OPTIMIZING THE MANAGEMENT AND RENDERING OF RAIN

Anna Puig-Centelles, Oscar Ripollesa and Miguel Chover
Computer and Languages Department, Universitat Jaume I, Castellon, Spain

Keywords: Real-time Rain, Particle Systems, Multiresolution, Virtual Environments, GPU, Geometry Shader.

Abstract: Current rain solutions are capable of offering very realistic images. Nevertheless, this realism often involves extremely high rendering costs. Our proposal is aimed at facilitating the creation and management of rain scenes. On the one hand, we define the areas in which it is raining and, on the other hand, we perform a suitable management of the particle systems inside them. The latter step is reached by means of level-of-detail techniques which are applied directly in the Geometry Shader by modifying the size and the number of rendered particles. Our results prove that fully-integrating the level-of-detail management in the GPU considerably increases the final performance.

1 INTRODUCTION

Realistic-looking rain greatly enhances scenes of outdoor reality, with applications including computer games and motion pictures. Rain rendering has been addressed by different authors in order to describe methods which define not only the rainfall (Tatarchuk, 2006; Changbo et al., 2008), but also their interaction with other surfaces (Wang et al., 2005) or even the water accumulation (Fearing, 2000).

Traditionally, authors have developed rain simulation frameworks which are based on particle systems. These systems have been successfully used to simulate in real time different kinds of fuzzy phenomena, like smoke or fire. Nevertheless, they present some limitations due to the cost of translating and rendering the high amount of raindrops that must be represented in order to offer a realistic rain appearance.

A possible solution to overcome these limitations is the exploitation of the current GPU possibilities, whose constant evolution can considerably increase the final performance. Moreover, it is worth considering the possibilities of applying multiresolution techniques to existing particle systems.

Little work has been carried out in using level-of-detail approaches within particle systems. It is important to comment on the work presented in (O'Brien et al., 2001) where the authors propose a hierarchy of particles and consider the use of physical properties for switching among LODs in order to obtain a higher resolution when necessary. Regarding rain, we must also comment on the solution introduced in (Puig-Centelles et al., 2008) which gives some initial ideas on improving rain rendering by means of multiresolution techniques.

From a different perspective, it is interesting to consider that real-world raining scenarios include rainy areas and also non-rainy areas. Even when it is raining in a very wide area, we must assume that we could always find non-rainy areas nearby. Traditionally, rain simulation has not considered this issue and their rain systems do not provide smooth transitions between areas with different rain conditions.

The work presented in this paper is proposed as a solution for diminishing the management cost of par-

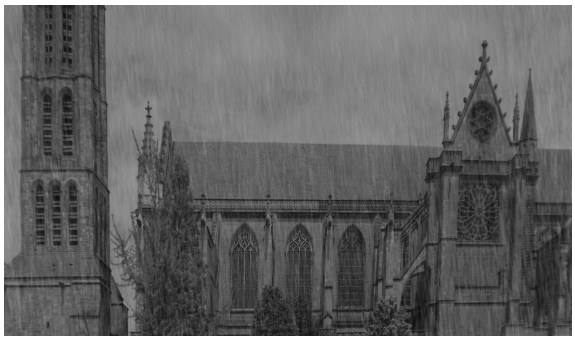

Figure 1: Rain sensation obtained with our solution in a scene with a building-textured skybox.

Puig-Centelles A., Ripollesa O. and Chover M.
OPTIMIZING THE MANAGEMENT AND RENDERING OF RAIN.
DOI: 10.5220/0001801803730378
In *Proceedings of the Fourth International Conference on Computer Graphics Theory and Applications (VISIGRAPP 2009)*, page
ISBN: 978-989-8111-67-8

ticle systems. To achieve this objective we propose a method for automatically generating rain environments with the definition of rain areas and with an adequate management of the particles created inside these areas. Furthermore, we include the LOD concept in the GPU in order to adjust the size and the number of particles to the conditions of the scene. In Figure 1 we have an example of a rain scenario obtained with our solution.

