USD at Scale

Jon-Patrick Collins
Animal Logic
Sydney, NSW, Australia
jonc@al.com.au

Romain Maurer
Animal Logic
Sydney, NSW, Australia
romainm@al.com.au

Fabrice Macagno
Animal Logic
Sydney, NSW, Australia
fabricem@al.com.au

Christian Lopez Barron
Animal Logic
Sydney, NSW, Australia
christianl@al.com.au

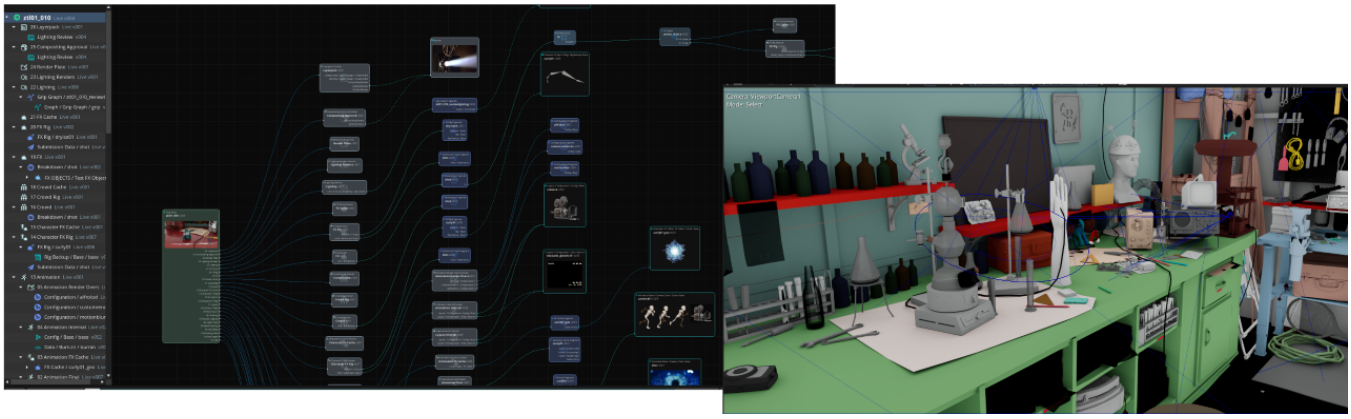

Figure 1: USD Asset Graph component, presenting the physical USD structure of a typical production shot, alongside a live 3D viewport render of the same shot. The graph flows from left to right, with the shot entity to the left, connected to downstream domains, subdomains, fragments and technical variants. This USD ALab Open Source Scene is available for public use at https://animallogic.com/usd-alab/.

ABSTRACT

We describe key steps in the process by which an animation and VFX studio (Animal Logic) integrated Pixar's Universal Scene Description™ into a large existing legacy pipeline. We discuss various architectural choices, as well as software systems developed to support these patterns. This successful USD migration has enabled the studio to significantly improve its toolchain productivity, supporting the simultaneous development of multiple feature films.

CCS CONCEPTS

• **Computing methodologies** → **Graphics systems and interfaces**.

KEYWORDS

USD, Pipeline, VFX, Workflow

ACM Reference Format:
Jon-Patrick Collins, Romain Maurer, Fabrice Macagno, and Christian Lopez Barron. 2022. USD at Scale. In *The Digital Production Symposium (DigiPro '22), August 7, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3543664.3543677

DigiPro '22, August 7, 2022, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9418-5/22/08…$15.00
https://doi.org/10.1145/3543664.3543677

1 INTRODUCTION

Historically, Animal Logic used a *generative* approach to scene construction, whereby atomic assets were assembled by tools based on distinct workflow requirements, without requiring the use of an explicit scene description. This system was mostly database-driven, and required extensive tooling. This approach was labor-intensive and required large amounts of software code and manual configuration. Whilst our toolchain itself was shared across workflows, the underlying assets were heterogeneous, with many file formats involved.

A short-lived attempt to develop an in-house Common Scene Description (CSD) format ended with the acknowledgment that developing such a system is a very significant undertaking that was beyond our development scope.

Following the open-sourcing of Pixar's Universal Scene Description™ in 2016, we embarked on a phased rollout of USD, with a successful pilot project by the Animation team on *Peter Rabbit* (2018) with the **Forge**™ shot-building application [Baillet et al. 2018]. Following this success, we undertook a major migration of the entire pipeline to USD with *Peter Rabbit 2* (2021).

