

Hello! I'm Mike Murray and this is Alan Blevins. We both work for DreamWorks Animation and today we're going to talk about our studio's experience of adopting Pixar's Universal Scene Description (or USD) for our feature production pipeline.



Just a quick note... we ask that you don't record any part of this presentation.. And with that out of the way...

1

2



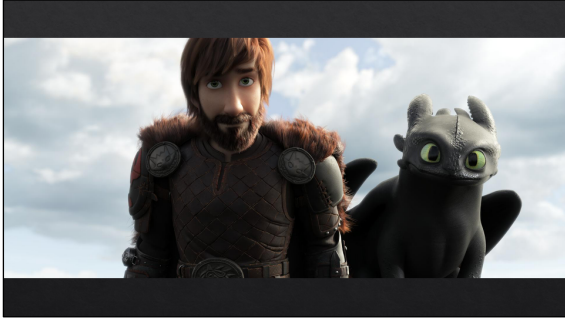
Today we'll briefly be touching on our approach to adoption, our first-pass at asset and shot design, how we integrated USD with our existing production framework, and the successes and challenges we faced along the way



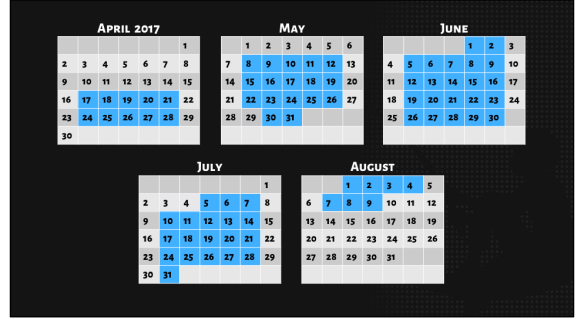
To give you a quick backstory...

3

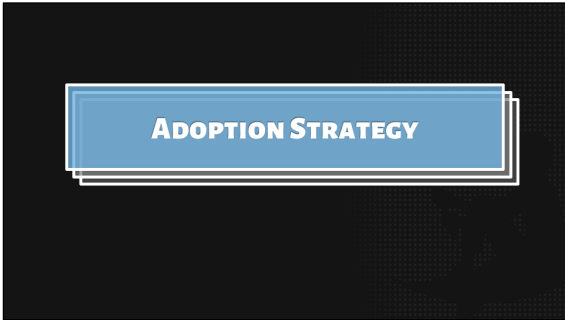
4



For How To Train Your Dragon: The Hidden World, we made an ambitious decision to adopt a brand new MCRT renderer called "Moonray" in a very short timeframe. As part of this move, we needed to replace our legacy scene description. We were searching for something fast, scalable, and ready-to-go... which is where USD came into play



This decision was made in Spring of last year. With Dragons production ramping up near the end of the year, we had a very short window to get everything in place



We needed a solid strategy to give ourselves the greatest chance for success


ADOPTION STRATEGY

1. ASSEMBLE A TEAM
2. MAKE SOME CALLS
3. DRAW THE PLANS
4. SCHEDULE THE WORK
5. TEST IN PRODUCTION

We started by forming a team we called the USD Steering Committee. It was made up of representatives from R&D, Pipeline, and TDs with expertise in all different fields. Additionally, the members' priorities ranged from immediate production needs to long-term ideal solutions. We had many conflicting ideas of how USD should be used so the diversity of stake holders prevented us from straying from the path and allowed us to converge towards robust, forward looking solutions.

ADOPTION STRATEGY


1. ASSEMBLE A TEAM
2. MAKE SOME CALLS
3. DRAW THE PLANS
4. SCHEDULE THE WORK
5. TEST IN PRODUCTION



We knew of other studios in the process of adopting USD. We reached out to several and they graciously agreed to answer our questions about their integration experience. We gained a lot of great insight from their stories

ADOPTION STRATEGY


1. ASSEMBLE A TEAM
2. MAKE SOME CALLS
3. DRAW THE PLANS
4. SCHEDULE THE WORK
5. TEST IN PRODUCTION



Next, we translated our ideas into diagrams, design documents, and specifications, running it through the USD steering committee for feedback. After several iterations, we eventually finalized a version 1...