This paper is organized as follows. Section 2 considers the state of the art on rain rendering. Section 3 introduces the concept of raining area as well as its interactions. Section 4 presents the multiresolution implementation. Section 5 offers the results. Lastly, Section 6 contains some remarks on our solution.

2 STATE OF THE ART

Rain has been traditionally rendered in two ways, either as camera-centered geometry with *scrolling textures* or as a *particle system*.

Scrolling Textures. This approach is based on the idea of using a texture that covers the whole scene. Then, the application scrolls it by following the falling direction of the rain. However, this technique exhibits certain properties that are stationary in time (Soatto et al., 2001). In (Wang and Wade, 2004), the authors present a novel technique for rendering precipitation in scenes with moving camera positions. They map textures onto a double cone, and translate and elongate them using hardware texture transforms.

These methods fail to create a truly convincing rain impression because of the lack depth and the rain not reacting accurately to scene illumination. To overcome this limitation, some authors (Tatarchuk, 2006) have developed more complex solutions which include several layers of rain for simulating rainfall at different distances, although the problem is not completely solved.

Particle Systems. Traditionally, this has been the approach chosen for real-time rendering of rain, even though particle systems tend to be expensive, especially if we want to render heavy rain. Lately, rendering of rain has become very realistic, although the management of these systems in real-time applications still poses severe restrictions.

The work presented in (Kusamoto et al., 2001) introduced a physical motion model for rain rendering with particle systems. Later, the authors of (Feng et al., 2006) considered the physical properties from a different point of view. They developed a collision detection method for raindrops and created a particle subsystem of raindrops splashing after collision.

Wang et al. presented in (Wang et al., 2006) a system formed by two parts: off-line image analysis of rain videos and real-time particle-based synthesis of rain. This solution is cost-effective and capable of realistic rendering of rain in real time.

Following a similar approach, N. Tatarchuk (Tatarchuk, 2006) developed a hybrid system of an image-space approach for the rainfall and particle-based effects for dripping raindrops and splashes. It presents a detailed rainy environment and provides a high degree of artistic control. The main issue is that it requires 300 unique shaders dedicated to rain alone. Moreover, the simulation requires the camera to maintains a fixed viewing direction.

Rousseau et al. (Rousseau et al.,) propose a rain rendering method that simulates the refraction of the scene inside a raindrop. The scene is captured to a texture which is distorted according to the optical properties of raindrops and mapped onto each raindrop by means of a vertex shader.

In paper (Tariq, 2007), Tariq proposes a realistic rain application that works entirely on the GPU. Rain particles are animated over time and in each frame they are expanded into billboards to be rendered using the *Geometry Shader*. The rendering of the rain particles uses a library of textures, which encodes the appearance of the raindrops under different viewpoint and lighting directions (Garg and Nayar, 2006).

More recently, the work presented in (Changbo et al., 2008) introduces a new framework which thoroughly address physical properties of rain, visual appearance, foggy effects, light interactions and scattering. The main drawback of this approach is that, despite offering very realistic simulations, their method cannot render a scene with a sufficient framerate to offer interactive walkthroughs.

Lastly, it is worth mentioning the study made in (Puig-Centelles et al., 2008), where the authors give some initial ideas and results about the application of level-of-detail techniques when rendering realistic rain. They propose the adaptation of the size of the particles in the GPU, although the number of particles and their distribution are initially fixed in the CPU. Moreover, their exploitation of the Geometry Shader is quite limited.

3 RAIN MANAGEMENT

One of the main objectives of our model is to be able to generate rainy environments automatically by cre-

ating raining areas where thousands of drops are simulated in real time. Within each rain area, we will simulate rain with a particle system in which we include a new level-of-detail framework in order to improve performance.

