ASH - A Case For Layered Shading

Luke Emrose
Animal Logic
Sydney, NSW, Australia
luke.emrose@al.com.au

Curtis Black
Animal Logic
Sydney, NSW, Australia
curtis.black@al.com.au

Emanuel Schrade
Animal Logic
Sydney, NSW, Australia
emanuel.schrade@al.com.au

Figure 1: Animal Logic's USD ALab is an example of a typical production scene using ASH as a layered shading solution. Our custom UI enables artists to edit the material layers of scene objects. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

ABSTRACT

For the past 8 years, Animal Logic has been using its custom **A**nimal Logic **SH**ading System (ASH) material definition and rendering technology for all film projects within our proprietary pathtracer *Glimpse*. We compare existing solutions for material binding and layering from MaterialX, PRMan, USD/USDShade, MDL and more, and show how our own system provides desirable features and solutions absent from other shading solutions and material binding/definition specifications. We propose that existing Open Source projects adopt support for true layered binding, shading and hierarchical assignment, and further propose such solutions provide controllable ordering to allow these layering mechanisms to adequately handle typical production scenarios and requirements. We provide and discuss production examples and further areas for research.

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

ACM ISBN 978-1-4503-9418-5/22/08…$15.00
https://doi.org/10.1145/3543664.3543675

CCS CONCEPTS

• **General and reference** → **Surveys and overviews**; **Reference works**; **General conference proceedings**; • **Computing methodologies** → *Reflectance modeling*.

KEYWORDS

Survey, Shader Binding, Shading, Material Definitions, Layered Materials

ACM Reference Format:
Luke Emrose, Curtis Black, and Emanuel Schrade. 2022. ASH - A Case For Layered Shading. In *The Digital Production Symposium (DigiPro '22), August 7, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3543664.3543675

1 INTRODUCTION

In 2013 Animal Logic developed *Glimpse*, a proprietary pathtracer, initially for *Walking with Dinosaurs: The Movie*, as a lighting preview tool and then for *The LEGO Movie* as a fully-fledged production renderer. This provided an opportunity for Animal Logic to rethink many aspects of rendering, such as the entire concept of material binding and layering, to include lessons learned from decades of prior experience. The resulting system, **A**nimal Logic **SH**ading System (ASH), was designed over months of meetings with a modest team of artists/technicians and developers and first used on the film *Allegiant*. Designed to address common production issues, such as layering of materials, complexity management and assisting in

modular material construction, it is still in use to this today on our entire slate of projects.

1.1 Production Requirements

ASH was designed with the following high-level production requirements in mind:

- Simple and flexible layering of multiple materials.
- Re-use of material components/functionality across multiple primitives, without the need to modify materials themselves.
- Hierarchical assignment/propagation of individual layers for material binding.
- Material layering complexity should not negatively affect rendering performance.

1.2 Technical Requirements

Our software engineering team established the following extended technical requirements which, together, fully address the production requirements:

- Hierarchical assignment of material layers to primitives.
- Automatic layering using an array of materials that can be assigned to primitives using an ordered and named binding mechanism we refer to as a *slot*.
- Clear separation between patterns and materials, allowing the two to be worked on independently and in parallel. This leads to improved turn-around times for asset look-development.
- An ability to connect shaders together with automatic binding of parameters by name-matching. Shaders can be connected either within materials using re-usable subgraphs, or within the binding system itself using a mechanism we call *material overrides*. This allows a slot to act as an *extension of functionality* to an existing slot, all supported directly within the binding mechanism itself.
- Allow users to add complexity to shaders without the need to modify the shaders themselves. An example is adding a layer of dust across an entire scene. In ASH this is trivially achieved by assigning a dust slot to the root of the scene and utilizing slot ordering to ensure it sits at the top of each composed material. Due to the one-to-one binding of material systems we had used in the past, this required tedious manual editing of all shaders within a scene; pasting the same dust node network within each material manually, or with error-prone automation scripts.
- Artist familiarity with Photoshop layering (which is, in essence, an alpha layering system with explicit ordering) made replicating the essence of this workflow appealing.
- Complex shader layering must not result in prohibitively expensive render times.
- Avoid the visual artifacts and lack of physical-basis for parameter-blending with *uber shaders*, and allow proper handling of metallic surfaces, dielectric surfaces and glossy surfaces all within a single flexible framework.
- Provide a rich set of material responses with complex vertical layering built into them directly.

2 BACKGROUND

The two major concepts we discuss in the context of our work are *material binding*, the connection between a renderable primitive and surface material, and *material layering*, the combination of BSDFs / lobes to form complex materials.

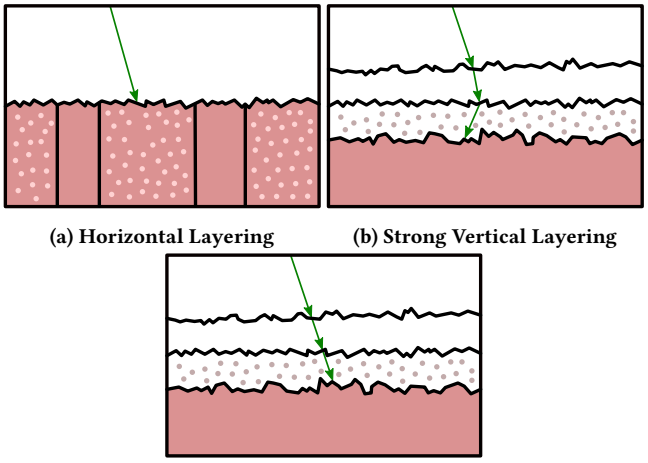
2.1 Material Binding

Material binding, also often referred to as material assignment, is in the majority of cases a one-to-one assignment of materials to primitives. For our binding system we drew much inspiration from the co-shader mechanism in Pixar's PRMan, which allows multiple-material bindings to a single render primitive. ASH supports a similar many-to-one binding behavior and introduces a structured approach to order resolution. At the time of writing, Pixar's Universal Scene Description (USD), the defacto VFX standard for scene description storage and manipulation, does not directly support the assignment of multiple materials to a single primitive (though we have managed to support this via custom schemas, more on this in: Appendix B). We hope that this document might inspire other scene-description implementations to support multi-material assignments to renderable primitives, or at least expose a mechanism to allow such assignments to be supported as a first-class concept.

2.2 Material Layering

Material layering, with reference to the terminology defined in: [Harrysson et al. 2021], describes the combination of lobes/BSDFs with horizontal (linear interpolation) and/or vertical layering (optical layering, such as [de Dinechin and Belcour 2022]). Layering mechanisms are typically implemented within material graphs themselves via node connections (see: [Pixar Animation Studios 2021]), and hence the binding of materials is not usually capable of defining any form of layering or extended functionality itself. One of the core ideas behind the design of ASH was to bring these different concepts together in a flexible and usage-driven way. We extend the established terminology slightly:

- Horizontal layering (see Figure 2a) – a shader equivalent of "Photoshop layering". Refers to the fact that in the real world, materials are often mixtures of different fundamental substances. Typical examples are paint sitting atop cement, surface wear and tear revealing other underlying materials, or decals sitting on top of other materials. This can be achieved by evaluating multiple materials and blending them using alpha-layering, or via stochastic evaluation of layers, which has a very direct physical meaning, since in the real world, each photon can only hit one type of material, with blending arising as a macro effect. In Open Shading Language (OSL) [Gritz et al. 2010], this is typically implemented as a linear interpolation of closures, but in Glimpse, we perform stochastic evaluation.
- Vertical layering (see Figure 2c and Figure 2b) – also sometimes referred to as "optical layering", represents dielectric materials of different densities as multiple coating layers. Typical examples are coated paints, intricate multi-layered glass work, and iridescence. The added complication of vertical layering is that light can be reflected and refracted by each layer transition, which needs to be tracked and handled



(a) Horizontal Layering (b) Strong Vertical Layering

(c) Weak Vertical Layering

Figure 2: Visualization of different layering techniques. A ray intersecting a *horizontally layered* material is affected by exactly one layer, depending on the weights of the layers at a given surface point (a). *Vertical layering* means, a ray may be affected by several (or all) layers of the material by reflecting or transmitting through interfaces between layers. We refer to systems that support physically-based light transport at interfaces and within layers as *strong vertical layering* (b). If rays, for example, pass through layers without scattering, we refer to this as *weak vertical layering* (c).

by the system supporting it. Efficient rendering of optically layered materials is not straightforward and has been of interest in the research community recently [Belcour 2018; Jakob et al. 2014; Weidlich and Wilkie 2007; Weier and Belcour 2020]. These papers focus on the light transport through vertically layered materials whereas we focus on the definition of layered materials and their properties. We further separate vertical layering into the following two categories:

- Weak vertical layering (see Figure 2c) – supports multiple vertically-stacked material layers, but rays traced through the layers do to necessarily support refraction, multiple-scattering between the layers, or physically-accurate material responses. Such a layering system may for example only tint the appearance of a material based on layers.
- Strong vertical layering (see Figure 2b) – supports refraction, multiple scattering between layers, and potentially other physical concepts such as anisotropy and layer roughness. This type of vertical layering can be considered physically-based.

