1 Introduction

For an upcoming short, “Wings of the Avalerion,” written by Joseph Guerrieri, an airplane set needed to be created that could be used in a number of configurations and from a number of angles. To easily accommodate the various shots, the decision was made to layout the airplane procedurally using Side Effects Software’s Houdini and to rely on Pixar’s RenderMan for flexible shading.

2 Procedural Modeling

The virtual set needed to consist of static elements that could be randomly varied, such as seat back angle, window shade opening and varied seatbelts. The individual components were modeled and UV’d in Autodesk’s Maya, then brought into Houdini to arrange, light, shade and render. The basis for the plane layout was a grid that determined the number of seats to be used on one side of the aisle, each corresponding to a specific vertex. The grid spacing was set using the bounding box of the seat cushion imported from Maya, and the grid was mirrored to create both sides of the plane. Then, using Houdini’s copy stamping, the previously modeled geometry was copied to each point of the grid, with a random variable based on point number creating the deviations.

For the plane walls, light consoles and overhead bins, the bounding box sizes were divided into the overall length of the base grid, to allow the procedural generation and placement of various sized elements. To generate the individual lights within the overhead consoles, instancing was again used, this time with lights rather than geometry. At the center of each light fixture, a single point was extracted containing position and normal orientation. A single spotlight was instantiated onto each point, allowing for a single lighting control to adjust all the overhead lights at once. Randomly deleting points from the array of points before instancing was then used to create a unique light set up, while hero lights could be extracted and controlled via separate lights.

3 Procedural Rendering

The procedural workflow that was used to create the model was extended to the shading and rendering, taking advantage of RenderMan Shading Language, as well as Houdini’s render outputs (ROPs). Each RSL surface shader was custom written, and included a standard set of arbitrary output variables (AOVs), including ambient, specular, diffuse, normals and others, to be output to separate layers of an OpenEXR file; these were to be used later when compositing the shot in post-production. Also, the point-based rendering workflow now available with Pixar’s RenderMan Pro Server was utilized to help with rendering computationally expensive passes, such as occlusion and color bleeding, without the necessity of raytracing.

In all of the plane shots, there was to be a static environment, as well as a number of moving objects, ranging from passengers to debris. Three ROPs were used to generate the final passes; in the first ROP, all static geometry was rendered through a static camera viewing the entire scene, with the results being output to a point cloud holding the dicing area of each micropolygon, surface color and lighting for a single frame. The second ROP then rendered only the animated geometry, baking the objects to point clouds for each frame of the animation. A post-frame render script was run at each frame which would combine the information from the static and animated point clouds. Each newly generated point-cloud was the run through a filtering process to generate brickmaps (octrees) [Christensen et al. 2004] holding the occlusion and color bleeding information. The third ROP was used to generate the final frames; using Houdini’s Take system, a different shader was assigned to all objects, which could read in the pre-computed information from the brickmaps previously generated.

4 Conclusion

While the initial setup of the scene took slightly longer than building a traditional set, the procedural flexibility in both modeling and shading was worth the expense. Individual frames were rendered in one quarter of the time due to the lack of raytracing, and precomputation of static geometry dramatically sped up the brickmap generation. Based on the success of this system in randomly generating an easily reusable set, the same techniques are being used on a jungle set for the same piece, with the additional step of rendering all of the L-System plant brickmaps as geometry, which hold all final color and sub-surface scattering data.

References