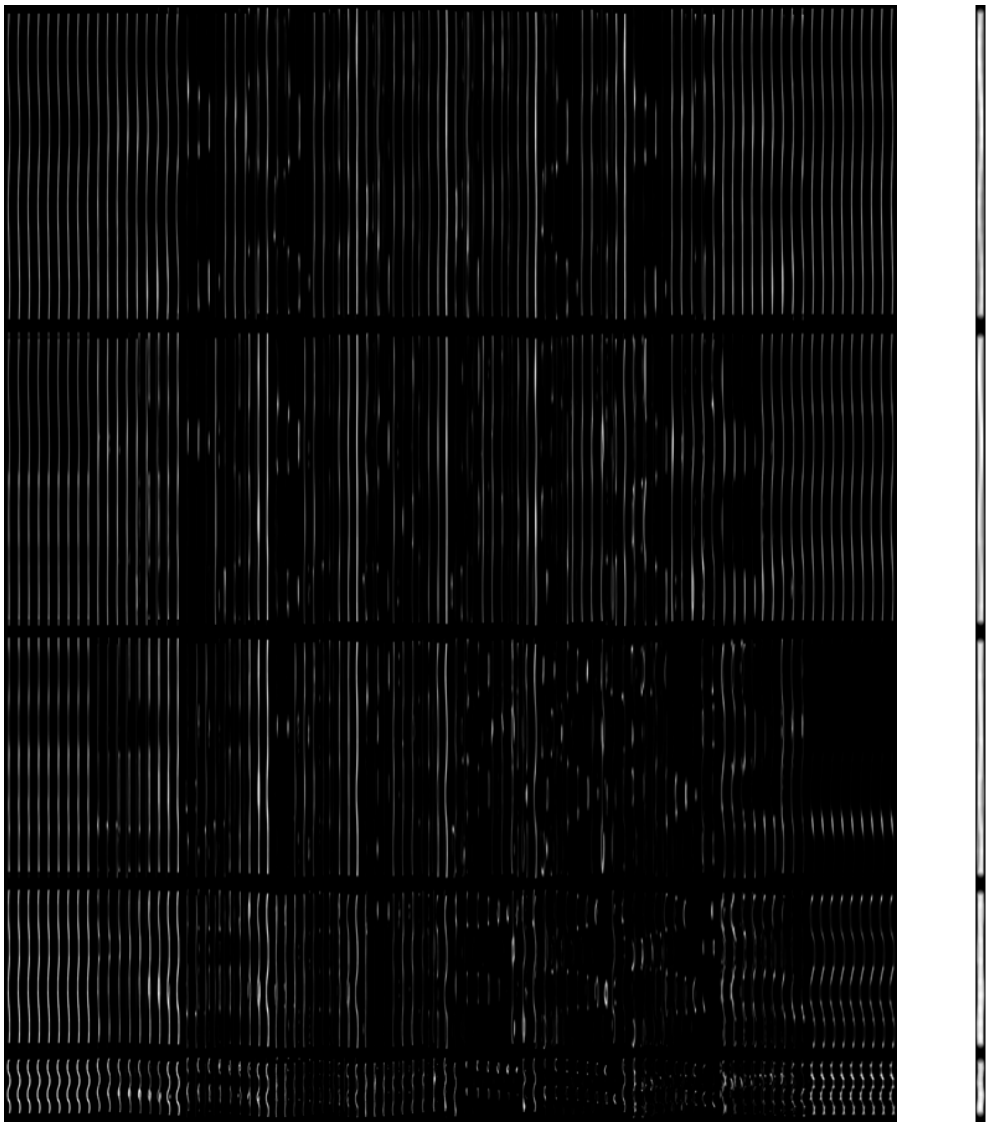
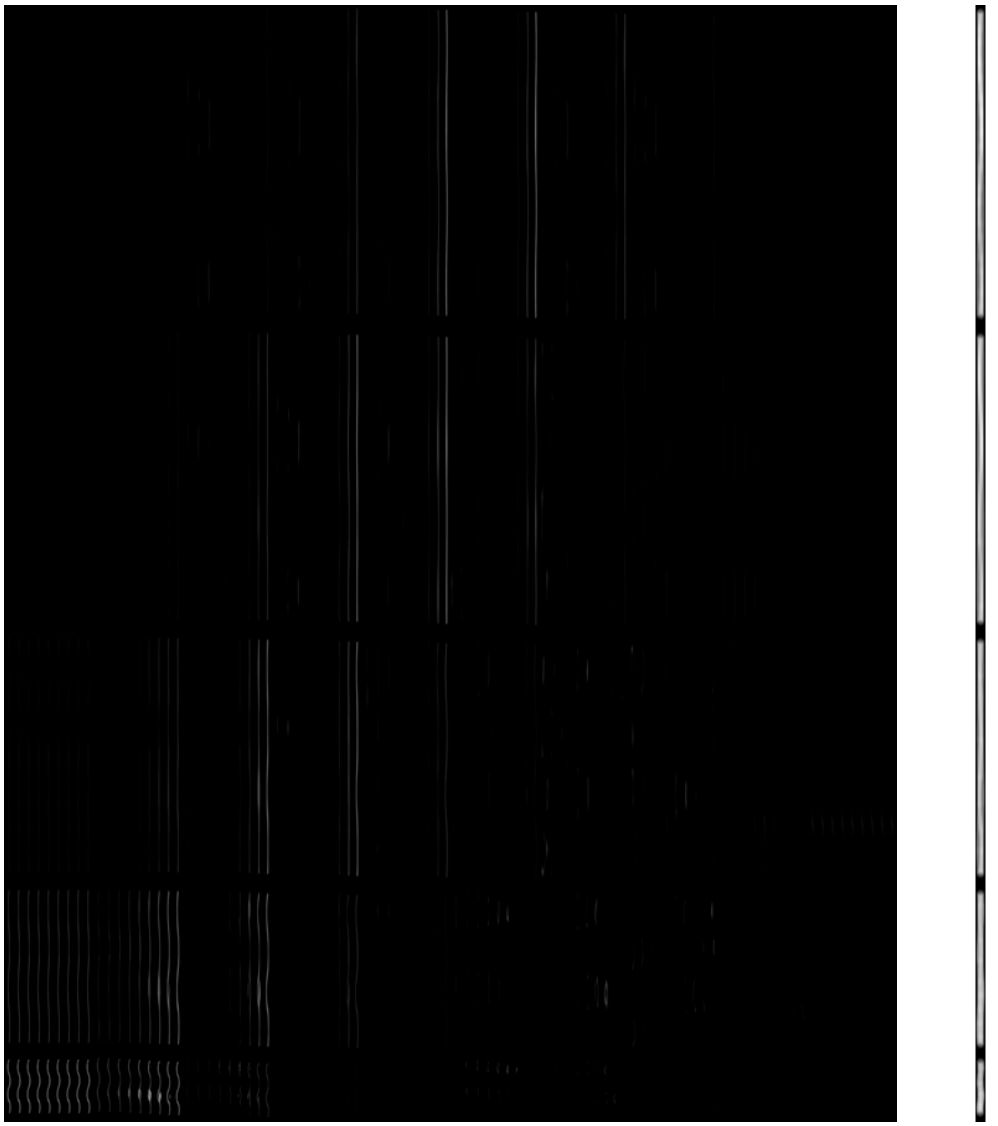


Figure 4.7: Organization of the rain streak atlases using the parameters of [Garg and Nayar, 2006]. Each row is used for a value of the θ_{view} angle parameter (in blue, from 0 to 80 degrees, with steps of 20 degrees). Each group of 9 consecutive columns corresponds to a value of θ_{light} (in green, from -90 to 90 degrees, with steps of 20 degrees) and a column in each group corresponds to a value of ϕ_{light} (in red, from 10 to 170 degrees, with steps of 20 degrees). This grid encodes the 450 different streaks associated to all the possible combinations of the lighting parameters considered by the authors. Environment lighting texture (on the right) only uses θ_{view}.



(a) (b)

Figure 4.8: Atlases that pack all the renders of a specific rain streak oscillation from [Garg and Nayar, 2006]'s database. The first image is for point light illumination, the second one for environment lighting. See Figure 4.7 for a explanation of their organization.



(a) *(b)*

Figure 4.9: Same as in Figure 4.8 but normalizing the brightness of each streak with the database scale factors.

The database has been created with 10 independent rain streak oscillations. The combinations of the lighting parameters that the authors have taken produce 370 different images for each streak oscillation when considering point lights, and 5 more for environment lighting. For each of the rain streak oscillations, we pack all its renders in two atlases, one with a resolution of 2880 by 3606 for the point light illumination and another one of 32 by 3606 for environment illumination. See in Figure 4.8(a) an example of the former and in Figure 4.8(b) the results for the latter. In Figure 4.9 the same atlases are shown when also taken into consideration the brightness scale factors provided in the database. In order to pack the resulting atlases of each of the 10 available oscillations, we store them in two texture arrays, one for point light sources and the other for environment illumination.

To avoid aliasing artifacts, a mipmap of the texture arrays is generated. The first four levels are directly taken from the original database, since 4 different resolutions are provided, the remaining 8 levels are automatically generated. Since each rain streak texture has a wide black margin, individual textures do not bleed color into their neighbors in the atlas during the first levels of these mipmap reductions. Moreover, streak brightness is lost during the reductions, thus the last levels have almost no contribution in color due to their low opacities.

4.3.2 Particle generation

Particle generation requires as input a few parameters to create the raindrops and the rain space. The most basic one is the need for an orthographic camera whose frustum is to be treated as the rain space (see the grey box in Figure 4.10). Information about the camera geometry transformation, i.e. its matrix, that we call *rain matrix*, is used during generation and real-time simulation. This camera is also used to create the depth map for the rain by rendering the scene from it.

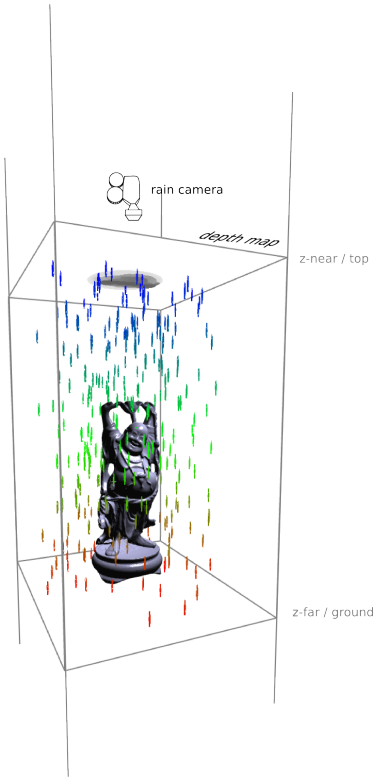


Figure 4.10: The camera whose frustum is used as the rain space. Also notice how the camera's z-near coincides with the top, and its z-far with the ground.

