Applying Volumetric Sampling Methods to the Rendering of Rain Droplets

October 10, 2024

AUTHOR
Mina Spijk

FIRST SUPERVISOR
Peter Vangorp

SECOND EXAMINER
Alexandru C. Telea

EXTERNAL SUPERVISOR
Jacco Bikker

Department of Information and Computing Sciences
Game and Media Technology
Utrecht University

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science

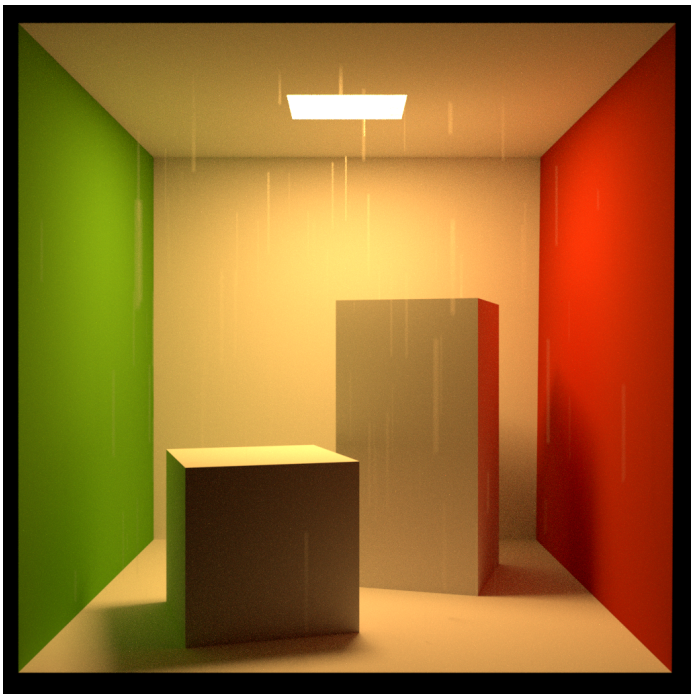


Figure 1: Cornell box with rain inside the box. The chance to hit a raindrop has been increased to make the result more visible.

1 Introduction

Rain is a common thematic element in media. It might be used to build tension, such as before the Battle of Helms Deep in Peter Jackson's *The Lord Of The Rings: The Two Towers* (2002). It can also be used to evoke emotions, as seen in Disney's *The Lion King* (2019). Rain is also frequently used to add a sense of ambience and realism, often seen in open-world games such as Nintendo's *The Legend of Zelda: Breath of the Wild* (2017) or Playground Games' *Forza Horizon 5* (2021). While live-action movies can be recorded in real rain or simulate rain using a sprinkler system, animated movies and games need to rely on computer-generated rain.

Although convincing techniques exist for rendering fog, rain clouds, wet surfaces and rain splashes, any existing methods of rendering the falling rain itself tend to appear unrealistic. This is not surprising: rain is a complicated visual phenomenon consisting of an incredible amount of raindrops, all falling at high speeds. Simulating every individual raindrop in a storm is infeasible, and existing methods tend to consider only the drops closest to the viewer.

This project aims to describe a method to render raindrops at a larger scale by applying volumetric sampling methods. These methods are already commonly used to render mist and smoke effects, as well as volumes for which the

Figure 2: Rainy scenes from various movies and games.
Top left: Peter Jackson's *The Lord Of The Rings: The Two Towers* (2002). Top right: Disney's *The Lion King* (2019). Bottom left: Nintendo's *The Legend of Zelda: Breath of the Wild* (2017). Bottom right: Playground Games' *Forza Horizon 5* (2021).

appearance is based on procedural generation. Raindrops have such a high density and large size that they cannot be treated as a volume directly, but this project aims to show the motion blur effect seen for drops falling at terminal velocity does allow treating a rain *streak* as a volume.

2 Literature Study

Information on the visual effects of rain exists across several domains, and a number of existing techniques are capable of rendering raindrops. Several techniques also exist for the rendering of effects that occur during rain that are separate from falling raindrops. In this section, relevant literature on these topics is discussed.

2.1 Physical properties of rain

The physical properties of rain have been researched for a number of reasons, including meteorological research and to research the impact of rain on crops. In this section the focus is on understanding the physical properties of which the effect can be observed by the human eye or a standard photo camera.

2.1.1 Velocity of raindrops

The terminal velocity of an object can be calculated by equating the gravitational force to the drag forces acting upon the object. Stokes' law describes the drag forces acting upon spherical objects falling in some medium and can thus

be used to calculate the terminal velocity of rigid spheres as

$$U = \frac{2}{9}\frac{\rho' - \rho}{\mu} g a^2$$

where U is the terminal velocity, ρ and ρ' the density of the sphere and the medium respectively, μ the dynamic viscosity of the medium, g the gravitational acceleration, and a the radius of the sphere [36, p. 589]. While it may be logical to interpret a water drop as such a sphere, the reality is that raindrops experience deformation while falling. This makes it so Stokes' law is not a perfect description of the terminal velocity for all raindrops.

The terminal velocity of water drops has been measured across multiple publications using different methods and under different circumstances, and a large number of these can be found in a survey paper by Serio et al. [57]. Notable is the paper by Gunn and Kinzer [22] due to its influence on future publications. The authors measure the terminal velocities of water droplets of various mass without disturbing the motion of the droplet itself. This paper concludes drops with a mass of less than 0.251 micrograms obey Stokes' law, and drops larger than 100,000 micrograms split into smaller drops when falling long enough. The terminal velocities of drops of sizes between these two extremes are tabulated and ranges between 0.18 and $9.17 m/s$, a rather broad range. Also useful may be the observation that the terminal velocity of a drop ranges between approximately 1600 and 4100 times its own diameter in a single second.

Foote and du Toit [14] generalise the measurements of Gunn & Kinzer, as well as measurements by Davies [12], to convert the terminal velocity of a drop in "standard conditions" (as used in Gunn & Kinzer's measurements) to velocities appropriate for a given air density and temperature.

Later papers [13, 69, 6, 2]. derive formulas fitted to Gunn & Kinzer's data to calculate the terminal velocity of a drop in "standard conditions" from the size or mass of a drop. All of these formulas are equally suited for the purposes of this thesis, though a preference may be expressed for the implementation of Wobus et al. [69] due to the relative simplicity of their formulas.

Montero-Martínez and García-García [42] show that raindrops can fall at speeds higher or lower than their terminal velocity when affected by wind. This effect is significant for drops smaller than 0.7mm in diameter. Prediction of the strength of this effect has not been sufficiently researched to simulate this difference without complex simulations. Given the small size of the affected drops and the difficulty with simulating this effect, replicating this effect is considered out of scope for this project.

It can be assumed that all raindrops falling from the sky have reached terminal velocity when seen by an observer. It may be desirable to be able to render falling water drops originating from different sources, such as water dripping from an umbrella or tree leaves, using the same techniques. Wang and Pruppacher [66] show that most drops fall a distance of at least 12 meters before reaching terminal velocity.

Starik and Werman [58] suggest that due to the high terminal velocity of raindrops, drops can be considered temporally independent across frames. Putting

frames from a rain scene in a random order or flipping the frames upside-down is shown to have no influence on human perception of rain. It should however be noted that the videos shown on the accompanying website use a framerate of 15 frames per second as compared to the more standard 24 frames per second for movies or 60 frames per second for video games. It is unclear whether the conclusion of Starik and Werman holds at higher frame rates. If the conclusion does hold, then it is acceptable for rain rendering systems to omit the continuity of raindrops across frames. In other words, raindrops could be rendered at random positions in every frame of an animation without losing realism.

Raindrops can gain a horizontal velocity due to the presence of wind, but little research exists on this topic outside of considerations for building construction (e.g. [10]). The horizontal velocity of particles being blown around by wind in general is better understood and can be applied to rain. Giudice et al. [21] use a turbulence model to simulate particles being transported in wind. These methods are often complex and implementing them is not trivial.

2.1.2 Diameter of raindrops

The diameter of raindrops in a rainstorm varies not only between different rainstorms, but also between different raindrops within the same rainstorm. The distribution of raindrop sizes is commonly referred to as the Drop Size Distribution, or DSD.

One of the first measurements of the DSD was performed in 1895 by Wiesner [67], using absorbent paper covered in dye to determine the largest size of raindrops in tropical rain. A mathematical approximation of drop size was out of scope for this paper.

Many similar papers measuring the DSD have been published since, using different methods, under different circumstances, and in different locations. These papers are too numerous and not relevant enough to discuss here, but a large number of such papers is listed in a survey paper by Serio et al. [57]. One observation to make based on this data is that rain is always composed of a significant variety in particle sizes, such that a generalisation to model only drops of a single size is clearly an oversimplification. An example of a measured drop size distribution as measured by Laws and Parsons [38] can be seen in figure 3.

The survey paper by Serio et al. [57] also lists four model types for the approximation of the DSD able to be fit to the existing measurements: a lognormal distribution, the Weibull distribution, the Marshall and Palmer distribution, and the Ulbrich or gamma distribution. Also mentioned is an experiment by Jiang et al. [31] where several approximation formulas are compared to data measured in Tokyo, Japan. The Weibull distribution gives the best fit to the data used, closely followed by the Ulbrich distribution. Both of these distributions accurately represent a peak in the distribution that shifts to larger raindrop sizes at higher rainfall intensity. The Marshall and Palmer distribution was only accurate at low rainfall intensities.

A paper published by Jameson and Kostinski [29] demonstrates that the

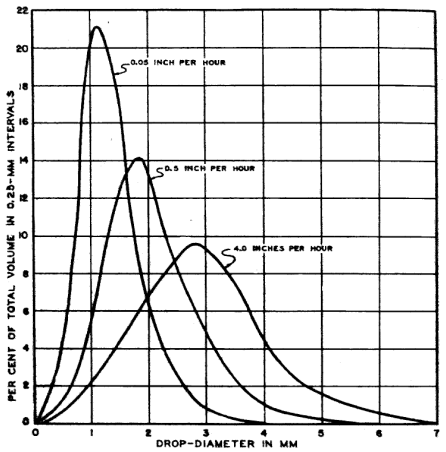


Figure 3: Drop size distributions measured for different rain intensities in Washington D.C. Figure reproduced from [38].

DSD should not be interpreted or used without fully understanding the meaning behind it. The DSD represents a sample that changes over time due to the many interactions a drop may experience during its fall. Additionally, three-dimensional information is lost in the DSD. This means directly using the DSD when simulating raindrops in 3D space may not be accurate to real rain. As an example, if small drops would generally be clustered close together, this information is lost upon the capturing of the DSD. An alternative to the DSD that does provide detailed 3D distributions does not appear to exist at this time, however.

2.1.3 Shape of raindrops

Raindrops deform due to experiencing drag when falling at high velocities. Beard and Chuang [5] study the equilibrium shape of a water drop falling in a wind tunnel, and provide equations to approximate this shape for a drop of a given size. Figure 4 shows the cross section of a drop for a number of drop sizes.

It is unknown whether the shape of a raindrop has any significant impact on our perception of a drop falling at high speeds, and it may provide useful insights on this topic to compare rendered images that do account for the deformation of a raindrop to ones that treat all raindrops as perfect spheres.

2.1.4 Refractive index of rain

For proper scattering of light, it is important to understand the refractive index of raindrops. While rain is not pure water (see for example Sanusi et al. [55]), it is assumed here to be pure enough that using the refractive index of water will not result in a significant visual difference. The refraction of light in water depends on the wavelength of light being scattered, which when viewed under the right

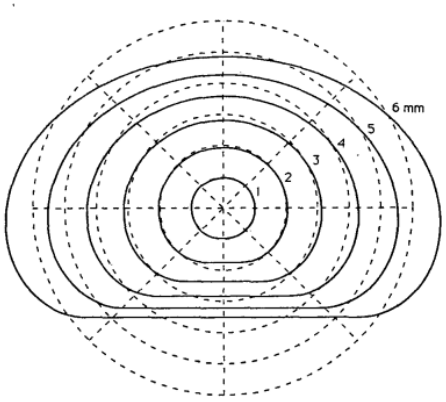


Figure 4: Drop cross-sections for diameters 1, 2, 3, 4, 5, and 6 mm. The dashed circles are provided as reference for circles of the same diameter. Figure reproduced from [5].

circumstances creates rainbows. Polyanskiy [53] provide a detailed database of the refractive indices of many materials for a large number of wavelengths, including water.

2.2 Rain in Computer Vision

Systems relying on computer vision often need to be robust against weather effects such as rain, or need to be able to accurately detect the presence of rain. The visual impact of raindrops on imaging systems has therefore been studied extensively in the field of computer vision. The contents of this section are largely based on a survey by Tripathi and Mukhopadhyay [62]

Weather conditions are generally classified into three categories for the purpose of computer vision: clear weather, steady weather such as fog and haze, and dynamic weather such as rain and snow.

Further distinction is made between the different types of videos obscured by the rain: the background video can be static (objects in the scene do not move) or dynamic (one or more objects in the scene are moving). The same distinction can be made for the camera: the camera can have a static viewpoint or can be moving.

Garg and Nayar [16] observe that camera parameters have a strong effect on the appearance of rain. By varying exposure time, aperture size, and distance of the focus plane, the visibility of rain could either be enhanced or reduced. This is less effective for heavy rain or for dynamic background video. The paper also only utilises a static camera.