Our USD toolchains continue to mature on our current slate of projects that include *DC League of Super-Pets* (2022) and *The Magician's Elephant* (2023), with a focus on building USD content at scale. We now generate many millions of USD files during the lifetime of a typical production. The transition to USD has allowed

us to consolidate many workflows and technology stacks, with an efficiency gain that now allows us to develop multiple feature films concurrently.

2 ARCHITECTURAL CHOICES

When designing our USD-based pipeline, we needed to consider many concerns such as: how to model pipeline content and workflows as atomic USD files; how to ensure that these USD files are composed in the most efficient way possible; how to automate the generation of USD files, including bulk rebuilds of many files; how to support packaging and versioning of USD files; how to balance the need for a centralized scene description with the decentralized nature of development teams across projects, locations and departments; and how to support parallel workflows across departments. These constraints led to the following design choices.

2.1 Fragment-Based Composition

A common software pattern in video game design is the *Entity-Component-System (ECS)* pattern, whereby entities are not defined by their "type" but rather by the components from which they are assembled. For example, an entity might be renderable if it contains a geometry component and might be simulatable if it contains a physics component.

We were inspired by this pattern to create a library of reusable global **fragments**, along with additional libraries of shot-specific fragments. Each fragment is defined by a USD file that typically references, sublayers or payloads other USD (and non-USD) files. Fragments represent units of reusable functionality, such as geometry, rigging or UV bindings. We then modeled top-level breakdown concepts (such as characters, props and shots) as **entities**, where each entity is defined by a USD file that contains a list of fragments that it references.

This clean separation between entities (aggregates of functionality) and fragments (units of functionality) is a key design choice from which much flows. A key result of this pattern is that it discourages a profusion of project-specific, typed entities and reduces the number of key concepts to be modeled by various software systems. This was a break from our previous pipeline, where every new pipeline concept required its own software and configuration stack, and where stale, project-specific types persisted in the codebase.

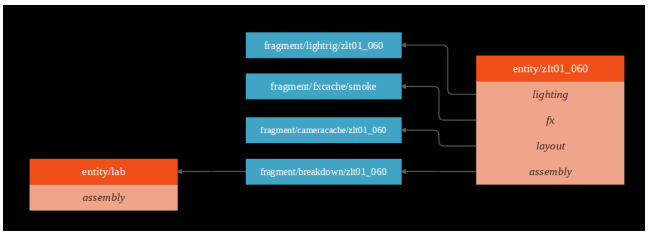

Figure 2: Composition arc diagram, used to describe the relationships between entities and fragments.

This pattern seeks to achieve complexity through the combination of fragments and their interplay, rather than by manually hard-coding each new top-level concept that arises during production. The pattern also seeks to create reusable functionality in the form of libraries, where fragments are the unit of reuse and where any given fragment may be referenced by multiple entities.

2.2 Entity-Based Composition

In the standard composition pattern, entities reference fragments, but fragments do not reference entities. However, in order to support entity assemblies such as environments, which are aggregates of many set pieces, we introduced certain fragments that reference entities. Notably, we utilize the **Assembly** and **Breakdown** fragments, both of which contain lists of references to other entity USD files.

We use the Assembly fragment to define entity assemblies such as environments, and we use the Breakdown fragment to populate shots with characters, props, environments, cameras and lights. The main distinction between the two is that assemblies are intended for more straightforward compounding of simpler entities, whereas entities added to a shot breakdown are intended to be heavily modified by many additional layers of tweaking, motion and other USD opinions.

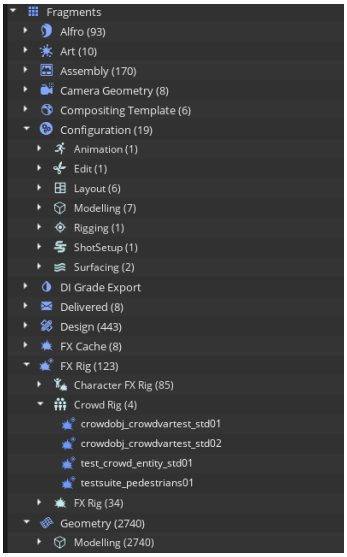

Figure 3: Global Fragment browser, with fragments categorized by their fragment type and compatible domains.