In order to properly distribute the drops, a probability density function must be provided. Our current approach is to derive it from an artist-generated density map whose 8-bit greyscale values are treated as the desired density, white being the densest and black being raindrop absence (see Figure 4.1(b) for a map example). This image is mapped to the camera's z-near plane and thus defines the distribution of drops as seen from the top. Drop position along the height of the rain space follows a uniform distribution.

Particle positions are created in the camera's clip space, in a way such that each coordinate is in $[-1, 1]$. Since the local top and local ground planes define a slice of the whole rain space where drops are to be simulated, the range of values for the Z coordinate is smaller than for the rest of the dimensions. If h is the separation between the local planes and H is the rain space height, then the Z range is $[1 - 2h/H, 1]$. Notice how the local ground and the rain space ground coincide when using this definition. Taking all this into account, placement of the particles in X and Y needs to choose values in $[-1, 1]$ following the distribution of the density map, and values for Z have to be uniformly distributed in $[1 - 2h/H, 1]$.

With the previously considered parameters, we know the volume to fill with drops and the probability distributions to use for them. The system uses two extra parameters in order to actually know how many drops to generate: maximum drops per unit area per second and drop falling speed. By dividing the two of them the maximum amount of drops per unit volume is found, and thus local space's total drop amount can be computed.

As a final step, when the particles are created, the system randomly assigns to each of them one of the 10 streak oscillations available in the atlases generated in Subsection 4.3.1.

4.3.3 Particle packets

Particles are distributed in packets following one of the three different available grouping schemes. These packets represent simulation and rendering units, so the system treats all the drops inside a packet at once. The three schemes have different target usages and imply different performances, so care must be taken when selecting between them. Packet desired size is another important factor that affects performance: if it is too big, resources may be wasted by particles that lay outside the viewing frustum, if it is too small, many packet transfers to the GPU are needed, hindering performance.

Worldwide packet approach

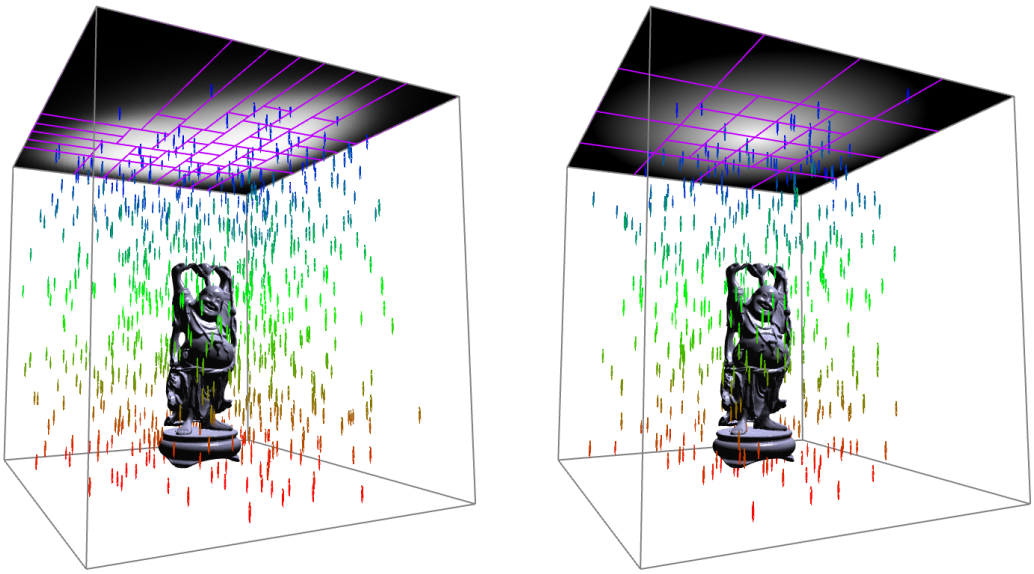
This is the simplest case: just one packet is used, holding all the particles generated. The existence of a single packet implies that only one packet is simulated and all the raindrops are rendered at each frame, assuming an infinite culling radius.

Tree packet approach

In this case, a partition of the rain space is used to group the particles. The approach taken is to perform two different 2-dimensional partitions on the top plane and then take the one that maximizes packet usage, i.e. the one with fewer leaves. One of the partitions used is a kd-tree-like algorithm: the plane is split into two so that each part has a similar amount of drops, if any of the children has more particles than the desired amount, it is then recursively subdivided (see Figure 4.11(a)). The other approach is a quad-tree, which is similar to the previous one except that, each time that a node has more drops than the desired amount, it is recursively subdivided in four equally sized parts (shown in Figure 4.11(b)).

Grid packet approach

This approach differs from the previous ones in that particles are not created distributing them in the local space as explained in Subsection 4.3.2. Instead, small *buckets* of particles are filled with varying densities. The idea is that these buckets can be latter used to recreate any distribution in local space: by selecting buckets with proper densities and placing them in the correct positions,



(a) *(b)*

Figure 4.11: The same scene using two different density maps, partitioned with the tree approach using a small packet size. The partition's packets are shown in purple at the top. The image on the left has a non-symmetric density map and is partitioned with the kd-tree-like algorithm. Notice how the packets at the extremes, specially the one at the top left corner of the image, are bigger than the ones in the high density area, the white zone. The image on the right has the circular gradient of Figure 4.1(b) as a density map and has been regularly partitioned with the quad-tree algorithm. In total, it uses three levels of subdivision, only needing this amount in the central part of the map.

the whole local space can be filled in a way such that the particle distribution matches the density map provided by the artist. In the example of Figure 4.12, 20 of the blue buckets are needed to occupy the whole local space (in green).

The first step in this approach is to divide the top plane using a regular grid with dimensions g_{width} and g_{height}. For each position in the grid its corresponding average density is computed by taking the texels of the density map that are inside its area and filtering them. If various texels are found, they are mipmap filtered, else linear interpolation of the four closest neighbor texels is used to find the correct density. Then the artist's density map is replaced with a new one of resolution g_{width} by g_{height} and filled with the densities of the grid. The dimensions of this new density map's texels match that of the buckets. Finally, a bucket has to be created for each density value found in the new density map. The particles of all buckets are always uniformly distributed such that, for any particle p, $x_p \in [-1, -1 + 2/g_{width}]$, $y_p \in [-1, -1 + 2/g_{height}]$ and $z_p \in [1 - 2h/H, 1]$ (where h is the local space height and H the rain space height, as explained in Subsection 4.3.2), i.e. particles are created inside the lower left prism of the grid division (shown in blue in Figure 4.12). Even though particles in the buckets are uniformly distributed, the original artist's distribution can be approximated by properly selecting and placing the existing buckets in specific positions of the grid. This approximation is exact if the filtered density map has equal or greater resolution than the original one.

The values of g_{width} and g_{height} have to be chosen in a way such that the maximum density bucket does not exceed the desired amount of particles per packet.

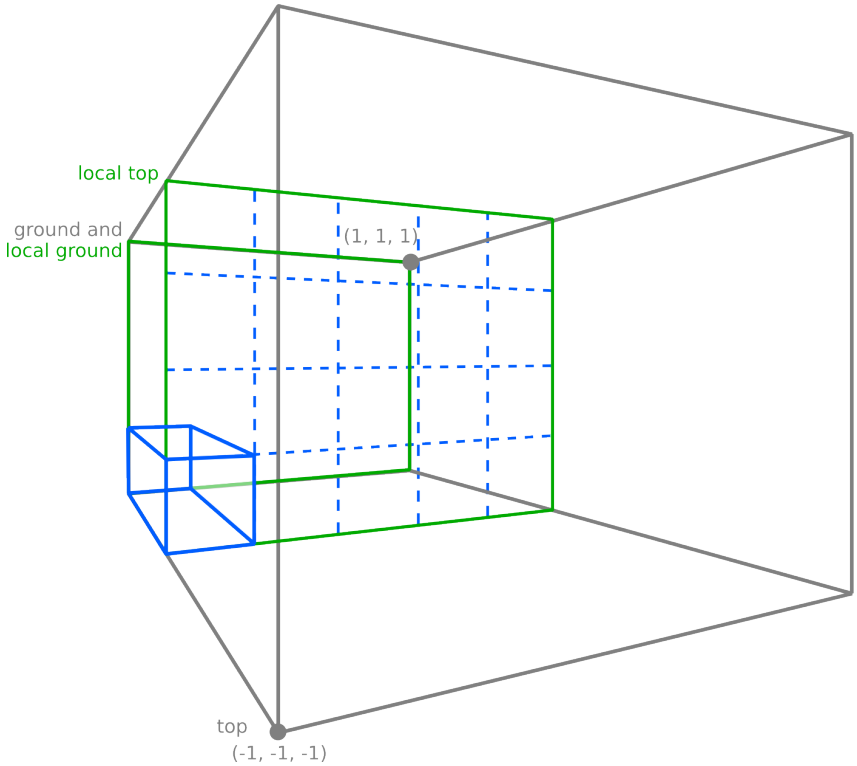


Figure 4.12: A bucket is depicted in blue inside the rain space (in grey), all elements of the figure are in clip coordinates. Bucket dimensions are computed by subdividing the local space (in green) into a regular grid of prisms, in the example it is a 5 by 4 grid. Coordinates of any bucket correspond always to the lower left prism as shown in the figure, regardless of the grid resolution or the particle density.

Automatic packet approach selection

Our present implementation of the algorithm automatically chooses one of the packing approaches taking into consideration the total amount of particles and the desired packet size: the approach that needs less packets is chosen. The rationale behind this decision is that a packet is considered to have the optimal size in terms of simulation and rendering speeds, so taking the one with fewer packets allows the system to achieve better performance.

4.4 Real-time simulation

In this last section of the present chapter, we explain the steps of the simulation that are executed in real-time. We detail them in order of precedence, so they sequentially detail the flow of the algorithm. We start with the computations done in the CPU and then proceed with the steps done in the programmable graphics hardware.

4.4.1 CPU steps

In order to optimize simulation time, the computations done in the CPU never handle particles directly, just their packets. Three main steps are entirely performed in the CPU: computation of the particle fall by checking the time elapsed since the last frame, computation of the updated position of the particles and finally selection of the packets that need to be sent to the GPU, placing them in their corresponding positions.

Time animation

To animate the drop fall, a single global parameter is used. This parameter, the *run position*, specifies how much distance from the local top the particles have advanced as a whole. Its value is normalized in $[0, 1)$ such that 1 corresponds to having advanced the whole local space height. The parameter is incremented at each frame with

$$\frac{\Delta time}{height_{local}/velocity_{fall}}$$

where $\Delta time$ is the time elapsed since the last frame, $height_{local}$ is the height of the local space and $velocity_{fall}$ is the falling speed of the drops. Finally, only the fractional part of the value is taken in order to keep it in $[0, 1)$.