3 MATERIAL BINDING PREVIOUS WORK

We review previous work on material binding mechanisms, since any suitably open programmable shading system is implicitly capable of supporting horizontal or vertical layering via node connections and material code if desired. Please note that this section is quite short, because we were unable to find many previous examples of innovation over the standard one-to-one material binding.

It is relevant to note that encapsulating layering mechanisms in one-to-one binding systems forces the complexity to be within each material itself, or hidden within scene-description data such as attributes or primitive variables. This obfuscated complexity can be difficult to manage for users. All systems we have worked with in the past have suffered from this same issue.

3.1 Co-shaders

To the authors' knowledge, released around 2007 in Renderman 13.5 [Pixar 2022a], co-shaders were the first introduction of multiple-material bindings to a single render primitive also supporting hierarchical assignment. Despite deprecation in newer PRMan versions, the co-shader mechanism is a powerful concept, allowing the user to partition material functionality into small, well-defined co-shader *modules*, and refer to them freely from an underlying material which orchestrates some chosen aggregate functionality. Allowing these assignments to be hierarchical further aids the simplicity of the scene description by allowing users to "push" common functionality within the scene "up" the hierarchy, effectively de-duplicating what would otherwise require redundant per-primitive assignments.

ASH's binding mechanism could be partially mimicked using co-shaders, which we demonstrate in the example rib-file in Listing 2 in the Appendix. RIB and RSL compliant renderers which could replicate the given example include (non-exhaustive list): 3Delight [Illumination Research Pte Ltd. 2021], AIR [Iverson 2014], Aqsis [The Aqsis Team 2015], and Pixie [Arikan 2005]. Rendering this example would produce the co-shader ordering listed in the comments of Listing 2 for each primitive. A strict ordering, let's say alphanumeric for the sake of argument, would however require either runtime sorting, which adds unacceptable performance overhead to shader evaluation, or using Scene Filters.

3.2 Scene Filters

Scene description filters are user-programmable modules that take as input a section of scene, and return it with additions, modifications, or deletions. As such, they provide numerous possibilities for implementing material binding mechanisms. An immediately-recognizable scene description filter is the PRMan Ri Filter which allows for modifying the scene through a C or Python interface. Sorting slots through a scene filter would be possible in a custom implementation of a scene filter but adds restrictions regarding sharing of scenes across different studios and renderers.

We solve the problem of slot ordering without introducing an additional render time overhead by sorting the slots during the scene traversal phase of the renderer loading a scene. Making this mechanism a first class concept within open-source scene description formats is desirable for cross-renderer compatibility.

4 MATERIAL LAYERING PREVIOUS WORK

In comparison to the limited existing work on material binding, material layering has received quite a lot of work and attention from the graphics research community. Here we review the approaches to material layering used by existing commercial and open source renderers.

- **Horizontal Layering:** As complex materials can usually be defined by combining hard-coded base materials in a graph structure, there are often *mixing* nodes to describe horizontal layering of materials. Based on blending weights, the materials that are connected to the inputs of the mixing node are combined to form the mixing node's output.
  Mixing nodes are part of all major renderers and material description standards with public documentation, such as Arnold [Autodesk 2022], Cycles [Blender Documentation Team 2022], Katana [Pixar 2022b], Mantra [SideFX 2022], MaterialX [Harrysson et al. 2021], MDL [NVIDIA Corporation 2019], Mitsuba [Jakob 2014], PBRT [Pharr et al. 2022], and PRMan [Pixar 2022b]. Regarding USD, USDShade [Pixar Animation Studios 2017, 2022] itself defines neither vertical or horizontal layering but leaves these implementation choices up to the user. There have been usd-interest forum discussions about layered materials [1][2][3] but none are yet concretely implemented by USDShade.
  Providing horizontal layering via mixing nodes implies a material must encode layering via node connections. This means that modifications to the layering cannot be managed freely at the scene level. In most cases this can be mitigated by careful interface parameterization, but changing a layer from one node type to another, cannot be handled without manually rewiring the graph itself.
- **Vertical Layering:** Some renderers allow for vertical layering in predefined materials, e.g. in specific materials such as the *car paint* material in Katana, or the *thin-film* material node in Arnold which can be applied to other materials. Such predefined nodes for vertical layering are also available in Cycles, Mantra, Mitsuba, PBRT, and PRMan, as well as in ASH. MDL and MaterialX allow for combining materials by vertical layering through mixing nodes. This allows for great flexibility in defining optical layered materials. While MDL currently supports weak vertical layering, MaterialX appears to support both weak and strong vertical layering.

Table 1 shows the material layering capabilities of different renderers. The (W) naming refers to weak vertical layering, all other vertical layering is assumed to be strong. Glimpse, with ASH material binding and layering is the only system we know of that currently supports horizontal material layering at the binding level.

Table 1: Comparison of Material Layering.

Renderer	Horizontal Layering	Vertical Layering
Arnold	Mixing Shader Node	Explicit Nodes
Cycles	Mixing Shader Node	Explicit Nodes
Katana	Mixing Shader Node	Explicit Nodes
Mantra	Mixing Shader Node	Explicit Nodes
MaterialX	Mixing Shader Node	Mixing Shader Node
MDL	Mixing Shader Node	Mixing Shader Node (W)
Mitsuba	Mixing Shader Node	Explicit Nodes
Octane	Mixing Shader Node	Mixing Shader Node
PBRT v3	Mixing Shader Node	Explicit Nodes
PBRT v4	Mixing Shader Node	Explicit Nodes
RenderMan	Mixing Shader Node	Explicit Nodes
UsdShade	Implementation Defined	Implementation Defined
VRay	Mixing Shader Node	Mixing Shader Node
Glimpse/ASH	Binding Assignment	Explicit Nodes

5 ASH

At Animal Logic we developed ASH as the shading system integrated into our proprietary production path-tracer Glimpse. It provides an abstraction layer between the scene description and the renderer for material assignments/binding and material graphs. ASH material graphs are processed and turned into OSL code for runtime evaluation. ASH consists of three major components:

- Hierarchical multiple-material per primitive scene composition binding.
- A C++ API for defining shading graphs and connections with special features, based on an OSL runtime JIT.
- Runtime stochastic evaluation of layering, masking, surface shading, and displacement.

These components allow us to efficiently and effectively surface multiple assets using shared material definitions without the need to edit the internal node connections or content of existing material graphs.

Traditionally complex scene modifications such as "adding dust across the entire scene" or "changing the look of all the metal for each primitive in a particular asset" are comparatively trivial in ASH due to the way it has been designed. These are precisely the type of production use-cases it was created to address.

Animal Logic has long focussed on photorealistic and physically-based rendering. As a result, we have moved away from uber-shader based approaches, due to the unnatural blending look obtained from parameter blending and instead prefer correct metallic specular responses and physical fresnel terms. As such, we require a material layering system that supports truly physically-based blending of parameters. We achieve this using stochastic blending, which represents the physical property of material mixtures represented as the probabilistic presence of multiple mixed materials. Or, in short, in the real world a ray or photon gets reflected or refracted by a single material at a time. Horizontal layering is therefore an aggregate result of multiple surfaces effecting the light propagation. We model this directly.