Garg and Nayar [15] and the later Garg and Nayar [18] make a comprehensive analysis of the visual effects of rain on imaging systems, with the latter publication extending the first. Discussed are many physical properties as in section 2.1 as well as more precise effects on the camera, such as the average

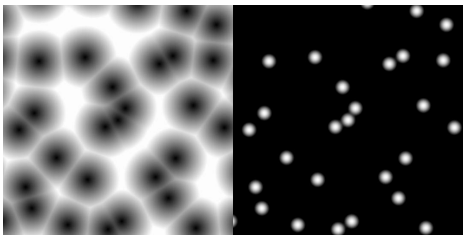


Figure 5: Standard Worley noise (left) and a modified version of Worley noise (right). Figure reproduced from [20].

brightness of a raindrop and the observation that raindrops will always appear as a streak when captured using a camera. A rain detection and removal algorithm is suggested based on some of these properties, namely by first detecting high intensity fluctuations between frames and then checking whether enough such pixels align within a frame to form a streak. Detected rain can then be removed by assigning the pixel a value based on the average value of the same pixel in frames without rain.

Similar methods have been developed by Zhang et al. [73], Brewer and Liu [7], Subhani and Oakley [59], Barnum et al. [4]. Zhang et al. [73] make the additional observation that colours are affected differently due to the refractive indices of water. Yet in the context of rain detection, the difference in brightness across colour channels can be treated as non-existent when compared to other moving objects in the scene. The method utilises this property to differentiate between raindrops and a dynamic background.

2.3 Sampling functions

Raindrops are distributed uniformly throughout the scene, as described in section 2.1. This section looks at a number of sampling strategies to cover a domain with random values.

Perlin noise [49, 50] provides an n-dimensional deterministic noise function; given the same coordinates, the function will always return the same random value. Due to its deterministic nature, Perlin noise is useful in situations where storing all sampled values in memory is infeasible. Another property of Perlin noise is that it is continuous: coordinates located close to one another provide a similar value for the noise function. This makes Perlin noise useful in the generation of terrain [1], but the continuous nature of the noise function may be unwanted when generating discrete particles such as raindrops. Later gradient-based noise functions such as Wavelet noise [11] retain the same properties.

Worley noise [70] similarly provides a continuous n-dimensional deterministic noise function by using the distance to nearby random points distributed on a grid. The pattern generated by Worley noise looks little like rain, but modifications exist that give a closer resemblance to rain (see figure 5) and this has been used to generate rain ripples on the surface of water [20].

The sampling of points on a grid as used by Worley noise is also known as stratified jitter or stratified sampling, and cannot be traced back to any one source. Stratified jitter can be generated by subdividing the sampling space into equally spaced grid cells, and generating a random position or multiple random positions within each cell. This method appears well-suited to the uniform distribution of rain, but minimum distance between samples is not guaranteed without information about nearby cells.

Poisson disk sampling and its three-dimensional equivalent, Poisson sphere sampling, distributes sample points randomly with a minimum distance between points. A major downside of Poisson sampling is the memory cost: most methods (e.g. [8, 72]) require keeping a data structure of previously generated samples. At least one sampling strategy exists such that samples can be generated without tracking all previous samples [47], using a similar approach to stratified jitter but using a grid of polyominoes instead of squares. This method is however restricted to two-dimensional sampling.

2.4 Existing methods for rendering rain

Rendering rain is not a new topic. Several methods to render falling raindrops exist with different objectives.

One family of methods is focused on generating a convincing but not necessarily realistic rain effect at a real-time framerate. This behaviour is suitable for video games, where the user may be more concerned with the responsiveness of the game than the overall realism. Wang et al. [63] use a particle system, where each particle represents a random but predefined rain streak. Wang et al. [64] use a similar method but takes into account more physical properties of rain, such as the DSD and the difference in velocity of raindrops with different sizes. Mizukami et al. [41] similarly use particle systems, but show how non-uniform wind can be represented. Rousseau et al. [54] use a particle system where the surroundings of a particle are mapped onto the particle as a texture, simulating refraction. Across all of these particle-based methods, raindrops are present only in a small range around the observer. Tatarchuk [60] render a number of two-dimensional rain textures in the world facing the camera, blur part of the scene to simulate smaller and distant raindrops, and simulate water dripping off of objects using a particle system.

A number of other methods instead focus on adding rain to existing image or video material. Starik and Werman [58] generate a two-dimensional mask of raindrop positions for each frame. Pixels affected by the mask are then blurred and the pixel is brightened by an amount dependent on the original intensity, such that a dark pixel is brightened more than an already bright pixel. This is a simplification of the complex light scattering of a raindrop, which generally causes a pixel to become brighter than the background by scattering light from the sky towards the camera. The model also assumes temporal independence of rain across multiple frames, as discussed in section 2.1.1. In other words, the rain mask for each frame is generated without using information about the positions of raindrops in the frame before it. Jaeger [28] utilise a similar two-dimensional

Figure 6: Rain streaks added to an image of a traffic light. Figure reproduced from [17].

particle system but darken the rest of the scene in a more controlled manner. Additionally, this method adds fog based on a depth map provided with the scene. This allows simulating the effect of raindrops at a larger distance from the viewer. Garg and Nayar [17] use a user-provided depth map to occlude rain streaks, and take into account illumination from light sources in the scene.

Some methods (e.g. [23]) focus on simulating rain as it appears in sensor data, such as seen in self-driving cars. This kind of rain is often sufficiently realistic to test the robustness of computer vision systems, but does not necessarily appear convincing to the human eye. Figure 8 shows an example of rain generated using such methods.

2.5 Additional effects during rainy weather

Raindrops alone are only a small part of the experience of a rainy day. To help create a convincing image, it may be useful to include a number of additional features in a rain rendering system. This section discusses a number of possible effects that can be employed to further increase the realism of a rainy scene. These techniques are not further discussed or required for the method described in this project and are provided solely to aid future implementations of rendering comprehensive rain scenes, but a small number of techniques are demonstrated on the Bistro scene in figure 15 and used in the final renders.

2.5.1 Wet materials

The wet appearance of streets is a major visual indicator of rain. Jensen et al. [30] describe an appearance model for wet materials based on both the water inside the rendered material and the thin surface layer of water on top of it. A two-layer surface reflection model is used for the thin surface layer of water, and a phase function model is introduced for materials with subsurface scattering.

Figure 7: Different kinds of rain added over an original image (top left). Figure reproduced from [28].

(a) Camera: Image without rain (b) Camera: Image with rain (c) Camera: Image with simulated rain

Figure 8: Camera sensor data for car sensors. While (c) introduces similar features to the data as (b), it does not appear similar to the human eye. Figure reproduced from [23].

Figure 9: A rendered image of wet asphalt. The entirety of the road is more reflective than usual, and certain parts of the road form "puddles" that are even more reflective. Figure reproduced from [43].

Both approaches are commonly supported by existing Monte Carlo ray tracing software.

A number of papers focus specifically on the rendering of wet asphalt. Nakamae et al. [43] build an appearance model for asphalt where a wet road uses a specular reflection component as well as a two-layer reflection model adapted to the local geometry, such that visible puddles form. An example of this method is shown in figure 9.

Bajo et al. [3] simulate the absorption, diffusion, and evaporation of water inside absorbent materials to determine the humidity of the surface, and present a number of ways to use this information to change the appearance of the material.

Wang et al. [65] model water drops on the surface of a material using a method inspired by fluid simulation. This allows for the rendering of small drops that stay behind on surfaces such as leaves or umbrellas, and the method is temporally consistent such that it can be animated for drops landing on a material.

2.5.2 Splashes and ripples

When looking at rain, water splashes on hard surfaces and ripples in puddles or bodies of water are a major visual clue for rain.

Garg et al. [19] record a large number of splash events under different circumstances and on different materials. Based on these recordings, a probability distribution is suggested for splash parameters, which in turn are used to generate particles in a particle system. An example of this method can be seen in figure 11. Splashes are measured on a limited number of surfaces, but linear interpolation between different materials is possible or novel materials can be introduced by estimating the parameters.

Figure 10: Flattened water drop resting on the surface of a leaf. Figure reproduced from [65].

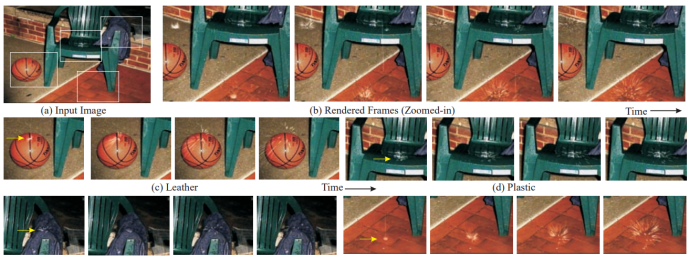


Figure 11: Material-based splashing rendered in a backyard scene with different materials. (a) shows the original image and selected regions. (b) shows frames of the rendered animation with splashes. (c-f) show the image regions in (a). Figure reproduced from [19].

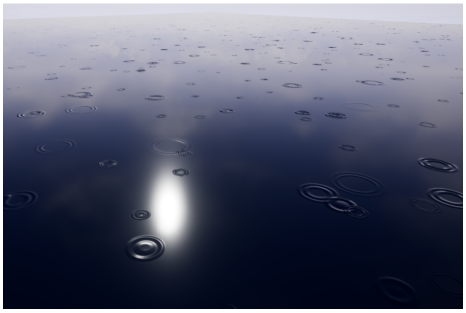


Figure 12: Rain ripples generated using Worley noise rendered using Unreal Engine 4. Figure reproduced from [20].

Figure 13: Photo of rain with out-of-focus raindrops on the lens of the camera. Photo by PINKE [52].

Gawron and Boryczka [20] demonstrate a technique for generating rain ripples on a surface of water by using modified Worley noise to change the normal map of the water surface. An example is shown in figure 12.

2.5.3 Water droplets on cameras

When capturing a photo using a camera in the rain, it is possible for raindrops to hit the camera directly and distort the camera's view. An example of this effect can be seen in figure 13. Although no publications appear to focus on the effect of water on a lens, research on the effect of water on a car's windscreen is available for driving simulators and for testing computer vision systems.

Nakata [44] model each water drop on the windshield as a particle, such that physical simulation of the drops is possible. Small drops can then be rendered as a normal map on the windscreen while larger drops are rendered as pseudo-hemispheres. An example of this method is shown in figure 14.

In comparison, Yang et al. [71] render a distortion model separately, which is then used to distort the image of the scene appropriately.

Figure 14: Rain rendered on the windshield of a car. Figure reproduced from [44].

2.6 Methods for rendering volumes

Similar to clouds and other volumes, rain consists of a large number of small particles. For certain weather conditions such as fog, a volumetric approach is already commonly used (e.g. [24]), and is directly applicable to the smallest of raindrops or rain at a distance. More specifically, raindrops that are small enough to occupy less than a single pixel in the resulting image and that are spaced less than a pixel apart from one another could be treated as a standard volume. Interpreting larger raindrops as a volume is not as trivial due to a raindrop being a solid object, but may be possible when representing a rain streak with motion blur. In this section a number of rendering techniques for non-homogeneous volumes are discussed. A larger number of techniques are discussed more extensively in the survey paper by Novák et al. [45].

A comprehensive equation of light reaching the camera, including light scattered and absorption by participating media, was published by Kajiya [33] and Immel et al. [27] simultaneously. These formulas are largely theoretical description of light transport and the rendered images shown in these publications only consider a small part of the full rendering equation, but these descriptions of light transport provide a foundation for later volumetric rendering techniques.

The technique of ray marching has been applied to the rendering of procedurally generated objects and volumes [48] and works by tracing along a ray in regular incremental steps until a scattering event is encountered or the ray is absorbed. Although the method works well, it is biased in that small features may be missed due to the fixed ray step size. The step size parameter in the method therefore provides a trade-off between the accurate rendering of small features and the speed of the algorithm.

Decomposition tracking [34] allows for faster ray tracing in volumes with a minimum density by decomposing the heterogeneous volume into a homogeneous volume for the minimum density and a sparser residual heterogeneous volume. In rain rendering, this technique could be used when some amount of the rain consists of very small drops that form a sort of mist, or for rain at a distance such that only the largest of drops need to be rendered using a heterogeneous volume.

(a) No modifications

(b) Application of a wet material over the street surface. The wet street is simulated by increasing the specular component of the materials as well as using a scattering layer over the material.

(c) A single stationary raindrop at small distance to the camera

(d) Using an overcast skydome instead of a sunny one.

(e) Approximately 11K raindrops distributed using Poisson spheres falling at terminal velocity in a small range around the viewer. Size distribution is uniformly random between 0.039mm and 2.88mm.

(f) Zoomed-in portion of (f). Three visible streaks have been highlighted.

Figure 15: Bistro scene under various circumstances. (a-d) rendered at 4096 samples per pixel, (e) rendered at 2048 samples per pixel.

A recent advance in the rendering of participating media is the null-scattering path integral formulation of light transport by Miller et al. [40]. Not only is this method unbiased, the method also has a known PDF and thus techniques such as Hero Wavelength Sampling [68] can be utilised to reduce noise. Like ray marching, the method takes steps through the volume to check for scattering or absorption events, but the step size is determined randomly. This random step size is influenced by the global maximum density of the volume.

An important distinction between volumetric rendering algorithms is the difference between isotropic and anisotropic volume rendering techniques. Novák et al. [45] describe a volume as isotropic when the chance for scattering or absorbance of light is independent of the direction the light is travelling in, and anisotropic otherwise. While generalising volumetric rendering methods to anisotropic volumes is possible, this remains an open challenge for some methods [45] and is unimplemented in many existing applications.

It has been shown by Castillo et al. [9] that volume rendering techniques can be applied to the rendering of fabric appearance, as fabric consists of a large number of small fibers. This can be seen as proof that volume rendering techniques can be applied to a large number of materials, so long as a sensible description can be found to interpret the material as a volume.

The book "GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics" [26] contains a chapter on volume visualisation on the GPU. Methods described in this book use proxy geometry, slices of transparent geometry that simulate the effect of a volume, and the focus is on techniques that can be implemented into rasterisation rendering algorithms. Proxy geometry can be used in much the same way for path tracing algorithms that results in a very different approach to rendering volumes than the previously described methods. This does not appear to be a common approach for path tracing and little research has been devoted to it.