For the grid packet approach, each position of the grid has a random offset in $[0, 1]$ that is added to the run position. This offset is needed in order to avoid visual particle repetition when the same bucket is rendered at various grid positions.

Height correction

As has been commented in Subsection 4.2.1, the local planes are moved up and down in order to keep the observer at an equidistant position from both of them. When particles are generated, they are placed in rain clip space assuming that the local ground coincides with the rain space ground plane (see Subsection 4.3.2). This implies that movement of the local planes requires only a translation along the Z-coordinate in the rain space clip coordinates. If h_o is the distance of the observer to the top plane (in clip space) and h_l the height of the local space (also in clip space), then the translation needed is $h_o - 1 + h_l/2$. Figure 4.13 shows how the local space is moved when a camera changes position.

A secondary effect of the previous transform, besides translating the local space along the observer's movement, is that particles also follow the observer. In order to keep them in a fixed position in the rain space regardless of the observer, the run position is modified. The rationale behind this is that, when observer moves up, the local space also does, and so particles must move down in order to stay in the same position. Respectively, a symmetric effect happens when the observer moves down. If h'_o is the observer's distance to the top plane during the previous animation frame (in clip space), then the run position must be incremented with $(h'_o - h_o)/h_l$ and then set again in $[0, 1)$. The effect of this correction is highlighted in Figure 4.13.

Packet selection and placement

The packets that are scheduled for rendering and their correct placement in world coordinates depend on the packet approach used. The world packet approach is the simplest one: the single existing packet is rendered, and its placement is done by transforming the particles by the inverse rain matrix.

In the case of the tree packet approach, placement is done in the same way, but selection takes a few extra steps. First, the culling radius around the observer is projected to the top plane, defining a rectangular area. Then, in order to optimize the selection, the frustum of the observer's camera is also projected to the top plane and a rectangle bounding it is taken as an approximation (see Figure 4.14). Then, the system searches the nodes of the packet tree that intersect with both rectangles. The leaves that do so are the packets rendered.

The grid packet approach takes some extra work. As in the tree approach, the culling radius around the observer and the frustum of the observer's camera are projected to the top plane to obtain two rectangles. Their intersection defines the area where simulation occurs. For each grid position in this area, the bucket matching the density of the position is selected to be rendered. Placement of the bucket takes two steps. First, a translation in the X and Y coordinates in clip space moves the bucket to the corresponding position of the grid. Then the inverse rain matrix sets the particles in world coordinates.

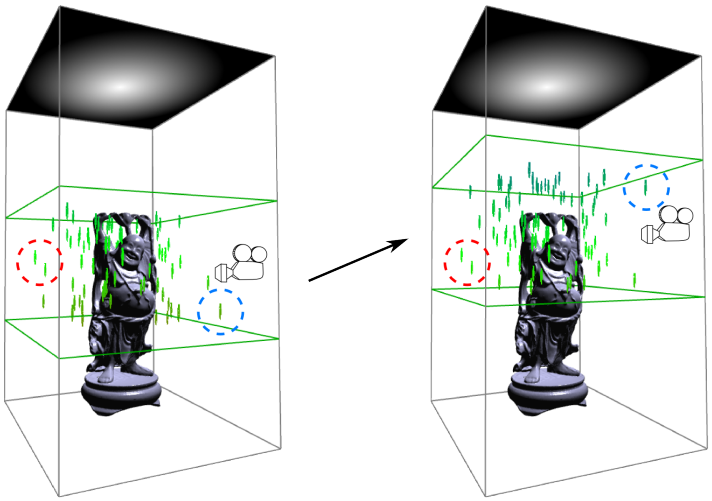


Figure 4.13: Two different renders of the same scene with different camera positions: left image has the observer lower than the right image. Local space is moved with the observer, but particles remain static in their positions thanks to the height correction (red circles). Particles that go beyond the local ground appear again at the local top (blue circles). If the observer's movement were downwards, particles would be going beyond the local top and appearing at the local ground.

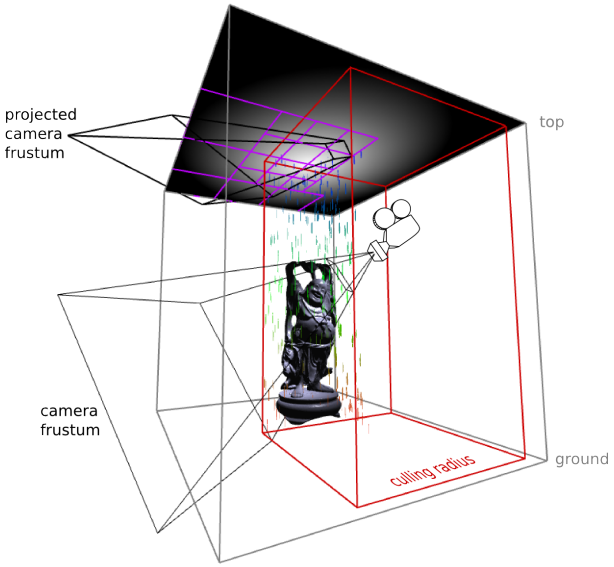


Figure 4.14: In this image we show an example of packet selection using the same scene as in Figure 4.11(b), i.e. on that has employed the tree packet approach with a quad-tree partition. Part of the partition is seen in purple at the top. The culling radius defines a frustum around the observer where simulation should occur (in red). In order to further reduce this volume, the frustum from the observer's camera is projected to the top plane (shown in black at the top). A rectangle bounding this projection defines the area of the top plane that is visible. Intersecting the packets inside this area (the ones shown in purple at the top) with the packets in the culling radius, the final set of particles to render is obtained.

4.4.2 GPU steps

The rendering state setup used for rain rendering consists in activating alpha blending, creating vertex buffer objects (VBO) for the particle packets and disabling write in the depth buffer. Alpha blending is needed in order to add the rain into the scene using transparencies. The VBO provide an efficient way to render geometry, in our case they are used to send the particles to the graphics hardware. Since the particles are not sorted by their distance to the observer, write in the depth buffer is disabled to avoid that close particles occlude the ones further away. Each particle, and thus also each vertex of the buffers, needs 3 floats for its own data. Two of the floats are used to store the particle position in the local top plane (in clip coordinates). The third float stores two different informations at the same time in order to reduce the VBO size: the particle's starting height inside the local space and also the oscillation texture assigned to the particle. These values are encoded together by just adding them. To latter recover their original values, we need to take the fractional and integer parts of the sum, the former for the starting height and the latter for the oscillation texture. This is possible since the texture identification is an integer index in $\{0, \ldots, 9\}$ and the starting height in the local space is always a value in $[0, 1)$.

All the simulation in the GPU is done through its programmable functionalities. Three stages of the programmable pipeline are used: vertex, geometry and fragment shaders.

Vertex shader

The first stage takes the original drop position and transforms it into world coordinates by applying the transforms computed in the CPU steps: first the translation with the height correction (see Subsection 4.4.1) and then the packet placement (see Subsection 4.4.1). The end result is the drop in world coordinates, at the local top. Then a translation in the fall direction is computed using the particle's starting height inside the local space and the global run position. By transforming the particle with this translation, its final position is obtained.

Instead of using the drop position to check collisions with the scene, coordinates of the rain streak billboard are computed and its top position is used for the check. By doing so, when a hit is found it is guaranteed that the rain streak is already fully below the collided surface, and thus it is no longer visible. In the collision check, the mentioned position is projected to the depth map and compared with the depth stored in it: if the value on the map is bigger, then the particle is a raindrop, else it has collided and it is either a splash or it is not visible (decided in the next stage).

The current version of our system allows the use of a static depth map as generated in Subsection 4.3.2 or a dynamic depth map. In the former case, the depth map covers the whole rain space. In the latter, the render of the scene is taken just around the observer, using the culling radius to compute the needed frustum, as depicted in Figure 4.15.

Geometry shader

This intermediate stage creates the billboards and computes lighting parameters. If the particle has collided, the drop fall velocity and the particle's depth below the hit height are used to compute how much time has elapsed since the collision (see Figure 4.2). This time and the splash animation duration are used to know which frame of the animation has to be used. If the shader detects that the animation has finished, the particle is discarded by not emitting any geometry. In any other case, a billboard is generated: for rain streak particles, a billboard perpendicular to the view vector and oriented in the fall direction is computed; for splashes, the scene's natural up vector is used as the billboard up. In order to add some variability, the splash billboards are randomly flipped and scaled by a small percentage.

The lighting parameters of Figure 4.6 are computed for rain streak particles, i.e. camera's θ_{view} elevation angle and light's θ_{light} elevation and ϕ_{light} azimuthal angles are computed for each light source. Specific subimages of the atlas corresponding to these values can be identified by taking into account how the individual textures have been placed in the atlas (see Figure 4.7). The atlas only contains renders that match discretized combinations of the angle parameters. Therefore, we round to get matches for θ_{light} and ϕ_{light}, but for θ_{view} we compute the two closest streaks

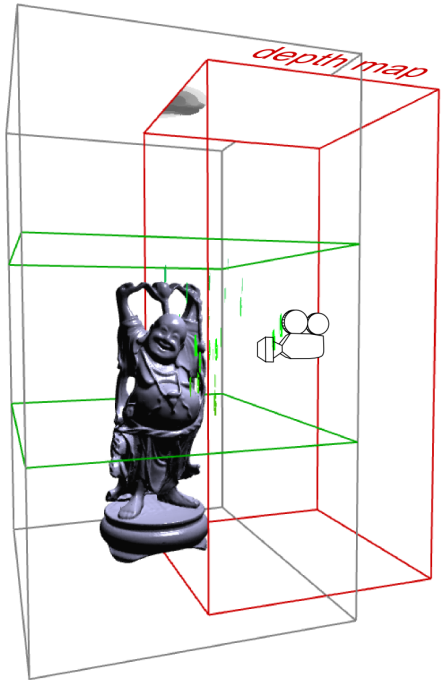


Figure 4.15: Scheme of a rain scene analogous to the one shown in Figure 4.4. As in the previous example, the red box corresponds to the culling radius volume. A render of the scene from the top of the box is used to capture the dynamic depth map, shown in the figure at the top, where only half of the Buddha model can be seen. Also notice how raindrops are only rendered in the intersection of the local space and the culling radius volume.