2.3 Entity Domains

We introduced an additional layer of organization between entities and fragments, the **entity domain**. Entity domains represent contributions to an entity from a single logical workflow such as Modeling, Rigging or Surfacing. Entities are then constructed as a stack of domain sublayers, each of which references potentially many fragments.

Entity domains provide a natural mapping to departments and workflows, and the relative strength of domains (as configured on a per-project basis) is reflected in the relative order in which they are appended to the entity sublayer stack. A key result of this design pattern is that it clearly demarcates the area of responsibility for

each department, and ensures that the final deliverable of each departmental workflow is an entity domain USD file where all USD content can be readily sublayered by a referencing entity.

2.4 Technical Variants

We separated out the heavy payload data of fragments into separate **technical variant** asset files. Many technical variants are stored as native USD files (typically .usdc files for heavy content and .usda files for lighter content), but many are also external file formats (such as Foundry Nuke™ .nk or SideFX Houdini™ .hou files) or standard data formats such as XML, YAML or TOML.

Fragments will usually define a USD variant set, where each named variant references a technical variant asset. For example, the Modeling entity domain references a Geometry fragment, which then references multiple geometry technical variants via a variant set, where each technical variant is a different resolution or use-case, such as *BaseMesh*, *PoseMesh* or *HighResolutionDeformationMesh*.

2.5 Packaging

Prior to USD, Animal Logic used a convoluted asset packaging system that had evolved over many years with an opaque toolchain. Migrating to USD provided us with the opportunity to streamline and modernize this toolset.

In order to package USD content, we simply snapshot USD files but *swap unversioned identifiers with versioned identifiers*. We optionally also post-process these USD snapshots as needed, with some content pruning and injection of metadata (such as bounding box extents). This creates a simple and powerful way to create snapshots of asset hierarchies, since we can repeat this process recursively through the USD graph topology. We only create package files for lightweight USD scaffolding content (namely entities, domains and fragments); we do not create snapshots for technical variants.

We store packaged assets with identifiers that have been extended to include the **package state**, with values such as *Live*, *Output* or *Delivered*; these are published as new assets on disk. In addition, at stage load time, we can also specify **dynamic overrides** of package states and versions in order to control which assets we want to use for any given workflow. Different workflows may require a blend of assets using different packaging rules; for example, a given workflow may call for the loading of a *Live* entity (where all content resolves to latest versions) but with *Delivered* modeling geometry (resolving to the latest delivered version, even if there is a newer non-delivered version).

3 IMPLEMENTATION

Our first approach to generating USD content was centered around our existing production database. Users would explicitly add new content to the database, which would trigger a script that would rebuild USD content. This system was fairly rigid and rapidly devolved into a large collection of complex scripts. The key flaw, however, was that USD generation first required updating the production database. Our next iteration of software sought to remove any dependency on the production database, allowing tools to create production-ready USD assets without needing to populate the production database with spurious or experimental content.

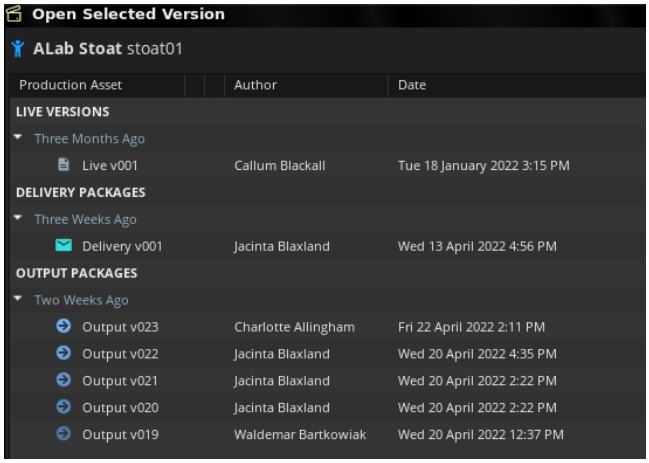

Figure 4: Open Selected Version dialog, a utility window for artists to select a specific version of a given entity to open. The dialog displays the various package states and their associated versions discovered for the given entity.