5.1 Overview

We first introduce the terminology and a few main ideas overarching ASH before explaining them in more detail:

- Multiple hierarchical material assignment, ASH's fundamental binding concept, allows for all named material assignments for a primitive (from now on referred to as *slots*) to be combined through horizontal layering. Slots are propagated hierarchically and sorted alphanumerically within each primitive.
- Materials (or *substances* in ASH) are represented by a node graph using OSL. Each substance can contain up to one of each of the following outputs (which we call *products*): *surface* (e.g. the BSDF), *displacement*, *normal* (for normal

[1] https://groups.google.com/g/usd-interest/c/EGJMkTbTnDE/m/5JizUHQuAwAJ
[2] https://groups.google.com/g/usd-interest/c/_hEPO2Z3nzI/m/EzlaiWLFAwAJ
[3] https://groups.google.com/g/usd-interest/c/-pzQUZQP6p0/m/HKGnxQ5UBwAJ

mapping) and *mask* (for the ability to set the substance's transparency).

- *Overriding* of slots is implemented through the specific naming convention `X::Y`, which means that we connect all matching output attributes of `Y` to the corresponding input attributes of `X` with the same name. This allows for overriding functionality through automatic rewiring of the substance graph by relying on naming conventions.
- Substance slots defined at a higher level in the hierarchy can be *replaced* at a lower level by re-defining the same slot. Similarly, assigning an empty slot (no associated substance graph) removes any previous assignment.
- Displacement and bump mapping are evaluated and combined from the bottom to the top layer. Normal mapping for each slot is evaluated directly following the displacement and bump mapping.

5.2 Hierarchical Assignment

Each substance is assigned to a named slot as part of the material assignment process. Assignments can be made to any primitive path in a scene, and all are propagated hierarchically (i.e. assignment even to groups and transforms is supported where appropriate). When a renderable primitive is found in the scene, its final singular material is composed as the union of all hierarchical substance assignments at this point from all ancestors.

Slot names are sorted to define the layer ordering. In our implementation this sorting is defined alphanumerically. Theoretically other sorting methods could be used, but alphanumeric sorting has worked well for us.

The concept of combining multiple substances via slot assignments is where the utility of this system is clearest. Scene subsections which contain visually similar renderable elements can all be surfaced with a single assignment. Further detail within each scene subsection can be refined with assignments at deeper levels in the form of layering or replacements.

A slot replacement occurs when a slot assignment is made to part of the scene which has the same slot name as a previous assignment higher up hierarchically. Replacement can also be used to deactivate higher slot assignments, this occurs when a slot is assigned an empty substance. A new layer occurs when a slot assignment is made anywhere above the current renderable primitive with a new unique slot name.

Listing 1 demonstrates these hierarchical assignment concepts using a simplified pseudo-USD scene.

The resulting composed materials for each object are:

- ObjectA: SubstanceA
- ObjectB: SubstanceB on top of SubstanceA
- ObjectC: SubstanceB on top of SubstanceC
- ObjectD: SubstanceB

5.3 Stochastic Evaluation

To reduce the cost of evaluating multiple substance layers, Glimpse uses stochastic evaluation based on mask product outputs. For each integrator sample, mask products for all slots are evaluated to produce per-layer opacities. These opacities are considered from top-to-bottom to compute per-layer weights. If the sum of these

Listing 1: "Pseudo-USD Hierarchical Assignment Example."

```
def "ObjectA"
{
  rel slot:000 = </SubstanceA>
  def "ObjectB"
  {
    // applies a new layer on top of '000'
    rel slot:010 = </SubstanceB>
    def "ObjectC"
    {
      // replaces the previous slot '000'
      rel slot:000 = </SubstanceC>
    }
    def "ObjectD"
    {
      rel slot:000 = null // removes the previous slot '000'
    }
  }
}
```

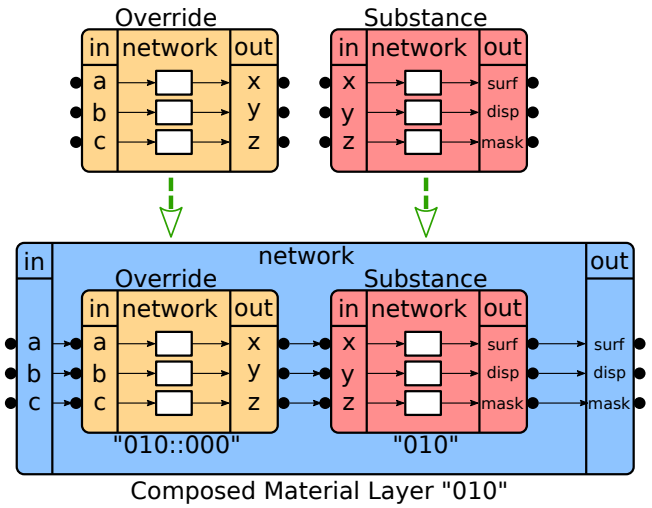


Figure 3: Overrides Example. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

weights is less than one, the remaining weight is considered to be the primitive transparency.

For example, consider the case of ObjectB in the example in Listing 1, where SubstanceB is on top of SubstanceA. Let the opacities of SubstanceA and SubstanceB be m_A and m_B, respectively. Then the weights will be:

$$w_B = m_B, \tag{1}$$

$$w_A = m_A \cdot (1 - m_B), \tag{2}$$

$$w_T = 1 - w_A - w_B \tag{3}$$

where w_T is the implicit weight of the object being fully transparent. During evaluation of materials, one of the substances or the transparent case will be selected stochastically, proportional to the weights w_A, w_B, and w_T.

5.4 Overrides

Overrides are substance graphs that provide output attributes to be consumed by the substances they are overriding. You can think

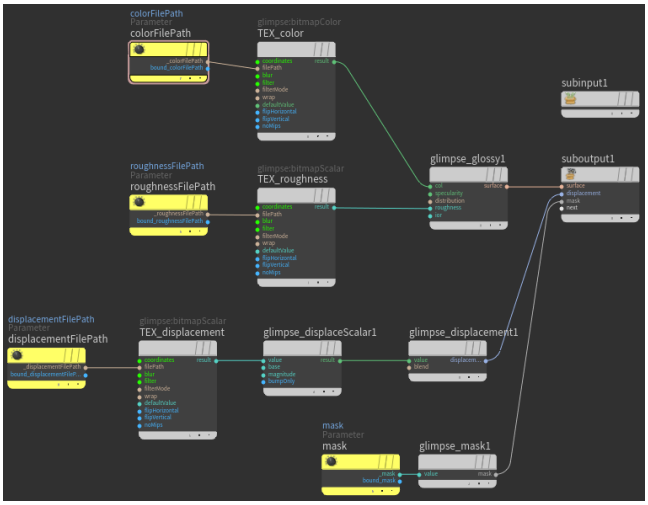


Figure 4: ASH Graph Houdini Integration. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

of them as providing patterns with output parameters that override existing input parameters for another graph. We illustrate an override example in Figure 3. Say a substance is assigned to layer slot `"010"`. Assume this substance contains the input parameters "x", "y" and "z". These parameters can be set directly as constant values, or an override with output parameters "x", "y" and "z" could be assigned to slot override `"010::000"`. When the final material layer `"010"` is composed by ASH, all overrides assigned to `"010"` (in this example, `"010::000"`) attempt to connect their outputs to any matching inputs of the `"010"` substance. The connections are automatically established if matching parameter names are found. The highest level override output parameter will "win" if multiple overrides provide the same named output parameter. At a basic level this allows substance inputs to be defined in one process, then driven by overrides containing complex graphs with textures, procedurals or some other complicated process in another. Naming conventions are established to ensure that the override process behaves as expected.