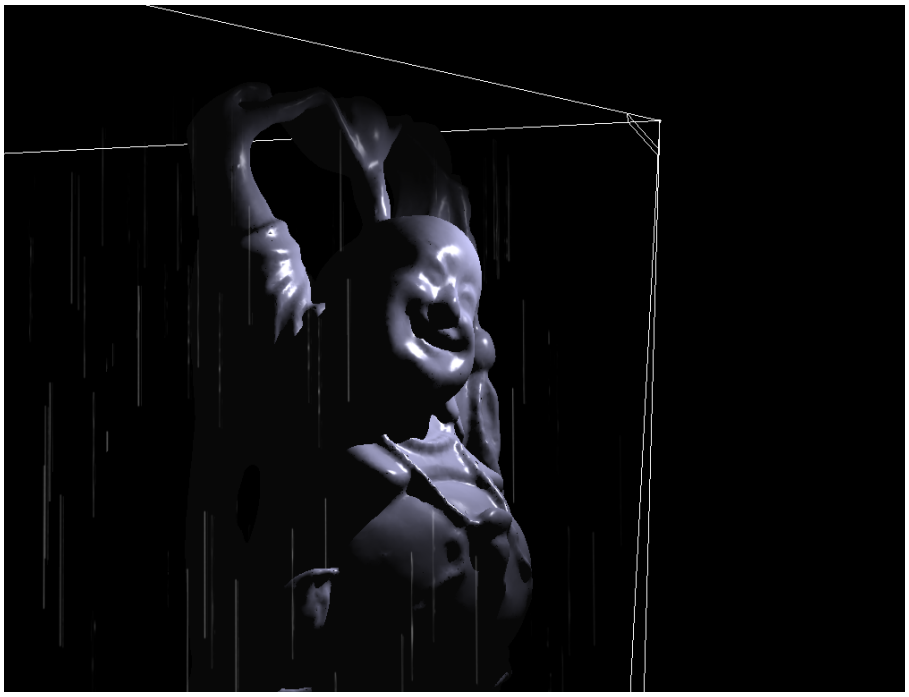
stored in the atlas in order to latter interpolate between the two of them and reduce visualization artifacts caused by the discretization. Note that environment lighting contribution only needs to consider the θ_{view}.

Fragment shader

The last stage takes two different execution paths depending on the type of particle the fragment corresponds to. For splashes, the current time in the animation is used to fetch the two closest frames, linearly interpolating between them. Then the value is lighted using Lambert's illumination for each light. In the case of rain streaks, collision of the fragment with the scene is computed in order to discard fragments of the billboard that are below the scene's surface. The fragments that are not discarded fetch, for each light source, their two textures from the database atlas, interpolate between them and finally apply Lambert's illumination. A Shadow Maps technique is used to cast shadows on the raindrops and splashes. Note that, since write in the depth buffer is disabled, rain particles cannot cast shadows on the scene nor between them.

Finally, the resulting value is blended onto the color buffer. In Figure 4.16 a final render of the example scene is presented.

(a)



(b)

Figure 4.16: Final result of the example scene with the textured rain streaks blended onto the image. The straight white lines are the single light's frustum. Notice how the streaks inside the frustum show complex illumination patterns due to the drops' oscillations. These effects are highlighted in the second image by not taking into account environment lighting in the drops. It can also be seen how the raindrops are shadowed by the figure.

Chapter 5

Results, Conclusion and Future work

In this final chapter, we evaluate how the performance of our method behaves when changing its configuration to extreme values. We start describing the test machine and the settings of the experiments, and we latter detail the tests performed and the results obtained. Finally, we discuss some of the drawbacks of our approach, its main strengths compared to the current state of the art presented in Chapter 3 and also possible future extensions to improve and optimize the simulations.

5.1 Test settings

In order to measure the performance of our implementation, series of tests have been executed using the same equipment and configuration. Our test machine is an Intel® Core™ 2 Duo CPU running at 3 GHz, with an NVIDIA® GeForce® GTX 280 with 1GB of memory. All the renders use a viewport of 1280 by 720 pixels and the same example scene is used in all the experiments (shown in Figure 5.1). The test scene is a model of a city with 779287 polygons and 149 textures (totaling 58 megapixels, not considering the mipmap reductions). Roughly, its rain space has an area of 400 square meters and it is 230 meters high. The circular gradient of Figure 4.1(b) is used to distribute the particles in the tests, making the rain denser at the city center and absent at the extremes. We have chosen a rain configuration that mimics heavy rain in order to stress the test machine. When considering the whole mentioned rain space, a total of 375139000 particles would be needed to fill our scene with heavy rain (each particle requiring 3 floats, see Subsection 4.4.2), but recall that subspace optimizations reduce this value to just a small portion of it, approximately the part inside the observer's viewing frustum (see Subsection 4.2.1). Unless stated otherwise, the local space used has 20 meters high, the culling radius has a 20 meters length (see a scheme explaining these settings in Figure 4.4), the packets store 50000 particles at most and one single point light is used. In all the tests, rain fall is completely perpendicular to the ground and collisions are checked with a static depth map of 8192 by 8192 pixels.

In order to texture the rain streaks, all the atlases generated in Subsection 4.3.1 are used. The mipmap levels of these rain streak atlases have resolutions as detailed in Table 5.1: each oscillation has a 32 by 3606 pixel atlas for its environment textures (153919 pixels in its entire mipmap) and a 2880 by 3606 for the one corresponding to the point light sources (13846232 pixels in total). Overall, 140 megapixels are needed to store the 10 oscillation textures.

The splash animation is taken from [Tatarchuk, 2006]. It consists of 21 individual frames, each of them with 6 mipmap levels using a 48 by 32 texture at its first level (see their resolutions in Table 5.2). Therefore, the whole animation and its mipmap filtering need in total 42987 pixels. The splash animation is a high speed capture of a milk drop colliding onto the floor. In that

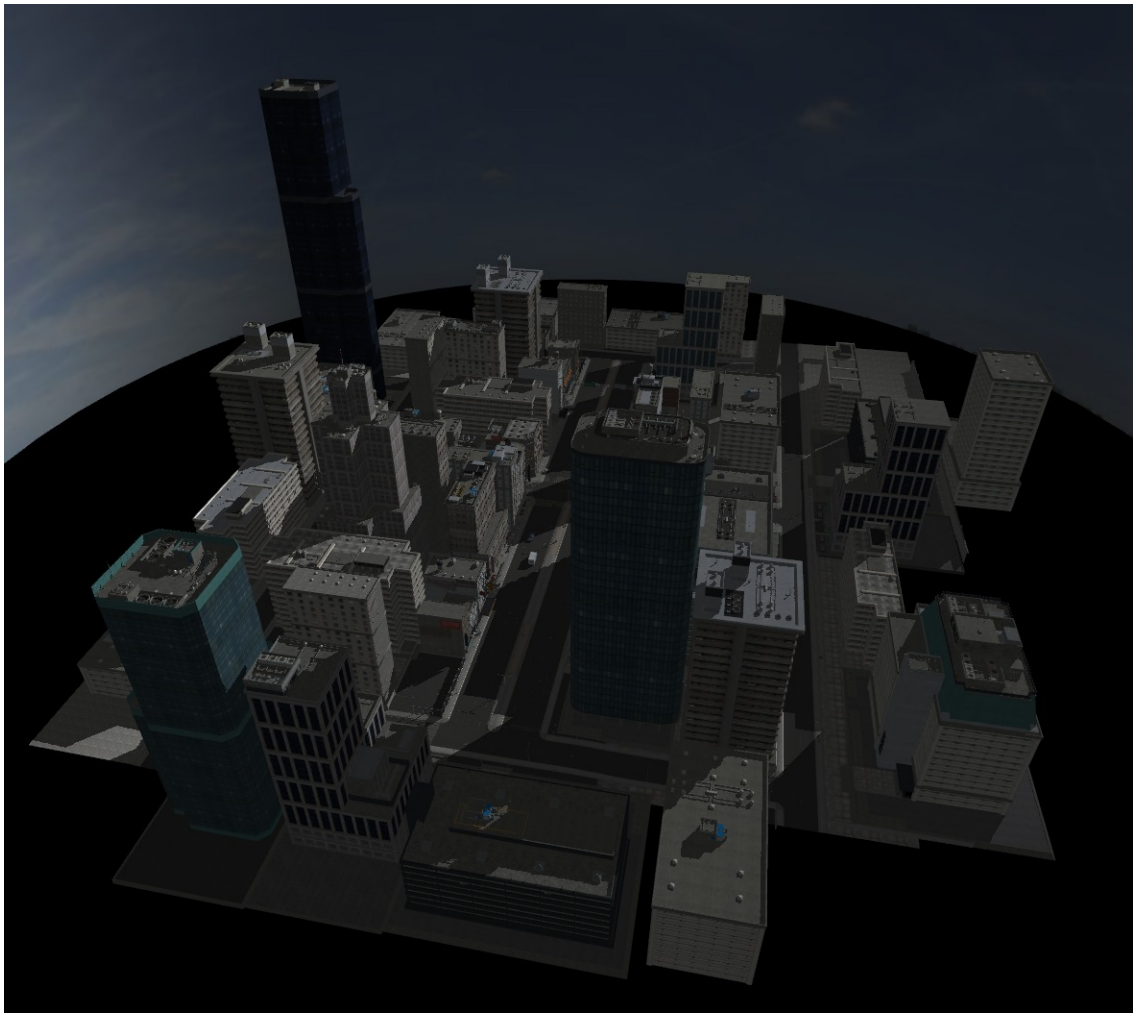


Figure 5.1: City model used in the tests, by Daz3D (www.daz3d.com).

work, the authors explain that milk is used instead of water in order to obtain clearer results when rendering.

The remaining data used in the tests, i.e. the rain particles, are different for each of our experiments and will be detailed when appropriate.

5.2 Performance

We have performed a series of different tests in order to infer how efficient our proposed rain rendering method is and to evaluate how its various parameters affect the framerate. We first check how the subspace dimensions improve simulation times (see Subsection 4.2.1 for subspace explanation), then we continue by analyzing particle packet sizes (see Subsection 4.3.3) and finally we test how the system behaves when increasing the amount of light sources.

5.2.1 Local space height

In this test we change the local space height while keeping the rest of parameters fixed, i.e. a culling radius of 20 meters, packet size of 50000 particles and one single light. Local space defines

a subvolume of the rain space where particles need to be simulated, so the narrower it is, the faster the system is able to render since less particles are considered. But this also increases visualization artifacts: missing particles con be noticed in the output. Thus a trade-off between render time and artifacts must be met for each particular scene.

Two different positions in our test scene are used in order to highlight the artifacts produced with narrow local spaces. In the first position, the sky looks almost particle-free for the 5 meter local space (see a comparison in Figure 5.2) and in the second one, a strip of particles at the horizon becomes evident when using thin local spaces (see Figure 5.3). Numerical results for both of the positions are listed in Table 5.3. As expected, performance decreases as local space is widened. In our example, 20 meters are a good compromise of visual quality and framerate.