3 Research question

This project aims to answer the following question: How can raindrops falling at terminal velocity, which scatter light in a three-dimensional scene, seen from a stationary camera with a non-zero integration time, be rendered using a Monte Carlo path tracing algorithm to produce a photo-realistic image by using or modifying existing techniques dedicated to the rendering of participating media?

To aid in answering this question, it will be useful or even necessary to answer the following sub-questions:

- What does a ground-truth approach to the rendering of raindrops look like, without concern for speed or noise?
- Is the visual impact of two raindrops colliding without merging significant and frequent enough that this should be accounted for when rendering raindrops?

- What sampling strategy is most suited for the procedural generation of raindrops?
- Is it viable to generate rain as a volume on the fly, such that any 3D coordinate can be sampled for the presence or probability that a raindrop overlaps this coordinate at a specific point in time without storing a full description of the volume in memory?
- If it is viable to generate rain as a volume on the fly, is this preferable over using a motion blur approach in terms of performance, noise, and memory usage?
- How does treating raindrops with motion blur as a volume affect the physical correctness of the image?

4 Distributing raindrops

For the purpose of this project, raindrops can be assumed to be uniformly distributed in three-dimensional space. In the case of the ground truth algorithm, raindrop positions are calculated before ray tracing commences. This allows the usage of both procedural and non-procedural sampling strategies. The volumetric sampling algorithm will require procedural sampling strategies. The goal is to achieve a uniform distribution of raindrops throughout the scene, such that on average each cubic meter contains around 1000 raindrops (matching the DSD measured by Jiang et al. [31]).

Poisson sphere sampling gives possibly the lowest discrepancy in the spread of raindrops due to the guaranteed minimum distance between drops. This method can be used to generate raindrops by first filling the sampling domain with Poisson spheres of a size slightly larger than the largest raindrop that could appear, and then randomly removing samples until the desired number of raindrops is reached. The low discrepancy comes at the cost of memory, making the method very costly to use for large scenes to the point that it is impractical even for a precomputed method. This can be demonstrated with the method by Bridson [8] for generating raindrops in $O(N)$ time. The focus of this calculation is purely on the memory usage of the grid itself, as the memory cost for the samples themselves will be present for any precomputed method. For three-dimensional samples, this method uses a three-dimensional grid with grid size $r/\sqrt{3}$. Using a maximum drop size of $r = 0.05m$ and 4 bytes of memory per grid cell, filling a cube with edges of length x uses $4 * (\frac{x}{r/\sqrt{3}})^3 = 4 * (x^3 * 3\sqrt{3})/r^3 = 96000\sqrt{3} * x^3$ bytes of memory. Allowing the grid to use 8GB of memory for generation purposes, this means an area of just under 37x37x37 meters can be filled with raindrops. This is sufficient for scenes with limited visibility such as the Amazon Bistro scene [39], but scenes with fewer obstructions can easily have visible distances larger than 40 meters and memory usage experiences cubic growth with respect to the area to be filled.

Stratified jitter allows faster and procedural generation, in exchange for a slightly worse distribution. Every coordinate in the scene is assigned to a grid cell. Whether a cell contains a raindrop, as well as the position and size of the drop within the cell, is then determined by a random number generator that uses the coordinates of the grid cell as a seed. A hashing function designed for generating random numbers is ideal for this purpose. In pseudocode, this sampling strategy can be described as in algorithm 1, where HASH is a function that takes any number of arguments and returns a floating-point value between 0 and 1

To ensure drops never overlap with neighbouring cells in the grid, stratified jitter as described in algorithm 1 will not generate any drops along the edges of the grid. This might cause visible grid lines to the viewer. A variation of stratified jitter would allow this overlap, at the cost of checking more grid cells when sampling a coordinate for the presence of a raindrop. This approach is shown in algorithm 2.

When allowing raindrops to overlap with neighbouring grid cells there is also a chance that two raindrops overlap at their initial positions. This can be prevented by checking the distance to raindrops in neighbouring cells on fixed sides, and cancelling the generation of a drop if it would overlap.

Table 1 is a comparison of the three variants of stratified jitter in speed. For the precomputation test, a loop was done over each grid cell to generate a list of each raindrop in a 10-meter cube. This kind of precomputation is required for the ground truth algorithm. For the sample test, random positions within the same cube were sampled for the presence of a drop, resembling the sampling done by the volumetric sampling algorithm. Drop sizes are as described in section 5 for a rain rate of 50mm/h and the grid size was set to 1 centimeter. This experiment was performed single-threaded with an AMD Ryzen 7 5800H CPU and 32GB of RAM.

The difference in speed between the methods can be explained by the steps needed to calculate them. Stratified jitter is significantly faster in sampling than the two variants with overlap between grid cells because it only ever needs to check a single cell for the presence of a drop, whereas the other methods need to check the surrounding cells as well. This is not necessary for precomputation and the difference there is thus smaller; in fact, stratified jitter is slower than the variant with overlap between the cells, because the calculation for a position within the cell is more costly than the calculation that ignores the cell boundaries. Checking whether a drop overlaps with other drops for the safe overlapping variant only needs to be done when a drop is present in a cell. This can explain why it is not significantly slower when sampling: only a small number of samples will actually find a drop.

The stratified jitter method with safe overlapping discarded an average of 3 drops across different seeds (out of the total 1M drops). As it only prevents drops from colliding in the initial generation and drops might still collide when moving, the extra effort to check for collisions is only rarely worth it.

Figure 16 shows a visual comparison between stratified jitter with and without overlap, rendered using the volumetric sampling algorithm that will be

Algorithm 1 Stratified jitter for grid size 1 cm. All distance units in cm.

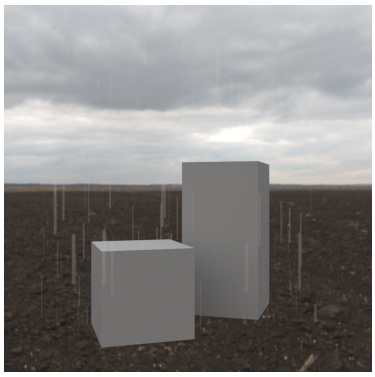
function GETCELL(x,y,z)
  return FLOOR$(x + 0.5)$, FLOOR$(y + 0.5)$, FLOOR$(z + 0.5)$
end function
function HASDROP(c_x, c_y, c_z, chance, seed)
  $random \leftarrow$ HASH$(c_x, c_y, c_z, seed, ID_{hasdrop})$
  return $random > chance$
end function
function DROPRADIUS(c_x, c_y, c_z, seed)
  $random \leftarrow$ HASH$(c_x, c_y, c_z, seed, ID_{radius})$
  return ... ▷ Described in section 5
end function
function DROPPOSITION(x_c, y_c, z_c, radius, seed)
  $max\_shift \leftarrow 0.5 - radius$
  $x_d \leftarrow max\_shift * (\text{HASH}(x_c, y_c, z_c, seed, ID_x)) * 2 - 1$
  $y_d \leftarrow max\_shift * (\text{HASH}(x_c, y_c, z_c, seed, ID_y)) * 2 - 1$
  $z_d \leftarrow max\_shift * (\text{HASH}(x_c, y_c, z_c, seed, ID_z)) * 2 - 1$
  return x_d, y_d, z_d
end function
function SAMPLE(x, y, z, seed)
  $x_c, y_c, z_c \leftarrow$ GETCELL(x, y, z)
  if HASDROP$(x_c, y_c, z_c, 0.001, seed)$ **then**
    $radius \leftarrow$ DROPRADIUS$(x_c, y_c, z_c, seed)$
    $x_d, y_d, z_d \leftarrow$ DROPPOSITION$((x_c, y_c, z_c, radius, seed)$
    return $x_d, y_d, z_d, radius$
  end if
end function
function PRECOMPUTE(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}, seed)
  for $x = x_{min}, \ldots, x_{max}$ **do**
    for $y = y_{min}, \ldots, y_{max}$ **do**
      for $z = z_{min}, \ldots, z_{max}$ **do**
        $x_c, y_c, z_c \leftarrow$ GETCELL(x, y, z)
        if HASDROP$(x_c, y_c, z_c, 0.001, seed)$ **then**
          $radius \leftarrow$ DROPRADIUS$(x_c, y_c, z_c, seed)$
          $x_d, y_d, z_d \leftarrow$ DROPPOSITION$(x_c, y_c, z_c, radius, seed)$
          output $x_d, y_d, z_d, radius$
        end if
      end for
    end for
  end for
end function

Algorithm 2 Overlapping stratified jitter for grid size 1 cm. All distance units in cm. The functions GetCell, HasDrop DropRadius, and Precompute are identical to those described in algorithm 1

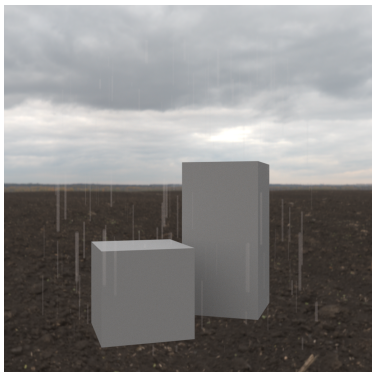
function DropPosition(x_c, y_c, z_c, radius, seed)
  $x_d \leftarrow c_x + radius + \text{HASH}(x_c, y_c, z_c, seed, ID_x) - 0.5$
  $y_d \leftarrow c_x + radius + \text{HASH}(x_c, y_c, z_c, seed, ID_y) - 0.5$
  $z_d \leftarrow c_x + radius + \text{HASH}(x_c, y_c, z_c, seed, ID_z) - 0.5$
  return x_d, y_d, z_d
end function
function Sample(x, y, z, seed)
  $x_c, y_c, z_c \leftarrow \text{GetCell}(x, y, z)$
  for $x = x_c - 1, \ldots, x_c$ **do**
    for $y = y_c - 1, \ldots, y_c$ **do**
      for $z = z_c - 1, \ldots, z_c$ **do**
        if HasDrop$(x_c, y_c, z_c, 0.001, seed)$ **then**
          $radius \leftarrow \text{DropRadius}(x_c, y_c, z_c, seed)$
          $x_d, y_d, z_d \leftarrow \text{DropPosition}((x_c, y_c, z_c, radius, seed)$
          output $x_d, y_d, z_d, radius$
        end if
      end for
    end for
  end for
end function

Method	Time precomputation	Time 100M samples
Stratified jitter	18.8s	5.6s
With overlap	18.3s	14.6s
With safe overlap	19.8s	14.7s

Table 1: Speed comparison of stratified jitter, stratified jitter with overlap, and stratified jitter with safe overlap



(a) Drops are placed randomly within the confines of the grid cell they are generated in.



(b) The origins of drops are placed randomly within the confines of the grid cell, but the drops themselves may overlap with neighbouring grid cells.

Figure 16: Comparison between stratified jitter without (a) and with (b) overlap between the grid cells, rendered using the method described in section 9. The density of the medium has been increased to make the streaks clearer to see.

discussed in section 9. Notice how the visual difference is minimal, and "grid lines" of empty space between the drops caused by the lack of overlap between the cells are not directly visible in this scene. Choosing to use stratified jitter without overlap when sampling randomly is thus a good option for better performance. Further comparisons between the two methods are made in section 10.

5 Size of raindrops

For each raindrop, a size should randomly be chosen with size probabilities corresponding to the drop size distribution or DSD. As discussed in 2.1.2 a number of approximation formulas exist for the DSD. Given that the Weibull distribution is both accurate and easy to randomly sample from, it is a good fit. The rest of this section will describe the DSD as a Weibull distribution and how this distribution is sampled from.

Kızılersü et al. [35] describe the Weibull distribution as a function that tells us how likely something is given a chance of failure over time. The Weibull distribution can be used to describe the distribution of raindrop sizes, which grow due to collisions until a "failure" such as turbulence breaks them up into smaller drops again [37, 57]. The Weibull distribution is described by two parameters: a shape parameter α that determines the shape of the distribution and a scale parameter β that stretches the distribution across time, such that

$W(x) = \frac{\alpha}{\beta}(\frac{x}{\beta})^{\alpha-1} * e^{-(x/\beta)^{\alpha}}$ describes the probability of a raindrop having radius x. Sekine and Lind [56] fit these parameters as $\alpha = 0.95 * R^{0.14}$ and $\beta = 0.26 * R^{0.44}$ where R is the rain intensity in millimeter per hour, with 1000 drops per cubic meter.

Johnson et al. [32] show that the Weibull distribution can be transformed to easily generate random samples on the distribution from a uniformly generated number as

$$X(U) = \beta(-ln(U))^{1/\alpha}$$

where U is a random number in the range $(0, 1]$ and X is a random sample distributed according to the Weibull distribution.

It is thus possible to sample the Weibull DSD by calculating the shape and scale parameters for the intensity of the simulated rain, generating a uniformly random number in the range $(0, 1]$, and using the result of formula 5 as the radius of the drop. The shape and scale parameters of the distribution only have to be calculated once for a given rain intensity.

It is theoretically possible for a drop to be assigned an arbitrarily large radius when sampling using the Weibull distribution, albeit increasingly unlikely. To prevent this causing issues during rendering, an upper bound r_{max} on the radius is set and drops larger than this radius are instead assigned the radius r_{max}. For comparison, Gunn and Kinzer [22] find that drops with a radius larger than 2.88 millimeters are unstable and tend to break up into smaller drops, and Sekine and Lind [56] mention that drops with a radius larger than 2.5 millimeters are quite rare. The choice for a larger radius was made to represent the existence of larger drops that have collided recently and are not yet broken up into smaller drops. Additionally, a drop with a radius of 5 millimeters fits nicely in a single grid cell with a grid size of 1 centimeter.

