A Pipeline Retrospective on USD & Conduit

Rebecca Hallac
Blue Sky Studios
Connecticut, USA

Tim Hoff
Blue Sky Studios
Connecticut, USA

Chris Rydalch
Blue Sky Studios
Connecticut, USA

Oliver Staeubli
Blue Sky Studios
Connecticut, USA

Ryan Bland
Blue Sky Studios
Connecticut, USA

Karyn Buczek Monschein
Blue Sky Studios
Connecticut, USA

Mark McGuire
Blue Sky Studios
Connecticut, USA

ABSTRACT

Over the past three years, Blue Sky Studios built a USD-centric layer on top of its next generation pipeline framework, Conduit. This transition involved mapping the legacy Blue Sky workflows into USD constructs. In addition, direct artist feedback during the delivery of six short films provided insights that informed the evolution of the Conduit backend to support these modernized workflows.

CCS CONCEPTS

• **General and reference → Design**; • **Computing methodologies → Graphics systems and interfaces**; • **Software and its engineering → Collaboration in software development**.

KEYWORDS

Pipeline, animation, USD, workflow

ACM Reference Format:
Rebecca Hallac, Tim Hoff, Chris Rydalch, Oliver Staeubli, Ryan Bland, Karyn Buczek Monschein, and Mark McGuire. 2021. A Pipeline Retrospective on USD & Conduit. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '21 Talks), August 09-13, 2021.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3450623.3464633

1 INTRODUCTION

The development of Conduit was guided by a set of key objectives gravitating around a unified experience, flexible workflows, and robust tracking of data dependencies. We approached these requirements using *Products* (repositories managed by Conduit) with *PRIs* (Pipeline Resource Identifier) as the building blocks of the pipeline. [Staeubli et al. 2019] A product is a unit of production data whose behavior is defined and customized by an *Archetype*. Below, we summarize our learnings from designing these products in the areas of asset & shot structure, propagating breaking changes, and exposing complexity.

SIGGRAPH '21 Talks, August 09-13, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8373-8/21/08.
https://doi.org/10.1145/3450623.3464633

2 USD PRODUCT STRUCTURE

In our pipeline, there are two sides to a USD product: its behaviour and its USD content. Every shot and asset is an *Entity* product, which is divided into *Element* products to allow for parallel discipline-specific contributions. Each element's USD layer is a *sublayer* in its entity's USD layer, where the strength order is defined in its archetype. Artists contribute to elements in parallel using the entity-element structure, however, given the simultaneous contributing, communication between artists is critical. In addition to adhering to this structure, our assets have a few additional goals: leveraging USD features, efficient instancing and variants built into assets.

2.1 Kinds, Payloads and Classes

In USD we organize our assets with *model kind*. Assets are generally split into two categories: *components* and *assemblies* where the kind informs artist workflows. We extended the available kinds from the base component or assembly.

Component-based assets contain heavy geometry. As a result, components all contain a *payload*, to defer heavy parts when opening USD stages. Assemblies, which are aggregate assets that reference other assemblies or components together, do not need an additional payload. In unique cases where an assembly needs to introduce new geometry, those prims are stored behind their own payload.

All assets *inherit* from a *class* primitive, allowing edits to be broadcast to all instances in a shot or scene. We structure our class primitive hierarchically, using tokens from the asset PRI, to help artists manage complicated primitive tree interfaces.

2.2 Instancing

For scalability, we rely on native instancing in USD in addition to PointInstancers. Our USD structure contains a payload prim beneath the asset root which was instanceable by default. This enabled VariantSets on the model root to share the same payload instance. Artists and tooling thus know that variants defined *below* the payload prim are expected to be tied to model changes, while variants defined *above* the payload prim are not tied to model variants. For example, material variants which generally only define light weight primvars, could be adjusted independently.

2.3 Variants

Model variants within an asset was a new concept for production. The initial implementation of variants included separate products for each variant. However, these extra products restrained artists' workflows and we transitioned to defining variants solely in USD. We provide three VariantSets, referred to as *core variants*, as the default way assets are setup and contributed to:

- Model - Geometry variations
- Resolution - For level of detail
- Representation - Generally for character rigs

Our variant convention allows for artists to add additional VariantSets as needed as well as ignore the core variants. Each core variant has a *layer* and *payload* USD file, which correspond to the core variant contributions that are above or below the payload.