5.2.2 Culling radius distance

As in the previous case, the culling radius directly affects performance by determining the amount of particles that are scheduled for rendering. In Table 5.4 performance results are shown for various values of the radius. Small values achieve faster framerates thanks to the smaller amount of raindrops that they need to render. Unfortunately, they also have visual artifacts: the user can perceive how rain disappears in the immediate vicinity (see Figure 5.4). Again, 20 meters come as a good trade-off between simulation fidelity and performance.

5.2.3 Packet size

In the packet size experiments, we set the camera in a fixed place in the scene and we load the same rain particles grouped in varying packet sizes. This time, visual results are the same for any of the tests since the rain density is the same (see Figure 5.5), but performance changes depending on the packets, as shown in Table 5.5. Packet size has a subtle effect on performance: when packets are too small, the system has poor performance since many rendering calls must be executed; when they are too big, packet selection (see Subsection 4.4.1) schedules for render packets that may lay mostly outside the visible frustum. This causes that, in Table 5.5, the first row of the results has worse performance than when using bigger packet sizes, and that then again framerates decrease notably when going beyond the 200000 particles per packet.

5.2.4 Number of light sources

For the light source amount test, a rain scene setup with 1057872 visible particles is used (a render is depicted in Figure 5.6). The results of Table 5.6 show that lights rapidly decrease performance. This is so because, for each light, the fragment shader blends together 4 different streak textures taken from the atlas (see Subsection 4.4.2), so the impact on performance becomes noticeable. Also memory usage is increased when considering the high resolution shadow maps of each light, diminishing cache coherence.

5.3 Discussion

Our approach, as explained this far, has a few limitations worth mentioning. One of them, already stated in Section 4.1, is that our method cannot handle volumetric data, nor treat scenes with semi-transparent geometry. This limitation comes from the fact that the rain is blended once the scene has been fully rendered, relying on its depth map in order to properly cull occluded particles. Volumetric data does not produce well-defined depth maps, so rain should be added in the same rendering step as the volumetric data instead of being blended in a postprocess of the result. Semi-transparent geometry also represents a problem since it updates the depth buffer, causing particles behind transparent objects to be occluded. No easy solution for this case is possible since raindrops are also semi-transparent, implying that a depth sorting must be performed to obtain correct results. The cost of this approach, however, is prohibitive when dealing with thousands of particles.

		Mipmap reduction level											
		0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
Environment streak	*width*	32	16	8	4	2	1	1	1	1	1	1	1
	height	3606	1803	901	450	225	112	56	28	14	7	3	1
Point light streak	*width*	2880	1440	720	360	180	90	45	22	11	5	2	1
	height	3606	1803	901	450	225	112	56	28	14	7	3	1

Table 5.1: Texture resolutions in pixels of the atlases generated in Subsection 4.3.1. Each oscillation has 153919 pixels for the environment textures and 13846232 for the point light sources. The ten available oscillations total 140 megapixels.

		Mipmap reduction level					
		0	*1*	*2*	*3*	*4*	*5*
Splash	*width*	48	24	12	6	3	1
	height	32	16	8	4	2	1

Table 5.2: Resolution in pixels of each frame of the splash animation, taken from [Tatarchuk, 2006]. To store the whole 21-frame animation with mipmaping, 42987 pixels are needed.

Height	Total particles	First Camera			Second Camera		
		Particles	FPS	FPS *(ng)*	Particles	FPS	FPS *(ng)*
5	13744500	430116	37	56	643442	33	44
10	27489100	426718	35	51	472634	36	49
15	41233600	518903	32	44	786653	28	35
20	54978100	776864	25	32	985408	23.5	28
25	68722700	987093	21	26	1200496	20.5	24
30	82467200	1117591	20	24	1426966	18	21
35	96211700	1218490	18	22	1611413	16.5	18.5
40	109956000	1534222	15	18	1956267	13.5	15
230	625231000	7612526	4	4.2	10307565	3	3

Table 5.3: Results obtained in Subsection 5.2.1 to test how local space height influences performance. The first column is the height of the local space (in meters) and the second column the total amount of particles inside it. Two different positions of the camera in the scene have been used, such that the so called Second Camera is able to see particles further away than the other one. For each camera position, the table details the amount of visible particles, the framerate when rendering everything and the framerate when no geometry is sent to be rendered. The last row entry of the table shows performance when local space equals the rain space. Recall that, even in this case, we use a culling radius, so only part of the whole rain space needs to be simulated.

Culling radius	5	10	15	20	25	30	35	40
Particles	70303	198818	395778	784812	1204536	1659172	2190468	3133179
FPS	60	53	40	27	20	16	12.5	9
FPS *(ng)*	200	100	60	37	25	18	14	10

Table 5.4: Results of Subsection 5.2.2 that show how the culling radius (in meters) affects the performance. In the second row, the amount of particles inside each culling radius is detailed, from a total of 54978100 particles in the whole local space. Third and fourth rows are the frames per second when everything is rendered or when discarding geometry, respectively.

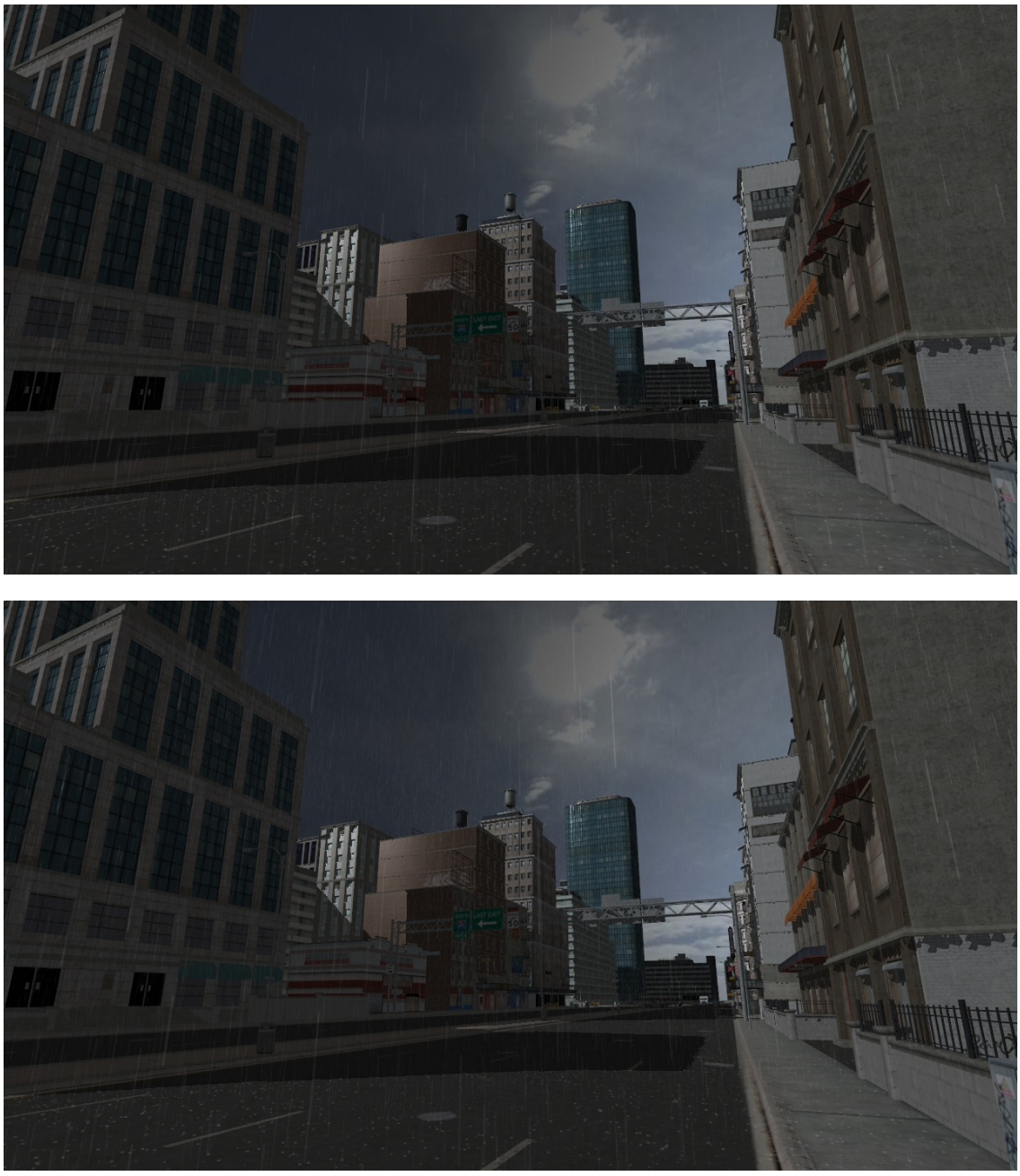


Figure 5.2: The images depict renders for the first camera position of the local space tests of Subsection 5.2.1. Top image uses a 5 meters height, bottom image a 20 meters. Notice how the second one has more particle density when looking at the sky.

As discussed in the tests of the local space (Subsection 5.2.1) and culling radius (Subsection 5.2.2), the subspaces provide an efficient way to limit the simulation volume at the price of possibly introducing visual artifacts: rain disappearing close to the observer. Just increasing their volume to solve this problem does not completely fix the handicaps since rendering performance is diminished. Computing tighter bounds of the visible volume would be a better approach to improve performance, but our current rain simulation model is highly dependent on the present

