ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 2, Issue 2, pp: (30-38), Month: April 2014 – June 2014, Available at: www.researchpublish.com

Rain: Graphical Rendering

Mr. Anurag Rana*

Abstract: **In recent times, increase in the computational speed of graphics hardware has been a many fold. Power afforded by modern graphics cards enables the possibility of simulating complex environmental phenomena, such as atmospheric special effects. The main objective of the paper is to create realistic rain falling effect in real time taking into account the refraction and reflection of raindrop by modeling the world as a cube map, and using environment mapping technique.**

We observe that the reflection property of a raindrop is also important since it contribute to its visibility in the dark regions. Hence, in this paper we propose a new method that takes into account both refraction and reflection property of raindrop.

Keywords: **Realistic rain falling effect, environment mapping technique.**

I. Introduction

Atmospheric effects such as rain, cloud, snow and other outdoor scenes in interactive applications (video games, training systems...) are important in creating realistic environments. As current graphics hardware's computation speed is improving, their high degree of realism is also required to immerse the user in a visually convincing environment. However, rendering these effects is a hard problem, especially in real time. Rain is a complex atmospheric physical phenomenon and consists of numerous visual cues as shown in Figure 1.1. There are different forms of rainfall like light, moderate, heavy, and extreme rainfall.

Figure 1.1: Photograph of effects in rainy environment.

In general, Particle Systems are used to generate rain in many video games by using particles falling vertically from the sky. Each particle is a translucent white streak. This may be a segment geometrically composed of two points, or approximated by a texture plated on a rectangle stretched vertically (commonly called billboard). The second technique has an advantage of allowing different drop sizes depending on the depth relative to observer, unlike the use of segments which have a fixed width whatever their position in the scene.

Langer et al. [1] presented interactive an image-based spectral synthesis method to render snow and rain. The proposed method that is a combination of a Particle System and a spectral analysis technique, which creates a dynamic rainy or snowy environment texture. The spectrum of the falling snow or rain texture is defined by a dispersion relation in the image plane.
Wang and Wade [2] presented a technique that maps textures onto a double cone. The orientation of cones is determined by the position and speed of camera movement. Several textures of rain and snow are simultaneously scrolled on the double cone with different speed to give a motion impression. This technique is faster than Particle System, but does not allow any kind of interaction between articles and their environment, whether physical interaction (bounces and other forces applied to particles) or optical (refraction and reflection of the scene visible in the drops). Besides, the textures must be drawn by artists for all the conditions shown in the scene taking into account the density of precipitation and level of brightness of the scene in which the textures will be used. ATI and Nvidia have integrated programs including a simulation of rain in their software development kits to illustrate the possibilities of their hardware.

ATI was the first to publish a demo, called Toyshop Demo, of a rainy night in a city. This demo contains various effects detailed in [3], [4], and [5]. Among these effects, the rain simulation interests us the most. The technique is based on a post-processing image-space technique, in which the rain is added once the rest of the scene was rendered. Several layers of raindrops are simulated with different speeds at varied depths in a single rendering layer in order to get a feeling of raindrop motion parallax (a strong visual cue in any dynamic environment). The fact that the animation takes place at night, and therefore with a low intensity enables not to bother with simulating refraction and reflection in the drops. Hence, the used textures which contain a constant stain and intensity are modulated by the intensity of a global luminous scene. This enables a realistic interaction of rain with the lightning which shortly lightens the scene from times to times. The demo also contains a simulation of raindrops falling off objects, for example raindrops pouring from a gutter pipe or falling off the rooftop ledge. Drops are simulated as independent particles and animated using graphics hardware of the method described in [6]. Finally, this demo takes into account the splashes coming from drops collision with a wet surface. The positions of the collisions are random and uncorrelated with the positions of the parallax planes that contain the falling drops. The rendering of the splashes is done by pre-rendering a high-quality splash sequence for a milk drop. This sequence is then deformed to avoid obvious similarities between apparent splashes, and afterward played step by step at each splash position.

Nvidia also proposed a demo of rain with its software of development kit for DirectX 10 to illustrate the possibilities of its models Geforce 8. The demo is detailed in [7] and uses database of images created by [8]. Some parameters have been eliminated and the database was restricted to 300 textures. Some of the textures, the angle of viewpoint and the light direction are selected and interpolated to render raindrop particles. This demo uses a night scene and only the interaction with light is simulated. It does not take into account the refraction and reflection of the raindrops appearance.