ADOPTION STRATEGY


1. ASSEMBLE A TEAM
2. MAKE SOME CALLS
3. DRAW THE PLANS
4. SCHEDULE THE WORK
5. TEST IN PRODUCTION



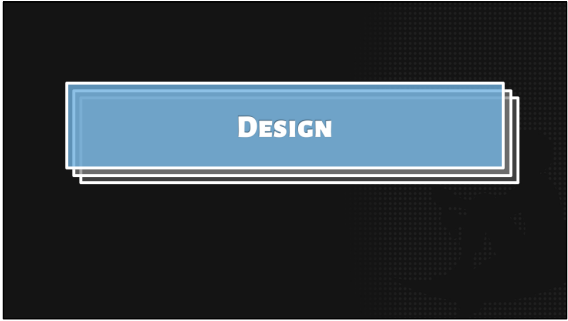
And triaged the development. It wasn't too long before we had working prototypes, but we needed a place to thoroughly test them... So we

ADOPTION STRATEGY

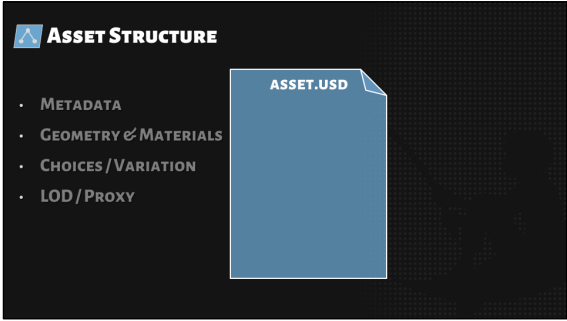
1. ASSEMBLE A TEAM
2. MAKE SOME CALLS
3. DRAW THE PLANS
4. SCHEDULE THE WORK
5. TEST IN PRODUCTION



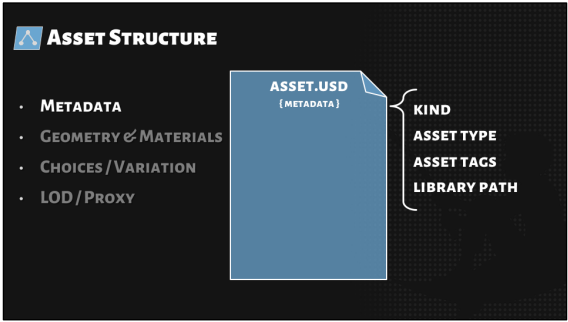
Deployed our new pipeline on a short called, "Bilby." The short provided the perfect testbed for hammering on our new workflows. Leadership emphasized the precarious state of the technology on the short and the crew maintained an understanding attitude through production. It certainly helped that artists were now producing some of the best renders to come out of DreamWorks.



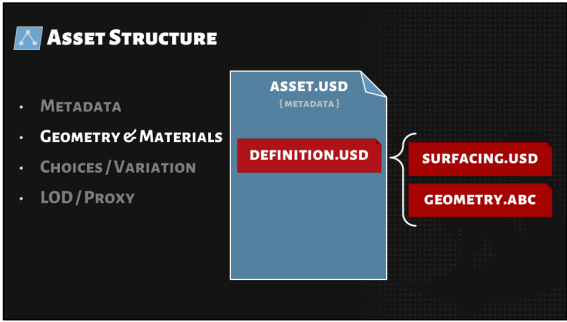
Moving on to some of the specifics of our design...



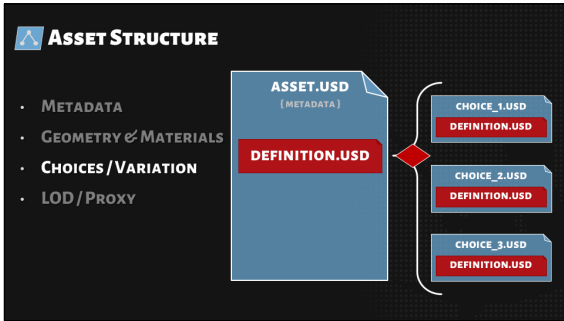
In drafting our new asset structure, our goal was clear... provide the same control and flexibility that existed in our legacy asset model



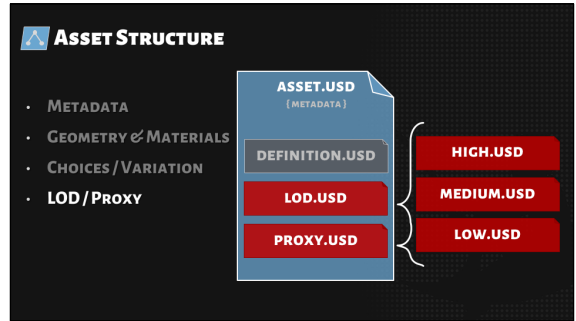
We start by defining a single, primary layer. We include important metadata storing information about the assets' type, USD structure, and path in the library.