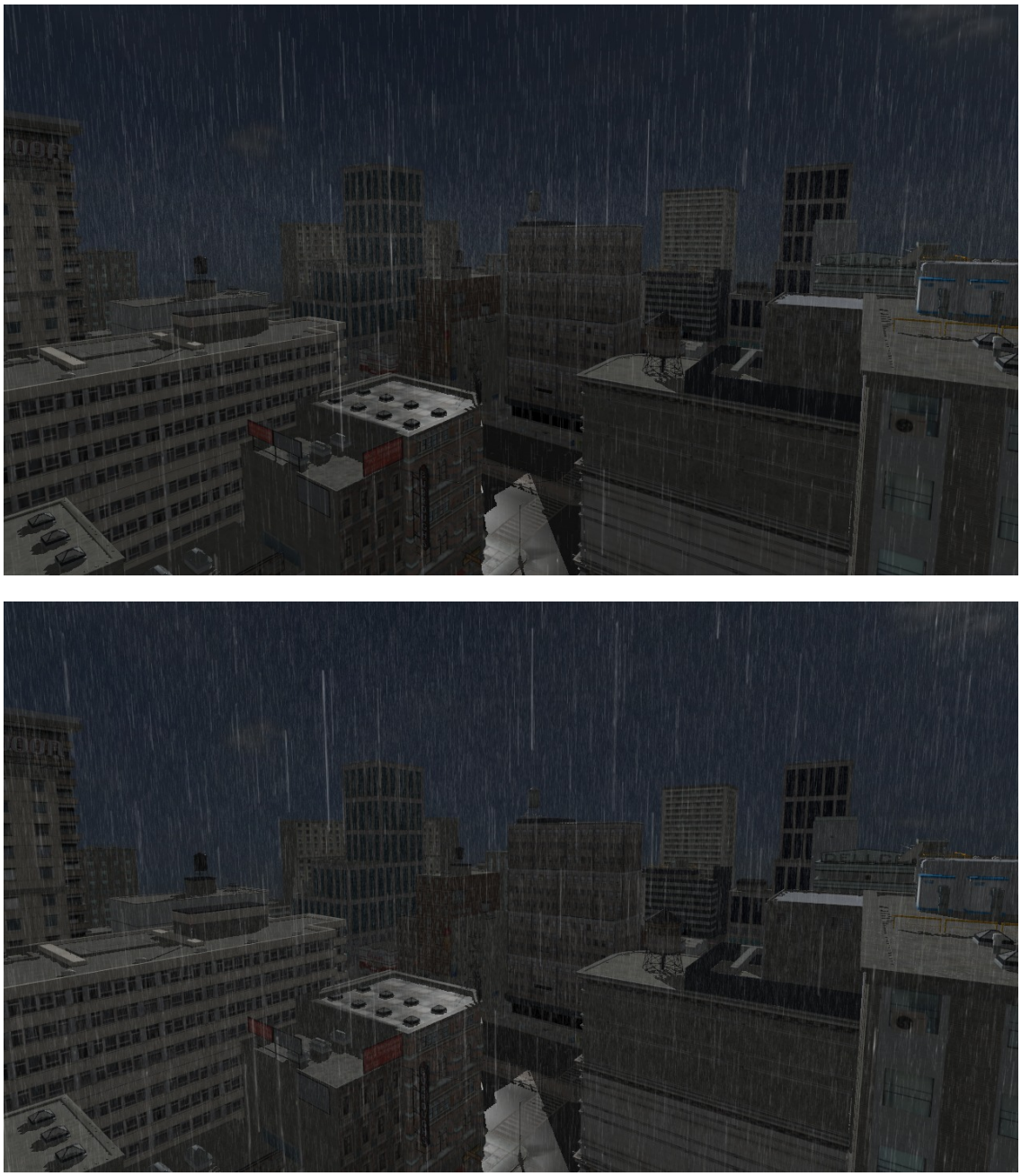


Figure 5.3: Same as Figure 5.2 but at the Second Camera position of the test of Subsection 5.2.1. The top image corresponds to the test with 5 meters height for the local space. Notice how a strip of far away particles can be seen at the horizon (subte effect).

subspace organization, constraining the available options. For instance, it is important to highlight that, while culling radius can be freely defined in real-time in our implementation, the local space height is used as a parameter for the particle generation. Thus, adjustment of the local space becomes cumbersome since particles need to be recreated at each modification.

Finally, particle render should be further studied. Currently, splashes use a single animation, thus becoming repetitive. In the case of rain streaks, a 140 megapixel texture array is used when

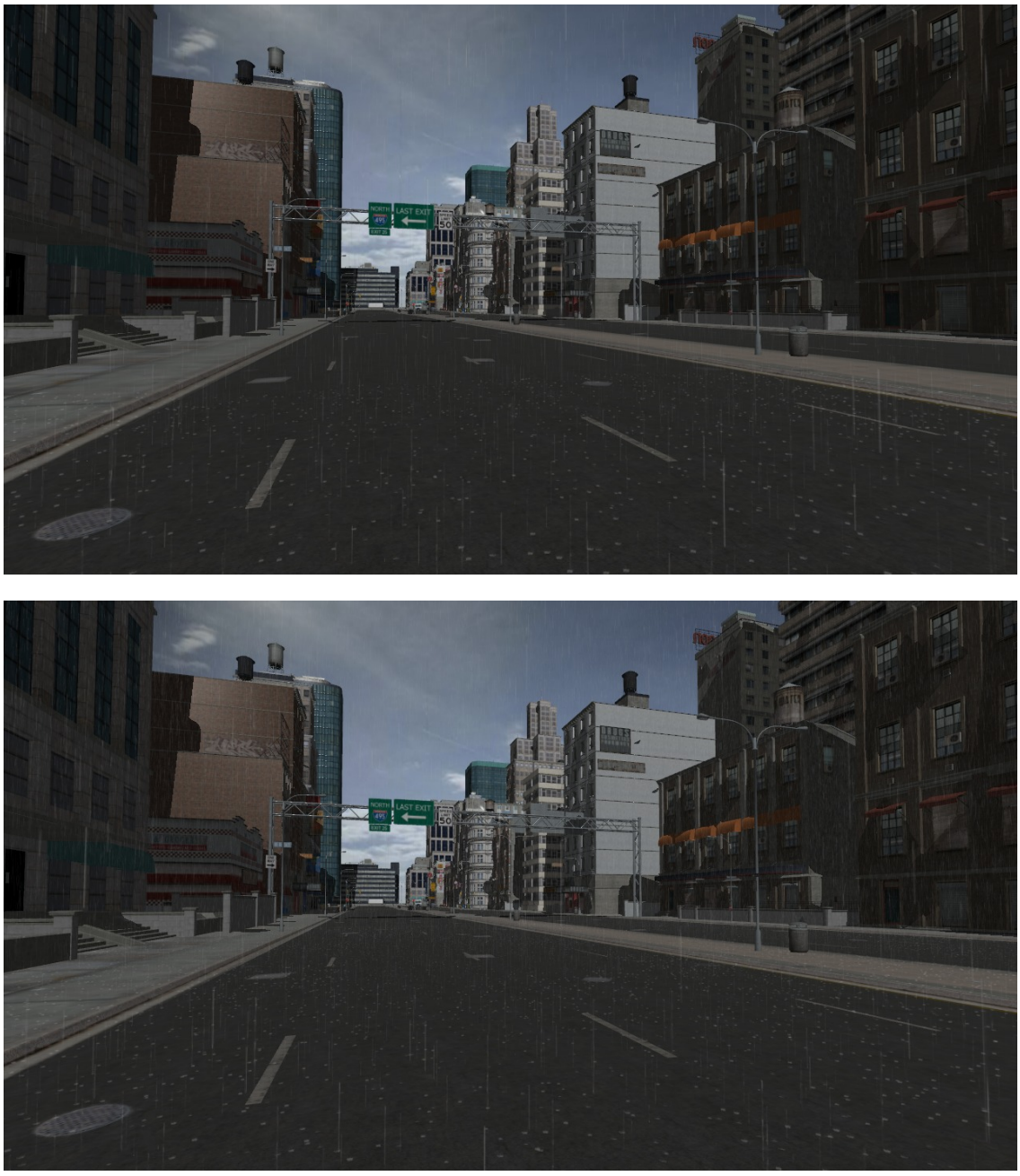


Figure 5.4: Renders obtained during the culling radius tests of Subsection 5.2.2. The first image uses a radius of 5 meters; notice that rain disappearing in the vicinity can be perceived (specially by looking at the splashes on the ground). Second image uses a 20 meter radius and this artifact is no longer visible.

influence on screen of individual drops is small. Analysis of the texture array usage could highlight that parts of its information are wasting resources because they are hardly used during rendering, thus pointing out how to optimize the atlas generation for specific scenes.

Packet size	Particles	FPS	FPS *(ng)*
100	895322	22.5	32
1000	938875	28	36
5000	938875	26.5	34
25000	934112	26.5	33.5
50000	999467	25.5	32
100000	1055737	24	29.5
150000	981267	24.5	33
200000	1190512	22.2	30
250000	2532449	11.5	13
500000	3787474	7	7.5
1000000	5094436	5	5

Table 5.5: Results for the packet size analysis of Subsection 5.2.3. The first column shows the sizes considered, the second the amount of particles rendered and the third and fourth columns detail the frames per second obtained when rendering everything and when ignoring geometry, respectively. Notice how small packets (100 particles) have bad performance, and that, when size is bigger than 200000 particles, performance decreases again in our test machine.

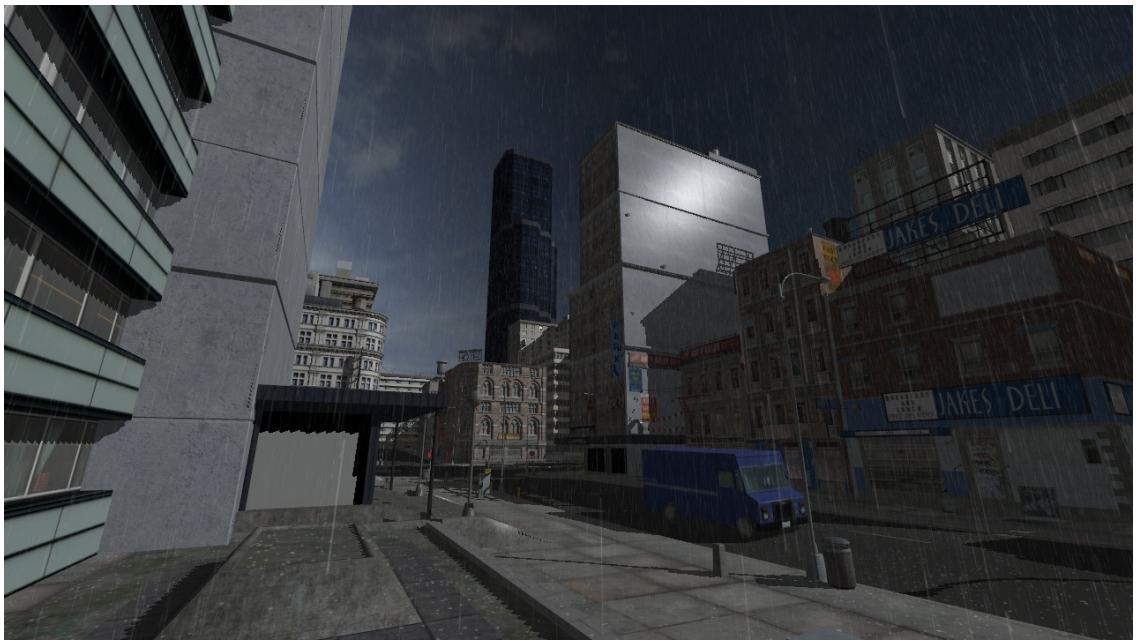


Figure 5.5: Part of the scene where packet size is tested. As mentioned in Subsection 5.2.3, this time all the tests have the same visual quality, regardless of the packet size, since roughly the same particles are rendered. In the figure, the 50000 particle packet test is shown.