The rain framework we are introducing tries to establish with reasonable precision where it is raining and where it is not raining inside a given scenario. Furthermore, we also want to handle transitions in order to create a visually realistic raining effect. An important aspect of this realism is the possibility of looking at a far rain area from a non-rainy place.

We have considered that a rain area is simulated as an ellipse with two initially given radii.

3.1 Rain Container

It is necessary to create and manage a *rain container*, which encloses the whole set of raindrops included in our particle system. It will follow the camera movements continuously so that the user does not lose the rain perception.

In the literature, it is possible to find methods which use a box-shaped container (Rousseau et al.,), a cylindrical container (Tariq, 2007) and a semi-cilyndrical rain container (Puig-Centelles et al., 2008). Under our circumstances, we consider that a semi-cilyndrical container with an ellipsoidal base is more adequate for our purposes. It is more ergonomically adjusted to the user's field-of-view than a box, it is more efficient than a whole cilynder and, in addition, it allows the user to perform small turns to the right or to the left without requiring to reposition the whole rain container. More precisely, if the user makes changes in its orientation of less than 45° in any direction then it will not be necessary to reorient the container as the rain perception is maintained. This feature is possible due to the selected shape of the container and improves previous solutions.

It is also important to take into account the size of the container. The previously mentioned solutions had difficulties in rendering scenes with heavy rain and a continuously moving user. This happened due to the fact that the management cost of their particle systems makes it impossible to manage these huge number of particles.

In order to overcome this limitation and use a bigger container, our solution incorporates LOD techniques to efficiently modify the particle size and location while preserving the realistic appearance of rain. Thus, by combining these techniques with the selected shape of the container, our method offers an appropriate solution for reacting efficiently to changes in the position and direction of the *frustum* of the user.

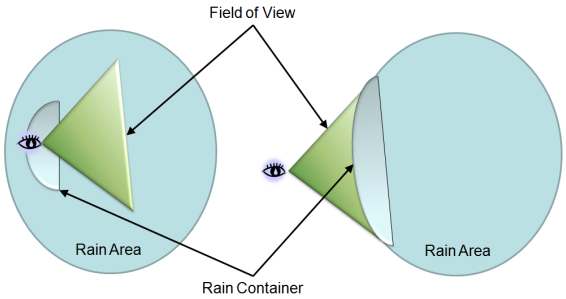

Figure 2: User point of view of the rain area, showing the two possibilities.

3.2 Simulating Rain Areas

The user could interact with a rain area in two different ways. On one hand, the user can be inside the rain area, completely surrounded by raindrops. We have named this situation *Close Rain*. On the other hand, the user can be outside the rain area, while looking at it from a distant place or moving away from the area. In these cases we use the *Far Rain* model.

These two possibilities are shown in Figure 2. It is important to note that the rain container has different behaviors depending on whether we are using the close rain or the far rain approach. So, the image on the left refers to close rain while the other one shows a possible circumstance in which we must use far rain.

3.3 Close Rain

This case occurs when the observer is inside the rain area. The rain container is initially centered on the observer. Nevertheless, the location of this container is updated in order to follow the user's movements. So, if the user moves then the container moves with him; if the user changes the view direction then the container will be reoriented too. As we have mentioned above, we establish different thresholds that must be overcome before re-locating the rain container in order to avoid updating its location continuously.

3.4 Far Rain

Far rain refers to a scenario where the observer is located far away from the rain area. This situation happens when the camera is placed inside an area where it is not raining and the camera can look at the area where it is actually raining.

In order to check whether the user can see the rain area, we do a simple test against the *frustum*. If the test is passed, the first step consists in centering the rain container within the perimeter of the ellipsoidal rain area, at the closest point to the observer. Then,

we should modify the size of the rain container in order to cover the whole part of the rain area that the observer can see. To do so, we consider again the frustum to determine which is its relation with the ellipse that defines the rain area. Once we have tested the intersections between the frustum and the ellipse, we will know whether the observer can see the whole rain area or just a part of it. If the observer can only see a part of the container, the same test will tell us the size of this part in order to adjust the size of the container correctly. Figure 2 shows a possible adaptation of the far rain container with regard to the features of the observer view. Moreover, we must perform similar tests to control the height of the rain container, expanding it as necessary in order to assure that the user perceives that the raindrops fall from the sky.

