Real-Time Rendering of Realistic Rain

Lifeng Wang[1] Zhouchen Lin[1] Tian Fang[2] Xu Yang[3] Xuan Yu[4] Sing Bing Kang[5]
[1] Microsoft Research Asia, {lfwang|zhoulin}@microsoft.com [2] South China University of Technology, P.R. China
[3] Nankai University, P.R. China [4] Shanghai Jiao Tong University, P.R. China [5] Microsoft Research, sbkang@microsoft.com

Abstract

We propose a new GPU based technique to render realistic rain in real-time. It consists of two parts: off-line image analysis of rain videos, and real-time particle-based synthesis of rain. Videos of real rain are analyzed to extract the rain mattes; random samples of these rain stroke mattes are then used for online synthesis. We incorporate pre-computed radiance transfer (PRT) in our particle system to take into account the scene radiance. The transfer function of a raindrop can be highly efficiently evaluated with a closed-form solution. Our approach achieves high realism with low computation cost and a small memory footprint. Our technique applies to a variety of scenarios, from synthetic 3D scenes to real videos.

1 Our Technique

A major challenge of rain simulation is the realism of the rain appearance, including the naturalness of the rain shape and the environmental lighting effect on the rain. The existing techniques either manually design limited number of the rain mattes (e.g. [ATI Demo 2005]) or simply assume straight rain strokes (as in traditional particle systems). Moreover, the rain color and intensity is either prescribed (e.g. [Starik and Werman 2002]) or is computed using the light sources only (e.g. [ATI Demo 2005]). Our technique aims at enhancing the realism of the rain *strokes* by applying the rain mattes extracted from rain videos and incorporating the effect of the *whole* environment map on the rain strokes using a particle system that utilizes the transfer function of the raindrops.

We formulate the rain extraction as an optimization problem:

$$(\mathbf{S},\alpha) = \arg\min_{\mathbf{S},\alpha} \sum_{i,j,k} \left(I^k_{i,j} - (1-\alpha^k_i)S_{i,j} - \alpha^k_i C_j\right)^2 + \lambda \sum_{i,j} ||\nabla S_{i,j}||^2 + \mu \sum_{i,k} \left(\nabla \alpha^k_i \cdot \mathbf{D}_0\right)^2,$$

where indices i, j, and k are used for referencing pixels, color channels, and frames, respectively. I is the observed color, α is the fractional blending weight due to rain, S is the color of the static background, (C_R, C_G, C_B) is the "color" of the rain (due to the environment), $\mathbf{D}_0$ is the principal rain direction, and λ and μ are weights. This is more effective than the naive median filter used in [Starik and Werman 2002]. Some image processing techniques such as morphological erosion and dilation and directional blurring along $\mathbf{D}_0$ may be applied to remove noise and enhance the shape of rain strokes. Then the best-looking individual rain strokes can be easily cut out as samples.

In our particle system that computes the rain color and intensity in real-time, each particle has a sphere model with the refractive index of water, a random rain stroke matte, and other physical attributes such as position and velocity. The particles are assumed to be uniformly distributed in space. The length of the matte shape is estimated based on the particle's current velocity and the exposure time of the virtual camera. The matte shape is achieved by shrinking or stretching its original shape accordingly through interpolation. The diameter of the sphere that models the raindrop corresponds to the width of the rain matte after being mapped to the scene coordinate.

Our rendering technique is purely GPU based. After the 3D position and orientation of each particle have been computed by a vertex shader, the next step is to color the particle. The GPU first determines which part of the screen is to be shaded by mapping the rain matte of the particle. A per-pixel PRT shader then computes the intensity distribution in the rain matte of the particle, where the location-dependent environment maps have been stitched from the input video with the existing computer vision techniques and the transfer vectors of raindrops can be highly efficiently evaluated from a closed-form solution, which is derived using geometric optics thanks to the roughly spherical shape of raindrops.

Figure 1: Adding synthetic rain to a real video. Left: Part of an original frame of resolution 720×480. Right: Result of adding rain. Notice how the synthetic rain was affected by the environment.

Finally, the GPU blends the appearance of the particle with the background using the computed intensity and alpha distributions of the rain mattes. If rough geometry of the scene is known, by rendering a virtual scene surface (defined using a mesh of a synthetic scene or the depth map of a real video) to the z buffer, the particle will be automatically clipped by the rendering engine at the scene surface. This gives a volumetric appearance of rain. The realism of rain can be further enhanced if scene decoloring and darkening, and rain fog, etc., are taken into account. Adding manually designed rain splatters to the ground is also very helpful to the realism.

2 Results

Figure 1 shows an example of adding rain to a real video. Please notice how the synthetic rain is affected by the environment. The memory footprint dedicated to the lighting effect of the rain, namely memory costs for the rain stroke samples and the raindrop transfer vectors, is only 6MB (without compression).

We used a PC with Intel Pentium® 4 3.2GHz CPU, 1GB RAM, and an nVIDIA® GeForce 7800 GT graphics card. Even with **80,000** rain strokes for each frame (all the rain strokes are visible to the virtual camera), the frame rate we obtained was 77 fps.

References

ATI DEMO, 2005. ATI Radeon series X1000, Demo: ToyShop. http://www.noticias3d.com/articulo.asp?idarticulo=527&pag=10.

STARIK, S., AND WERMAN, M. 2002. Simulation of rain in video. In *Proc. 3rd Int'l Workshop on Texture Analysis and Synthesis*, 95–100.