II. Problem Statement

Objective of the paper is to create realistic rain falling effect taking into account the refraction and reflection of raindrop by modeling the world as a cube map, and using environment matting techniques. Unfortunately, this takes a lot of computation to render an image. Hence, we also focus on optimization running time of this technique in order make this effect run in real-time (which means the number of image or frame generated per second (frame-rate) is greater than 20).

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 2, Issue 2, pp: (30-38), Month: April 2014 – June 2014, Available at: www.researchpublish.com

III. New Technique

New technique is primarily based on the aforementioned three functions, Pattern Generation, Map generation, and Relighting and Compositing. It has two steps: pre-processing step and processing step.

In pre-processing step, we make an imaginary"huge" cube and set the camera (the viewpoint) in the middle of it. The huge cube is then divided into L layers and each layer are divided into R rows and C columns, so we can imagine that we have L*R*C small cubes inside that huge cube. Figure 1.2 presents division of huge cube in L layers, R rows, and C columns in order to get L*R*C small cubes. Afterwards, we use Pattern Generation and Map Generation to generate maps of a raindrop put in the middle of the small cube one by one. Eventually, we get M*R*C maps which are used in processing step. All theses maps are indexed from one to M*R*C according to the position of small cubes so that we can easily find them.

In processing step, we capture six images of the scene and mapped these images on six faces of the huge cube. Then we check position of each raindrop and used the correspond map to render those raindrops. For example if a raindrop positions in a small cube numbered three than the map numbered three is used by Relighting and Compositing function to render that raindrop. Capturing six images of the scene: we use a second camera which is positioned at the same location as the primary camera (observer) and directed parallel to the ground. Field Of View (FOV) of the camera is set to 90o. Front, left, right, top, down, and back image are generated by capturing the scene after the camera is rotated 0o, 90o to left, 90o to right, 90o to up, 90o to down, and 180o to up respectively.

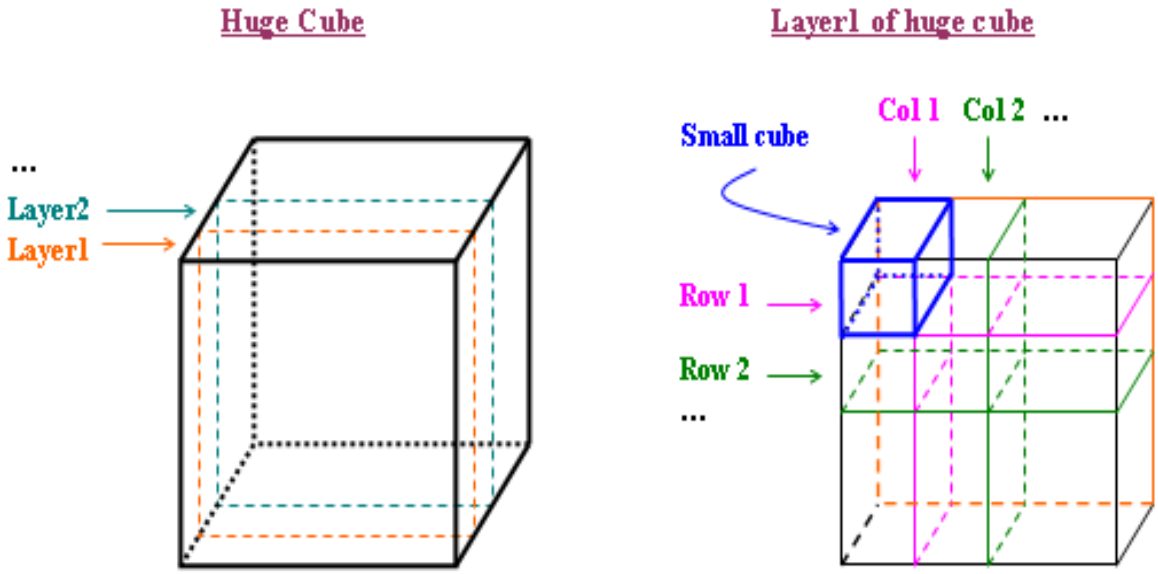

Figure 1.2: Division of huge cube in L layers, R rows, and C columns in order to get L*R*C small cubes

Movement of the camera: wherever the camera moves (forward, backward, leftward, rightward, upward, downward), the cube is also moved along with it since we want that camera to be always located in the middle of the cube. This is because all the maps are generated when the camera is only positioned in the middle of the huge cube in pre-processing step. Besides, when the camera is changed from a position A to a new position B, six images of the scene have to be recaptured so that raindrops

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 2, Issue 2, pp: (30-38), Month: April 2014 – June 2014, Available at: www.researchpublish.com

are looked reasonable in the scene by rendering with these new images. But, when it is rotated, the scene needs not to be recaptured.

