Introduction

The Nodal Scene Interface

A flexible, modern API for renderers

Authors
Olivier Paquet, Aghiles Kheffache, François Colbert, Berj Bannayan

January 22, 2018

The Nodal Scene Interface (NSI) was developed as the next generation rendering API. It aims to replace existing APIs in our renderer which are showing their age. Designed in the 80s, and extended several times since, they include features which are no longer relevant and design decisions which do not reflect modern needs. This makes some features more complex to implement than they should be or prevents implementing others. The design of the NSI is shaped with the following goals:

Simplicity

The interface itself should be simple to understand and use – even if complex things can be done with it. This simplicity is carried into everything which derives from the interface.

Interactive rendering and scene edits

Scene edit operations should not be a special case and be extremely efficient. There should be no difference between scene description and scene edi ts. In other words, a scene description is a series of edits and vice versa.

Tight integration with Open Shading Language

OSL integration is not superficial and affects scene definition. For example, there are no explicit light sources in NSI: light sources are created by connected shaders with an emission() closure to a geometry.

Scripting

The interface should be accessible from a scripting language – one that is platform independent, efficient and easily accessible. Scripts can be used to add render time intelligence to a scene description.

Performance and multi-threading

All API design decisions are made with performance in mind and this includes the possibility to run all API calls in a concurrent, multi-threaded environment. Nearly all software today which deals with large data sets needs to use multiple threads at some point. It is important for the interface to support this directly so it does not become a single thread communication bottleneck. This is why commands are self-contained and do not rely on every call.

Support for serialization

The interface calls should be serializable. This implies a mostly unidirectional dataflow from the client application to the renderer and allows greater implementation flexibility.

Extensibility

The interface should have as few built-in assumptions as possible about which features the renderer supports. It should also be abstract enough that new features can be added without looking out of place.