3.5 Transitions between Rain Cases

Within the proposed solution for simulating rain particles, it is essential to control properly the transitions that an observer can experiment when changing from *close rain* to *far rain* and vice versa. We must control whether it is necessary to perform the changes in order to preserve the realism of our rain system.

The transition from close rain to far rain occurs when the user leaves the rain area. Our system detects that change when the user is positioned on the perimeter of the ellipse that defines the rain area. In that case, the rain container remains static at that point. The amount of raindrops will decrease progressively while the user moves away from the rain container which is no longer following the user's movements.

Regarding the transition from far rain to close rain, we again use the perimeter of the rain area. As we have commented before, we calculate the size of the rain area that the user can see, in order to adapt the shape of the rain container. As the observer gets closer to the rain area, the size of the user's perceived area becomes smaller and the container reacts accordingly, decreasing its size. This decrease in size assures that once the user reaches the perimeter of the rain area the container will have the original size of the container. Consequently, we can assure that changing from far to close rain is almost imperceptible for the user.

4 MULTIRESOLUTION IMPLEMENTATION

In our work we decided to follow the particle distribution presented in (Puig-Centelles et al., 2008). These authors described a level-of-detail distribution of particles which assured that there were more rain particles and smaller in those areas close to the viewer and less particles but bigger in the areas far from the user. Therefore, in the farther zones we will render less particles but with a bigger size so that the global view of the system resembles the same intensity of rain.

The solution we are presenting proposes adapting the level-of-detail directly in the GPU. The previous multiresolution rain model (Puig-Centelles et al., 2008) obliged the system to initially define the position of each particles and their quantity. As a consequence, the spatial distribution of the raindrops was fixed and initially defined in the CPU. Our objective consists in deciding to render more or less particles directly in the GPU.

In order to accomplish this objective, our system needs two rendering passes. The first one is in charge of updating the position of each particle, while the second one applies the multiresolution techniques and also renders the final particle system. During the development of this framework we tried to perform all these tasks in only one single pass. The difficulties appeared when working in the pipeline at the same time with points and quads. The first pipeline is in charge of translating vertically each raindrop to simulate the rainfall. In this pass we use points as the rendering primitive. Following that, the second pipeline pass will initially receive points but will output quads for the rain rendering. By updating directly the quads we would be obliged to translate, re-locate and output four vertices instead of one for each raindrop, which our tests have proven to be much more costly.

4.1 Patterns for Creating Raindrops

The main idea of our solution is to use the GPU to dynamically create more particles in those areas that are closer to the user. The *Geometry Shader* facilitates this task by enabling to use each particle as a seed to create many more replics (quads). As we have mentioned before, the distance is the criterion used to decide the most suitable size for a particle. This criterion is also used to decide how many particles we should render from each seed particle.

In Figure 3 we present three sample patterns that are used to generate more particles from an initial one. The seed particle is depicted in blue and the new generated ones in green. It is relevant to note that we have decided to create a maximum number of 6 raindrops per seed particles. Our tests have proven that with that number it is enough to get a proper rain impression without increasing too much the rendering cost. Moreover, we have applied those three patterns in our

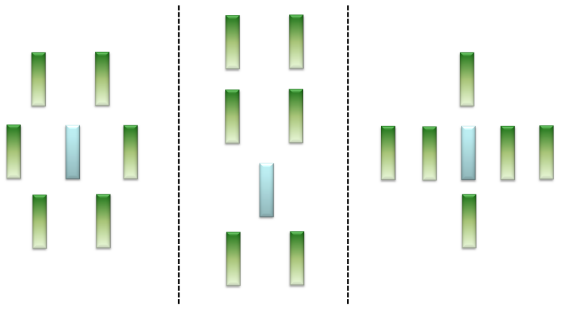

Figure 3: Sample patterns for rain generation.

test as they are sufficient to avoid the user to perceive repeated patterns. Nevertheless, both the number of generated particles and the patterns followed can be easily modified in our framework.