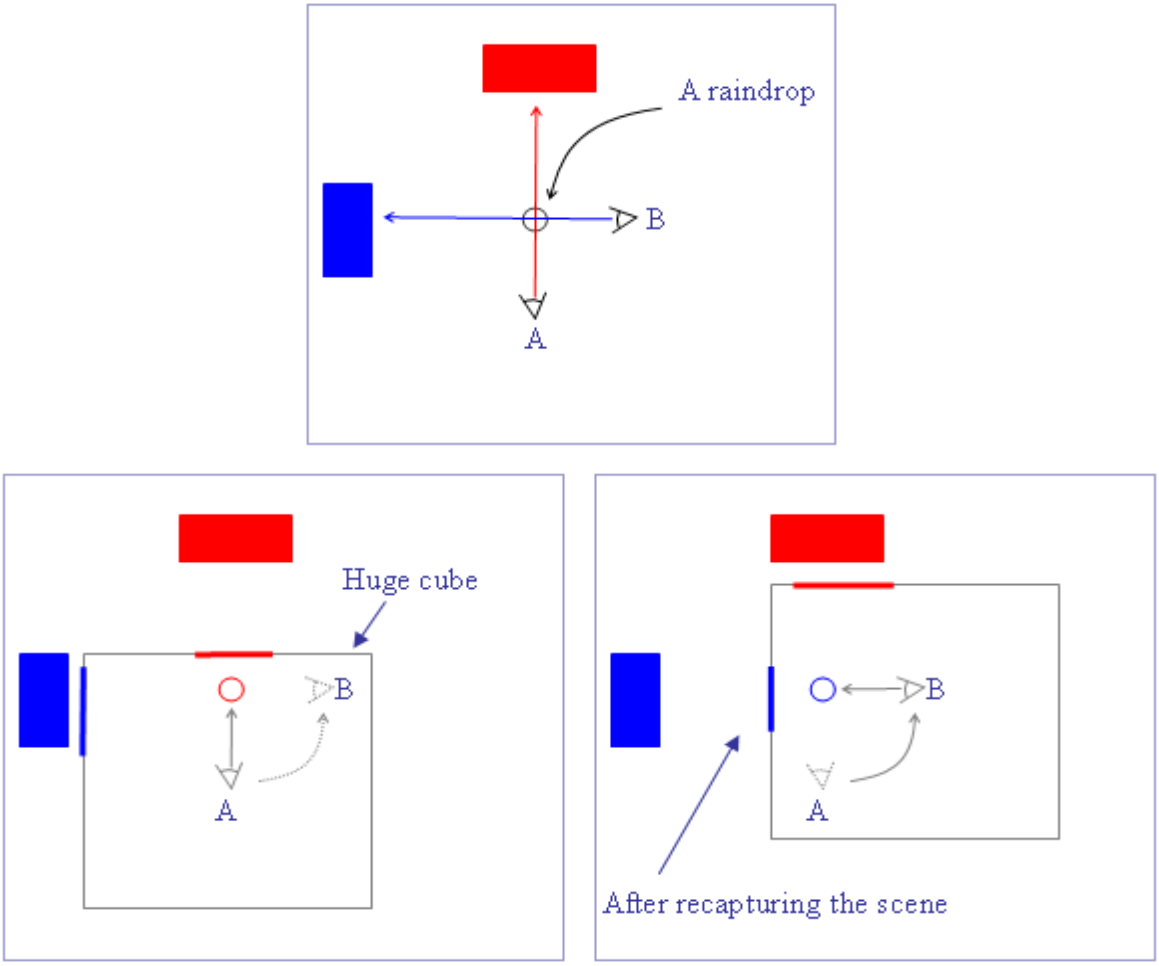

Figure 1.3: Top image shows what a raindrop look like when it is viewed from position A or B. Bottom images show the movement of the camera from position A to a new position B by new technique

The top image of figure 1.3 presents the scene which is viewed from the top to bottom. The scene contains a blue object, a red object, and a raindrop. In reality, if the raindrop is seen from the position A, it appears red; if that raindrop is seen from the position B, it appears blue. In the bottom images of figure 1.3, our technique shows the movement of camera from position A to B. In bottom left image, the camera is positioned at A and the raindrop looks red since it is primarily contributed by the red color which is on the huge cube. In bottom right image, when the camera is moved to B, the huge cube is also moved along with it and the scene is recaptured so that the raindrop looks blue.