6 Interpretation of density

In traditional volumetric rendering techniques, density is typically multiplied with the cross-sectional area for absorbing and scattering particles (typically constant across the volume), resulting in an absorption and scattering coefficient representing the probability a ray of light gets absorbed or scattered per unit length travelled throughout the volume [45]. When the density is spatially varying, it can be approximated by taking local samples. This allows rain to be rendered using existing volumetric rendering techniques by reinterpreting the presence of raindrops as density for such a local sample. More formally, given sample position s and time $0 \leq t \leq 1$ where $t = 0$ is the time at which the camera's shutter opens and $t = 1$ the time at which the shutter closes, what is the expected number of raindrop collisions ρ along a ray segment r of length l through s at a random time t?

Let D be the collection of all raindrops that might possibly pass through s. Every drop d in D has a radius r and a center position c_t moving in a straight line dependent on t. A probability p_d can be calculated from this to express the

chance d overlaps with s, as well as ρ_d for the expected number of interactions between d and r. p_d and ρ_d are calculated in appendix A.

It is clear that if D is empty, ρ is zero. When D contains at least one drop, $\rho = \sum_{d \in D} \rho_d$.

As visible from the equation given for ρ_d in appendix A, ρ_d is dependent on the direction of the ray. This makes intuitive sense as well: a ray of light passing through some point in a drop's path and going in the same direction as a drop is guaranteed to hit the drop, while a ray through the same point but oriented directly perpendicular to the drop's movement direction only hits it at very specific times. This creates an anisotropic volume when using ρ_d, and as discussed in section 2.6 not all volumetric rendering methods and importance sampling methods support this.

Instead, ρ_d can be approximated for any ray through a point s using p_d. Let the approximation be denoted by $\hat{\rho}_d$. To make this generalisation, first move the origin of the ray o to s. As s is a point on the ray, shifting the origin like this has no effect on the final image. The generalisation of the duration a drop intersects with a ray has already been described as p_d: this can be observed by taking $\vec{r} = (0, 0, 0)$, essentially averaging over every possible direction a ray could have. In that case, $\hat{\rho}_d = \dfrac{p_d}{a_{max} - a_{min}}$. Generalising the length of the ray segment that intersects with the drop is more difficult. One option is to use $\hat{\rho}_d = \dfrac{p_d}{r}$, which gives good results but is not entirely accurate. Conceptually, it makes sense to use the radius: the length a ray can stay within a drop logically depends on the size of said drop. However, this approximation underestimates the length of the ray segment for rays directly perpendicular to the movement direction of the drop: in that case $a_{max} - amin = r$ when the ray also passes through any point c_t and $a_{max} - amin > r$ otherwise. Meanwhile, it overestimates the length of the ray segment parallel to the movement direction of the drop, in which case $a_{max} - amin \geq m_y$. Nevertheless, $\hat{\rho}_d = \dfrac{p_d}{r}$ gives a good enough approximation to be indistinguishable from the real probability of scattering when used for the volumetric sampling algorithm.

6.1 Upper bound for density

A number of volumetric rendering techniques, such as null scattering [40], require an upper bound to the absorption and scattering coefficients. With a constant cross section, this means the maximum density for the volume should be decided. A closer match to the true maximum density allows taking fewer samples and thus higher rendering speeds, but if the maximum density is too low then bias is introduced. As such the upper bound should match the largest value $\hat{\rho}$ as closely as possible, which is the maximum value of $\hat{\rho}_d$ multiplied by the maximum number of drops passing through a single point. The upper bound for $\hat{\rho}_d$ is $(2 * r/m_y)/r = 2/m_y$. m_y is dependent on the size of drops and grows larger with larger drops, thus the upper bound for $\hat{\rho}_d$ is determined by the smallest possible drop (see figure 17), expressed as $\hat{\rho}_{smallest}$. The number

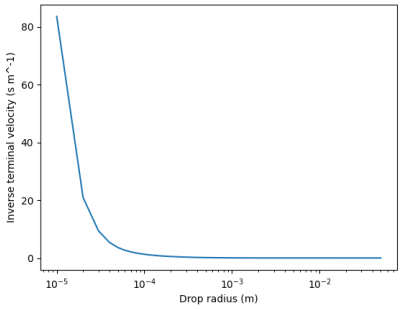


Figure 17: The inverse of the terminal velocity corresponding to drop radius. The maximum value $\hat{\rho}_d$ for a drop is proportional to the inverse of its terminal velocity, and thus largest for smaller drops.

of drops that might pass through a single point is equal to the number of grid cells checked, but is practically much lower: it is incredibly unlikely for multiple small drops to pass through the same point. An upper bound of $2 * \hat{\rho}_{smallest}$ is sufficient in nearly all situations.

With the upper bound dependent on the smallest drops and drops being potentially infinitely small, a problem arises: what should be the practical upper bound? The choice is made to set a minimum radius for drops r_{min} in addition to the existing upper bound on the radius described in section 5. Drops below this size are not rendered. Discarding the smallest drops has minimal impact due to them being barely visible in the first place. An alternative for visualising these anyway will be discussed in section 6.2.

6.2 Lower bound for density

The smallest of raindrops are nearly invisible individually. Instead, drops smaller than r_{min} can be simulated as a homogeneous fog (imagine the mist visible during a light drizzle of rain). This fog should be assigned a constant density equal to the chance of a ray hitting any of the discarded drops.

The expected number of raindrop interactions with drops of radius x along a ray is equal to $1000 * W(x) * \pi * x^2$ per meter. This value equals the number of raindrops per cubic meter multiplied by the visible area of each drop. Using the cumulative distribution function (CDF) of the Weibull distribution allows a linear estimation of the constant density of the discarded drops. The chance the radius of a drop is less than or equal to a certain radius x as

$$CDF(x) = 1 - e^{-(x/\beta)^{\alpha}}$$

Expressing $x(i) = i * r_{min}/n$, the density required for the homogeneous medium

can be estimated in n steps as

$$\sum_{i=0}^{n}(CDF(x(i)) - CDF(x(i-1))) * 1000 * W(x(i)) * \pi * x(i)^2$$

This value is constant for a given value R and can thus be precomputed before rendering.

7 Interpretation of phase function

When a ray scatters as determined by the null-scattering algorithm, the direction is chosen by a phase function that describes the likelihood of choosing a direction based on the original direction of the ray. For raindrops, this direction would depend on the angle light makes with the surface of the drop, which in turn depends on the position of the drop.

One way to do this is to create a physically accurate model of the scattering that happens inside of a real raindrop. A random position for a raindrop can be generated by sampling a uniformly random value for t within the limits calculated in equation A.2. The ray can then be intersected with the sphere representing the raindrop as usual in path tracing, allowing fully accurate scattering, reflection, and absorbance. This can either be done to determine the new direction of travel for the ray (in which case sampling the medium is only done to check for collisions) or used as a model for the phase function (in which case the direction of the exiting ray is a sample of the phase function).

When intersecting with the discarded part of the volume simulated through the homogeneous medium, an additional step is required for this physical model as there is no predetermined size or position for the drop that got hit. A radius for the drop can be chosen by first calculating the uniform number that would generate a drop of size r_{min} by solving $\beta(-ln(U_{min}))^{1/\alpha} = r_{min}$ and then sampling the Weibull distribution (equation 5) with an input uniformly random between 0 and U_{min}. U_min only has to be calculated once for a constant rain rate and r_{min}. After determining the size as $r_{simulated}$, a position for the simulated drop should randomly be chosen within a distance of $r_{simulated}$ from the sample position. At that point scattering within the drop can be modelled the same as described previously.

A major problem when using the physical model as a phase function is the unknown PDF: although taking a sample is relatively easy, the probability of said sample is hard to determine due to the many possible ways light can be reflected within the drop, the duration the ray intersects with the drop, and the presence of discarded raindrops. Without an accurate PDF it is nearly impossible to do any kind of importance sampling, which in turn results in a noisy image. Reverse engineering the PDF for this model phase function is out of scope for this project, and this phase function is therefore not used for the volumetric sampling method described in section 9.

Replacing the physically-accurate scattering model described above with the widely-used Henyey-Greenstein phase function [25] still gives acceptable results.

The phase function parameter was set experimentally, using images rendered with the ground truth algorithm as a goal. The value $g = 0.6$ was found to be a good match (see also figures 18 and 19). This value can intuitively be reasoned to be a good match too: raindrops scatter light forward from most angles, but it is not rare for light to be scattered at a sharp angle when hitting the side of a drop. Reflection also occurs, but is extremely unlikely. This suggests a positive and relatively large g-value.

A major advantage of using the Henyey-Greenstein phase function is its common usage: most rendering software already have this phase function implemented for volume rendering. Another advantage is the known PDF of the function and thus allowing importance sampling. A major disadvantage is the physical accuracy: the Henyey-Greenstein phase function is not accurate to the true refraction and reflection inside raindrops, preventing the formation of physical phenomena such as rainbows.

8 Ground truth algorithm

The ground truth algorithm is built on top of the PBRT software [51] by generating a scene with raindrops. The requirements for this implementation are support for motion blur and dielectric materials with wavelength-dependent refractive indices and absorbance. The implementation is run before rendering starts and can be a separate program to generate a scene.

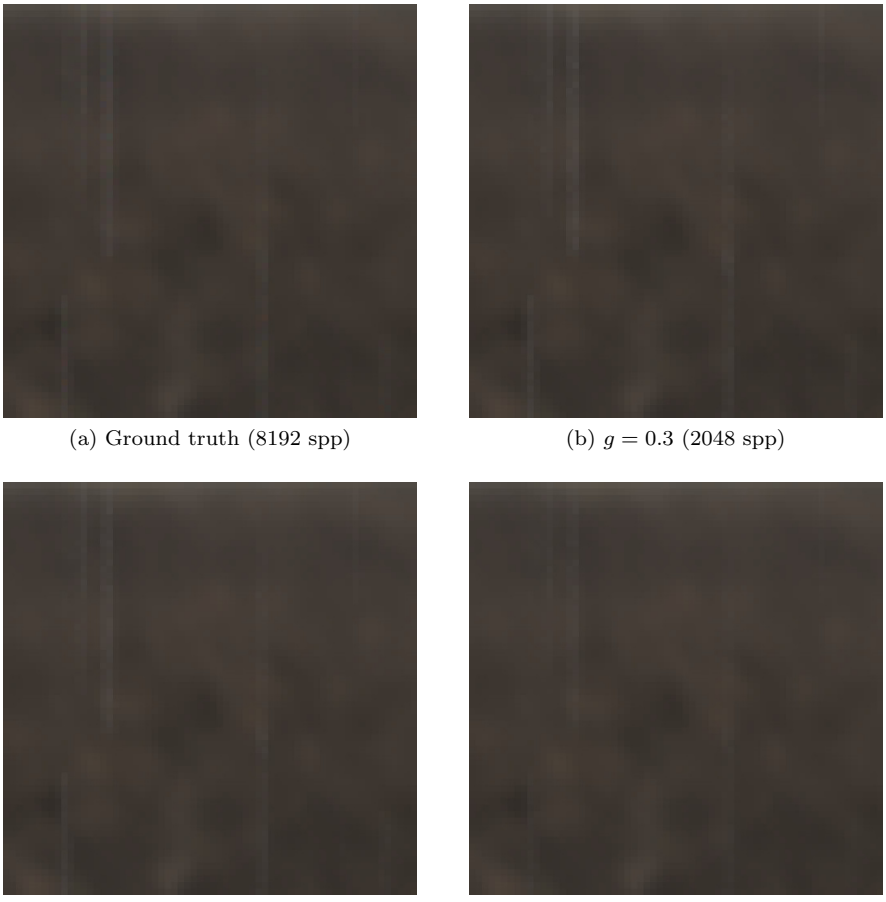
Given a set of bounds for the area to fill with raindrops and the shutter time of the camera, the ground truth algorithm loops over every grid cell within the bounds. Whether this cell contains a raindrop, and if so at what position and of what size, is determined using one of the stratified jitter methods described in section 4. The terminal velocity for the raindrop is then calculated according to the formula by Wobus et al. [69] and multiplied by the shutter time to determine the total distance the drop moves. A sphere is inserted into the scene with the appropriate radius, moving from the initial position to a position that is the correct distance down. The sphere is assigned a material matching the parameters found by Polyanskiy [53].

Once the algorithm has passed all grid cells and raindrops have been inserted in the correct locations, rendering can commence as usual.

8.1 Effect of colliding drops

Due to the difference in velocities for drops of different sizes, it is possible for two drops to collide no matter which spatial distribution is used. It can be assumed that a collision between two raindrops has a low enough visual impact in real life that properly simulating raindrop collisions can be skipped for computer renders: collision between raindrops was first observed by Testik and Rahman [61] and that required the use of a high-speed optical disdrometer.

Nevertheless, improperly handling collisions might lead to artifacts in the final render: most rendering algorithms do not distinguish between entering a



(a) Ground truth (8192 spp)

(b) $g = 0.3$ (2048 spp)

(c) $g = 0.6$ (2048 spp)

(d) $g = 0.9$ (2048 spp)

Figure 18: Zoomed-in comparison of rain streaks with a varying Henyey-Greenstein parameter value. The streaks for $g = 0.3$ are slightly brighter than the ground truth, while the streaks in $g = 0.9$ are darker. 19 shows vertical slices from each of these images for a closer comparison.