3.1 LEAF Toolkit

To support standalone entity-fragment content creation, we developed the **LEAF (Layered Entities and Fragments) Toolkit**, a collection of Python libraries that provides a suite of functionality for USD file creation and editing. Some key characteristics of LEAF include:

- A collection of *general-purpose facade classes* such as `FnEntity`, `FnDomain`, `FnFragment` and `FnTechVariant` that model our entity-fragment concepts. We use this API layer to ensure a consistent approach to creating, editing and traversing USD content, ensuring it conforms to our USD design patterns.
- An extensive registry of *factory builder classes* used to construct and modify specific domain and fragment types. Currently, we have builders registered for over 60 different domain and fragment types, each maintained by technical directors (TDs) from their relevant craft groups. Builders provide a managed, extensible ecosystem for the creation of USD content, allowing contributors from across the organization to extend the overall USD scene description but within a constrained framework.
- An extensive library of *configuration files* that define various default entity-fragment topologies when creating new entities. Configuration files allow for a clean separation of code and data; these files are stored in TOML format and maintained by TDs for each active project.

3.2 AssetWorkshop Toolkit

The next phase of work involved developing the **AssetWorkshop Toolkit**, an ecosystem of tools that allow artists and TDs to view and edit USD graphs using intuitive hypergraph-style user interfaces, with rich interactive tools for selection, context menu commands, drag-drop capabilities and more. This work introduced:

- **AssetWorkshop API**, a higher-level Python library that encapsulates LEAF with a simple, object-oriented API that

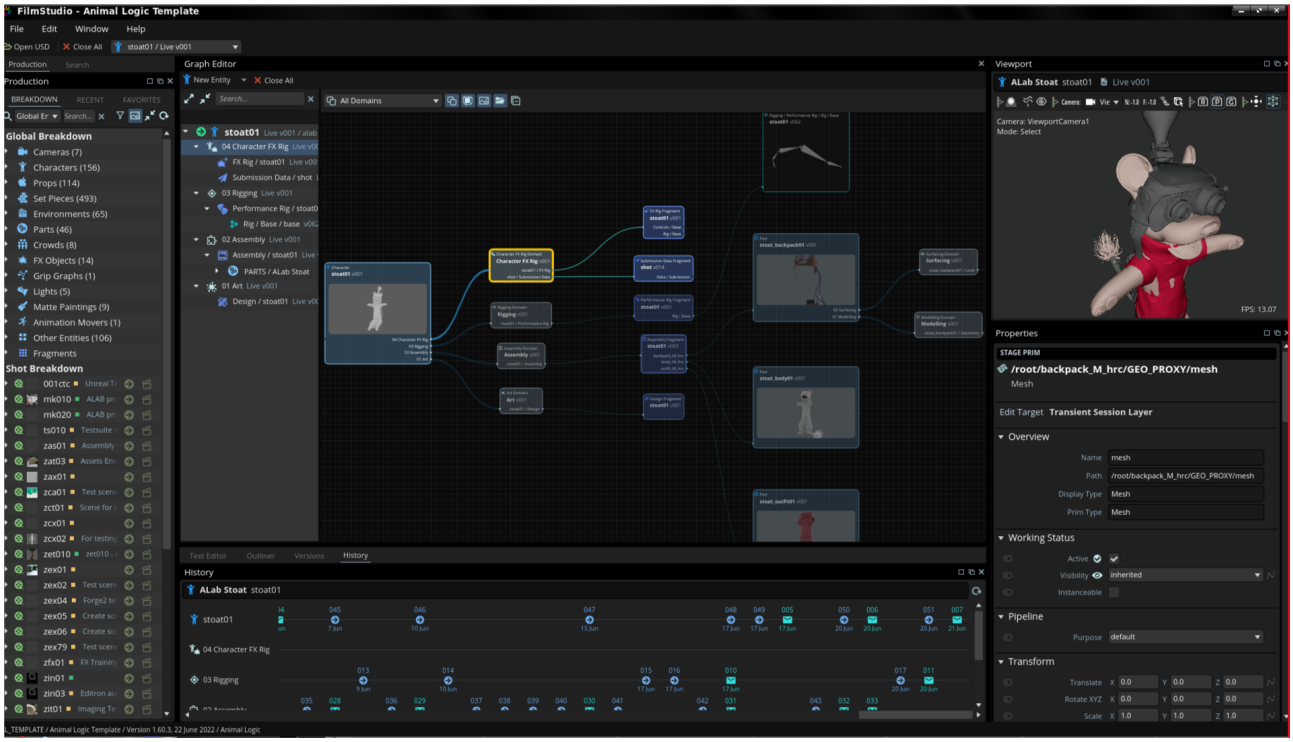

Figure 5: FilmStudio™, a comprehensive tool for viewing and editing USD entities and fragments.

includes tighter integration with our production database; a rich collection of commands for modifying USD graphs, such as inserting domains and fragments; and improved UI support with production media, tooltips and other UI elements.