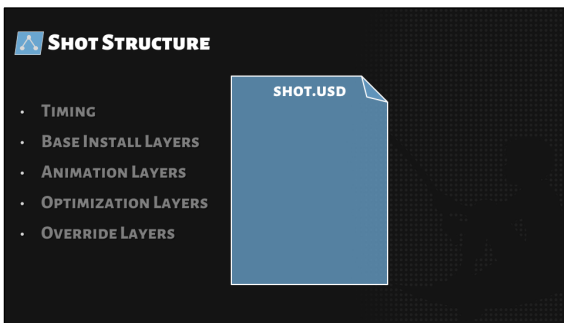
Next, the Modeling department contributes alembic geometry, and the Surfacing department publishes the shading network making up the various looks of the asset. These files form the base definition of the leaf-level asset.



For more complex assets, such as character, consisting of child assets, we provide a mechanism for making selections about which child asset to choose. For example, our Art department might designate which combinations of garments are appropriate for a character. We then group these presets together as a selectable option on the top-level asset.

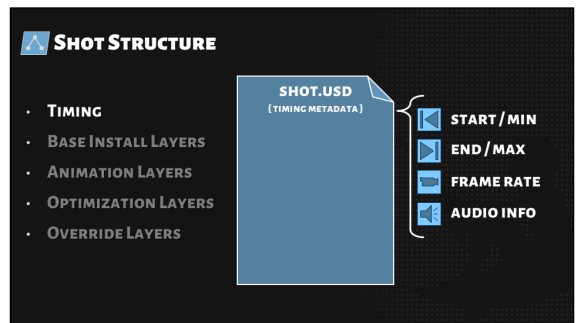


In addition to looks variations, we run a process to decimate geometry and bake-down shading networks in order to provide lighter-weight options for both rendering and display in the 3D viewport.

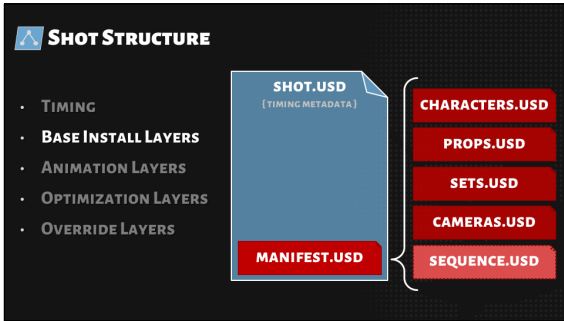


Jumping over to shot structure...

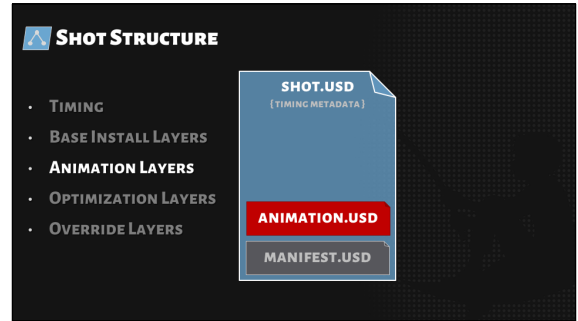
Our USD shots parallel a lot of similar constructs and functionality we had in our previous scene description. For every production shot, artists could expect there to be a primary USD file representing the official, published version of the scene.



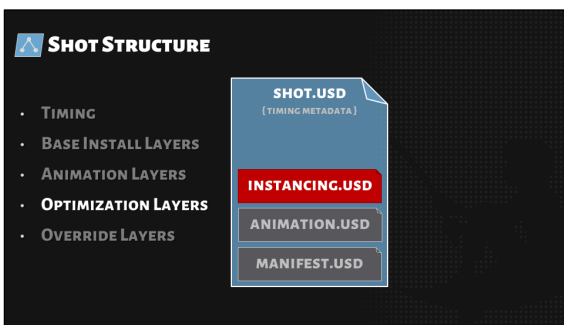
In the main layer, we store timing information as metadata. These timing values are updated with each new cut produced by Editorial.