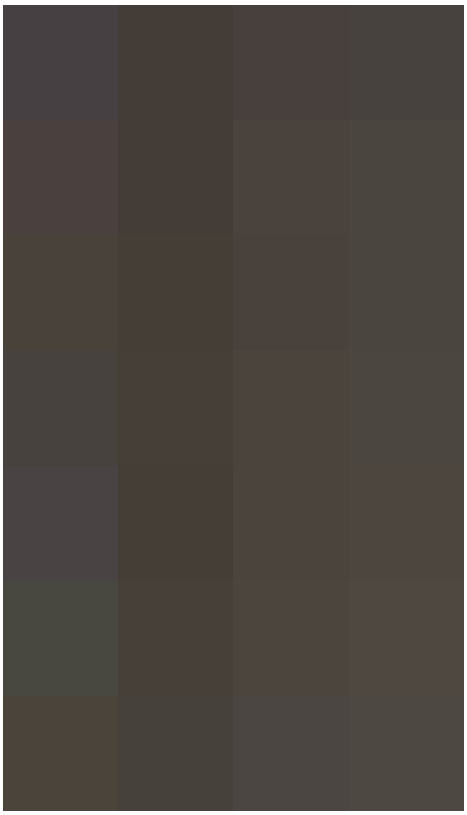


Figure 19: Vertical slices from rain streaks with a varying Henyey-Greenstein parameter value. From left to right: ground truth render, $g = 0.9$, $g = 0.6$, $g = 0.3$. The streaks for $g = 0.3$ are slightly brighter than the ground truth, while the streaks in $g = 0.9$ are darker.

(a) Two drops overlapping, merged into a single shape.

(b) Two drops overlapping but not merged. This introduces the possibility for rendering artifacts, but none are visible.

(c) The difference between the two renders, consisting of almost pure noise and displaying no artifacts.

Figure 20: Comparison between overlapping drops when merged into a single shape or kept as separate objects, rendered at 4096 spp. No significant difference is visible between the two renders.

second material and exiting the first, thus overlapping geometry will be treated as an air bubble. Using the size and distribution described in the previous two sections, it is found that (on average) no more than five in a million drops collide at a time. Figure 20 additionally shows a comparison between a render of two stationary raindrops merged into a single shape (such that overlapping geometry is removed) and a render of two stationary raindrops overlapping without merging. The two renders are similar enough that the visual impact on the ground truth render can safely be assumed to be negligible too.

9 Volumetric sampling algorithm

The algorithm using a simulated volume is built on top of the PBRT software [51] as well. The requirement for this implementation is that the software supports heterogeneous isotropic volumes. The core functionality that needs to be implemented for this algorithm is the method for acquiring a local density sample of the volume. This type of sampling is common for many volumetric rendering techniques, such as ray marching or null scattering. While the implementation described here is specifically implemented with null scattering, other methods will follow roughly the same approach.

Given a position p and a seed s, the position's cell is determined using one of the methods described in section 4. If the method for spreading raindrops does not allow overlap between cells, the current cell and a number of cells above it are checked for the presence of raindrops at $t = 0$. If overlap is possible, the cells directly or diagonally adjacent to these cells are checked as well. All drops are falling in the same direction, which is aligned with one of the major axes of the grid. If a uniform wind velocity is assigned to the drops such that they

move diagonally instead of straight down, this can be achieved by rotating the grid accordingly.

The number of cells to check above the current position depends on the shutter time, the cell size, and the velocity of the fastest raindrop relative to its own radius. Enough cells need to be checked such that the starting cell of a raindrop can be found if it would pass through a certain point when the shutter closes. The fastest raindrop travels just under $10ms^{-1}$ and the maximum number of grid cells a drop travels through can thus be calculated by $10 * shutter/cell$ where *shutter* is the number of seconds the shutter is open and *cell* the size of a cell in meters. With a typical shutter speed of $0.01s$ and a grid size of $1cm$ the center of this drop would thus pass through 10 grid cells in a single frame.

The null-scattering implementation of PBRT [51] is used to sample and render this medium, using the Henyey-Greenstein phase function described in section 7 for scattering. Sampling the medium coefficients is done by determining the density by summing over the value of $\hat{\rho}_d$ for all drops in the checked cells and adding the value calculated using equation 6.2. The upper bound for the coefficients is as determined in section 6.1 and constant across the medium.

10 Evaluation

This section compares the visual results of the volumetric sampling algorithm to those of the ground truth algorithm, as well as the efficiency with which these results are produced. Timings were run with an AMD Ryzen 7 5800H CPU (utilising 12 of the 16 available cores), NVIDIA GeForce RTX 3050Ti GPU, and 32GB of RAM. r_{min} was set to 0.01 cm for the volumetric method, corresponding to the smallest 1% of drops at a rain rate of 50mm/h. PBRT's "volpath" integrator is used, a uni-directional path tracing algorithm.

Four scenes are used for comparison. The first scene is the Cornell box [46] with rain generated within a sphere fitting inside the box. The second scene uses the same cubes and sphere as the first scene, but replaces the exterior walls and light with an outdoor skydome. The third scene shows rain generated around a small red light source with walls around the light source. See figure 22 for a top-down perspective of this scene. Fourth is the Bistro scene by Lumberyard [39], modified to use an overcast skydome and a wet street (see figure 15). Due to the size and complexity of this scene, it is only rendered at low sample counts on the CPU with rain limited to a small distance around the camera and the volumetric sampling algorithm with overlapping jitter is skipped. A single render of scene 4 on the GPU is shown in figure 21 with rain extending beyond the visible area of the scene. Precomputation timings for the ground truth method are ignored in all cases. Accurate motion blur techniques for the GPU are rare, which makes it impossible to run the ground truth algorithm on the GPU. This demonstrates one of the most significant advantages of the volumetric sampling algorithm.

The following abbreviations are used: NO for renders with no rain, GT for renders using the ground truth method, VN for renders using the volumetric

Figure 21: Bistro scene rendered at 256 spp on the GPU with $R = 50$. The red region is displayed in the top-right corner.

Rain rate	GT (CPU)	VN (CPU)	VO (CPU)	VN (GPU)	VO (GPU)
10mm/h	2.5 min	11 min	22.5 min	47 s	2.5 min
50mm/h	2.5 min	11 min	22 min	29 s	1.5 min
200mm/h	2.5 min	11.5 min	22 min	23 s	55 s

Table 2: Timings for rendering scene 1 at 64 spp. Rendering with no rain took 2.5 minutes on the CPU and 9 seconds on the GPU.

sampling algorithm with non-overlapping stratified jitter used for the distribution of rain, and VO for renders using the volumetric sampling algorithm with overlapping stratified jitter.

With few raindrops, the ground truth method is barely any slower than renders without any rain at all for simple scenes. The volumetric sampling method is significantly slower on the CPU, but due to its capability to run on the GPU it can easily outperform the ground truth method for these scenes. Also very important is the visibility of rain: rain is barely visible at low sample counts with the ground truth method, and remains noisy even at high sample counts. This means that the extra performance of the ground truth method on the CPU does not necessarily result in better images even when granted the same amount of rendering time, as demonstrated in figure 26.

In general, rendering with non-overlapping stratified jitter is around two to four times faster than using overlapping stratified jitter, with negligible impact on the quality of the result.

In scenes with strong directional light, a clear difference between the ground truth and the volumetric sampling method can be observed on the sides of

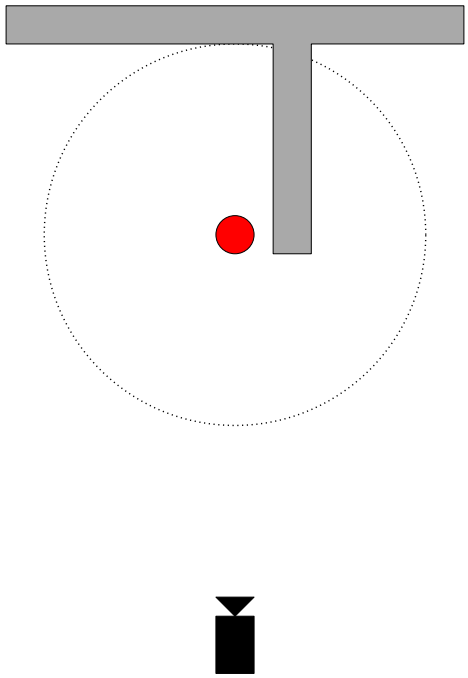


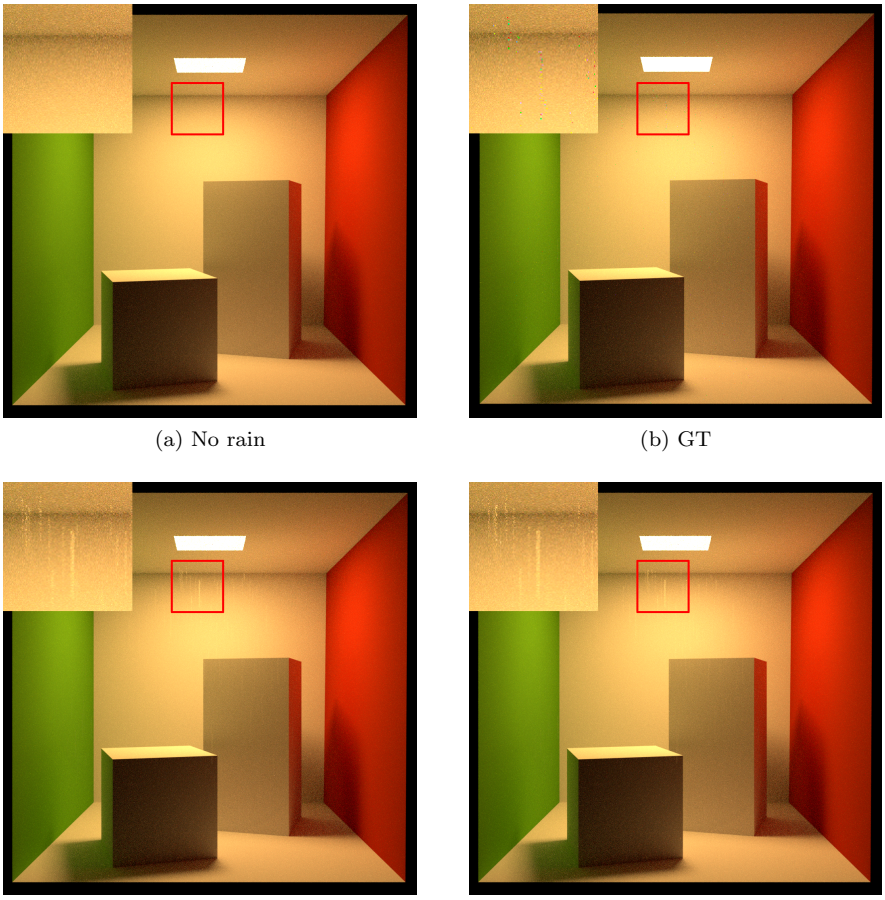
Figure 22: Setup of the red light scene. The dotted circle represents the sphere within which raindrops are placed. The red sphere represents the light source, and the gray shape the walls within the scene.

Rain rate	GT (CPU)	VN (CPU)	VO (CPU)	VN (GPU)	VO (GPU)
10mm/h	42 s	7.5 min	16 min	6.5 min	26.5 min
50mm/h	42 s	7.5 min	15.5 min	3 min	8.5 min
200mm/h	45 s	7.5 min	15.5 min	52 s	3 min

Table 3: Timings for rendering scene 2 at 64 spp. Rendering with no rain took 40 seconds on the CPU and 5 seconds on the GPU.

Rain rate	GT (CPU)	VN (CPU)	VO (CPU)	VN (GPU)	VO (GPU)
10mm/h	1.5 min	7.5 min	19 min	2 min	8 min
50mm/h	1.5 min	8 min	19 min	1.5 min	4.5 min
200mm/h	1.5 min	7.5 min	18.5 min	42 s	2 min

Table 4: Timings for rendering scene 3 at 64 spp. Rendering with no rain took 1.5 minutes on the CPU and 12 seconds on the GPU.



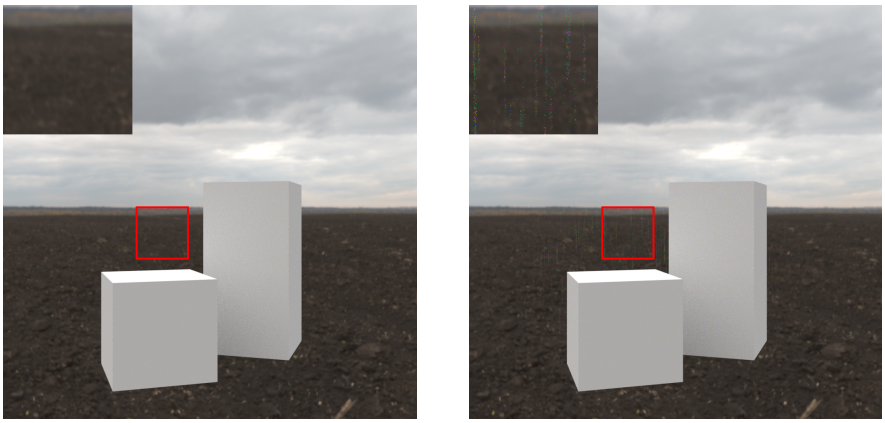
(a) No rain (b) GT

(c) VN (d) VO

Figure 23: Images of scene 1 at 64 spp with $R = 200$. The red region is displayed in the top-left corner.

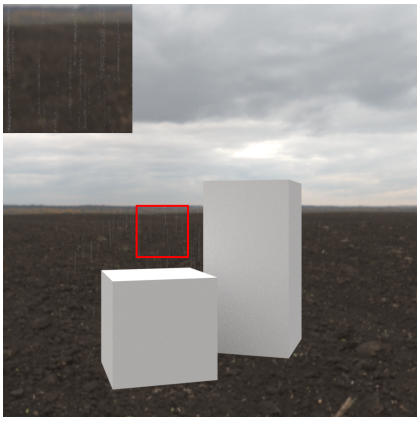
Rain rate	GT (CPU)	VN (CPU)
10mm/h	35 s	41 min
50mm/h	40 s	40 min
200mm/h	43 s	42 min

Table 5: Timings for rendering scene 4 at 16 spp. Rendering with no rain took 28 seconds on the CPU.