- **USD Asset Graph**, a reusable user-interface component developed in Qt and available for use by in-house applications. Powered by AssetWorkshop, this view includes context menu commands for inserting and deleting graph content; drag-drop support for adding content; and visual feedback indicating the status of each USD file.

3.3 VirtualBreakdown Toolkit

Our more recent phase of work has involved developing the **VirtualBreakdown Toolkit**, a high level collection of tools based around "virtual breakdowns"—recipes that describe one or more entities from a logical perspective. The motivation for VirtualBreakdown was two-fold.

In some workflows, users are more interested in a higher-level, logical view of content than they are in a lower-level asset-based view. For example, they may be interested in working with a shot manifest that describes the content of a shot, such as its characters, props, lights, environment and cameras.

Also, users may be working with large numbers of entities at a time, and do not require opening every entity as a separate in-memory USD stage. In some cases, they may be updating dozens or hundreds of shots and require an efficient way to work with these without incurring a non-trivial (often over 30 second) stage load time per shot. This work built on LEAF in the following ways:

- **VirtualBreakdown API**, a higher-level Python library that describes a hierarchical asset recipe that can be processed locally to generate USD content; or can be *published*, which involves offline processing to generate and check-in USD content as well as updating the production database.
- **Virtual Breakdown Editor**, a user-interface component developed in Qt and available for use by in-house applications. Powered by VirtualBreakdown API, this component uses an outliner-style approach to displaying entities and their logical content.

4 MILESTONES

The transition from our classic generative pipeline to our modern, USD-based pipeline occurred over an extended period, punctuated with key milestones associated with certain productions. We used the **same asset management system across all versions of the pipeline**, allowing us to support multiple pipelines in parallel during this transition. In particular, we were able to maintain our classic pipeline on *The Lego Movie* (2014), *The Lego Batman Movie* (2017), *The Lego Ninjago Movie* (2017) and *The Lego Movie 2: The Second Part* (2019) whilst developing a parallel USD-based pipeline.

4.1 USD pipeline prototype

We sought to integrate USD into a **limited subset of our pipeline**, in order to manage risk and with an existing toolset as a fallback

option. On *Peter Rabbit* (2018), we selected the Animation pipeline as a suitable prototype test-case. One motivation for this choice was that this pipeline was already undergoing an abrupt platform change from SOFTIMAGE|XSI™ to Autodesk Maya™. Development included:

- A suite of **USD generators** that process the production database to generate corresponding USD layers (noting that asset formats themselves remained unchanged, such as the use of Alembic as our baked geometry format);
- An artist-facing application (**Animal Logic Forge™**) that allows animators to load and manipulate USD content within a Maya context, including the ability to toggle visibility, active status and active variant of individual USD layers;
- A custom Maya plugin (**AL_USDMaya**) to translate live USD composed stage content into native Maya data.

4.2 Technical migration

We reflected on the generalized design requirements that would be needed to roll out USD across all pipelines. This design phase culminated in the **entity-fragment design pattern** and the development of **LEAF Toolkit** to manage USD content generation.

Certain less-successful features of the USD prototype were abandoned, such as the storage of **Python code** as USD attributes, which had been an attempt describing workflows inside USD assets; and the storage of **user-interface configuration** using USD layers, which was beyond the natural scope of USD.

Embarking on a wide rollout of USD on *Peter Rabbit 2* (2021), we generated USD content in parallel with existing assets for all pipelines, and introducing **USD-based packaging**.

