High Speed Rendering Method for Complex Scenes

Matthew W. Johnson† 玉木徹† 金田和文† 水上嘉樹†† 多田村克己††
†(広島大学大学院工学研究科) ††(山口大学大学院理工学研究科)

Introduction:

Recently there has been an increasing pressure on researchers to develop ways to render even the most complex scenes both quickly and accurately for simulations and entertainment. One such example is for driving simulators which train their users to handle adverse weather conditions such as rain[1]. However rendering tens of thousands of raindrops accurately in a high speed application is cost prohibitive using current methods. We are developing a method which allows an application to render a large number of drops through interpolation during preprocessing thus freeing up the processor during run time to manage the application.

Problems of Traditional Ray Tracing:

Current ray tracers which can calculate both reflection and refraction accurately are cost prohibitive when it is necessary to render a large number of objects. This inherent aspect of ray tracing makes it difficult to use in real time applications. The main cause of this is because every time the camera position changes the entire screen must be recalculated. The reason is because ray tracing calculates pixel color for each pixel, even when environment maps are used. Quickly calculating ten thousand raindrops with rays reflecting and refracting quickly becomes cost prohibitive.

Proposed Method:

In our method we propose to store not pixel data but instead store Environment Map Intersection Locations (EMILs) saved as θ,φ. By ray tracing a few representative raindrops during preprocessing, we can quickly interpolate between intermediate drops which allows us to avoid calculating ray intersections and refractions for each movement of the camera.

Each raindrop is given its own 'screen' and in that screen we store EMILs, rather than pixel color data. We begin by ray tracing a set grid of raindrops which have predefined positions relative to the viewpoint.(see figure 1) Using those ray-traced drops we can quickly interpolate numerous intermediate drops. The use of Environment Map Intersection Locations and not pixel color data allows screen data to be interpolated with few calculations. Later when a simulation is run it will use the rain position data and find its closest neighbor which is stored in a raindrop database. The raindrop in the simulator will acquire the 'screen' data from the drop stored in the database and retrieve the pixel color data from the current environment map background.

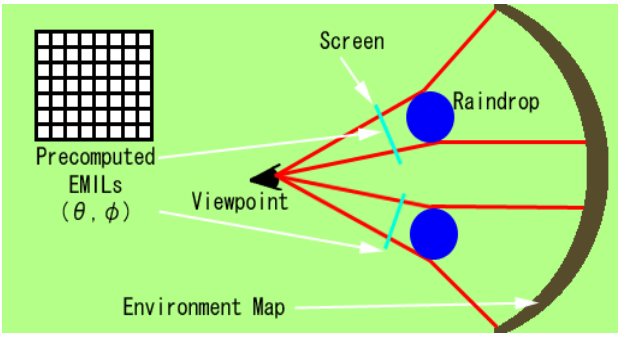

Fig.1: An above view of the layout for the proposed method.

Since the viewpoint is static in relation to the environment map, even though the environment map image changes rapidly as in the case of a video, it is not necessary to re-calculate any refracting rays and the raindrop image can be retrieved quickly.

Results:

Using a simple background we have ray traced two raindrops A and B. By interpolating based on relative distances we have interpolated raindrop C's values. Below you can see ray traced drops A and B and interpolated drop C. Raindrop D is the ray traced version of drop C. As you can see the values are quite similar.

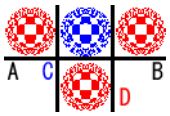

Fig.2: Ray Traced drops A and B located 'left' and 'right' of the camera. Interpolated drop C located dead center. Drop D - Ray traced version of drop C.

Conclusions Future Work:

Through our method rendering thousands, possibly tens of thousands of raindrops in a single scene both quickly and accurately will be possible. We avoid the expensive costs of calculating ray intersections and refractions and reduce it to an easy call to memory.

In the future we plan to implement an anti-aliasing step which makes use of a ray's width when projected to the environment map. Implementing a type of mip-mapping process will ensure both a high quality image and high speed rendering times.

References:

K. Kaneda, S. Ikeda, H. Yamashita, 'Animation of Water Droplets Moving Down a surface', The Journal of Visualization and Computer Animation, 10,15-26 (1999).