Making rain falling effect as mentioned above is a physically-based method, thus all the raindrops look like spherical or ellipsoidal. Even though these raindrops are physically correct, they seem to lack realism since human eye actually see raindrops like streaks. This is because of human's retina which takes about 70 ms to form an image and raindrop moves very quickly. During that time, the raindrop keeps falling, and all its different positions compose a continuous sequence of images on the retina.

Since the streak is the result of one moving drop, we used an image of our raindrop model to make a new vertical streak image as follows (Fig 1.4):

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 2, Issue 2, pp: (30-38), Month: April 2014 – June 2014, Available at: www.researchpublish.com

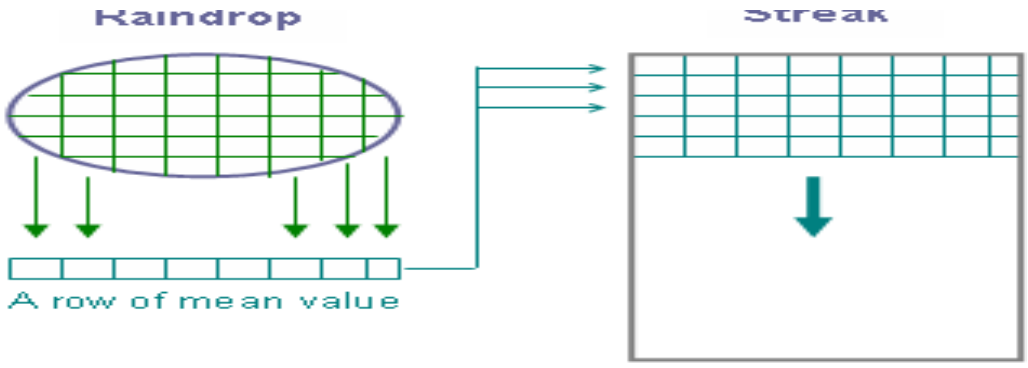

Figure 1.4: Making streak raindrop

1. Compute a mean of each pixel column of a raindrop to get a row of mean values
2. Use this row to make a streak
3. Lower alpha value in order to blur the streak.

IV. Algorithm

Algorithm is like a rendering technique which takes a particle as input and outputs a beautiful raindrop image matching the environment.

The complexity of this algorithm depends on number of particles visible to the camera, and computation time of relighting function (which in turn depends on resolution of input image). Assume that input image resolution is MxN and P particles visible to camera. P is the sum of P1 and P2, where P1 is number of particles within S small cube and P2 is number of particles in the same small cubes as P1. So the complexity can be written as below:

$$O((MxN)^{P1} + P2)$$

ALGORITHM:

1. Pre-processing step:

Step 1: Generate patterns.
Step 2: Map these patterns on the huge cube.
Step 3: for each small cube in the huge cube do.
Step 4: place a raindrop in the middle of the small cube.
Step 5: camera direction is looked at the raindrop.
Step 6: camera field of view is decreased to see only the raindrop.
Step 7: render an input image.
Step 8: end for.
Step 9: Generate L*R*C maps using these input images.

2. Processing step:

Step 1: Load maps generated in pre-processing step.
Step 2: Capture 6 images of surrounding environment.
Step 3: for each particle that is visible to camera do.
Step 4: Get the particle position.
Step 5: Check the particle if it is inside which small cube using its position.

Step 6: if the small cube is not already used then.
Step 7: Compute raindrop texture (raindrop color) using captured images and a map correspond to the small cube.
Step 8: if Streak raindrop is enable then.
Step 9: Compute streak raindrop texture using raindrop texture above.
Step 10: end if.
Step 11: Store this texture in memory.
Step 12: Mark that small cube as used.
Step 13: else
Step 14: Load raindrop texture (raindrop color) from memory.
Step 15: end if
Step 16: Map this texture to the particle.
Step 17: end for.