4.2 GPU Implementation

In order to avoid all the raindrops following similar falling paths, we initially give each particle a slightly different falling speed and direction. Each particle is given a falling vector, which indicates the displacement direction and amount that have to be added every time we update the position of the particles. The speed is calculated by following the information given in (Ross, 2000), trying to simulate the way that real rain behaves.

Similarly, creating the particles strictly following the presented patterns would entail noticeable repeating images. To avoid these visual artifacts, we have included in the information of each particle a different random value. This value will be used to modify the final positions of the particles created using the patterns.

For each particle, the Geometry Shader of the second pass will be in charge of applying the level-of-detail. Thus, for each particle we calculate the appropriate size according to the distance. Then, we generate the 4 vertices of the quad using the information of the original size and the new size adjusted to the distance. We must also consider the position and orientation of the user. Finally, if the particle is close enough, we calculate the number of replicas we want to create. For each of these new quads, we should calculate the position of the new vertices using the original ones and the random value.

4.3 Far Rain Implementation

In the previous paragraphs of this section we have presented the different characteristics of our multiresolution model for rendering rain in real time. Nevertheless, these features are only useful for the *close rain* approach. The *far rain* needs further considerations. On one hand, the rain container has to be expanded in order to cover all the extension of the rain area that the user can see. This size is calculated in the CPU and its value is uploaded into the GPU, where all the particles will be displaced horizontally and vertically in order to cover the new rain container. On the other hand, the rain particles should be much bigger to cover the whole area of the expanded rain container.

5 RESULTS

In order to examine our presented rain framework, we have conducted several tests to analyze both the visual appearance and the obtained performance. These set of tests were done with a Pentium D 2.8 GHz. with 2 GB. RAM and an nVidia GeForce 8800 GT graphics card. The implementation of this solution is coded in HSLS and C++ by using the possibilities of the recent DirectX10.

One of the main objectives of our work is to increase the performance of the rain simulation. We present in Table 1 the frame rate obtained for the different rain intensities proposed above. In order to compare the performance of the different rain frameworks we render the amount of particles indicated, which are necessary to simulate a similar rain intensity in the different frameworks compared. It can be seen how our approach can render the rain scenes with a frame-rate increased in a 55%.

We have also considered interesting to test the difference in performance that can be obtained when creating the raindrops in the GPU. In Figure 4 we depict in blue the frame rate obtained when rendering different amounts of raindrops without creating multiple quads on the GPU. In the same Figure, we show in red the *fps* obtained when applying our new method. We translate 20,000 particles but in the GPU we replicate them by 2, 3 and so on in order to obtain as many raindrops as desired. We can see how the replication method outperforms drastically the previous method. In this study we have considered that the seed raindrop only replicates into 6 more raindrops, rendering at most 140,000 particles. The reason for selecting this number of replicas is because it fits more properly with the objectives of our multiresolution proposal.

6 CONCLUSIONS

In this paper we have introduced a new framework for optimizing the management and rendering of rain.

Table 1: Performance comparison for an intense rain scenario.