Our existing artist-facing tools were re-engineered internally to work with USD assets as inputs, but were otherwise unchanged. This phase was technically risky, as the project success was predicated on this novel design pattern. Also, artist disenchantment was considerable, as many tools suffered workflow breakages, poor performance or perceived degraded functionality. We developed:

- An artist-facing application (**Animal Logic Environment Studio™**) that allows set dressing artists to load and manipulate USD content within a Maya context. This was our first studio tool based on the new entity-fragment paradigm.
- The Performance (Layout and Animation) pipeline was upgraded with a slightly modified version of Forge™ compatible with new LEAF pipeline.

4.3 Initial workflow migration

In order to fully realize the value of a USD-based pipeline, we needed to **upgrade the full range of artist-facing tools** to work with USD concepts natively, rather than simply using USD as a back-end. This process had a number of challenges, including artist resistance to redesigning existing tools, and the need to communicate the entity-fragment design pattern more fully. This work was carried out during the development of *DC League of Super-Pets* (2022). Development included:

- The **AssetWorkshop Toolkit**, and its deployment in a new artist-facing application (**Animal Logic FilmStudio™**) to allow asset TDs and coordinators to explore production content in a fully entity-fragment compliant manner, with the ability to create and edit assets graphically using interactive visual controls;
- Upgrading studio tools such **Animal Logic Filament™** (our proprietary lighting tool) to use the entity-fragment pattern;
- Dropping support for non-USD formats for technical variants where possible, such as removing the generation of Alembic baked geometry data in favor of USD binary (.usdc) format.

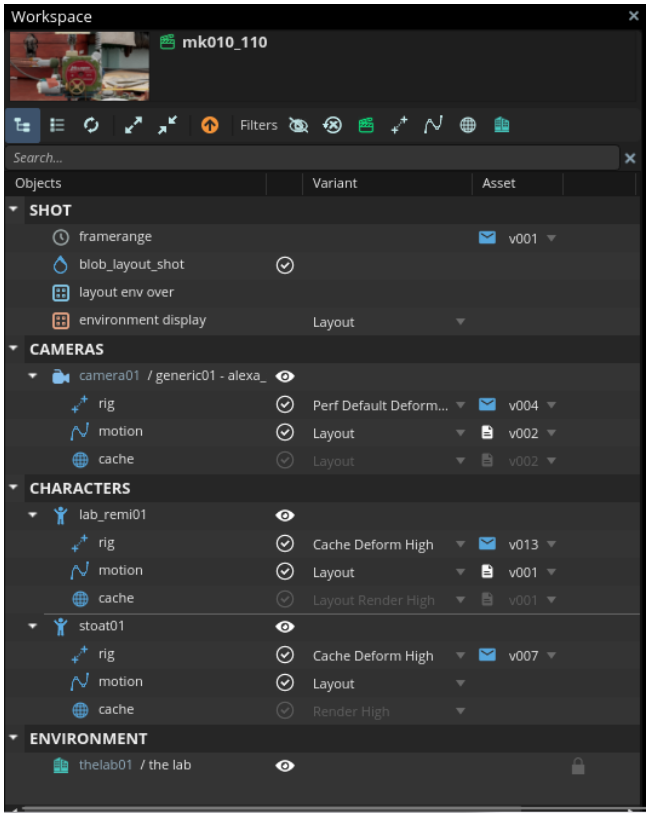

Figure 6: Workspace view deployed with Forge™ to manipulate scene content.

4.4 Further workflow migration

We have continued enhancing our USD-based workflows on *The Magician's Elephant* (2023), *The Shrinking of Treehorn* (2023), and two further as-yet-unannounced projects. We are still working to complete feature parity with our classic pipeline. Most workflows have been migrated to use USD entities as their entry point. Recently development has included:

- The **VirtualBreakdown Toolkit** and its deployment in FilmStudio™ as a next-generation workflow for assembly artists to populate shots;
- New workflow tools, including **Animal Logic Modeling Studio™** for modeling artists to work with Maya in a USD-based manner.

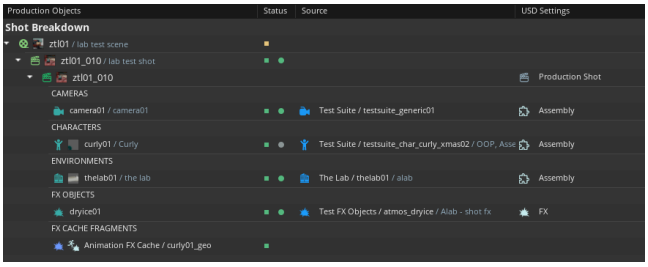

Figure 7: Virtual Breakdown view, presenting the logical structure of a typical production shot.