5.4 Conclusion

We have presented a new approach that builds on existing state of the art methods ([Garg and Nayar, 2006; Tariq, 2007]) and improves them by considering extra significant phenomena. In particular, we are able to use the entire database of [Garg and Nayar, 2006] instead of simplifying the illumination parameters as done in [Tariq, 2007], thus achieving high quality images with complex illumination effects (shown in Figure 5.7). We also apply shadowing techniques in the particle illumination to enhance the results, see Figure 5.8 for an example.

Light sources	1	2	3	4	5	6
Geometry	21.6	12.5	8.8	4.5	1.66	1.51
No geometry	24.5	13.7	8.8	4	1.36	1.34

Table 5.6: Results of Subsection 5.2.4 analyzing performance impact of the light sources. First row corresponds to the amount of active lights in the scene, second and third rows are the framerates obtained.

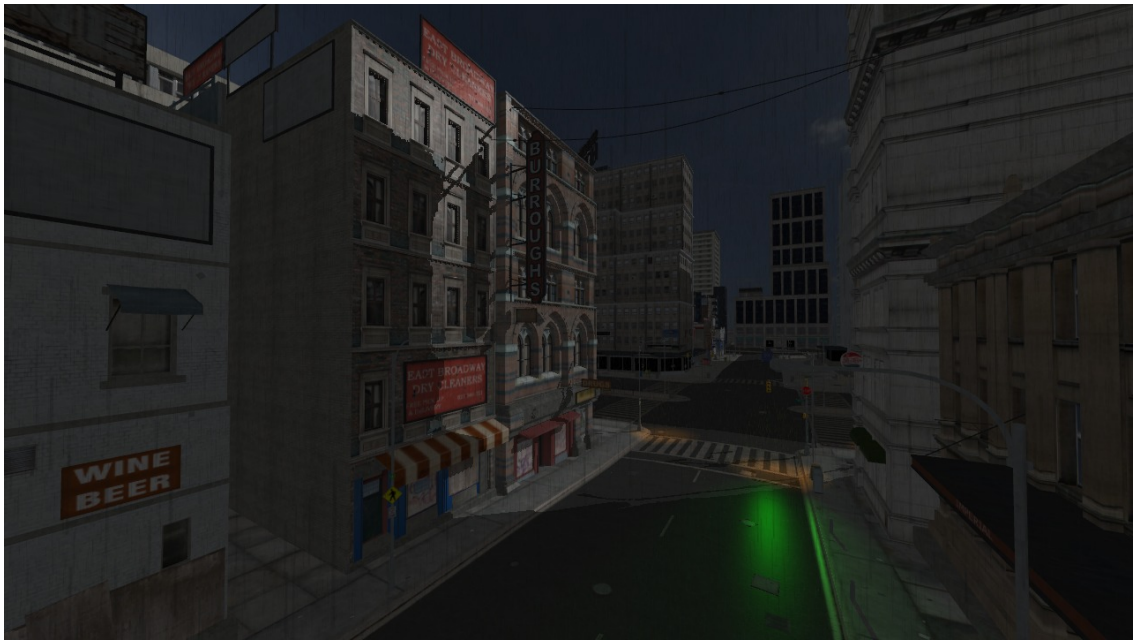


Figure 5.6: Render of the scene using 6 different light sources, two of them are almost occluded by the geometry, another one is the global sun light, the remaining three of them are evident in the image. Notice that our system does not render the actual light source, so they are invisible except for the lighting effects they cause in their surroundings.

Moreover, our method is designed to allow easier rain configuration and more precise scene interaction than previous approaches. For the former, we have devised a particle system that places each particle following an easy to create density map, thus simplifying artistic distribution of rain in a scene. For the latter, we have considered the particles as an integral part of the scene and so we simulate them accordingly. In particular, we take into account the geometry to avoid rendering rain inside solid objects and to produce splashes on the ground. The splashes are actually rendered in the exact collision points, being the first real-time method that has a direct correlation between a raindrop particle and its splash. Collisions are computed with a depth map that, in our implementation, can be updated periodically to handle dynamic scenes.

Finally, another novel aspect of our algorithm is that it automatically adapts the simulation volume in order to increase performance by reducing the amount of particles to render. Even though the computations are conceptually tricky, involving a few space transformations, their implementation is straightforward and implies a small amount of operations. Therefore, we are able to execute them per-frame even when handling over a million of particles, although, in our experiments, fewer particles were enough to simulate heavy rain without noticeable artifacts. The remaining costly computations of the simulation are done per particle and are left to be executed in the programmable graphics hardware, increasing performance.

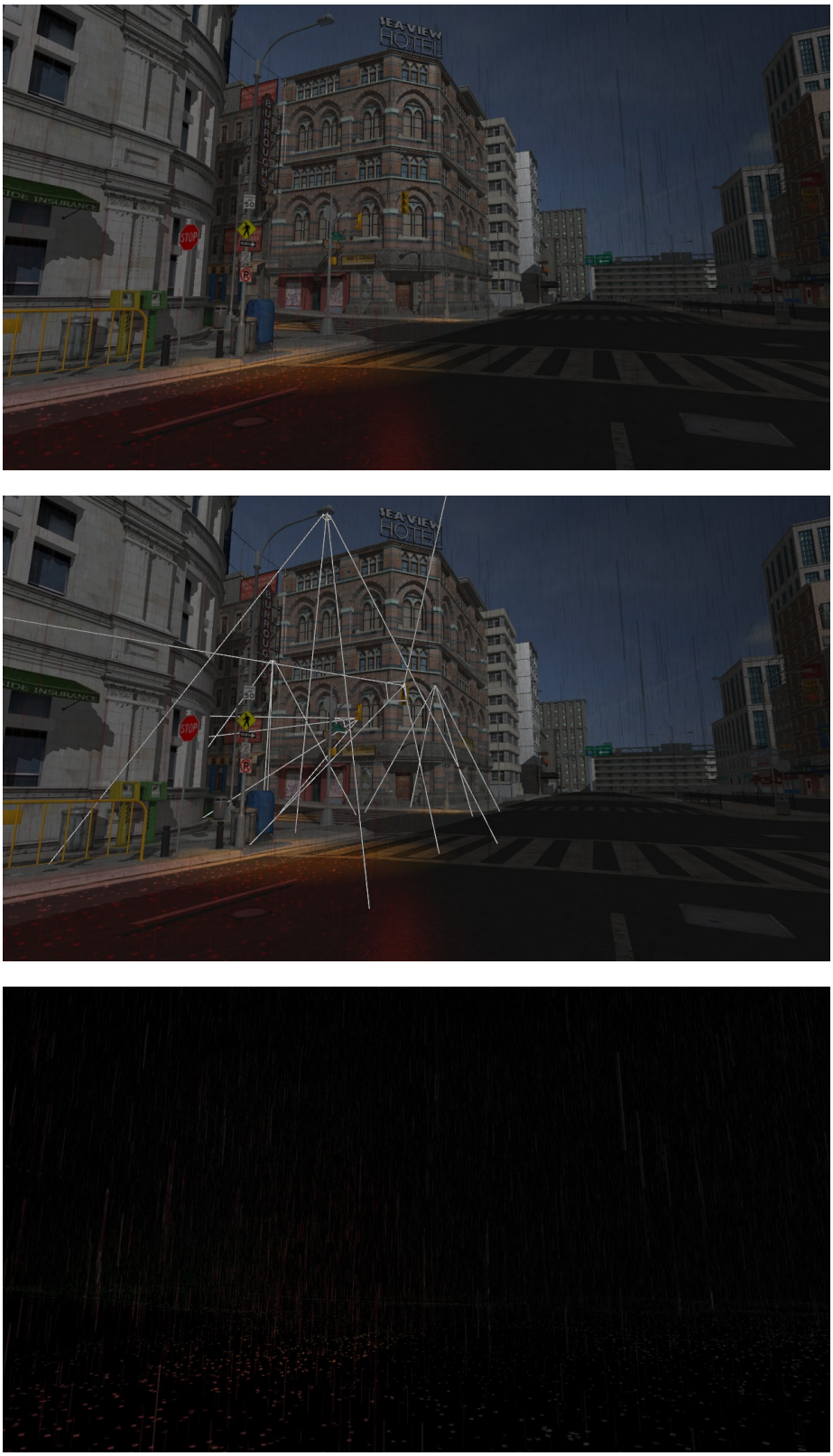


Figure 5.7: Scene lighted with six different sources, their frustums are shown in the central image: red and green lights for the semaphores, yellow for the three street lamps and white for the sun. Notice in the first image how rain streaks and splashes on the left part of the picture are both red and yellow colored, while on the right part they are in the shadow and look darker. In the third image particles are highlighted by discarding the geometry and not casting shadows.