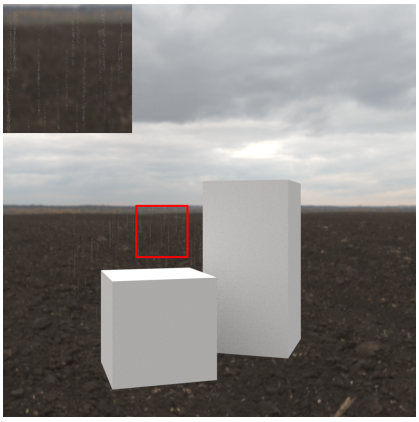


(a) No rain

(b) GT

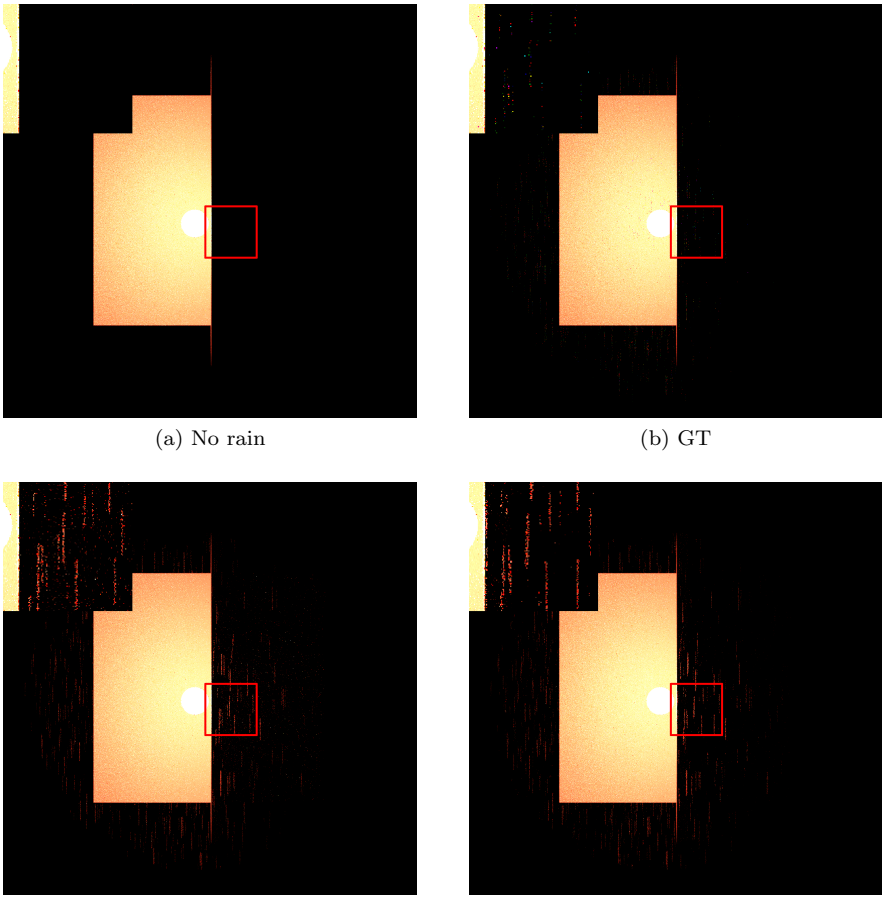


(c) VN



(d) VO

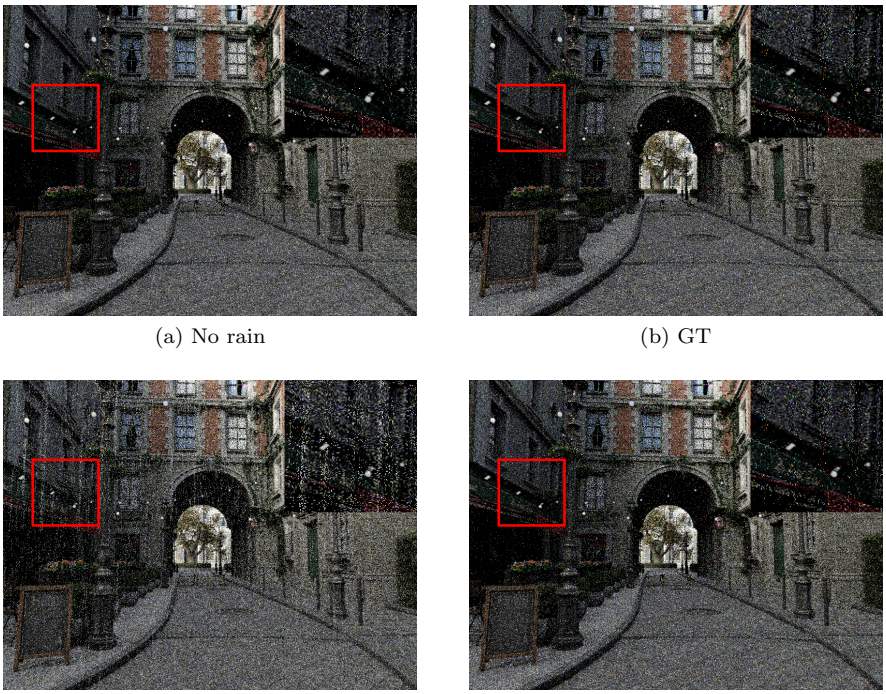
Figure 24: Images of scene 2 at 64 spp with $R = 200$. The red region is displayed in the top-left corner.



(a) No rain (b) GT

(c) VN (GPU) (d) VO (CPU)

Figure 25: Images of scene 3 at 64 spp with $R = 50$. The red region is displayed in the top-left corner. The GPU version has more noise on the right wall, caused by a difference in the sampling strategy PBRT uses between CPU and GPU rendering.



(a) No rain (b) GT

(c) VN (d) GT (40 minutes)

Figure 26: Images of scene 4 with $R = 50$. a, b, c rendered at 64 spp, d rendered for 40 minutes (the same amount of time required for rendering c). The red region is displayed in the top-right corner.

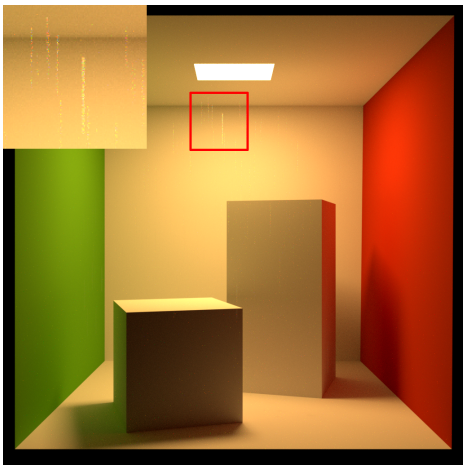


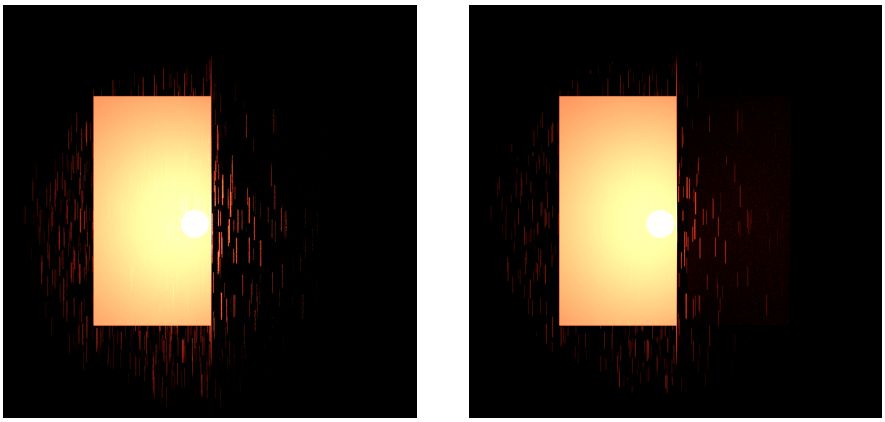
Figure 27: Scene 1 rendered using the ground truth method at 1024 spp. The red region is displayed in the top-left corner.

raindrops furthest away from the light source. This effect is clearly demonstrated in figure 28, where drops to the left have a glint on their left side in the ground truth (and the opposite on the right) and the same glint is not present with the volumetric rendering technique. This is caused by the approximation of the phase function: realistically, light is mostly refracted to the right on the left side of a raindrop and the other way around. The Henyey-Greenstein phase function does not make this distinction and scatters light independent of which side of the drop a ray is passing through. The physically accurate phase function described in section 7 would not suffer the same issue.

The volumetric sampling method also has more adjustable parameters than the ground truth method. The medium's density can be multiplied with an arbitrary number to increase or decrease the visual impact of rain (like used in figure 16), which may be useful in situations where realism is less important. r_{min} provides an easy way to trade accuracy for performance by switching the smallest drops out for a uniform medium, as demonstrated in table 6. Figure 29 shows the results of these different renders.

11 Future work

The implementation outlined in this document focused on constructing a volume with proper densities for simulating rain. Also important for the proper appearance of a volume is the use of a correct phase function. The physically accurate phase function described in section 7 satisfies this requirement, but has an unknown PDF making it unusable with importance sampling methods. Completing the PDF for this phase function or developing an alternate phase function with better accuracy than the Henyey-Greenstein phase function used

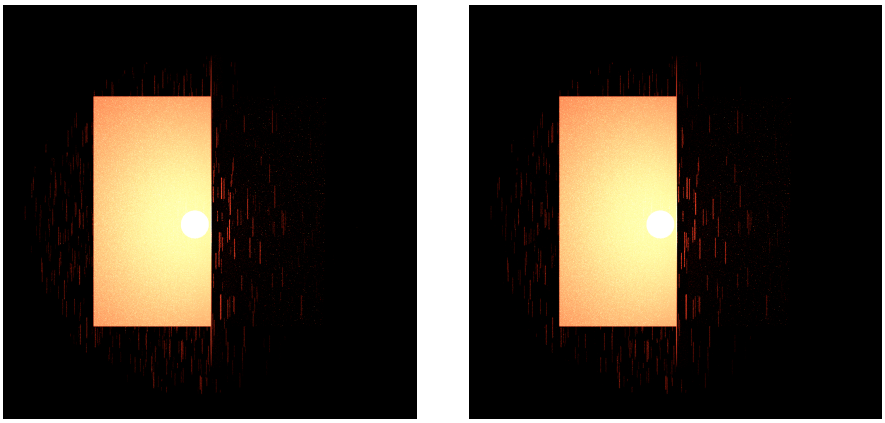


(a) Ground truth (b) Volumetric sampling

Figure 28: Comparison between the ground truth and volumetric method for scene 3, both rendered at 65K spp. The rain streaks in the ground truth have a white glint on the side opposing the light, whereas the streaks generated by the volumetric sampling method vary less in color.

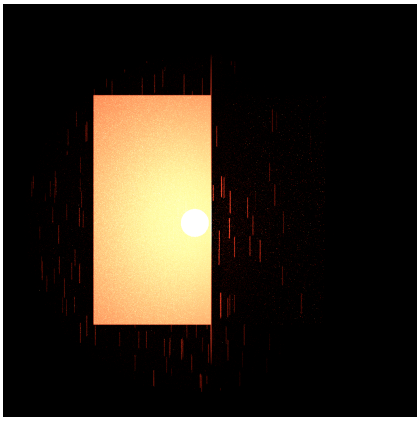
r_{min} in cm	Render time
0.01	14.5 min
0.02	7.5 min
0.03	5 min
0.05	4 min
0.1	3.5 min
0.15	3 min
0.2	2.5 min

Table 6: Render time for scene 3 at 512 spp using the VN method on the GPU, at $R = 50$.

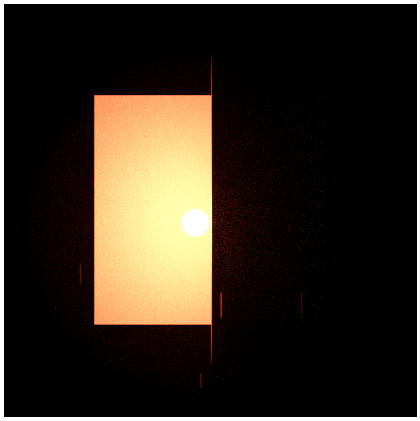


(a) $r_{min} = 0.01$

(b) $r_{min} = 0.05$



(c) $r_{min} = 0.1$



(d) $r_{min} = 0.2$

Figure 29: Comparison of different values for r_{min} in scene 3. All images are rendered at 512 spp with $R = 50$. At a higher rain rate, fewer individual raindrops are visible but a uniform fog begins to fill the area instead.

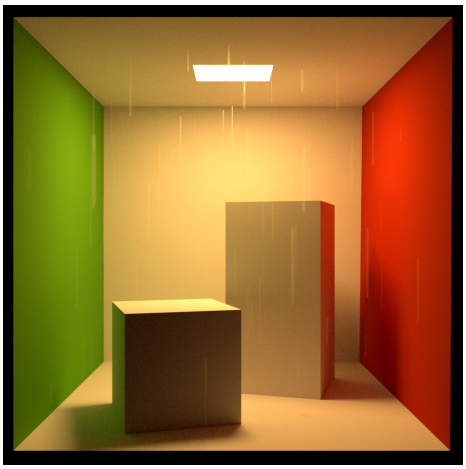


Figure 30: Scene 1 at 256 spp with exaggerated rain density using the VN method: all rain is 3x more likely to be hit.

here will result in more accurate renders and allows real-life optical effects such as rainbows to emerge.

The density approximation described in section 6 is close enough to the true density to give results with minimal difference to the ground truth for the chance to hit a drop, but the division by r is one that is not backed up mathematically. A better alternative might exist, or it might be possible to construct proof that the calculation for $\hat{\rho}_d$ used in this project is indeed one of the best approximations for ρ_d. Another possibility for future work is to develop or use a rendering algorithm that supports anisotropic media to implement a volume sampling algorithm that uses ρ_d directly.