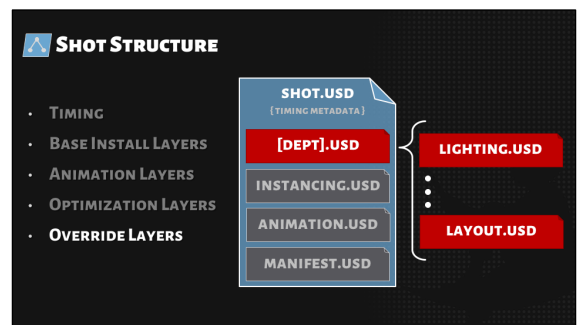
Our shots bring in assets via a base install layer which we call the manifest. The manifest pulls in any characters, props, set pieces, and cameras installed by our Layout department. The manifest may also bring in other shot's manifests or pull from a sequence-level install layer.



When the assets are installed in the shot, they will be in their default library pose. In order to apply animation, we override each asset's model paths to point to baked deforming geometry on disk.



Next, we run an optimization pass to enable instancing for assets we deem "safe" to do so. We saw significant drops in memory by doing this. We gave these overrides their own layer so that it's easy to turn off when trouble-shooting and simple to update on demand.



Finally, we include our department override layers into the shot. Each department has a layer where they are expected to make overrides specific to their workflow. These layers are strength ordered so back-end departments' overrides take precedence over front-end departments'.

USD EXTENSIONS

- BUILD / DEPLOYMENT
- ASSET RESOLVER
- ASSET / LAYER METADATA
- PROCEDURAL SCHEMA
- MOONRAYOBJECTAPI

While we aimed for the most universal design, there were various components of our new pipeline that required DreamWorks-specific logic.

USD EXTENSIONS

- BUILD / DEPLOYMENT
- ASSET RESOLVER
- ASSET / LAYER METADATA
- PROCEDURAL SCHEMA
- MOONRAYOBJECTAPI

One of the first things we found ourselves modifying was our USD deployment. In addition to creating a Scons-based build system, we broke up and released USD as smaller packages for extra control and to minimize dependencies.

USD EXTENSIONS

- BUILD / DEPLOYMENT
- ASSET RESOLVER
- ASSET / LAYER METADATA
- PROCEDURAL SCHEMA
- MOONRAYOBJECTAPI

For over a decade, DreamWorks has used a proprietary environment-variable like system for overrides and navigation within the production environment. We wrote a custom Asset Resolver allowing USD to recognize and expand these paths which map to our file system.

USD EXTENSIONS

- BUILD / DEPLOYMENT
- ASSET RESOLVER
- ASSET / LAYER METADATA
- PROCEDURAL SCHEMA
- MOONRAYOBJECTAPI

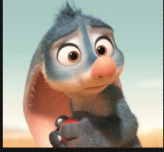
PRIM METADATA	
FIELD	EXAMPLE
ASSETTYPE	"ENVIR"
ASSETTAGS	["FOLIAGE"]

LAYER METADATA	
FIELD	EXAMPLE
MINTIMECODE	100
MAXTIMECODE	200
AUDIOFILE	\${A_AUDIO_FILE}.AIFC
AUDIOOFFSET	91

USD makes it easy to create custom metadata. We recreated several DreamWorks-specific pieces of information in USD, mainly for consistency and compatibility with legacy toolsets.

USD EXTENSIONS

- BUILD/DEPLOYMENT
- ASSET RESOLVER
- ASSET/LAYER METADATA
- PROCEDURAL SCHEMA
- MOONRAYOBJECTAPI



```
def Procedural "bilby_fur"
{
  uniform token procedural:class = ...
  custom float procedural:density = ...
  ...
  rel material:binding = ...
}
```

When it came to representing Moonray Geometry in USD, there were several types which had no USD schema mapping. While we'd love to be part of the discussion for formalizing schemas for objects such as fur, we simply didn't have time. Instead, we opted for a new generic "Procedural" schema, which allowed us to pass objects without a first-class Schema through USD until we translated them to our renderer format. We find usefulness in this Procedural schema beyond the DreamWorks pipeline and plan on contributing it back.