Model	Our Model	(Rousseau et al.,)	(Tariq, 2007)	(Puig-Centelles et al., 2008)
Number of Particles	**25,000**	*95,000*	*167,000*	*100,000*
Frame rate	**140**	*74*	*66*	*86*

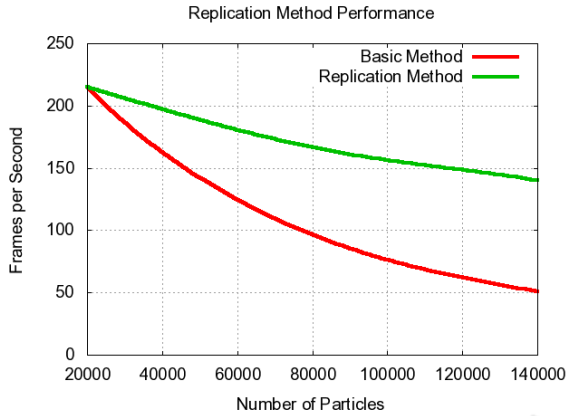

Figure 4: Comparison of the performance obtained with and without replicating particles on the GPU.

The solution presented here provides different approaches, depending on the relation between the user location and the rain area location. In addition, we have included multiresolution techniques which are applied directly and uniquely in the GPU.

Our presented solution is capable of offering similar rain intensity sensations with much less particles. This reduction in particles directly involves an increase of the obtained performance. Moreover, the use of level-of-detail techniques that are fully integrated into the *Geometry Shader* strongly decreases the temporal cost of the rain system.

For future work we are currently considering the interactions between the rain and the environment. In this sense, we consider interesting the extension of our model to include collisions, light interaction and material modifications.

ACKNOWLEDGEMENTS

This work was supported by the Spanish Ministry of Science and Technology (TSI-2004-02940, TIN2007-68066-C04-02) and Bancaja (P1 1B2007-56).

REFERENCES

Changbo, W., Wang, Z., Zhang, X., Huang, L., Yang, Z., and Peng, Q. (2008). Real-time modeling and rendering of raining scenes. *Visual Computer*, 24:605–616.

Fearing, P. (2000). Computer modelling of fallen snow. In *SIGGRAPH '00*, pages 37–46.

Feng, Z.-X., Tang, M., Dong, J., and Chou, S.-C. (2006). Real-time rain simulation. In *CSCWD 2005, LNCS 3865*, pages 626–635.

Garg, K. and Nayar, S. K. (2006). Photorealistic rendering of rain streaks. *ACM Transactions on Graphics*, 25(3):996–1002.

Kusamoto, K., Tadamura, K., and Tabuchi, Y. (2001). A method for rendering realistic rain-fall animation with motion of view. *IPSJ SIG Notes*, 1109(106):21–26.

O'Brien, D., Fisher, S., and Lin, M. (2001). Automatic simplification of particle system dynamics. *Computer Animation*, pages 210–257.

Puig-Centelles, A., Ripolles, O., and Chover, M. (2008). Multiresolution techniques for rain rendering in virtual environments. In *23 International Symposium on Computer and Information Sciences*.

Ross, O. N. (2000). Optical remote sensing of rainfall micro-structures. Technical report, Freien Universitt Berlin. Diplomarbeit thesis.

Rousseau, P., Jolivet, V., and Ghazanfarpour, D. Realistic real-time rain rendering. *Computers & Graphics*, 30(4):507–518.

Soatto, S., Doretto, G., and Wu, Y. N. (2001). Dynamic textures. In *International Conference on Computer Vision*, pages 439–446.

Tariq, S. (2007). Rain. nvidia whitepaper. http://developer.nvidia.com.

Tatarchuk, N. (2006). Artist-directable real-time rain rendering in city environments. In *ACM SIGGRAPH Courses*, pages 23–64.

Wang, H., Mucha, P. J., and Turk, G. (2005). Water drops on surfaces. *ACM Trans. Graph.*, 24(3):921–929.

Wang, L., Lin, Z., Fang, T., Yang, X., Yu, X., and Kang, S. (2006). Real-time rendering of realistic rain. In *ACM SIGGRAPH Sketches*, page 156.

Wang, N. and Wade, B. (2004). Rendering falling rain and snow. In *ACM SIGGRAPH 2004 Sketches*, page 14.