5.5 Custom Channels - AOVs

Custom Arbitrary Output Variables (AOVs) can be defined in ASH by simply placing a `customChannel` node within a substance graph and either setting it with a constant input value, or feeding it with a computed value. This node is detected as a custom channel and exposed to Glimpse automatically. Custom channel nodes exist for each output data type supported by Glimpse: `customChannelColor`, `customChannelInteger`, etc. Substances can compute an arbitrary number of custom output channels, and substance overrides can add them to existing substances at any time. One special feature of custom channel nodes is that even though they do not provide an output connection, they are still retained by the OSL shader compiler. Typically nodes that do not provide connected outputs are truncated from the graph.

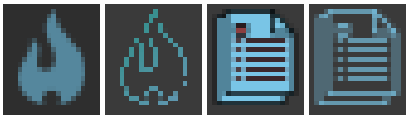


Figure 5: Houdini ASH Slot Assignment Icons. Slot assigned: (a). Slot inherited: (b). Override assigned: (c). Override inherited: (d). Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

5.6 Houdini User Interface

Surfacing and look development of USD stages is performed in the SideFX Houdini LOPs UI. Standard Houdini material library nodes provide the containers that hold USDShade materials. Subnets within these material library nodes contain the shaders themselves. The ash substance graphs are edited within the VOP context. ASH automatically generates VOP nodes from the internal Glimpse node definitions, and these node types are filtered within the Glimpse ASH VOP UI, in order to only show valid substance nodes. The interface for a typical ASH graph is shown in: Figure 4. Note that all the products provided by this graph are connected to a Houdini VOP suboutput node to collect them. In this particular example, the surface product is `glimpse_glossy1`, the displacement product is `glimpse_displacement1`, there is no normal product and the mask product is `glimpse_mask1`.

Hierarchical slot assignments are handled by a dedicated UI that provides a novel visual reference to help our artists understand their application within a scene. The hierarchy of the USD stage is shown, along with icons at each primitive to represent the aggregation of slots at that point. The icons in Figure 5 provide a visual representation of the hierarchical slot assignments to the primitives in the scene. The icons represent a new slot assignment (Figure 5a), an inherited slot assignment (Figure 5b), a new override assignment (Figure 5c), and an inherited override assignment (Figure 5d).

A production example of the UI is shown in Figure 6. There are 3 columns for the *Scene Graph Path*, the *Slot* name, and the *Material Prim Path* for the scene path to the substance or override assigned at that point. Slots are always shown, at each primitive, sorted from the base layer at the top, to the top layer at the bottom. To figure out the location of a slot assignment or override, the user just needs to look at which icon is bright. A bright icon is an application point, and a dark icon indicates inheritance from a parent level. Reading a row at the primitive level from left to right, you can also see the slot and override assignments from the lowest to the highest layer. This means that even when the hierarchy is collapsed, you can still tell, at a glance, what the substance assignments are. It is then possible to further explore the detail by expanding the hierarchy and looking at which substance is assigned to which slot. We have used multiple variations of this general UI across all of our Digital Content Creation (DCC) tools and they work well to represent the ASH hierarchical assignments at a glance.

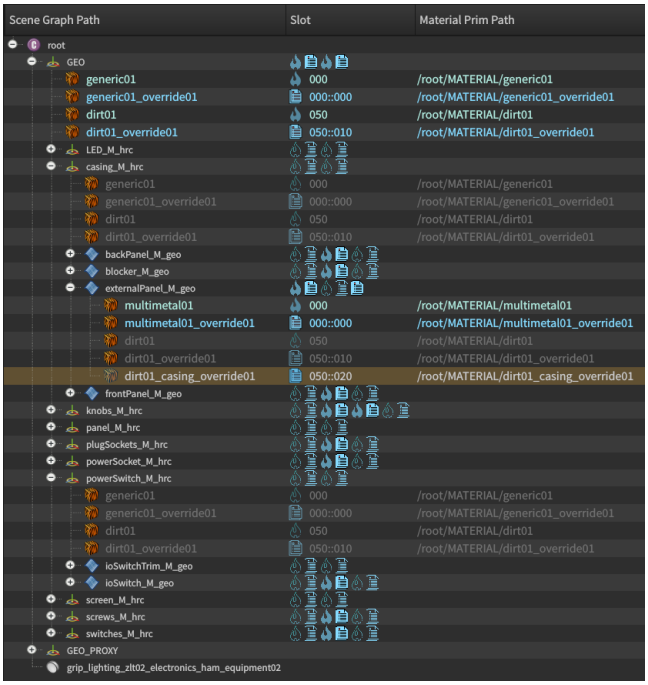


Figure 6: Houdini Slot Assignment UI. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

Figure 7: Production Example Overview: viewport preview and final render. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

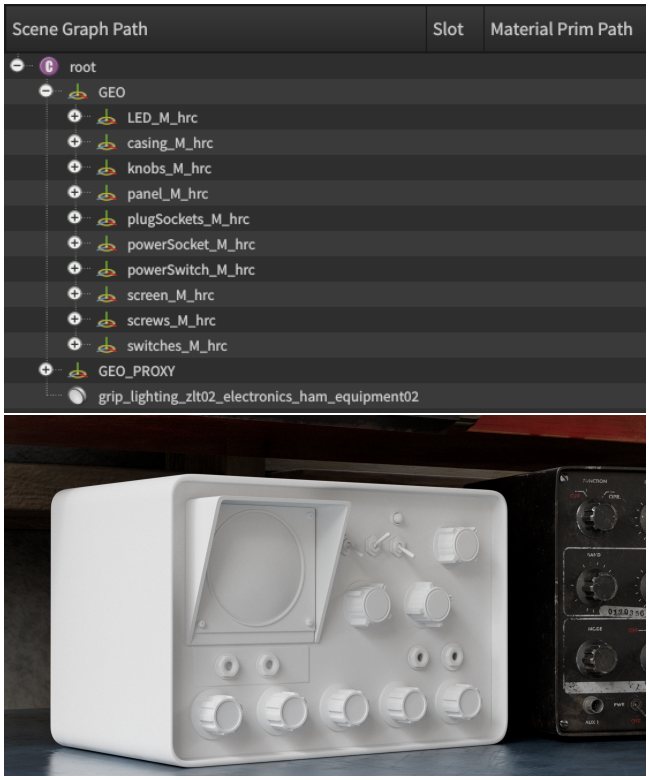


Figure 8: Default Substance. When no slot assignments are made in the DCC, assets render with a default white diffuse. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

6 PRODUCTION EXAMPLE

We now show how the concepts we have introduced can be used when surfacing a production asset. We choose an oscilloscope asset from the Animal Logic ALab. The Houdini viewport view is compared to the final Glimpse render in Figure 7 as a starting point.

6.1 Default Substance

In a fresh scene with no assignments (Figure 8) all geometry renders with a default white diffuse product.

6.2 Hierarchical Layering

Hierarchical layering (Figure 9) is used to set up the *base* substance layer slot of the asset. At the root of the asset, `generic01` (a typical glossy shader) is applied as a base layer at slot `"000"`. This produces a render with a darker grey tint, since our default glossy product is a mid-grey. Note also that the assignment UI shows the inherited assignments down the scene hierarchy.