USD EXTENSIONS

- BUILD/DEPLOYMENT
- ASSET RESOLVER
- ASSET/LAYER METADATA
- PROCEDURAL SCHEMA
- MOONRAYOBJECTAPI

```
>> from pxr import MoonrayObject as MO
>> mo = MO.MoonrayObjectAPI(prim)
>> attr = mo.CreateAttribute(
    'subasset',
    Sdf.ValueTypeNames.String)
```

Finally, we found it beneficial to create a USD API tailored specifically for authoring prims and attributes unique to our renderer

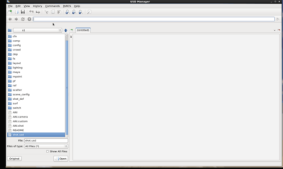
CUSTOM TOOLS

- USD FILE BROWSER
- PRE-LIGHTING RENDER

USD ships with a ton of useful tools, like usdview. We also created a few which we found helpful.

CUSTOM TOOLS

- USD FILE BROWSER
- PRE-LIGHTING RENDER

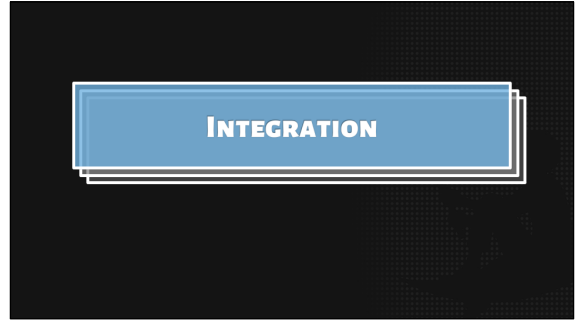


One of them is a USD file browser, which displays the ASCII representation of a USD layer, and allows you to navigate through file references in a web browser like interface. We are in the process of open sourcing this tool.

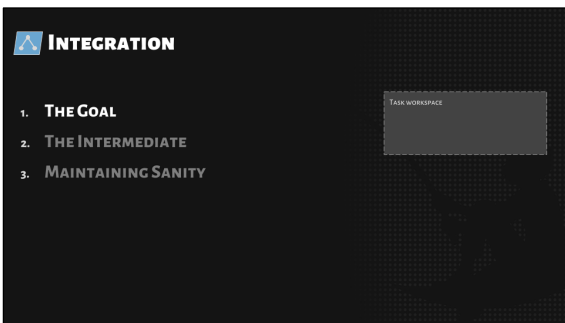


Our legacy pipeline had a tool used by front-end departments to easily kick-off pre-lighting renders. We re-created the tool from scratch to take in a USD stage, install a default light rig, and render the scene. The tool includes options to render all assets or select a subset, frustum cull, set various LODs, provide a custom comp script, and much more.

Now I'm going to turn it over to Alan to talk about USD integration...



I'll be talking a bit about the scope of our USD adoption across the DreamWorks pipeline.

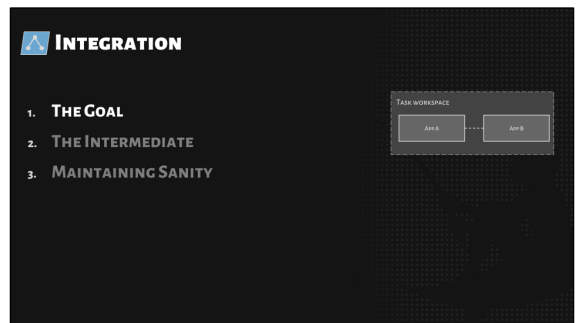


Our goal with USD adoption is to have USD as the common interchange format for scene data across all departments.

At DreamWorks, our pipeline works like this:

<click>

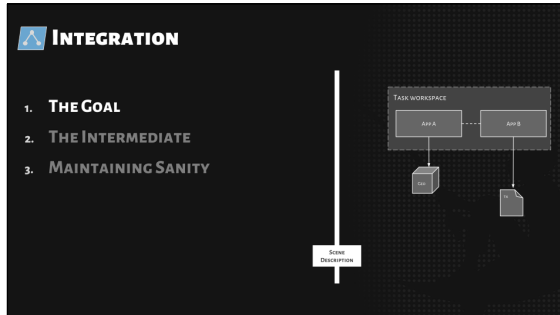
All artists do their work in a workspace specific to their task.



In that workspace, artists will use any number of applications to complete their task.



