UsdShade in the Pixar Pipeline

Florian Hecht
fhecht@pixar.com
Pixar Animation Studios
USA

Daniel McCoy
mccoy@pixar.com
Pixar Animation Studios
USA

Stephen LaVietes
stevel@pixar.com
Pixar Animation Studios
USA

F. Sebastian Grassia
spiff@pixar.com
Pixar Animation Studios
USA

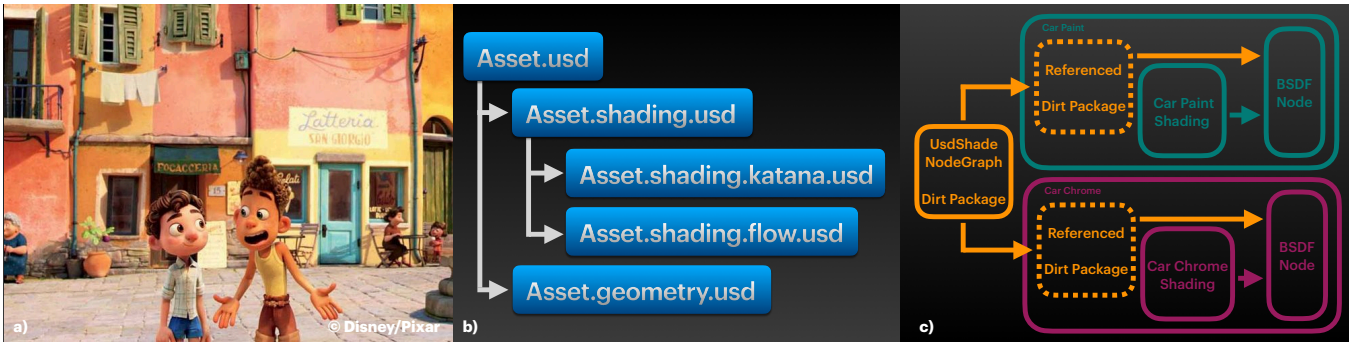

Figure 1: a) A frame from Pixar's *Luca*. The shading of every character and object is represented in *UsdShade*. b) Assets are structured with multiple *USD* layers, including multiple shading layers. c) *UsdShadeNodeGraph* prims allow shading with reusable packages that can be included in multiple networks. ©Disney/Pixar.

ABSTRACT

The VFX and animation industry is widely adopting Pixar's **USD** (Universal Scene Description) format to describe and manipulate scene information throughout production of CG content. A key part to a complete description of a scene is the representation of shading of all the parts, which is the description of materials that will be used to render it. **UsdShade** is the submodule of USD that is designed to handle material description. *UsdShade* was developed at Pixar in 2014 during the production of *Finding Dory* and has been used on all following productions. Since then we have learned a lot and have refined our practices to get the most out of *UsdShade*. We want to share our best practices and learnings to guide others to great success in using *UsdShade* in their pipelines.

ACM Reference Format:
Florian Hecht, Daniel McCoy, Stephen LaVietes, and F. Sebastian Grassia. 2021. UsdShade in the Pixar Pipeline. 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/10.1145/3450623.3464680

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SIGGRAPH '21 Talks, August 09-13, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8373-8/21/08.
https://doi.org/10.1145/3450623.3464680

1 INTRODUCTION

Pixar has been using procedural shading networks to control the appearance of CG objects since the introduction of *SLIM* in 2003. The procedural nature of this form of shading description has made it easier for the shading work to withstand changes in the underlying geometry and to react more quickly to notes from the director. This has been essential to getting reuse and variation out of the initial creation of the shading network.

1.1 UsdShade Components

UsdShade is built on-top of the core *USD* elements of layers that contain definitions and overrides of prims, which in turn are made up of attributes and connections. *UsdShade* defines three prim types:

- **UsdShadeShader**, which represents an instance of a shading node that can be directly created in a renderer. It has inputs for constant values and incoming signals as well as outputs signals.
- **UsdShadeNodeGraph**, which represents an encapsulated group of nodes. It can have both *UsdShadeShader* and *UsdShadeNodeGraph* nodes as children and has its own interface of inputs and outputs.
- **UsdShadeMaterial**, which represents a complete material network that can be bound to geometry. It is a *UsdShadeNodeGraph* itself and can have a (public) interface with inputs that control the high-level functionality of the network. It defines the end-points that will represent the complete *BSDF* or *displacement* computation.

1.2 USD Shading Layers

The shading work at Pixar is organized in asset, set, sequence and shot components. An individual asset is made up of multiple *USD* layers (see Fig. 1b), which includes a layer for the geometry of the object and multiple layers for shading - one for each application generating the shading.