3 PROPAGATING BREAKING CHANGES

The initial implementation of our pipeline was optimized for a "fail forward" push workflow with the intention of introducing breakpoints, in form of collectors and flattening of USD data, as the pipeline matured. We quickly discovered that due to the sparse nature of USD overrides, the need for these breakpoints became critical. This problem appeared in many areas of the pipeline but most prominently in topology changes.

3.1 Topology Changes

A caching step happened at the shot level which described asset edits and animation as sparse overrides. This led to mismatched data when assets had a breaking topology change. To correct this discrepancy, all instances of the asset in shots would have to be re-cached. While we were moving towards more automated solutions this process was manual for artists. As a result, absorbing breaking topology changes was very time consuming and costly.

3.2 Collectors

Part of the solution to breaking changes was to introduce *Collectors*. Collectors would lock down versions and dependencies and potentially flatten USD data down to a single layer. This would insulate artists from breaking changes downstream until they were ready to absorb these changes. We believed the cost to this solution would be losing the flexibility of swapping versions of individual elements in upstream entities. However, from experience, this flexibility is not worth the increased complexity in the artist's workflow; collectors would have been a worthwhile solution to pursue.

4 EXPOSING COMPLEXITY

With a complex underlying system, we were challenged to expose a helpful level of control and complexity when designing artist-facing workflows and tools. This is compounded by the need for every department to absorb a wide range of new concepts. While this balance game manifested in many areas we will focus on three occurrences that are relevant to all disciplines.

4.1 Working with Workspaces

When initially designing workspaces we anticipated that artists would want the flexibility to define generic areas in which to author any number of assets, scenes, or shots. In practice, some artists felt more comfortable with a single workspace per task as the complexities of managing multiple shots were new and hard to manage with the nascent UIs. An alternative organization that while familiar would allow a multi-shot or multi-asset workflow was to create a task specific to the multi-shot/multi-asset editing that could be modified in a workspace.

4.2 Resolvers in Production

To facilitate a push mechanism, our dependency resolution is based on labels assigned to specific numeric versions. Our initial implementation employed the following three labels: *latest*, *published*, and *approved*. While the intent behind a label is for it to be used to propagate changes through the pipeline, the naming of the available labels introduced a conflict between the user perception of their purpose and their actual function. Specifically the label *approved* was mistaken as a certificate of approval rather than a mechanism for pushing the most stable version to production.

The use of a label based resolver avoids some complexities encountered in dependency resolution. However, when combined with the ability to have local overrides at any level, as well as the ability for the user to dictate the timing of incoming updates, the system can cause confusion. All this highlights the need for the aforementioned collector workflow to enable simple and stable dependency networks.

4.3 Named PRIs vs. UUIDs in Practice

The use of human readable content identifiers wherever possible helps artists to communicate with their peers and is more informative when encountered in authored data than a traditional UUID. The downside is the commitment to an immutable name when a product is created. To address this, we support a simple way to clone a product to a new PRI. Through their tokens, PRIs provide an implied hierarchy which can be leveraged for product discovery.

5 CONCLUSION

Transitioning Blue Sky Studios to a new pipeline was a learning experience for both artists and developers. Throughout this transition we evolved our USD structures, artists workflows, and tools to simplify the artists user experience. There is still much left to do, but after finishing our first production on the pipeline we feel like we were off to a good start.

ACKNOWLEDGMENTS

Our work would not have been possible without the amazing Artists, TDs, Engineers, and Production Management at Blue Sky Studios. Production was trusting and supportive throughout the process - in particular, Roberto Calvo.

REFERENCES

Oliver Staeubli, Tim Hoff, Ryan Bland, Rebecca Hallac, Josh Smeltzer, Chris Rydalch, Karyn Buczek Monschein, and Mark McGuire. 2019. Conduit: A Modern Pipeline for the Open Source World. In *ACM SIGGRAPH 2019 Talks* (Los Angeles, California) *(SIGGRAPH '19)*. Association for Computing Machinery, New York, NY, USA, Article 47, 2 pages. https://doi.org/10.1145/3306307.3328175