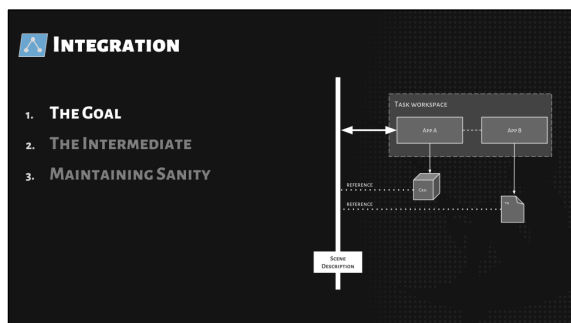
Artists create products that are consumed by the rest of the pipeline, such as geometry and textures.



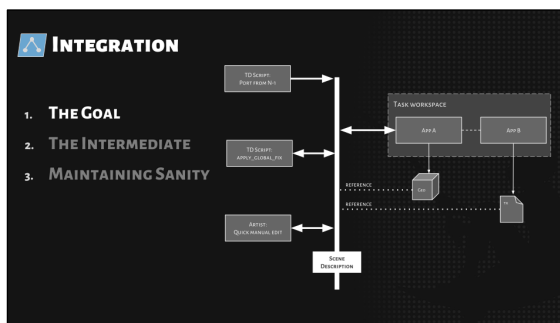
Shot and asset data is represented using a common scene description format.

38

39



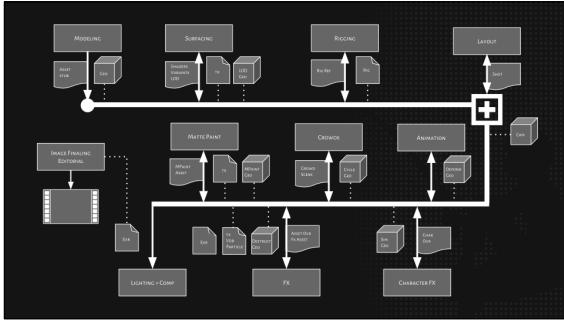
When an artist is ready to publish their work, the scene description is updated to reflect the artists's changes and reference any new products that may have been created.



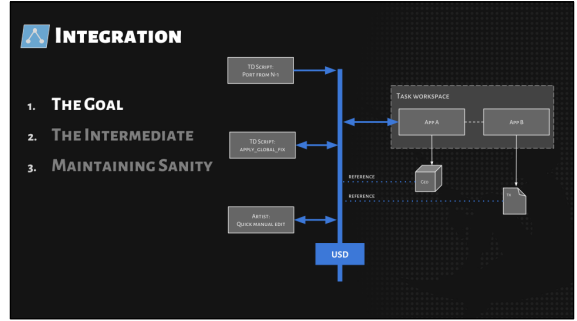
Other processes outside the workspace can also inspect and modify the scene description.

40

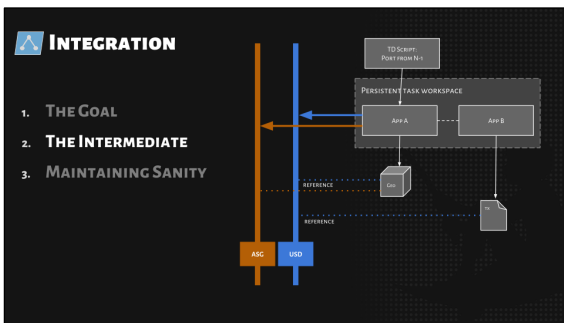
41



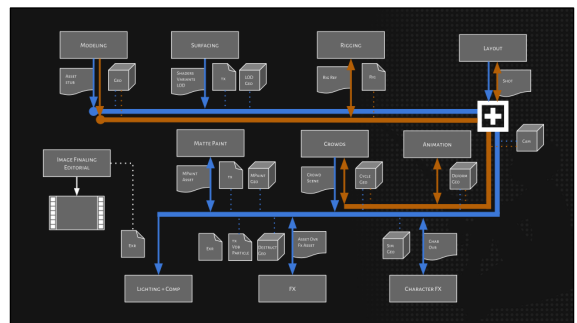
As a whole, the Dreamworks pipeline looks somewhat like this. Asset creation departments build individual assets. Layout combines assets together into shots. And then each shot department makes their contributions. Everything gets consumed by lighting and turned into final images for the film. The thread that ties all of these departments together is the common scene description.