V. Implementation

Matlab codes of three primary functions Pattern Generation, Map Generation and Relighting and Compositing, these three functions are needed for our implementation in C++, and we converted them to C++ codes. Interestingly, the new result of computation time is pretty fast. Both output result (output image) of Matlab and C++ functions are the same. Detailed result of our experiment is shown below:
The following are our assumption which is used by original Matlab, new Matlab, and c++ codes.

The number of reflection + refraction is 4 because we did the same experiment by varying that number from 1 to 10 and the result images of setting number from 4 to 10 are look exactly the same.
• Resolution of each cube face, input images, and output image: 512x512
• Number of colors for making unique color codes: 3
• Number of refraction + reflection: 4

The following is the result of original Matlab code:
• Map computation: 70 seconds (average)
• Relighting and Compositing: 30 seconds (average).

We modified the original Matlab codes using two steps:

1. By changing the map format and partly modifying codes in Map Generation and Relighting and Compositing function in order to use the new map, we got the following result:
• Map computation: 80 seconds (average)
• Relighting and Compositing: 20 seconds (average)

2. By optimizing the first step, that is modifying main part of the codes in Relighting and Compositing function, we got the following result:
• Map computation: 75 seconds (average)
• Relighting and Compositing: 1 seconds (average)

Finally, we converted Malab codes into c++ codes and got the following result:
• Map computation: 2 seconds (average)
• Relighting and Compositing: 0.04 seconds (average).

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 2, Issue 2, pp: (30-38), Month: April 2014 – June 2014, Available at: www.researchpublish.com

POVRay is used to make input images since its photorealistic rendering tool. Raindrop 3D model is also generated in POVRay using parametric function.

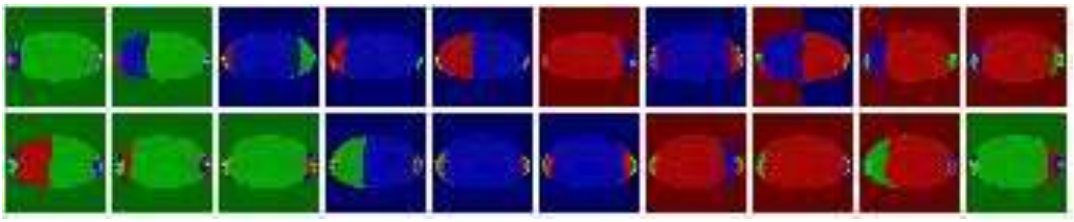

Figure 1.5: Input images of a raindrop undistorted radius 4.5mm

Pattern Generation function is used to generate patterns that are mapped on the huge cube face. For each small cube inside the huge cube, a raindrop model is positioned in the middle of that small cube, camera direction is looked at the raindrop, camera field of view is decreased to see only the raindrop, and then an input image is rendered. Figure 1.5 show some input images of a raindrop, undistorted radius 4.5mm.

For our implementation, we choose 10 meters, and 0.5 meter as the side of a huge cube and all small cubes respectively. All small cube sizes are the same since it's easy to manage their position. We can't choose the size of small cube side less than this due to memory problem.

According to the chosen value, we can know that there are 8000 small cubes within the huge cube so we need 8000 maps. This requires around 192 MB of memory to store them since one map require around 24 KB. If 0.25 meter is chosen as side of each small cube, it requires 64000 maps, which need 1536 MB of memory. Each small cube seems big to the raindrop and more than 100 drops can be positioned in the small cube. This might cause some raindrops that close to the camera look different from what they should be seen in the scene. To avoid this problem, all generated raindrops are positioned at least 2 meters away from camera (observer) because raindrops inside a small cube, which is 2 meters from camera, look quite similar to the one in the middle of the small cube. In addition, raindrops are very small and move very fast thus this drawback can be neglected.

VI. Result

The result images are rendered in Delta3D on a PC with Dual-Core Intel CPU 1.86GHz processor with 2GB RAM. Table 4.1 shows frame-rate of generating drops and streak drops in a scene that contains 1000, 10000, 20000, 30000, 40000 and 50000 drops. Our rain falling effect can be run in real-time in a scene with 10000 drops or streak drops. Figure 1.6 shows refraction and reflection of raindrops falling in the field:- Top left: when the sky is light; Top right: when the sky is dark and the ground is light; Bottom left: when the sky is dark, the ground is bright and camera is rotated 45o to the sky; Bottom right: when the sky is dark, the ground is bright and camera is rotated 90o to the sky. In top left image, we can clearly see different between rain drops on the top and bottom corner because of the refraction property of raindrop. Since raindrops have reflection property, we can also see the light brown color, which is contributed by brown color of the ground, on the border of raindrops. In the same situation, raindrops rendered by Physical Properties-Based Method will not be visible since it doesn't take into account the reflection property of raindrop.

