TaskProcessor: A Pipeline Execution Framework and IDE

Oliver Staeubli Mark McGuire
Blue Sky Studios

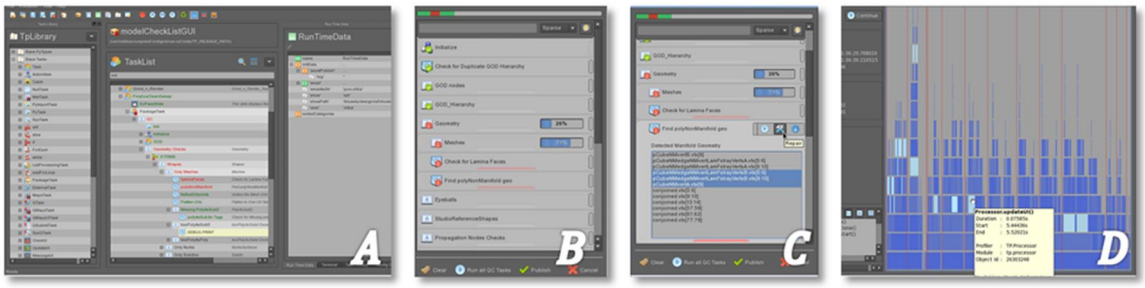

Figure 1: *A) main TaskProcessor window, B) QC Panel interface, C) repair interface, D) profiler*

Abstract

At Blue Sky Studios, we wanted a better way to manage and write cross package Python code. During production of *Horton Hears a Who*, the goal was to improve efficiency of rig construction in Maya by creating a custom Python engine that would execute code to distribute rigs. This project grew into the TaskProcessor, a tool that is now used across all production departments.

1 Overview

Over the past 6 years, TaskProcessor has helped modernize the Blue Sky Pipeline and standardize the development, deployment, and debugging of common pipeline processes. TaskProcessor consists of 1.) an integrated development environment (IDE) with visual drag & drop coding and flow control, 2.) a system for in-house Python library management integrated with SVN, 3.) an intuitive debugger and profiler that allows TDs and engineers to step through processing, 4.) a cross-package filtering system to distill large scenes and search for particular asset parameters, 5.) a multiprocess execution engine that provides realtime and configurable processing feedback to users.

2 Implementation and Challenges

TaskProcessor (TP) interfaces with Blue Sky's proprietary code distribution system to give developers access to a growing library of task types. Tasks can be programmatic constructs and iterators (e.g. if/elif/else, while, foreach), package specific templates (e.g. MayaTask, HoudiniTask, NukeTask), UI constructors (e.g. Slide) as well as generic system executions. Developers can subclass tasks to create custom workflow tasks (e.g. QMayaTask) all using standard Python OO syntax. TP provides developers access to two sets of data storage objects during execution: Configuration Data (CFG) and Run Time Data (RTD). CFG data is used to set defaults to the code and provide overrides via. UI input. It is immutable during execution. RTD data allows developers to pass calculated results throughout the execution process. Serializing these data objects while providing access to both local and farm machines for RTD modification presented challenges for locking and integrity. Due to Python's threading limitations, TaskProcessor implements multiprocessing through separate processes rather than threads and facilitates inter-process communication through network sockets.

This approach has the benefits of not being limited by the threading capabilities of potential host applications. The members of the processing tree can be distributed among several host machines. TaskProcessor manages the initialization and communication with its worker instances leaving developers free to concentrate on the high level workflow. Key to TaskProcessor's usability for TDs is the standardized debugging and logging. By controlling the execution of the Python code, TP allows developers to step through the execution graph. This execution can be recorded with the profiler to pinpoint inefficiencies in the code or workflows. TaskProcessor does not precompute the execution graph as it can be dynamically altered during the execution itself. A particularly novel feature of TaskProcessor is its ability to suspend execution to allow user input. A TP package can be executed until it reaches a certain point, save its state, and be continued at a later time. This is particularly useful for long running workflows that include human approval before continuing through the graph.

3 Related Work

Most third party tools including Maya, Houdini, and Nuke provide node-based execution engines that are highly customizable with a Python interpreter. Similarly, CROM from Rhythm & Hues and Fabric Engine are powerful platforms for development that optimize for GPU/CPU calculations. We took a different approach that focused purely on the development and deployment of pipeline code written in Python. TaskProcessor compliments third party tools with Python interpreters by allowing us to develop and debug packages of code that can be reused within all these applications. We have been able to develop QC tools that can execute across packages while using TaskProcessor's Python plackback and debugging tools. Because of its pure Python implementation, TaskProcessor doesn't compete in the space of GPU/CPU execution engines, but rather sits above those engines. TaskProcessor's own text editor can be used in place of or in addition to other dedicated IDEs such as Sublime.

SIGGRAPH 2015 Talks, August 09 – 13, 2015, Los Angeles, CA.
ACM 978-1-4503-3636-9/15/08.
http://dx.doi.org/10.1145/2775280.2792569