The lower bound to raindrop size discussed in section 6.1 directly influences the step size of the null-scattering algorithm. With individual raindrops further away from the camera being harder to see, it would be logical to increase this lower bound for samples taken further away from the camera and increase the homogeneous component of the medium as described in section 6.2 accordingly. Primary questions about this approach are the rate at which the lower bound can be increased without compromising the accuracy of results, and the effect on render times and noise.

References

[1] Travis Archer. Procedurally generating terrain. In *44th annual midwest instruction and computing symposium, Duluth*, pages 378–393, 2011.

[2] David Atlas and Carlton W. Ulbrich. Path- and area-integrated rainfall measurement by microwave attenuation in the 1–3 cm band. *Journal of Applied Meteorology and Climatology*, 16(12):1322

– 1331, 1977. doi: 10.1175/1520-0450(1977)016⟨1322:PAAIRM⟩2.0.CO; 2. URL https://journals.ametsoc.org/view/journals/apme/16/12/1520-0450_1977_016_1322_paairm_2_0_co_2.xml.

[3] Juan Miguel Bajo, Claudio Delrieux, and Gustavo Patow. Physically inspired technique for modeling wet absorbent materials. *The Visual Computer*, 37(8):2053–2068, Aug 2021. ISSN 1432-2315. doi: 10.1007/s00371-020-01963-w. URL https://doi.org/10.1007/s00371-020-01963-w.

[4] Peter C. Barnum, Srinivasa Narasimhan, and Takeo Kanade. Analysis of rain and snow in frequency space. *International Journal of Computer Vision*, 86(2):256–274, Jan 2010. ISSN 1573-1405. doi: 10.1007/s11263-008-0200-2. URL https://doi.org/10.1007/s11263-008-0200-2.

[5] Kenneth V. Beard and Catherine C. Chuang. A new model for the equilibrium shape of raindrops. *Journal of the Atmospheric Sciences*, 44: 1509–1524, 1987. URL https://api.semanticscholar.org/CorpusID:121259889.

[6] Edwin X Berry and Maarten R. Pranger. Equations for calculating the terminal velocities of water drops. *Journal of Applied Meteorology (1962-1982)*, 13(1):108–113, 1974. ISSN 00218952, 2163534X. URL http://www.jstor.org/stable/26176880.

[7] Nathan Brewer and Nianjun Liu. Using the shape characteristics of rain to identify and remove rain from video. In Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James T. Kwok, Michael Georgiopoulos, Georgios C. Anagnostopoulos, and Marco Loog, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 451–458, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-89689-0.

[8] Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, page 22–es, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781450347266. doi: 10.1145/1278780.1278807. URL https://doi.org/10.1145/1278780.1278807.

[9] Carlos Castillo, Jorge López-Moreno, and Carlos Aliaga. Recent advances in fabric appearance reproduction. *Comput. Graph.*, 84(C):103–121, nov 2019. ISSN 0097-8493. doi: 10.1016/j.cag.2019.07.007. URL https://doi.org/10.1016/j.cag.2019.07.007.

[10] E.C.C Choi. Simulation of wind-driven-rain around a building. *Journal of Wind Engineering and Industrial Aerodynamics*, 46-47:721–729, 1993. ISSN 0167-6105. doi: https://doi.org/10.1016/0167-6105(93)90342-L. URL https://www.sciencedirect.com/science/article/

pii/016761059390342L. Proceedings of the 1st International on Computational Wind Engineering.

[11] Robert L. Cook and Tony DeRose. Wavelet noise. *ACM Trans. Graph.*, 24(3):803–811, jul 2005. ISSN 0730-0301. doi: 10.1145/1073204.1073264. URL https://doi.org/10.1145/1073204.1073264.

[12] C.N. Davies. Unpublished rept. 1939.

[13] A. Nelson Dingle and Yean Lee. Terminal fallspeeds of raindrops. *Journal of Applied Meteorology (1962-1982)*, 11(5):877–879, 1972. ISSN 00218952, 2163534X. URL http://www.jstor.org/stable/26175465.

[14] G. B. Foote and P. S. du Toit. Terminal velocity of raindrops aloft. *Journal of Applied Meteorology (1962-1982)*, 8(2):249–253, 1969. ISSN 00218952, 2163534X. URL http://www.jstor.org/stable/26174522.

[15] K. Garg and S.K. Nayar. Detection and removal of rain from videos. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, 2004. doi: 10.1109/CVPR.2004.1315077.

[16] Kshitiz Garg and Shree K. Nayar. When does a camera see rain? *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, 2:1067–1074 Vol. 2, 2005. URL https://api.semanticscholar.org/CorpusID:206769439.

[17] Kshitiz Garg and Shree K. Nayar. Photorealistic rendering of rain streaks. *ACM Trans. Graph.*, 25(3):996–1002, jul 2006. ISSN 0730-0301. doi: 10.1145/1141911.1141985. URL https://doi.org/10.1145/1141911.1141985.

[18] Kshitiz Garg and Shree K. Nayar. Vision and rain. *International Journal of Computer Vision*, 75(1):3–27, Oct 2007. ISSN 1573-1405. doi: 10.1007/s11263-006-0028-6. URL https://doi.org/10.1007/s11263-006-0028-6.

[19] Kshitiz Garg, Gurunandan G. Krishnan, and Shree K. Nayar. Material Based Splashing of Water Drops. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques*. The Eurographics Association, 2007. ISBN 978-3-905673-52-4. doi: 10.2312/EGWR/EGSR07/171-182.

[20] Michał Gawron and Urszula Boryczka. Procedural rain ripples generated using worley noise. In Adam Wojciechowski and Piotr Napieralski, editors, *Computer Game Innovations*, page 39–48. Lodz University of Technology Press, 2018.

[21] Andrea Lo Giudice, Roberto Nuca, Luigi Preziosi, and Nicolas Coste. Wind-blown particulate transport: A review of computational fluid dynamics models. *Mathematics in Engineering*, 1(3):508–547, 2019. ISSN

2640-3501. doi: 10.3934/mine.2019.3.508. URL https://www.aimspress.com/article/doi/10.3934/mine.2019.3.508.

[22] Ross Gunn and Gilbert D. Kinzer. The terminal velocity of fall for water droplets in stagnant air. *Journal of Atmospheric Sciences*, 6(4): 243 – 248, 1949. doi: 10.1175/1520-0469(1949)006⟨0243:TTVOFF⟩2.0.CO;2. URL https://journals.ametsoc.org/view/journals/atsc/6/4/1520-0469_1949_006_0243_ttvoff_2_0_co_2.xml.

[23] Sinan Hasirlioglu and Andreas Riener. A general approach for simulating rain effects on sensor data in real and virtual environments. *IEEE Transactions on Intelligent Vehicles*, 5(3):426–438, 2020. doi: 10.1109/TIV.2019.2960944.

[24] James Hegarty. Geometric and path tracing methods for simulating light transport through volumes of water particles. *XRDS*, 15(1):16–20, sep 2008. ISSN 1528-4972. doi: 10.1145/1452012.1452016. URL https://doi.org/10.1145/1452012.1452016.

[25] Louis G Henyey and Jesse Leonard Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal, vol. 93, p. 70-83 (1941).*, 93:70–83, 1941.

[26] Milan Ikits, Joe Kniss, Aaron Lefohn, and Charles Hansen. Volume rendering techniques. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, chapter 39. Pearson Higher Education, 2004.

[27] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, page 133–142, New York, NY, USA, 1986. Association for Computing Machinery. ISBN 0897911962. doi: 10.1145/15922.15901. URL https://doi.org/10.1145/15922.15901.

[28] Marc Jaeger. Enhancing virtual natural scenes using quick and dirty image based recipes. In *2012 IEEE 4th International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications*, pages 164–171, 2012. doi: 10.1109/PMA.2012.6524829.

[29] A. R. Jameson and A. B. Kostinski. What is a raindrop size distribution? *Bulletin of the American Meteorological Society*, 82(6):1169 – 1178, 2001. doi: 10.1175/1520-0477(2001)082⟨1169:WIARSD⟩2.3.CO;2. URL https://journals.ametsoc.org/view/journals/bams/82/6/1520-0477_2001_082_1169_wiarsd_2_3_co_2.xml.

[30] Henrik Wann Jensen, Justin Legakis, and Julie Dorsey. Rendering of wet materials. In *Rendering Techniques' 99: Proceedings of the Eurographics Workshop in Granada, Spain, June 21–23, 1999 10*, pages 273–281. Springer, 1999.

[31] H. Jiang, M. Sano, and M. Sekine. Weibull raindrop-size distribution and its application to rain attenuation. *IEE Proceedings: Microwaves, Antennas and Propagation*, 144(3):197 – 200, 1997. doi: 10.1049/ip-map:19971193. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031164659&doi=10.1049%2fip-map%3a19971193&partnerID=40&md5=4ed23c0d03c0ed18b0b9bb9180d95f88. Cited by: 46.

[32] N.L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*. Number v. 1 in Continuous Univariate Distributions. J. Wiley, 1970. ISBN 9780471446262.

[33] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, page 143–150, New York, NY, USA, 1986. Association for Computing Machinery. ISBN 0897911962. doi: 10.1145/15922.15902. URL https://doi.org/10.1145/15922.15902.

[34] Peter Kutz, Ralf Habel, Yining Karl Li, and Jan Novák. Spectral and decomposition tracking for rendering heterogeneous volumes. *ACM Trans. Graph.*, 36(4), jul 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073665. URL https://doi.org/10.1145/3072959.3073665.

[35] Ayşe Kızılersü, Markus Kreer, and Anthony W. Thomas. The Weibull Distribution. *Significance*, 15(2):10–11, 04 2018. ISSN 1740-9705. doi: 10.1111/j.1740-9713.2018.01123.x. URL https://doi.org/10.1111/j.1740-9713.2018.01123.x.

[36] H. Lamb. *Hydrodynamics*. University Press, 1916.

[37] Irving Langmuir. The production of rain by a chain reaction in cumulus clouds at temperatures above freezing. *Journal of Atmospheric Sciences*, 5(5):175 – 192, 1948. doi: 10.1175/1520-0469(1948)005⟨0175:TPORBA⟩2.0.CO;2. URL https://journals.ametsoc.org/view/journals/atsc/5/5/1520-0469_1948_005_0175_tporba_2_0_co_2.xml.

[38] JOHN O. Laws and Donald A. Parsons. The relation of raindrop-size to intensity. *Eos, Transactions American Geophysical Union*, 24(2):452–460, 1943. doi: https://doi.org/10.1029/TR024i002p00452. URL https://api.semanticscholar.org/CorpusID:140141884.

[39] Amazon Lumberyard. Amazon lumberyard bistro, open research content archive (orca), July 2017. URL http://developer.nvidia.com/orca/amazon-lumberyard-bistro. http://developer.nvidia.com/orca/amazon-lumberyard-bistro.

[40] Bailey Miller, Iliyan Georgiev, and Wojciech Jarosz. A null-scattering path integral formulation of light transport. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 38(4), July 2019. ISSN 0730-0301. doi: 10/gf6rzb.

[41] Yoshiki Mizukami, Katsuhiro Sasaki, and Katsumi Tadamura. Realistic rain rendering. In *Proceedings of the Third International Conference on Computer Graphics Theory and Applications - Volume 1: GRAPP, (VISIGRAPP 2008)*, pages 273–280. INSTICC, SciTePress, 2008. ISBN 978-989-8111-20-3. doi: 10.5220/0001095702730280.

[42] Guillermo Montero-Martínez and Fernando García-García. On the behaviour of raindrop fall speed due to wind. *Quarterly Journal of the Royal Meteorological Society*, 142(698):2013–2020, 2016. doi: https://doi.org/10.1002/qj.2794. URL `https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.2794`.

[43] Eihachiro Nakamae, Kazufumi Kaneda, Takashi Okamoto, and Tomoyuki Nishita. A lighting model aiming at drive simulators. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, page 395–404, New York, NY, USA, 1990. Association for Computing Machinery. ISBN 0897913442. doi: 10.1145/97879.97922. URL `https://doi.org/10.1145/97879.97922`.

[44] Nobuyuki Nakata. Animation of water droplets on a hydrophobic windshield. 2012. URL `https://api.semanticscholar.org/CorpusID:17333937`.

[45] Jan Novák, Iliyan Georgiev, Johannes Hanika, and Wojciech Jarosz. Monte Carlo methods for volumetric light transport simulation. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)*, 37(2), May 2018. doi: 10/gd2jqq.

[46] Cornell University Program of Computer Graphics. Cornell box data. URL `https://www.graphics.cornell.edu/online/box/data.html`.

[47] Victor Ostromoukhov. Sampling with polyominoes. *ACM Trans. Graph.*, 26(3):78–es, jul 2007. ISSN 0730-0301. doi: 10.1145/1276377.1276475. URL `https://doi.org/10.1145/1276377.1276475`.

[48] K. Perlin and E. M. Hoffert. Hypertexture. *SIGGRAPH Comput. Graph.*, 23(3):253–262, jul 1989. ISSN 0097-8930. doi: 10.1145/74334.74359. URL `https://doi.org/10.1145/74334.74359`.

[49] Ken Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, page 287–296, New York, NY, USA, 1985. Association for Computing Machinery. ISBN 0897911660. doi: 10.1145/325334.325247. URL `https://doi.org/10.1145/325334.325247`.

[50] Ken Perlin. Improving noise. *ACM Trans. Graph.*, 21(3):681–682, jul 2002. ISSN 0730-0301. doi: 10.1145/566654.566636. URL `https://doi.org/10.1145/566654.566636`.

[51] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering, fourth edition: From Theory to Implementation*. MIT Press, 2023. ISBN 9780262048026. URL `https://pbr-book.org/`.

[52] PINKE. Rain!, 2009. URL `https://flic.kr/p/6c2454`.