Table 1: frame-rates of different number of drops in the scene

Number of Drop in the scene:						
	1000	**10000**	**20000**	**30000**	**40000**	**50000**
Drops	100	45	30	21	16	9
Streak drop	329	167	93	39	21.9	6.7

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 2, Issue 2, pp: (30-38), Month: April 2014 – June 2014, Available at: www.researchpublish.com

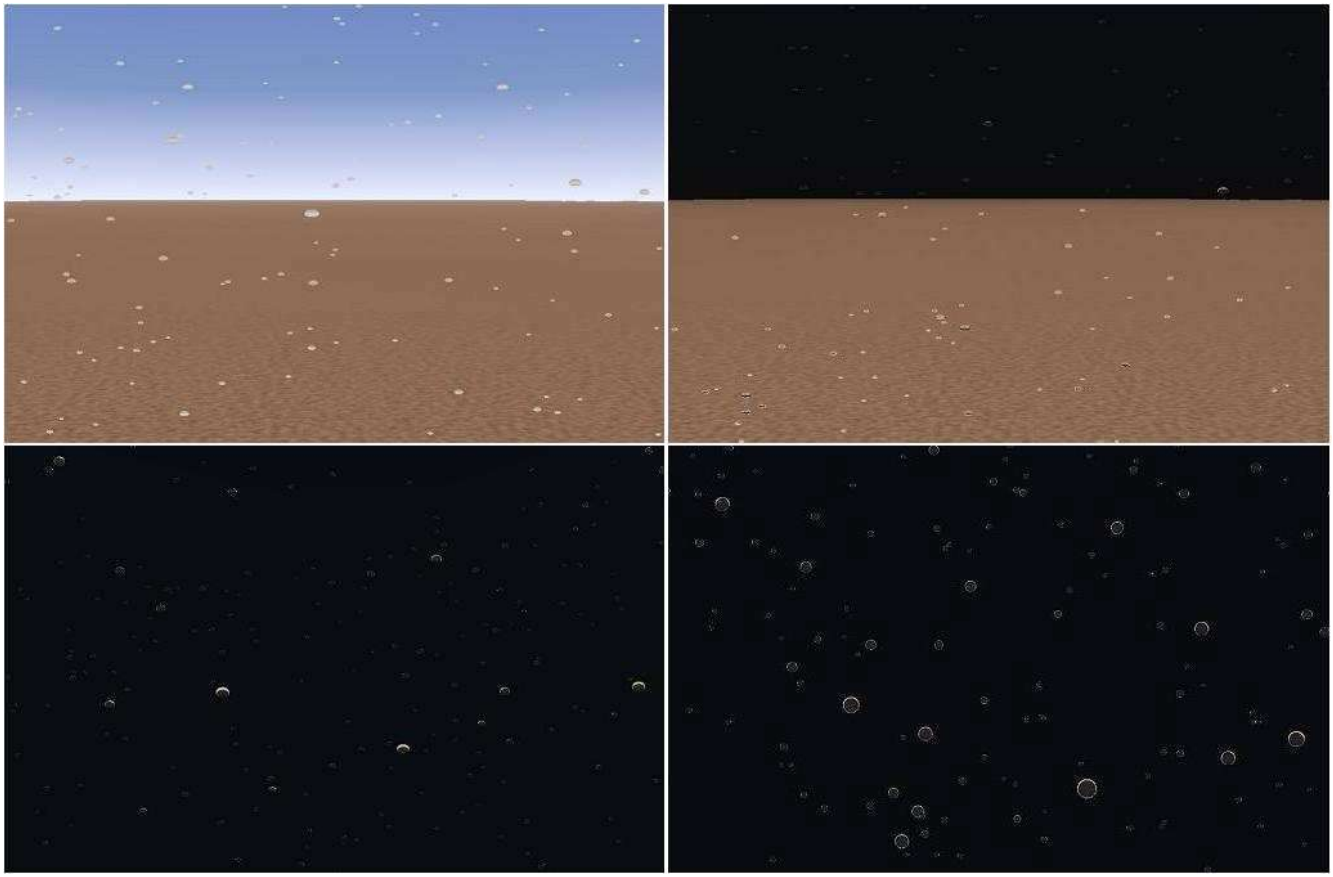

Figure 1.6: Refraction and reflection of raindrops in a field. Top left: the sky is light. Top right: the sky is dark and the ground is light. Bottom left: the sky is dark, the ground is light, and the camera is rotated 45o to the sky. Bottom right: the sky is dark, the ground is light, and the camera is rotated 90^{o} to the sky.