The order of these layers is fixed in our pipeline. We have a base layer, which is created by hand or script, that imports shading from another asset. We have a layer for simple shading from *Maya*, then a layer for shading data from *Flow*, our main in-house shading application, and lastly a shading layer from *Katana*.

Sets are aggregating assets themselves and have a shading *USD* layer, which is used to override, modify or replace per-asset shading.

Sequence and shot overrides are currently done in *Katana* and not stored in USD, but we have clear plans to express these operations in *USD* as well.

1.3 Shading Variants

We use the powerful variation mechanism in *USD*. Many of our assets have shading variation built in, which can be selected when making a set or changes in a shot. These shading variants can express different color choices, materials or environmental effects.

We always have a default shading variant and all other variants modified versions of the default. That way we don't start from scratch for each variant. In a variant we can tweak and override any value, add nodes or connections, and change material bindings.

2 SHARING & REUSE

Creating high-quality shading is expensive and so we try to reuse as much of it as possible. The procedural shading approach lends itself to that, but we can take advantage of *USD* composition features to make that even more flexible and powerful.

We mentioned how we can inherit shading from an asset archetype, e.g. every clothing item imports the shading of a base garment. Similarly we base the shading variants on the initial default variant.

2.1 Material Libraries

Artists on each film create libraries of materials that are designed for reuse. These materials are stored in *USD* files and referenced, in the *USD* sense, in any asset. This reference is live and if the library is improved, so will be the shading on any asset that uses it.

Users can modify the referenced material with overrides in the asset to any values as well as adding nodes and connections. The reference is still live and updates to the library will be combined with the local adjustments. This can create issues, so we provide an option to localize any referenced material.

2.2 Specialized Materials

USD has a composition arc designed for shading that was inspired by child materials in *Katana*. Instead of a child material B *referencing* a parent material A, B *specializes* A. At first this behaves identical to a reference and B is the same as material A, but B can have overrides. This becomes important when we adjust the shading of A in a downstream context, like a set or a shot. If B *specializes* A, these changes will affect B, unless overridden.

2.3 Shared Groups

UsdShadeNodeGraph can be used to manage node graph complexity by encapsulating subnetworks into containers.

Since they are regular *USD* prims, we can reference shared *NodeGraph* prims. These references are live, meaning it is possible to do shading work in one place and reuse it in many networks.

For example, we can have a procedural dirt module and reference it in multiple materials on a car (car paint, chrome, glass, etc.) such that all get a coherent dirt pattern (see Fig. 1c). Any changes to the shared dirt group will be propagated to all materials instantly.

3 SHADER MAINTENANCE

Shading systems in renderers are constantly evolving and with it are the associated best practices. Keeping up with these changes to get optimal rendering performance can be a lot of work. We present two techniques that have worked well for us.

3.1 BSDF Stand-in & Lighting UI

In a shading network, the terminal nodes, like the *BSDF* and *displacement* closures, are tied intricately to the renderer. If these nodes are explicitly encoded in each shading network, deploying new versions of these nodes becomes a daunting task.

To handle this better, we've introduced a synthetic terminal node which we call the **Network Material StandIn**. We transform it into the *PxrSurface* and *PxrDisplacement* nodes just before submitting to Renderman. together with all the bump mapping nodes. Since this transformation is done in code, we can change the processing to adapt to changes in the renderer and use it for optimization purposes.

3.2 Shader Registry & Version Upgrades

USD ships with the **Sdr** (Shader Definition Registry) library, which discovers and parses all available shaders (*PRMan* C++, *OSL*, *MaterialX*, etc.). On top of this system, we track the status of each shader. We mark shaders as hidden when in beta or as obsolete if we have an improved version, thus hiding them from the artist's UI.

When we mark a shader as obsolete, we can associate it with the shader that should be used instead and also a snippet of *Python* code, that can upgrade the shader. These upgrade scripts can be run in bulk on whole networks in our shading tools to make it easy to update assets to the latest versions.

4 CONCLUSIONS

UsdShade was designed to enable all kinds of ways to author, share and reuse shading. We take full advantage of that and are finding new ways to use *USD* composition.

This model has been so successful that we're using it in other contexts like a procedural vegetation system ([Dixon et al. 2020]) and we're working on using it for light filter networks in *UsdLux*.

REFERENCES

Dave Dixon, Matt Johnson, Andy Whittock, Peter Roe, Jamie Hecker, and Matt Kuruc. 2020. *Procedural Geometry with Open Shading Language on Pixar's Onward and Soul.* Association for Computing Machinery, New York, NY, USA.