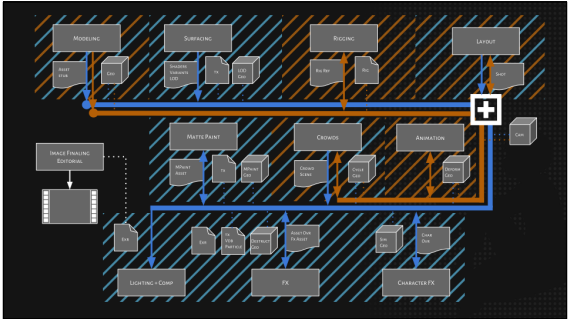
And so our goal is to have USD as the global scene description that ties everything together across the pipeline.



We've taken huge steps toward that goal, but we're not there yet. A few in-house apps and workflows are strongly tied to our legacy scene description called ASG, and there just wasn't time to update them for this first round of USD adoption. To accommodate this, we maintain a skeleton legacy pipeline with the minimal amount of data necessary to support those workflows. On export, some departments will write a partial set of legacy data in addition to a complete set of USD data.



This is our pipeline as of How to Train Your Dragon 3.



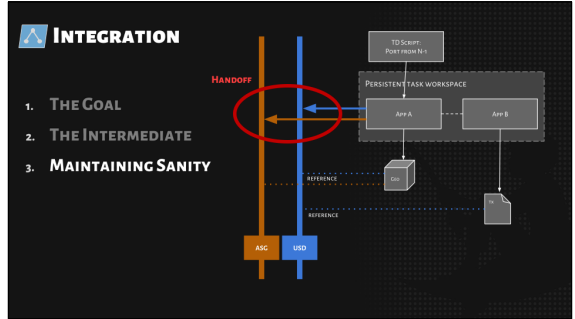
Most departments consider USD as the primary scene format

These departments work exclusively with USD. <click>

A few departments have a foot in both pipelines in order feed rigging, animation, and crowds.

Even so, all departments except rigging and animation author complete USD datasets. <click>

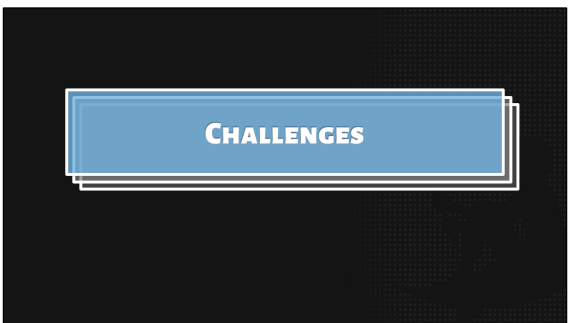
Animation and rigging are stuck in a legacy pipeline world (for now)<click>



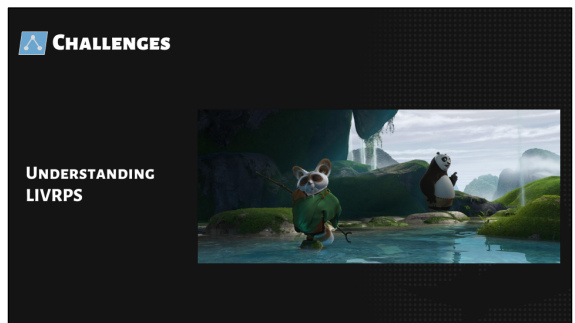
For departments that have to straddle the pipeline divide, we're able to maintain sanity due to our highly constrained check-in process, which we call handoff.

<click>

The only way for artist work to be added to the pipeline is to run a handoff process that performs any necessary conversions and validations as part of updating the global scene description. Because we have this controlled entry point, it's easy to leverage that process to ensure the correct data is exported to both scene descriptions and that parity is maintained.




I thought I could talk about some of the challenges we faced during this process.




USD composes individual scene description layers into meaningful assets and scenes using a scheme called LIVRPS.

While this has a lot of similarities to our legacy scene descriptions, it also has functionality that took some trial and error to fully understand. As a result, we restructured our shots and asses a few times as we got the hang of USD layer composition.

CHALLENGES

DWA

 PROCEDURALISM




At DreamWorks, our scene descriptions have always been full of proceduralism, including support for expressions, curve bindings, and runtime variable expansion.