VII. Conclusion and Future Work

Created rain falling effect using Delta3D Particle System Editor that allows the user to have the impression of a rainy environment, but it's not very realistic. We proposed a new method to generate visually convincing results of rain falling using Cube Map and Environment Mapping, which is a rendering technique used to create photorealistic images of an object by mapping the environment on the object. Results of this method are more realistic than the other two methods using: Delta3D Particle System Editor, which renders rain falling using two static textures of raindrops whose colors are independent to the environment, and Physical Properties-Based Method, which renders rain falling by mapping the background scene onto raindrops according their refraction property. We highly optimized the original Matlab codes of environment mapping by modifying map format and main part of the codes in Relighting and Compositing function. We also converted the optimized Matlab codes into c++ whose computation time is pretty fast. We proposed a new method of making streak raindrops based on our raindrop model. Finally, we implemented our new methods in Delta3D modifying its particle system to render reasonable raindrop textures instead of particles.

In future work, our method can be implemented in Graphics Processing Unit, which is very efficient at manipulating and displaying computer graphics, in order to render more than 10000 raindrops in real-time (Table 1). It can also be improved to handle collisions of the raindrops with the ground or objects in the scene.

References

[1] M. S. Langer, L. Zhang, A. W. Klein, A. Bhatia, J. Pereira, and D. Rekhi. A spectral-particle hybrid method for rendering falling snow. In Rendering Techniques 2004 (Eurographics Symposium on Rendering). ACM Press, june 2004.

[2] N.Wang and B.Wade. Rendering falling rain and snow. In ACM SIGGRAPH 2004 Technical Sketches Program, 2004.

[3] N. Tatarchuk. Artist-directable real-time rain rendering in city environments. In Proceedings of Game Developers Conference, 2006.

[4] N. Tatarchuk. Artist-directable real-time rain rendering in city environments. In Proceedings of the 2006 Symposium on Interactive 3D graphics and games Poster, page 30, New York, NY, USA, 2006. ACM Press.

[5] N. Tatarchuk and J. Isidoro. Artist-directable real-time rain rendering in city environments.In Eurographics Workshop on Natural Phenomena, New York, NY, USA, 2006. ACM Press.

[6] P. Kipfer, M. Segal, and R. Westermann. Uberflow : A GPU-based particle engine. In Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics hardware, pages 115-122, 2004.

[7] S. Tariq. Rain. Technical report, Nvidia, 2007.

[8] K. Garg and S.K. Nayar. Photorealistic Rendering of Rain Streaks. ACM Trans. on Graphics (also Proc. of ACM SIGGRAPH), July 2006.

[9] Biswarup Choudhury, Deepali Singla, Sharat Chandran. Fast Color-Space decomposition based Environment Matting. In Proceedings of the 2008 symposium on Interactive 3D graphics and games, pages 1-1. ACM, 2008.

[10] Delta3D. www.delta3d.org, last viewed on June, 05 2008.

[11] Open Scene Graph. http://www.openscenegraph.org/, last viewed on June, 05 2008.

[12] Open Dynamics Engine. http://www.ode.org/, last viewed on June, 05 2008.

[13] 3D Character Animation Library. https://gna.org/projects/cal3d/, last viewed on June, 05/2008.

[14] GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html, last viewed on June, 05 2008.

[15] Falling Rain Particle Effect. http://www.delta3d.org/filemgmt/visit.php?lid=80, last viewed on June, 05 2008.

*** Author** is currently employed as Educator at University institute of information Technology (UIIT) in Himachal Pardesh University (HPU) Shimla-5. He acquired M. Tech. Computer Science and Engineering from Arni School of Technology and Master of Computer Application from Arni School of Computer Science in Arni University (HP). His special interests include Artificial Intelligence, Artificial Neural Network, and Distributed System/Network.