5 DISCUSSION

The processes that we have described generate USD content that can be consumed by **any USD-aware system**. The only proprietary concepts in our USD scene descriptions are the usual USD extensions, such as kinds.

Our pipeline is built around the idea of **reusable USD content** in the form of fragments, where each fragment type has associated registered builder logic. Any part of the fragment library, from geometry to materials to motion, can potentially be referenced by any entity. Importantly, entities and fragments are ultimately an organizational approach to provide a taxonomy on top of our USD assets.

We have opted to use **entities as the common starting point to every workflow**. All relevant asset references are available and discoverable through the USD scene description, regardless of whether any given asset is a USD file or not. We represent non-USD assets using attributes in the scene description so that external tools can process them.

A useful characteristic of our current USD pipeline is that entities are now explicit collections of fragments that are reviewed to ensure the **correctness of the combination**. Our previous pipeline had no such explicit statement of valid combination, and no way of knowing which combinations of surfacing variation, rigging variation, FX variation or lighting variation were valid together.

Another important aspect of our USD pipeline is that content creation now occurs **independently of the production database**. Previously, breakdown content was required to be first registered with our production database, after which automated processes would be triggered to generate USD content as needed to sync with the database. Our USD pipeline inverts this relationship; the LEAF API creates production-ready USD content without the database being notified at all, and the VirtualBreakdown API notifies the production database but only as a final step for consistency, rather than as an initial step. Artists can now explore variations and ideas using fully production-ready USD content that can be imported into digital content creation (DCC) systems like Autodesk Maya™ without the database being clogged with endless churn.

Organizationally, the USD rollout has been a success, notwithstanding a transitional period in which artists were frustrated with broken and degraded workflows. Some indicators of this success include:

- A single USD API replacing many different workflows and file formats, making for a **more unified and coherent technology stack**, with most concepts expressed in USD, and less reliance on the production database;
- A more powerful **suite of workflow tools** has emerged to manage breakdown changes, with more visibility on the assets used;
- **Departmental contributions** are now more explicit due to domain USD assets now defined as first-class concepts (that map closely to departments), as well as via the improved USD-based packaging;
- With USD emerging as an industry standard, **onboarding and upskilling technical staff** is easier, with increased sharing of internal resources such as tutorials, glossaries, documentation and extension libraries.

6 FUTURE WORK

One area of active research and development is the **optimization of asset resolution**. For larger USD graphs with potentially thousands of asset identifiers to resolve, the combined asset resolution duration is a limiting factor in how rapidly we can regenerate USD content and load USD stages. We are also exploring ways to reduce the number of generated files, which will be a consideration for any **cloud migration**. As an example, some of our larger shots contain 15,000+ asset references, each requiring a separate call to a Web service.

Another area of ongoing work is providing tools to allow technical developers to **work at extreme scale** in order to perform updates of potentially many thousands of assets across a production. We are embarking on a range of tools to allow for rapid deployment of custom scripts in the context of a visual programming environment to help TDs work more efficiently and powerfully.

We are also aware that our robust migration of content and relationships out of our production database and into USD state has in some cases made certain lookups much slower. For example, we do not store entity-fragment relationships outside of USD itself, making it difficult to quickly determine which entities use a given fragment. We are exploring ways to **cache key USD content and relationships** in a way that permits rapid search and lookups.

Finally, we are exploring ways to improve our **quality assurance**, improving the way we analyze domain contributions for validation and compatibility in order to limit breaking deliveries.

ACKNOWLEDGMENTS

We would like to thank Jens Jebens, Eoin Murphy, Aidan Sarsfield, Oliver Dunn and Justen Marshall for their leadership in the development of the entity-fragment paradigm and packaging system.

REFERENCES

Aloys Baillet, Eoin Murphy, Oliver Dunn, and Miguel Gao. 2018. Forging a New Animation Pipeline with USD. In *ACM SIGGRAPH 2018 Talks* (Vancouver, British Columbia, Canada) *(SIGGRAPH '18)*. Association for Computing Machinery, New York, NY, USA, Article 54, 2 pages. https://doi.org/10.1145/3214745.3214779