6.3 Substance Overrides

Substance input values can be overridden by setting values on substance overrides. In Figure 10, `generic01_override01` provides colour and roughness maps. A distinct advantage to using ASH overrides is that the base library substance `generic01` can be updated at any time, and the override will remain the same. This allows for

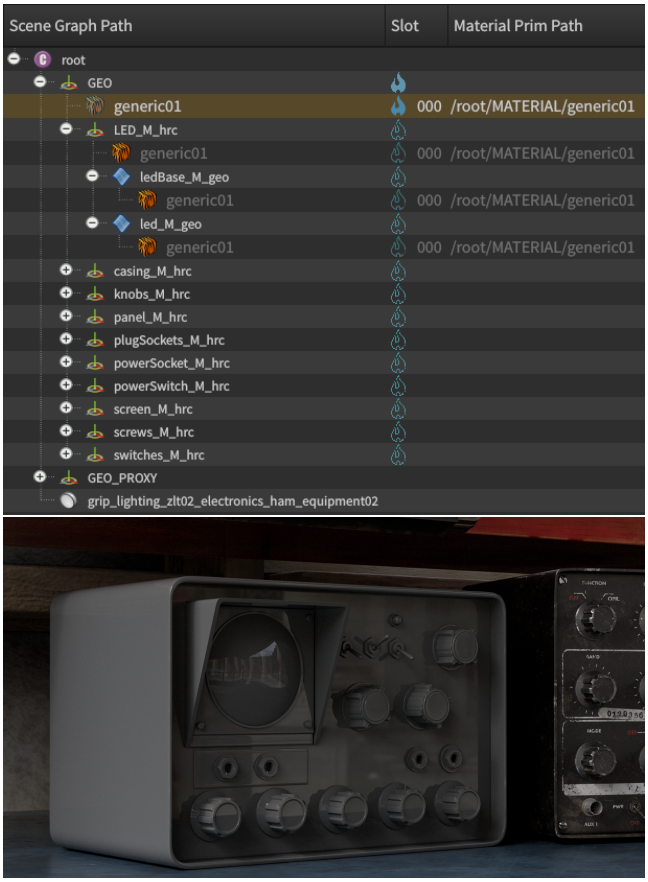


Figure 9: Hierarchical Layering. When a slot is assigned at the root of an asset, in this case a glossy substance, it is hierarchically applied to all children. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

a division of work between the team creating the base substances, and the team driving their input values. They can work in parallel, since the substance itself can be partitioned into these two distinct pieces. Working within a single substance graph would not allow for such a clear separation or ability to be able to parallelize the work.

A second `metal` substance `metal_stainless01` and substance override `metal_stainless01_override` are added lower in the hierarchy in Figure 11 and apply only to the child primitives. In this case note the metal elements behind the knobs are changed by these overrides compared to Figure 10.

6.4 Hierarchical Replacement

For primitives which require a different substance to `generic_01`, a substance can be assigned to slot `"000"` to replace the original `generic01`. This is shown in Figure 12 as the oscilloscope case changing look from Figure 11. It now uses a metal product `metal_stainless01` rather than a glossy product.

Figure 10: Substance Overrides. A substance override assigned at the asset root provides the basic color and roughness map pattern to be consumed by the previously applied glossy substance. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

6.5 Override Sharing

Because overrides are substance graphs themselves, they can be re-used and assigned to multiple slots on multiple primitives. This makes it simple to quickly change the colour of many primitives at the same time. This is seen in Figure 13 as the knobs changing to a red colour. The override *plastic_opaque01_red* overrides only the red colour, but the underlying substance itself remains unchanged.

Dirt and dust substances can be layered on top of base layers and have built-in masking so they automatically only appear in occluded areas or on top of surfaces. This is visible in Figure 14 as dirt, wear and tear on the various surfaces of the oscilloscope compared to Figure 13 via the `dirt01` substance and `dirt01_override01` override.

6.6 Override Layering

Because overrides are also assigned with sorted slots, they can be layered just like substances. For example, in Figure 15 the inherited dirt colour was changed at a different point in the hierarchy while retaining the original dirt mask.

The slot arrangement for this is:

(1) `"050" - dirt01` (inherited)
(2) `"050::010" - dirt01_override` (inherited)
(3) `"050::020' - dirt01_casing_override` (object specific)

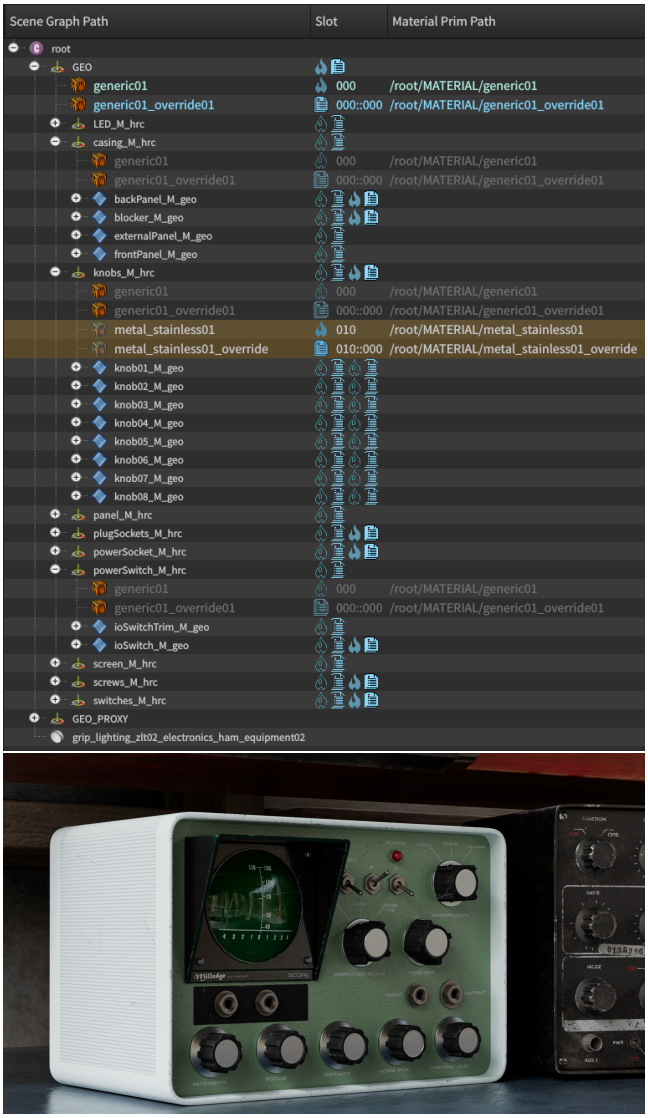


Figure 11: Substance Overrides. A new metal substance (with acompanying override) is assigned to a higher slot lower in the hierarchy, this applies only to the metal elements behind the knobs. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

This is visible in Figure 15 as a lighter colour for the dirt on the oscilloscope outer case compared to Figure 14.

6.7 Slot Removal

Slot assignments can be removed by assigning *empty* slots. For example to remove the inherited `dirt01` substance, we can assign an empty substance to slot `"050"` on the `frontPanel_M_geo` primitive. This is shown in Figure 16 as the dirt substance disappearing on the front panel, whilst being retained elsewhere.

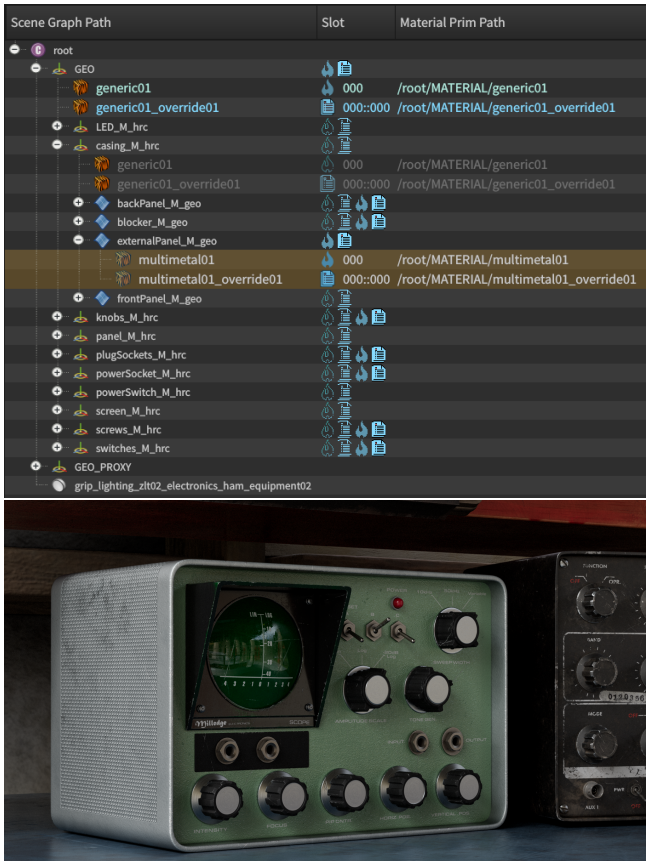


Figure 12: Hierarchical Replacement. The existing glossy substance is replaced with a metal substance on the outer shell by assigning the metal to the same slot name as the glossy, lower in the hierarchy. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

6.8 Adding Custom AOVs in Overrides

AOVs can also be added using slot overrides. As mentioned in subsection 1.2, substance overrides are just standard substance graphs containing nodes that are added to the substance they are targeting, so any valid individual nodes (like AOV nodes) will be added to the main substance; acting as if they had always been present within it. This allows us to add new functionality to an existing shader without ever having to rewire it or change any code. In Figure 17, a custom AOV (`glimpse_customChannelColor1`) has been added to an override to output an AOV for the presence of the dirt substance. Due to the automatic layering of substances, the AOV merely needs to output a value of 1. Glimpse will automatically stochastically sample it based on the probability of sampling the slot layer it is in. The result is an AOV that matches the layering and masking result of the substance containing that AOV. We have found this mechanism extremely useful for simply and quickly generating extra data for compositing passes with minimal effect on production, since no library assets need to be changed or republished.