[53] Mikhail N. Polyanskiy. Refractiveindex.info database of optical constants. *Scientific Data*, 11(1):94, Jan 2024. ISSN 2052-4463. doi: 10.1038/s41597-023-02898-2. URL https://doi.org/10.1038/s41597-023-02898-2.

[54] Pierre Rousseau, Vincent Jolivet, and Djamchid Ghazanfarpour. Realistic real-time rain rendering. *Computers & Graphics*, 30(4):507–518, 2006. ISSN 0097-8493. doi: https://doi.org/10.1016/j.cag.2006.03.013. URL https://www.sciencedirect.com/science/article/pii/S0097849306000859.

[55] Astrid Sanusi, Henri Wortham, Maurice Millet, and Philippe Mirabel. Chemical composition of rainwater in eastern france. *Atmospheric Environment*, 30(1):59–71, 1996. ISSN 1352-2310. doi: https://doi.org/10.1016/1352-2310(95)00237-S. URL https://www.sciencedirect.com/science/article/pii/135223109500237S.

[56] Matsuo Sekine and Goran Lind. Raindrop shape limitations on clutter cancellation ratio using circular polarization. *IEEE Transactions on Aerospace and Electronic Systems*, AES-19(4):631–633, 1983. doi: 10.1109/TAES.1983.309352.

[57] Maria A. Serio, Francesco G. Carollo, and Vito Ferro. Raindrop size distribution and terminal velocity for rainfall erosivity studies. a review. *Journal of Hydrology*, 576:210–228, 2019. ISSN 0022-1694. doi: https://doi.org/10.1016/j.jhydrol.2019.06.040. URL https://www.sciencedirect.com/science/article/pii/S0022169419305876.

[58] Sonia Starik and Michael Werman. Simulation of rain in videos. 2002. URL https://api.semanticscholar.org/CorpusID:1305628.

[59] M F Subhani and J. P. Oakley. Low latency mitigation of rain induced noise in images. In *5th European Conference on Visual Media Production (CVMP 2008)*, pages 1–4, 2008. doi: 10.1049/cp:20081070.

[60] Natalya Tatarchuk. Artist-directable real-time rain rendering in city environments. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, page 23–64, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933646. doi: 10.1145/1185657.1185828. URL https://doi.org/10.1145/1185657.1185828.

[61] Firat Testik and Md Rahman. First in-situ observations of binary raindrop collisions: Binary raindrop collisions. *Geophysical Research Letters*, 44, 01 2017. doi: 10.1002/2017GL072516.

[62] Abhishek Tripathi and Sudipta Mukhopadhyay. Removal of rain from videos: a review. *Signal, Image and Video Processing*, 8, 11 2012. doi: 10.1007/s11760-012-0373-6.

[63] Changbo Wang, Zhangye Wang, Xin Zhang, Lei Huang, Zhiliang Yang, and Qunsheng Peng. Real-time modeling and rendering of raining scenes. *The Visual Computer*, 24(7):605–616, Jul 2008. ISSN 1432-2315. doi: 10.1007/s00371-008-0241-0. URL https://doi.org/10.1007/s00371-008-0241-0.

[64] Cheng Wang, Meng Yang, Xuemin Liu, and Gang Yang. Realistic simulation for rainy scene. *J. Softw.*, 10:106–115, 2015. URL https://api.semanticscholar.org/CorpusID:34152456.

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

[66] P. K. Wang and H. R. Pruppacher. Acceleration to terminal velocity of cloud and raindrops. *Journal of Applied Meteorology (1962-1982)*, 16(3):275–280, 1977. ISSN 00218952, 2163534X. URL http://www.jstor.org/stable/26178108.

[67] J Wiesner. Beiträge zur kenntniss des tropischen regens. *Sitz Ber Math Nat Akad Wiss Kl*, 104:1397, 1895.

[68] A. Wilkie, S. Nawaz, M. Droske, A. Weidlich, and J. Hanika. Hero wavelength spectral sampling. *Computer Graphics Forum*, 33(4):123–131, 2014. doi: https://doi.org/10.1111/cgf.12419. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12419.

[69] Hermann B. Wobus, F. W. Murray, and L. R. Koenig. Calculation of the terminal velocity of water drops. *Journal of Applied Meteorology (1962-1982)*, 10(4):751–754, 1971. ISSN 00218952, 2163534X. URL http://www.jstor.org/stable/26175683.

[70] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 291–294, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464. doi: 10.1145/237170.237267. URL https://doi.org/10.1145/237170.237267.

[71] Y. Yang, X. Wang, and M. Beheshti. Blurry when wet: animating raindrop behavior. *IEEE Potentials*, 24(3):33–36, 2005. doi: 10.1109/MP.2005.1502504.

[72] Cem Yuksel. Sample elimination for generating poisson disk sample sets. *Computer Graphics Forum*, 34(2):25–32, 2015. doi: https://doi.org/10.1111/cgf.12538. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12538.

[73] Xiaopeng Zhang, Hao Li, Yingyi Qi, Wee Kheng Leow, and Teck Khim Ng. Rain removal in video by combining temporal and chromatic properties. In *2006 IEEE International Conference on Multimedia and Expo*, pages 461–464, 2006. doi: 10.1109/ICME.2006.262572.

A Calculation of Density

Let t be defined as a point in time, where $0 \leq t \leq 1$, such that $t = 0$ refers to the time at which the virtual shutter opens and $t = 1$ the time at which it closes. c_t refers to the center of the rain drop R at time t. r refers to the radius of the rain drop. See figure 31 for a visual representation.

$dist(a, b)$ is used to refer to the Euclidian distance between point a and point b.

Let c be the initial center of the drop and $\vec{m}$ the movement vector of the drop, with the length of $\vec{m}$ being nonzero. The center of the drop at time t is described as $c_t = c + \vec{m} * t$, and thus $c_0 = c$.

For convenience, let us take that drops only move along the y-axis, such that $m_y \neq 0$ and $\vec{m}$ is zero in all other dimensions. This already holds for drops falling down without diagonal movement in a scene where the y-axis points up. Otherwise, the coordinate system can be rotated such that $\vec{m}$ becomes zero in all other dimensions.

A.1 Calculation of ρ_d

For point c_t and a ray with origin o and normalised direction $\vec{r}$, let us calculate the range of values t such that there exists a value a for which $dist(o + \vec{r} * a, c + \vec{m} * t) \leq r$ holds and divide it by the length of the ray segment intersecting with the drop, this being the length of the range of values a such that there exists a value t (including values of t smaller than 0 or greater than 1) for which $dist(o + \vec{r} * a, c + \vec{m} * t) \leq r$ holds.

Let us define $\Delta_x = o_x - c_x$, $\Delta_y = o_y - c_y$ and $\Delta_z = o_z - c_z$ for convenience.

First, let us calculate the length of the interval of a for which $dist(o + \vec{r} * a, c + \vec{m} * t) \leq r$ for any value of t. This is the case when $(\Delta_x + r_x * a)^2 + (\Delta_z + r_z * a)^2 - r^2 \geq 0$. Name the minimum value of a for which this holds a_{min} and the maximum a_{max}. Then

$$a_{min} = \frac{\Delta_x r_x + \Delta_z r_z - 2\sqrt{2\Delta_x r_x \Delta_z r_z - (r_z^2 * \Delta_x^2 + r_x^2 * \Delta_z^2) + (r_x^2 + r_z^2) * 4r^2}}{r_x^2 + r_z^2}$$

and

$$a_{max} = \frac{\Delta_x r_x + \Delta_z r_z + 2\sqrt{2\Delta_x r_x \Delta_z r_z - (r_z^2 * \Delta_x^2 + r_x^2 * \Delta_z^2) + (r_x^2 + r_z^2) * 4r^2}}{r_x^2 + r_z^2}$$

This makes the interval between a_{max} and a_{min} as follows:

$$a_{max} - a_{min} = \frac{4\sqrt{2\Delta_x r_x \Delta_z r_z - (r_z^2 * \Delta_x^2 + r_x^2 * \Delta_z^2) + (r_x^2 + r_z^2) * 4r^2}}{r_x^2 + r_z^2}$$

Next, let us calculate the length of the interval of t for which $dist(o + r * a, c_t) = r$ for any value of a. This is the amount of time the ray intersects the drop. The first collision happens at a_{min} at the earliest, the last collision at a_{max} at the latest. Due to a_{min} and a_{max} being calculated without the constraint of

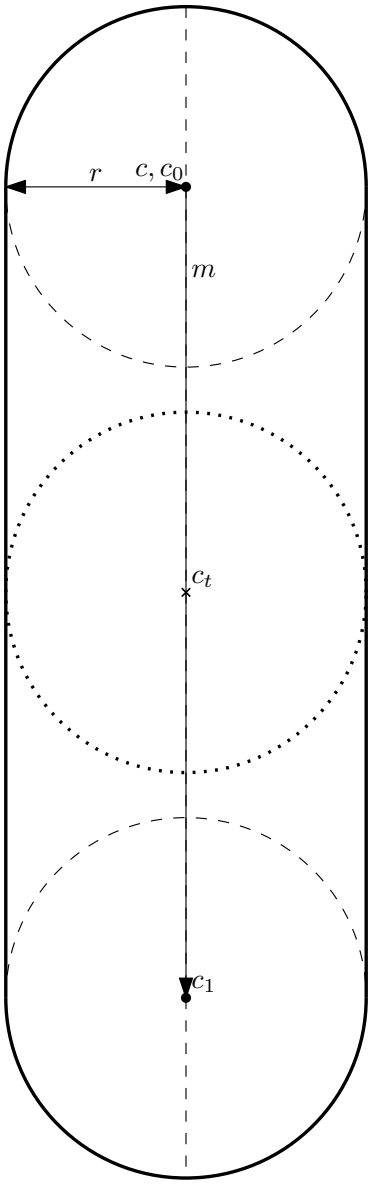


Figure 31: Visual representation of the variables used to describe a raindrop. $c = c_0$ refers to the origin of the drop when the shutter opens and c_1 to the origin of the drop when the shutter closes. r is the radius of the drop. $\vec{m}$ is the vector from c_0 to c_1 such that $c_t = c + t * \vec{m}$.

$0 \leq t \leq 1$ the true collision may be slightly earlier or later. As such, any value for t calculated with these values of a does still need to be clamped in the end.

First, let us calculate the values of t such that $dist(o + \vec{r} * a_{min}, c + \vec{m} * t = r$ for any a.

$(\Delta_x + r_x * a)^2 + (\Delta_y + r_y * a + m_y * t)^2 + (\Delta_z + r_z * a)^2 \leq r^2$

$$t = \frac{-2\Delta_y - 2r_y a \pm \sqrt{(\Delta_x + r_x * a)^2 + (\Delta_z + r_z * a)^2 - r^2}}{m_y}$$

$$t_{min} = \frac{-2\Delta_y - 2r_y a_{min} - \sqrt{(\Delta_x + r_x * a_{min})^2 + (\Delta_z + r_z * a_{min})^2 - r^2}}{m_y}$$

$$t_{max} = \frac{-2\Delta_y - 2r_y a_{max} + \sqrt{(\Delta_x + r_x * a_{max})^2 + (\Delta_z + r_z * a_{max})^2 - r^2}}{m_y}$$

Both t_{min} and t_{max} should be clamped to $clamp(t_{min}, 0, 1)$ and $clamp(t_{max}, 0, 1)$ respectively.

All in all,

$$\rho_d = \frac{clamp(t_{max}, 0, 1) - clamp(t_{min}, 0, 1)}{a_{max} - a_{min}}$$

.

A.2 Calculation of p_d

For point c_t and a sample position s, let us calculate the range of values t such that $dist(s, c_t) \leq r$ holds.

First, let us calculate the values of t for which $dist(s, c_t) = r$. $dist(s, c_0 + t * \vec{m}) = r$

$\sqrt{(s_x - c_x)^2 + (s_y - c_y - t * m_y)^2 + (s_z - c_z)^2} = r$

Let us define $\Delta_x = s_x - c_x$, $\Delta_y = s_y - c_y$ and $\Delta_z = s_z - c_z$ for convenience.

We then get that $\sqrt{\Delta_x^2 + (\Delta_y - t * m_y)^2 + \Delta_z^2} = r$, which can be rewritten to $t = \frac{\Delta_y \pm \sqrt{r^2 - \Delta_x^2 - \Delta_z^2}}{m_y}$.

Thus $dist(s, c_t) \leq r$ holds for

$$\frac{\Delta_y - \sqrt{r^2 - \Delta_x^2 - \Delta_z^2}}{m_y} \leq t \leq \frac{\Delta_y + \sqrt{r^2 - \Delta_x^2 - \Delta_z^2}}{m_y}$$

.

Note that in a single frame t only ranges between 0 and 1, and thus these values should be clamped before use. Taking

$$clamp(x, a, b) = \begin{cases} a, & \text{if } x < a \\ b, & \text{if } x > b \\ x, & \text{otherwise} \end{cases}$$

, we then get

$$clamp(\frac{\Delta_y - \sqrt{r^2 - \Delta_x^2 - \Delta_z^2}}{m_y}, 0, 1) \leq t \leq clamp(\frac{\Delta_y + \sqrt{r^2 - \Delta_x^2 - \Delta_z^2}}{m_y}, 0, 1)$$

The fraction of time $dist(s, c_t) \leq r$ holds in a given frame is equal to the length of the interval of t derived above:

$$p_d = clamp(\frac{\Delta_y + \sqrt{r^2 - \Delta_x^2 - \Delta_z^2}}{m_y}, 0, 1) - clamp(\frac{\Delta_y - \sqrt{r^2 - \Delta_x^2 - \Delta_z^2}}{m_y}, 0, 1)$$

If $r^2 - \Delta_x^2 - \Delta_z^2 < 0$, this equation has no solutions: the drop and the point never overlap and thus $p_d = 0$.