USD by comparison is very declarative, so it has taken some adjustment to adapt and build processes to bake or replace some procedural representations.

CHALLENGES


MANAGING
 COMPLEXITY




As we started production on larger shots, it became clear that we needed to investigate ways to keep scenes manageable.

We saw immediate benefits from applying instancing to our scenes. We were also able to see improvements in other places by restructuring data to use payloads, layer offsets, variantsets, PointInstancers, and other USD concepts.

CHALLENGES

EDUCATION

 EXPECTATIONS

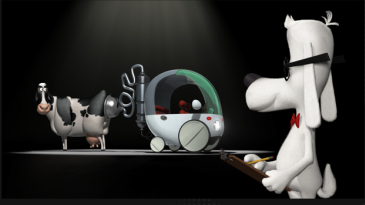


Because USD is new to almost everyone at DreamWorks, we've been making a concerted effort to maintain a set of wikis and documents that explain our USD deployment.

We've also prepared a number of classes that teach USD concepts, as well as all of our new DreamWorks USD toolset.

CHALLENGES

CONCURRENT
 TECHNOLOGY
 ADOPTION



In addition to adopting USD as our new scene description, we also simultaneously adopted a new renderer, a new lighting package, and a new surfacing tool.

I don't recommend this approach if it can be avoided.

CHALLENGES

ALEMBIC GEOMETRY



In order to maintain a bridge to some of our legacy workflows, we've been using Alembic with USD to represent much of the geometry in our scenes.

There's nothing wrong with Alembic itself, but USD is just inherently better at reading its own format called crate, both in terms of memory usage and scene load time.

We're currently in the process of moving to crate for all geometry as fast as possible.

54

SUCCESSSES

I want to end this talk on a high note, and mention everything that went well for us when adopting USD.

55

SUCCESSSES

EASY TO INTEGRATE
ROBUST PYTHON API
FLEXIBLE ASSET MODEL



Although USD concepts can be complex and have a bit of learning curve, it adds up to a data model that is both extremely flexible and powerful. We had no trouble representing our most complex asset types with USD once we got the hang of it.

Additionally, the APIs are well-designed, which made it straightforward to integrate into our applications and workflows.

56

SUCCESSSES

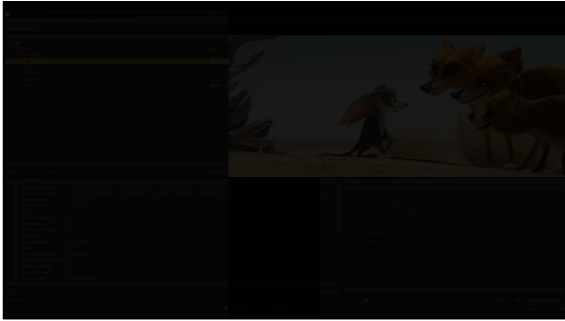
EXCELLENT DEBUGGING TOOLS

- USDVIEW
- KATANA PLUGIN



The tools that ship with USD have been very useful for inspecting our data for validation, debugging, and everyday queries.

57



In usdview for example, we can always find an answer to the important questions that all TDs have:

What is in this scene?

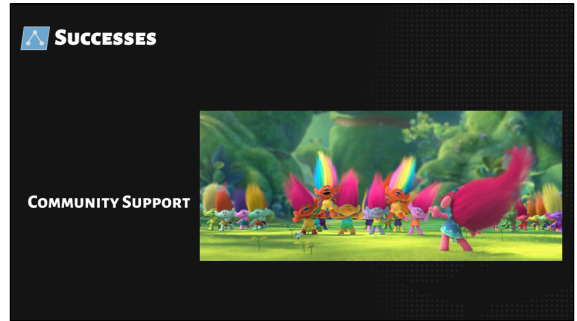
click

How did it get here?

click

Is this correct?

58



Finally, we've been able to reach out for help on several occasions.

The usd-interest google group is very active and has been insightful for both practical and philosophical USD questions.

And we've been fortunate to compare notes with a few other studios, which has been a big help as we tried to plan out our efforts.

59



That's all we have today. Thank you for coming, I hope this was a useful overview for anyone interested in adding USD to your own pipeline.

If you've seen the Dragon 3 trailer, it's worth noting that every frame was 100% USD backed.

And if you're interested in developing a USD pipeline, do checkout dreamworksanimation.com/careers

(time for questions)

60