Figure 13: Override Sharing. A single override can be assigned to many different elements to easily provide patterns (to the red knobs) without changing the substance type (which remains glossy). Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

Figure 14: Override Sharing. A dirt layer (with acompanying override) is layered on top of all elements in the asset. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

6.9 Decals and Projections

Decals can be implemented using additional substances with masks added as extra slot layers. In Figure 18, the sticker on the side has been added as an extra substance slot layer.

The decal sticker is a coated paper material with a projected mask override. It is layered above the base slot but below the dirt, paint, and dust.

6.10 Layered Displacement/Bump/Normal

Substance displacement/bump/normal products can be set to *add*, *mix* or *none* mode. The add and mix modes are limited in extent by the substance mask. The difference is that in mix mode the displacement is additionally blended from top to bottom in the same way as surface product substance layering (i.e. higher slots mask the lower slots). The none mode ignores all masking and layering and applies everywhere. The add and mix options are shown in Figure 19 as the sticker being either masked by or on top of the underlying material respectively.

7 LIMITATIONS AND FUTURE WORK

Having proven itself over many films, ASH has become a mature staple of Animal Logic's workflows. We now incorporated layered materials into every asset we create, and the flexibility of having individually adjustable layers and overrides provides us with more

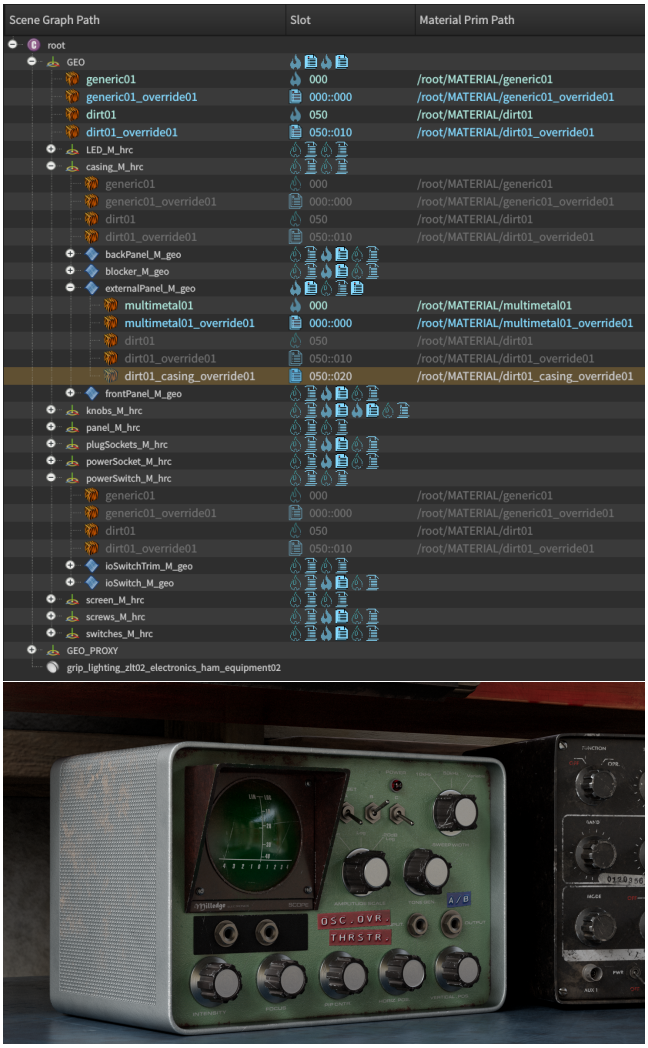


Figure 15: Override Layering. The pattern of the dirt layer on the outer metal is reduced by applying a substance override lower in the hierarchy. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

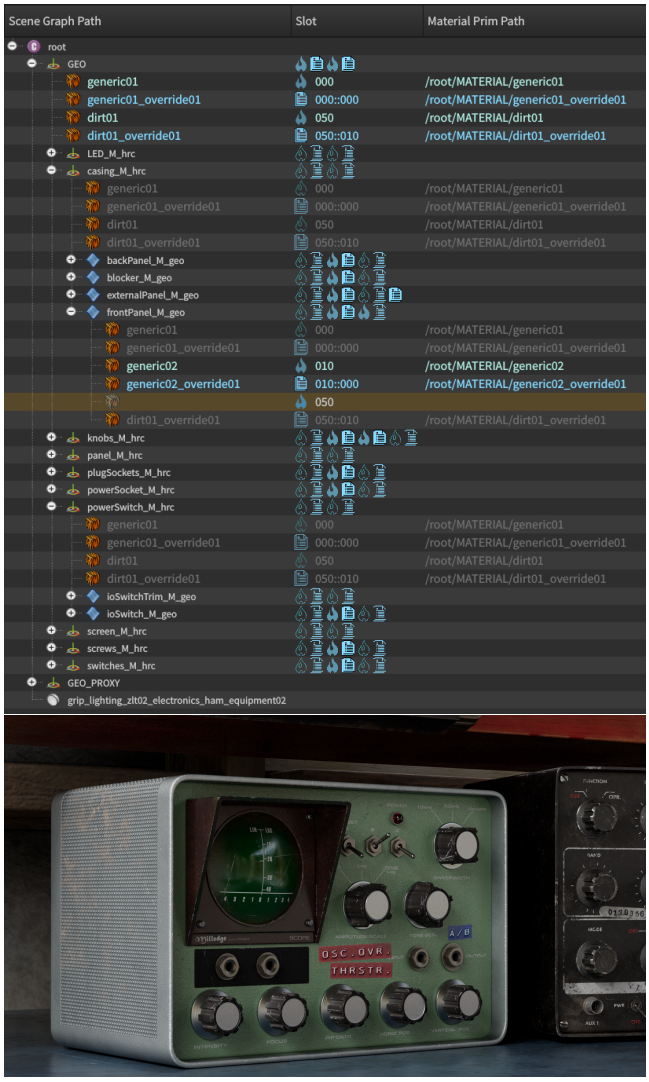


Figure 16: Slot Removal. Dirt is removed from the front panel by assigning an empty substance to the the previous dirt slot name, lower in the hierarchy. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

creative tools, and faster turn-around times in cases where last-minute changes need to be made. Despite this success, there are still many areas the Glimpse team have already identified for future extension/improvement. We also like to keep pushing our tools to deliver results for ever-increasing complexity and realism.

7.1 Layering Depth

The Glimpse ASH implementation currently imposes a limit of 8 substance slots/layers per primitive. This limitation is relatively arbitrary, and following the descriptions in this paper one could choose to support any number of layers.

7.2 Substance Product Customization

We do not support node graph connection-based horizontal mixing and interpolation of surface products like other systems such as MaterialX, Arnold and PRMan. We have found our layering system provides an adequate alternative (with clear advantages in terms of editing materials), but a case could be made to combine these two techniques together for an even more expressive system. Further discussions with other vendors in this area is something we intend to actively pursue.

