Statistics and Diagnostics

RenderMan 24.0 now includes an entirely new diagnostics sub-system, re-written from the ground up to incorporate live, configurable statistics.

> ⚠ Legacy Diagnostics will continue to be supported, unchanged, allowing customers to rely on the old system while transitioning to the new system.

Introduction

The RenderMan stats system has undergone an overhaul across the board, from the internal framework to the final user-facing results. The new system has been designed from day one for interactivity and extensibility. In other words, live stats with open-ended presentation options.

Overview

Much like the best VFX, the new stats system will mostly go unnoticed as you are running RenderMan 24.0. If you render to the Image Tool "it" you will see an improved HUD which includes overview stats for the in-progress render. Additionally, you may opt-in to live diagnostics in each of RenderMan's supported DCC plugins. Other presentation options are available, such as a JSON report or CSV telemetry output. These output options, or **Listeners,** are implemented as part of an extensible plugin system for diagnostic presentation. A Listener plugin attaches to a running render and registers interest in some or all of the metric data being collected by the renderer. As the Listener receives data from the renderer it is free to analyze and process the data as needed for its particular application.

Live Stats

RenderMan stats now provides access to the current state of render, whether it be a DCC preview render or a farm render. Out of the box, live stats will be displayed in the HUD of Pixar's Image Tool `it` with all bridge products. Optionally the system can be configured to stream live data via either gRPC or a WebSocket client.

Diagnostic Report

A new end-of-render report is also available as part of the default set of presentation plugins. Diagnostic data is written to a hierarchical JSON file, including data accumulated across checkpoints and canceled renders. The content of the output file is also configurable to include a few, most, or all of the metrics available in the system.

Checkpoints and Recovery

When the end-of-render report is enabled diagnostics are automatically written out on checkpoint exits and read back into the database on recovered renders.

Getting Started

The collection of stats is built into the renderer so there's no need to "enable" the stats system. On the other hand, Listeners need to be explicitly enabled and configured in order to introspect the data being collected. The default configuration allows for specific live metrics to be presented via the `it` image tool and the DCC Live Statistics windows. Upcoming releases will see an increase in the ability to configure these options, however, in the meantime, advanced configuration options are available through an INI-style configuration file.

Under the Hood

At the core of this new system is a centralized in-memory database of metric data. Stats are entered into the database by applications (renderer, bridge product, plugin, etc). Data is then extracted by the **Listener** presentation plugins. A Listener plugin is able to query specific data in order to present it according to that application's needs. One example of a Listener plugin would be a "report listener" which simply writes the data out to a file, mimicking the legacy (pre-24.0) functionality of stats. Another example of a Listener plugin would be an extension that extracts data for presentation in a HUD (DCC or "it"). The metric data held within the stats system is available for introspection by a Listener at any point during a render so an application has the option to continuously report live data and dynamically adjust the reporting rate.

Note that applications such as the renderer are no longer responsible for maintaining the layout or presentation of the data they are gathering. Their job is simply to report data updates. The stats system takes care of collation and synthesis and Listeners handle the data presentation.

Visual Debugging

PRMan also provides specialized Integrators for the debugging of Lighting and Shading:

- PxrDirectLighting
- PxrDefault
- PxrDebugShadingContext
- PxrValidateBxdf
- PxrVisualizer