Atmos: A System for Building Volume Shaders

Ferdi Scheepers
Pixar Animation Studios

Alexis Angelidis
Pixar Animation Studios

To support effects artists in creating volumetric effects for the film WALL•E, we developed a system to build shading networks for volume shaders. We briefly present the system’s architecture and elaborate on implementation considerations, specifically to ensure fast execution and a small memory footprint. The system is modular, flexible, easy to use, and easily extended. Our system is implemented as a collection of Slim templates. Programmers with access to Slim or a similar shader generation tool can easily implement their own versions.

1 Volumetric Effects

The film WALL•E includes many volumetric effects such as clouds, exhaust trails, and explosions. To render these effects, we use a volume shader to determine how light is scattered or absorbed. We create volumetric effects either procedurally or through fluid and particle simulations. Procedurally-defined effects, such as the nebula above, are created entirely within the shader. Three-dimensional (3D) fractal noise functions are used to generate the pattern. For fluid and particle simulations, the volume shader acts as a visualizer for the simulation results, although we often embellish those results with procedural elements in the shader.

To help effects artists create volume shaders, we’ve developed a set of templates and cooperating RSL plugins for Pixar’s Slim shader generation tool. Collectively called Atmos, our system provides a straightforward way to build complex volume shaders fine-tuned for specific tasks. Users already familiar with building surface shaders quickly learn to build volume shaders. Atmos is easily extended by adding new templates.

2 Architecture Overview

At the heart of Atmos is a ray-marcher node. Given a ray, and input color and opacity, the ray-marcher integrates along the path of the ray, modifying the color and opacity due to volumetric scattering, self-luminosity, and attenuation. In our implementation, the ray-marcher provides a sample point \( P_r \) and a volume fragment \( dP_v \) to other nodes in the shading network, in effect polluting them to evaluate their results for the volume sample \( (P_r, dP_v) \). One of these nodes, the illumination node, calculates the light and opacity at a given volume sample and provides these values to the ray-marcher node. We’ve implemented illumination nodes to model scattering due to different substances (i.e., gas, dust, and vapor).

Input to illumination nodes are provided by density, color and luminosity nodes. For example, a density node may combine data input from a particle simulation with a 3D fractal noise function to define the substance inside the volume, as illustrated above. Similarly, a color node could define the color of the substance at each volume sample, and a luminosity node could be used if the effect being created is incandescent.

Manifold nodes define the evaluation domain. A special Atmos manifold node is available to receive the volume sample \( (P_r, dP_v) \) and pass it on to other nodes in the shading network. We’ve also implemented manifold transformation nodes, fluid and particle evaluation nodes, and a variety of shape nodes. Finally, we’ve enhanced some of our existing Slim nodes, which were implemented specifically with surface shading in mind, to work for volume shading. Specifically, some noise nodes needed to be made aware of volume samples so that high frequency noise can be filtered properly.

3 Implementation Considerations

A primary motivation for Atmos is to allow volumetric effects to be integrated into productions with the same ease as particle effects. Therefore, issues such as hold-out mattes, self-shadowing, render speed and memory management needs careful consideration.

A benefit of using volume shaders for volumetric effects is that RenderMan takes care of the segmentation needed to create hold-out mattes. For self-shadowing, we use two-pass rendering. The first pass renders from the light’s viewpoint, accumulating opacity into a volume texture. The second pass reads the accumulated opacity and attenuates incoming light accordingly. Access to the volume texture is filtered by the size of volumes samples to remove high frequencies or to intentionally blur shadows.

Rendering speed is proportional to the number of volume samples. If volume sampling is done for all pixel samples in the rendered image, an excessive number of ray-marching calls would result. To improve performance, we overlay a lower-resolution volume grid to store illumination results. For each pixel sample, the grid, which is implemented in a hierarchical data structure, is queried along segments provided by RenderMan. If data are unavailable for a segment, the ray-marcher computes and stores the illumination in the grid for later references. For high-resolution renderers the grid cannot fit comfortably into memory; therefore, we’ve implemented a swapping mechanism that saves, loads and deletes memory as needed during render time.

Atmos is a rewrite of a surface shader of the same name, originally developed by Erdem Taylor for underwater explosions in Finding Nemo. Thanks to David MacCarthy, John Pottebaum, Jake Merrell, David Batte, Chris Chapman, Jason Johnston, Bill Watral, Enrique Vila and other effects artists at Pixar for helping out with Atmos.