7.3 Vertical Layering

Vertical layering is currently supported in ASH via hard-coded surface products that incorporate the vertical layers directly into their

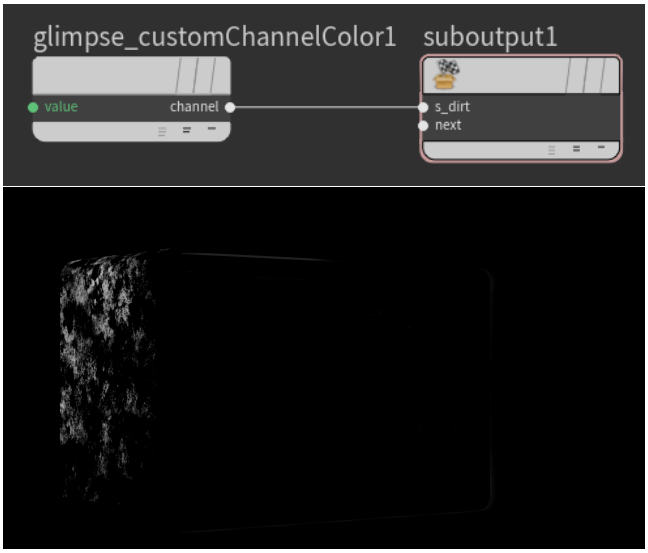


Figure 17: Adding Custom AOVs in Overrides. Substances and overrides can provide custom AOVs, in this case the dirt substance provides a matte for later use. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

Figure 18: Decals and Projections. A new decal substance and override are assigned to a slot in-between the base metal and the top dirt layers. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

Figure 19: Layered Displacement - Add and Mix modes. Displacement, bump, and normal mapping products can optionally blend on top of lower layers. In this case the decal displacement can optionally accumulate on top of the displacement on the lower metal layer. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

definitions. These vertical layers cannot be inherently edited outside of the exposed parameters on the surface products themselves. So far this hasn't shown to be particularly problematic, but some users desire more control and from a code-maintenance perspective, automated handling of vertical layers is appealing and expected to reduce code size and complexity.

We would like to incorporate recent work related to vertical layering, specifically: [Belcour 2018], [Weier and Belcour 2020], [de Dinechin and Belcour 2022], [Guo et al. 2018] and [Randrianandrasana et al. 2021]. An extension to the slot naming could support vertical layering for substances. Given that `"000"` is currently an example of a slot name for a horizontal layer, adding a vertical layer to that slot could be achieved by specifying `"000|010"`. This would add the vertical substance layer specified in slot `"000|010"` to slot `"000"`. If the override naming convention still applied, it would be possible to specify an override for `"000"` using `"000::010"` as before, but additionally the user could add an override to a vertical layer with `"000|010::010"`. The obvious issue here would be user-confusion with these naming conventions, but this could be mitigated by an appropriate UI to hide such complexities. One advantage such a system would provide over graph-based vertical layering systems, such as MaterialX, would be that the materials could be modified in a non-destructive manner, i.e. vertical layers

could be added or removed entirely at the binding level. Additionally, vertical layers could then be re-used across multiple assets, and if sorted in the same way that horizontal layers currently are, their level within the vertical stack could be accurately and reliably controlled.

7.4 Friendly Slot Naming

The alphanumerical naming and sorting used by ASH was chosen to provide a reasonable compromise to the problem of how to sort data consisting of potentially complicated layer names. As a result, at Animal Logic, we established a 3-digit alphanumerical naming convention:

The main slots are named `"010"`, `"020"`, `"030"`, ... , `"0X0"`. This leaves room for `"000"` to be placed at `"the bottom"` of the layer stack at any time, and the stride of 10 allows for further modifications to the stack order such as `"015"` between `"010"` and `"020"` etc. We decided to avoid non-numeric characters for simplicity and the sanity of our artists!

When presenting this to the OSL committee a few quite rightly pointed out that this is reminiscent of goto statement labels in the BASIC language. It is, also, upon first viewing, quite different to existing systems. We recognize this, and hence we'd like to engage in further discussion with the VFX community about what ideas they might have to improve or simplify the naming schemes and strategies we have adopted so far.

7.5 Hierarchy vs Collections

ASH hierarchical assignment relies on having a reasonable scene hierarchy as scaffolding for the slot binding to act upon. Some facilities might prefer to work with flatter object hierarchies. In these cases, extending hierarchical assignment to work with collection-based assignment would make a lot of sense. This would allow a "virtual" hierarchy to be formed from collections, and used to assign slots to multiple objects in a more expressive fashion. We have not yet embraced such an idea in Glimpse, but intend to raise this as a discussion point with other studios to further improve the flexibility of slot-based assignment.

8 CONCLUSION

Material layering and slot binding have allowed Animal Logic to express surfaces in a way that has improved the photorealism and efficiency of our work. The ability to decouple different parts of the pipeline such as base materials and pattern generation and work on them in parallel has improved our productivity compared to our work before ASH. Building assets with hierarchies that encode their structure has also helped us to further embrace the power of hierarchical slot binding, and assisted in the simplicity of our material assignment.

Moving away from one-to-one binding limitations has allowed us to simplify large-scale surfacing modifications in situations that would have previously required republishing thousands of material assets. Avoiding the industry trend of uber-shaders has further provided Animal Logic with a visual style and look more grounded in real-world physics, showcased in Figure 20, free from the sometimes unnatural artifacts of purely blending material parameters to represent layering. We can quite easily combine true metallic specular responses with glossy dielectric responses without the need for non-physical parameters such as "metalness", whilst retaining the power of Photoshop-style alpha layering.

Figure 20: Animal Logic's USD ALab demonstrates the results of layered shading using ASH within a complex environment. Copyright © 2022 Animal Logic Pty Ltd. All Rights Reserved.

Here is a quote from a user and advocate of Glimpse and ASH in production:

> Animal Logic ASH shading tools have allowed artists to quickly layer materials in complex hierarchies without pre-establishing the final shader composition. Glimpse layering mechanisms have allowed us to easily layer multiple weathering effects on complex assemblies or large sets without diving into the individual asset components. This approach, combined with our material referencing scheme, has contributed to a noticeable reduction in turnaround time in the look development department when dealing with complex assets.
>
> *Jean-Pascal leBlanc - Global Asset Supervisor*

We believe that presenting our work, and actively discussing it as we already have with the OSL, MaterialX and Pixar USD Technical Steering Committees will lead to improvements and modifications to our ideas that the entire industry may be able to benefit from. This excites and motivates us to continue improving and refining this work. Our long term goal is for open source scene description formats such as USD to adopt multiple-material per primitive assignments with hierarchical propagation and layer sorting to support the basis upon which binding-based material layering could be implemented. We would love to further extend and refine these ideas to include true vertical-layering support, and clear standards for slot naming, slot sorting and production usage.

If you have any questions, suggestions, or feedback we would love to hear from you and would encourage you to contact us to discuss any aspect of this work.

ACKNOWLEDGMENTS

We would like to thank the Glimpse and ASH teams, past and present: Max Liani, Daniel Heckenberg, Michael Balzer, Steve Agland, Simon Bunker, Jakub Jeziorski, Edoardo Dominici, Andy

Chan, Erik Pekkarinen, Antoine Roille, Matthew Reid, Thomas Caissard and Linas Beresna. A more amazing group of colleagues we cannot imagine. Thank you to the following members of our surfacing team for assistance with preparing the production example images: Camela Cheng, Sabri Badri, Roxanne Joyner, Callum Blackall and Jean Pascal leBlanc. Thank you to Bradley Webster from our Lighting team for assistance with our talk turntables. Thank you to all of the authors of all of the papers we have implemented in Glimpse. It's truly wonderful to get responses about highly technically specific nuanced questions to do with shading and light transport from the smartest people in the world in the field of rendering. Thank you to the ALab team for providing the wonderful assets we used throughout this paper. Thank you to all of our friends and family for supporting our passion, drive and long hours, we are all very grateful. Thank you to Animal Logic for supporting this work.