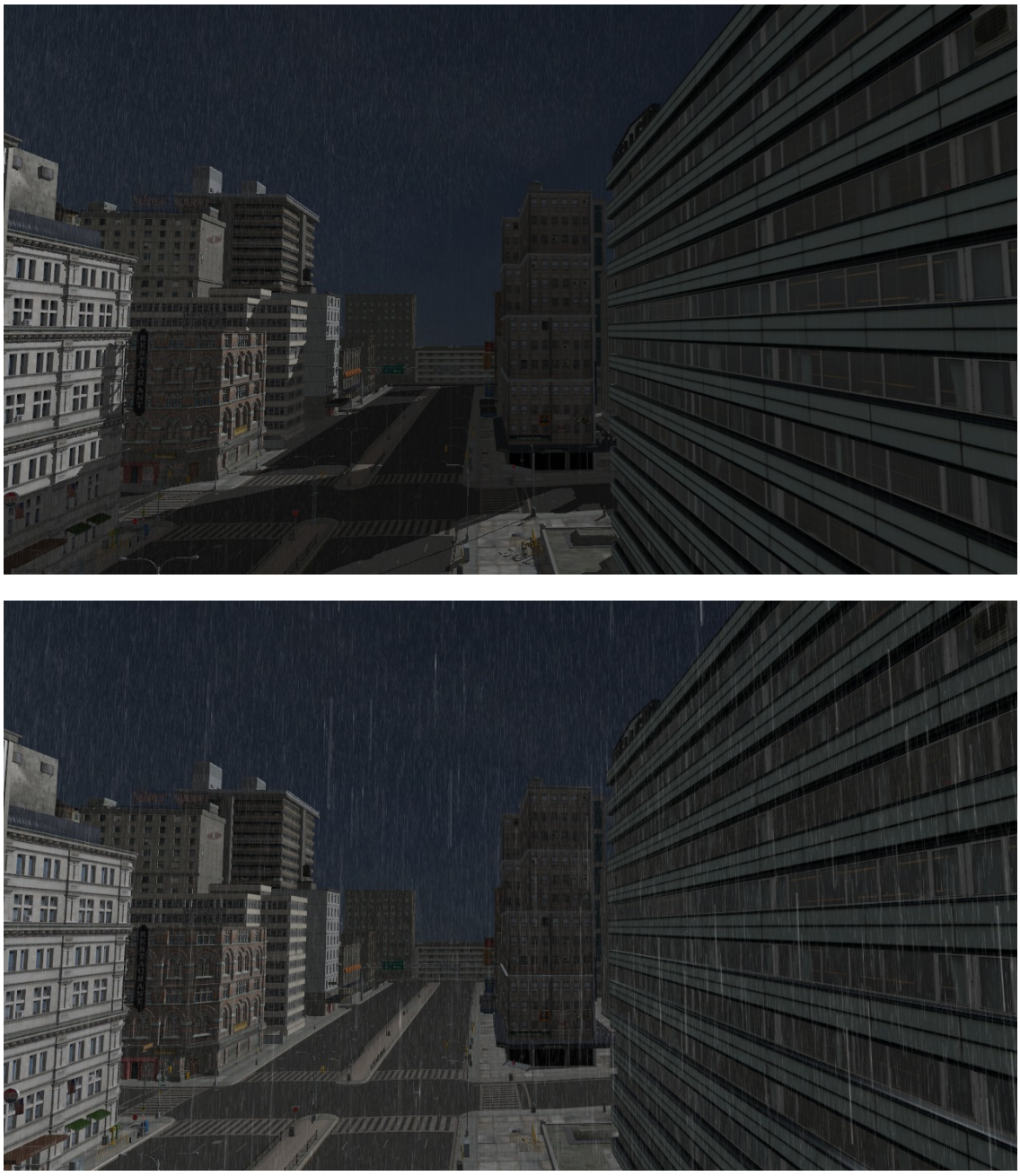


Figure 5.8: In the first image, the system casts shadows for the sun's light (which is located beyond the top right corner of the image). Rain streaks in front of the sky seam to disappear when reaching the building's shadow. In the second image, the shadows are ignored and particles become visible.

5.5 Future Work

In Section 5.3, we have enumerated the main flaws of the current proposal. Some of them are intrinsic to how we handle the particles and thus cannot be solved unless we change the whole system. Other ones, however, could be improved by performing deeper analysis on some of their characteristics. For instance, tighter bounds on the rendering volume would increase the framerate

without reducing image quality. Currently, rectangles are used to bound the projection of the observer and its viewing frustum onto the top plane. Computing their convex hull instead of rectangles is a future avenue of work that would reduce the final simulation area. Moreover, further adaptive techniques could be employed: far away particles do not require illumination computations as complex as the ones in the foreground, so they could be approximated using methods similar to the ones in Subsection 3.2.1, i.e. scrolling textures with precomputed rain streaks. This would effectively change the algorithm to consider two kinds of drops, the ones close to the observer rendered in high detail and the ones far away, that would use rough textures.

Our current approach for rendering splashes consists on just repeating a single real splash capture, regardless of the surface material, its wetness and angle of impact. More realistic splash models are analyzed in [Garg et al., 2007] and could be adapted to real-time simulation by precomputing a discretized amount of animations and storing them in atlases, following the same idea used for our rain streaks. A partition of the scene in different material types and the surface normal at the collision points could be used to select specific splash animations at the appropriate times.

Finally, we would also like to study methods to automatically compute subspace minimum dimensions that have no visual artifacts, instead of manually testing and fine-tuning them for each scene. This work would previously need to define a way to identify and measure errors, in order to latter adapt the subspace parameters using automatic evaluations. The artifacts due to incorrectly parameterized subspaces are made worse when rain is not fully perpendicular, but slanted. This is so because, in this case, the local space and the culling radius are also slanted with respect to the scene, so their volumes do not match so well the observer's view frustum, making artifacts more evident. Handling this case is important to widen the range of rain types that can be efficiently simulated without simulation errors.

Appendix A

Physics

Raindrops are a complex dynamic phenomenon that has been widely studied. Research in this area focuses on its dynamics and the optic properties that raindrops' shapes have. Specific studies on the shape of the drops can be found in [Lenard, 1904; Magono, 1954; Pruppacher and Beard, 1970; Beard and Chuang, 1987; Chuang and Beard, 1990; Ross and Bradley, 2002]. The aerodynamic pressure exercised on the drop deforms it as it falls, losing its spheric shape. Small drops below 0.5 mm retain almost a spherical form, while greater drops become ellipsoidal (see Figure A.1). This distortion changes its aerodynamic properties, causing more turbulent regimes in the air and slowing down the fall. In [Gunn and Kinzer, 1949; Ross, 2000] this speed at ground level has been studied and tabulated into various categories. Their results show that spheric drops (when radius is between 0.1 mm and 0.45 mm) have terminal velocities between 0.72 m/s and 3.67 m/s, bigger ellipsoidal drops (when radius is between 0.5 mm and 4.0 mm) can reach speeds from 4.0 m/s to 9.23 m/s. Even bigger drops are considered in other experiments, but they tend to split due to aerodynamic forces breaking them. In [Best, 1950] the authors found that raindrop size follows an exponential two-parametric distribution. The experiments mentioned are taken in laboratories with specific conditions, which usually include the use of distilled water, control of the atmospheric pressure and only analyzing drop falls through stagnant air. This implies that, in real rain conditions, drops may differ from the results found.

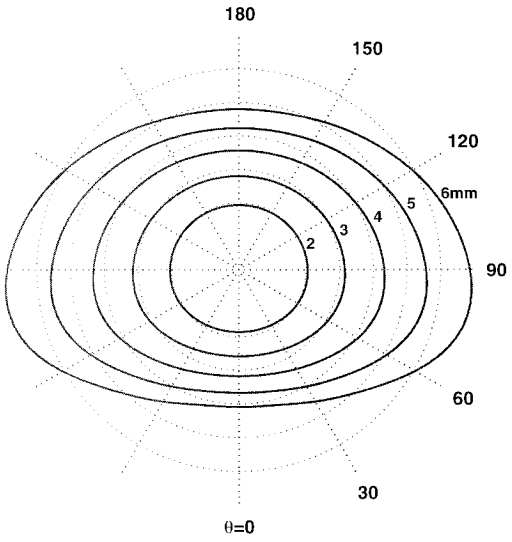


Figure A.1: Vertical slice of a raindrop, showing its shape as a function of its size. From [Ross and Bradley, 2002].

Splashes due to drop collisions are analyzed in [Stow and Stainer, 1977]. In this work the authors study the amount, size and distribution of the droplets produced in the splashes. They also consider the effect that the drop size and speed have on the amount of emitted droplets. These results along with empirical experimentation are used in [Garg et al., 2007] to infer a splash model. Although this model is not used in the present proposal, future extensions could improve fidelity by employing it in the simulations.

Bibliography

K. V. Beard and C. Chuang. A new model for the equilibrium shape of raindrops. *J. Atmos. Sci.*, 44:1509–1524, 1987.

A. C. Best. The size distribution of raindrops. *Quarterly Journal of the Royal Meteorological Society*, 76(327):16–36, 1950.

George Borshukov. Making of the superpunch. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 19, New York, NY, USA, 2005. ACM. doi: http://doi.acm.org/10.1145/1198555.1198599.

Wang Changbo, Zhangye Wang, Xin Zhang, Lei Huang, Zhiliang Yang, and Qunsheng Peng. Real-time modeling and rendering of raining scenes. *Vis. Comput.*, 24(7):605–616, 2008. ISSN 0178-2789. doi: http://dx.doi.org/10.1007/s00371-008-0241-0.

C. Chuang and K. V. Beard. A numerical model for the equilibrium shape of electrified raindrops. *J. Atmos. Sci.*, 47:1374–1389, 1990.

Kshitiz Garg and Shree K. Nayar. Photorealistic rendering of rain streaks. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 996–1002, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. doi: http://doi.acm.org/10.1145/1179352.1141985.

Kshitiz Garg, G. Krishnan, and Shree K. Nayar. Material Based Splashing of Water Drops. In *Proceedings of Eurographics Symposium on Rendering*, Jun 2007.

Ross Gunn and Gilbert D. Kinzer. The terminal velocity of fall for water droplets in stagnant air. *Journal of Meteorology*, 6:243–248, 1949.

Daniel Herman. Rainman: Fluid pseudodynamics with probabilistic control for stylized raindrops. In *Conference Abstracts and Applications ACM SIGGRAPH 2001*. ACM, 2001.

Kazufumi Kaneda, Shinya Ikeda, and Hideo Yamashita. Animation of water droplets moving down a surface. *Journal of Visualization and Computer Animation*, 10(1):15–26, 1999.

P. Lenard. Über regen. *Meteorol. Z.*, 21:249–260, 1904.

C. Magono. On the shape of water drops falling in stagnant air. *J. Meteorol.*, 11:77–79, 1954.

H. R. Pruppacher and K. V. Beard. A wind tunnel investigation of the internal circulation and shape of water drops falling at terminal velocity in air. *Q. J. R. Meteorol. Soc.*, 96:247–256, 1970.

Anna Puig-Centelles, Oscar Ripolles, and Miguel Chover. Creation and control of rain in virtual environments. *Vis. Comput.*, 25(11):1037–1052, 2009. ISSN 0178-2789. doi: http://dx.doi.org/10.1007/s00371-009-0366-9.

Oliver N. Ross. Optical remote sensing of rainfall micro-structures. Freie Universität Berlin Master's Thesis, 2000.

Oliver N. Ross and Stuart G. Bradley. Model for optical forward scattering by nonspherical raindrops. *Applied Optics*, 41(24):5130–5141, 2002.

Pierre Rousseau, Vincent Jolivet, and Djamchid Ghazanfarpour. Realistic real-time rain rendering. *Computers & Graphics*, 30(4):507–518, 2006.

C. D. Stow and R. D. Stainer. The physical products of a splashing water drop. *Journal of the Meteorological Society of Japan*, 55:518–531, 1977.

Sarah Tariq. Rain. NVIDIA White Paper, 2007.

Natalya Tatarchuk. Artist-directable real-time rain rendering in city environments. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 23–64, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. doi: http://doi.acm.org/10.1145/1185657.1185828.

Huamin Wang, Peter J. Mucha, and Greg Turk. Water drops on surfaces. *ACM Trans. Graph.*, 24(3):921–929, 2005. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1073204.1073284.

Lifeng Wang, Zhouchen Lin, Tian Fang, Xu Yang, Xuan Yu, and Sing Bing Kang. Real-time rendering of realistic rain. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 156, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. doi: http://doi.acm.org/10.1145/1179849.1180044.

Niniane Wang and Bretton Wade. Rendering falling rain and snow. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 14, New York, NY, USA, 2004. ACM. ISBN 1-59593-896-2. doi: http://doi.acm.org/10.1145/1186223.1186241.