REFERENCES

Okan Arikan. 2005. Pixie Documentation. http://www.okanarikan.com/project/2005/05/24/Pixie.html

Autodesk. 2022. Arnold Documentation. https://docs.arnoldrenderer.com/display/A5AFMUG/Layer+Shader

Laurent Belcour. 2018. Efficient Rendering of Layered Materials Using an Atomic Decomposition with Statistical Operators. *ACM Trans. Graph.* 37, 4, Article 73 (jul 2018), 15 pages. https://doi.org/10.1145/3197517.3201289

Blender Documentation Team. 2022. Blender 3.2 Manual - Cycles. https://docs.blender.org/manual/en/latest/render/cycles/index.html

Heloise de Dinechin and Laurent Belcour. 2022. Rendering Layered Materials with Diffuse Interfaces. *Proc. ACM Comput. Graph. Interact. Tech.* 5, 1, Article 13 (may 2022), 12 pages. https://doi.org/10.1145/3522620

Larry Gritz, Clifford Stein, Chris Kulla, and Alejandro Conty. 2010. Open Shading Language. In *ACM SIGGRAPH 2010 Talks* (Los Angeles, California) *(SIGGRAPH '10)*. Association for Computing Machinery, New York, NY, USA, Article 33, 1 pages. https://doi.org/10.1145/1837026.1837070

Yu Guo, Miloš Hašan, and Shuang Zhao. 2018. Position-Free Monte Carlo Simulation for Arbitrary Layered BSDFs. *ACM Trans. Graph.* 37, 6, Article 279 (dec 2018), 14 pages. https://doi.org/10.1145/3272127.3275053

Niklas Harrysson, Doug Smythe, and Jonathan Stone. 2021. MaterialX Physically-Based Shading Nodes. https://www.materialx.org/assets/MaterialX.v1.38.PBRSpec.pdf

Illumination Research Pte Ltd. 2021. 3Delight Documentation. https://documentation.3delightcloud.com/display/3DSP/Introduction

Scott Iverson. 2014. AIR User Manual. http://www.sitexgraphics.com/air.pdf

Wenzel Jakob. 2014. Mitsuba Documentation. https://www.mitsuba-renderer.org/releases/current/documentation.pdf

Wenzel Jakob, Eugene d'Eon, Otto Jakob, and Steve Marschner. 2014. A Comprehensive Framework for Rendering Layered Materials. *ACM Trans. Graph.* 33, 4, Article 118 (jul 2014), 14 pages. https://doi.org/10.1145/2601097.2601139

NVIDIA Corporation. 2019. MDL Documentation. https://developer.nvidia.com/designworks/dl/mdl_spec

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2022. PBRT v4 Documentation. https://www.pbrt.org/users-guide-v4

Pixar. 2022a. Renderman 13.5 Release Notes. https://renderman.pixar.com/resources/RenderMan_20/rnotes-13.5.html

Pixar. 2022b. Renderman Documentation. https://renderman.pixar.com/resources/RenderMan_20/rfhLayering.html

Pixar Animation Studios. 2017. Material Assignment in UsdShade with Collections and Purpose. https://graphics.pixar.com/usd/files/MaterialAssignmentinUsdShadewithCollectionsandPurpose.pdf

Pixar Animation Studios. 2021. Renderman 24 MaterialX Lama Documentation. https://rmanwiki.pixar.com/display/REN24/MaterialX+Lama

Pixar Animation Studios. 2022. USD Documentation - USDShade. https://graphics.pixar.com/usd/dev/api/usd_shade_page_front.html

Joël Randrianandrasana, Patrick Callet, and Laurent Lucas. 2021. Transfer Matrix Based Layered Materials Rendering. *ACM Trans. Graph.* 40, 4, Article 177 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459859

SideFX. 2022. Mantra Documentation. https://www.sidefx.com/docs/houdini/shade/layering.html

The Aqsis Team. 2015. Aqsis Documentation. https://www.aqsis.org/documentation/user_manual/index.html

Andrea Weidlich and Alexander Wilkie. 2007. Arbitrarily Layered Micro-Facet Surfaces. In *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia* (Perth, Australia) *(GRAPHITE '07)*. Association for Computing Machinery, New York, NY, USA, 171–178. https://doi.org/10.1145/1321261.1321292

Philippe Weier and Laurent Belcour. 2020. Rendering Layered Materials with Anisotropic Interfaces. *Journal of Computer Graphics Techniques (JCGT)* 9, 2 (jun 2020), 37–57. http://jcgt.org/published/0009/02/03/

A PRMAN RIB SLOT BINDING EXAMPLE USING CO-SHADERS

Listing 2: "RIB Slot Binding Example."

```
Display "test" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 320 240 1

Translate 0 0 3
Rotate 0  1 0 0
Rotate 90 0 1 0
Scale 1 1 -1

WorldBegin
    Shader "testslot" "010"
    Shader "testslot" "020"
    Surface "testbase"
    AttributeBegin
      TransformBegin
          # resulting order: 010, 020, 000
          # ASH ordering would be: 000, 010, 020
          Shader "testslot" "000"
          Translate 0 0 -0.5
          Sphere 0.15 -0.15 0.15 360
      TransformEnd
    AttributeEnd
    AttributeBegin
      TransformBegin
          # resulting order: 010, 020, 001 ordering
          # ASH ordering would be: 001, 010, 020
          Shader "testslot" "001"
          Translate 0 0 0.5
          Sphere 0.15 -0.15 0.15 360
      TransformEnd
    AttributeEnd
    AttributeBegin
      TransformBegin
          # resulting order: 010, 020, 011 ordering
          # ASH ordering would be: 010, 011, 020
          Shader "testslot" "011"
          Translate 0 -0.5 0
          Sphere 0.15 -0.15 0.15 360
      TransformEnd
    AttributeEnd
    AttributeBegin
      TransformBegin
          # resulting order: 010, 020, 021 ordering
          # ASH ordering would be: 010, 020, 021
          Shader "testslot" "021"
          Translate 0 0.5 0
          Sphere 0.15 -0.15 0.15 360
      TransformEnd
    AttributeEnd
WorldEnd
```

B MATERIAL SLOT BINDING API

Native USD material bindings are expressed by UsdRelationships. They are authored using Pixar's schema: UsdShadeMaterialBindingAPI. A direct material binding is specified in Listing 3.

Listing 3: "USD direct binding."

```
def Xform "Asset"
{
  rel material:binding = </DustMaterial>
}
```

The main limitation of Pixar's binding is that it does not support the binding of multiple materials at the same prim location (for a given material purpose). This design decision is also reflected in the way bound materials are resolved by Hydra (which uses the ComputeBoundMaterial() function of UsdShadeMaterialBindingAPI), returning a single resolved material per geometry primitive (for a given material purpose).

As Hydra support is not a requirement within our pipeline, we decided to use a custom encoding for Glimpse ASH bindings using our own API schema.

B.1 Slot bindings in USD

For our own schema, we decided to stay as close as possible to the native USD bindings. Slot bindings are represented by UsdRelationships, where the slot name is encoded in the name of the relationship, as shown in Listing 4.

Listing 4: "USD Slot Binding."

```
def "brick_6141_91"
{
  rel material:slotBinding:_000 = </LegoPlastic>
  rel material:slotBinding:_050 = </DustMaterial>
}
```

All material relationships representing a slot binding begin with the `material:slotBinding` namespace and the rest of the relationship's name encodes the slot name. The devil is in the details, as USD naming conventions are quite strict:

- The names of any UsdProperty (including UsdRelationship) are made of a single identifier or a list of identifiers separated by colons.
- An identifier is considered valid if it follows the USD identifier convention; that is, it must not be empty, must start with a letter or underscore, and may contain only letters, underscores, or numerals.

Therefore, we encode Glimpse slot names as valid USD identifiers using the following steps:

For example to encode a slot override `"050::000"`:

(1) split the string with `":"` as the separator: `["050","","000"]`
(2) transform each element into a valid USD identifier: `["_050","_","_000"]`
(3) join each element with `":"` as the separator, giving: `"_050:_:_000"`

This means that a Glimpse binding on slot `"050::000"` to a material called "PlasticOverride" would be encoded as shown in Listing 5.

Listing 5: "USD Slot Binding with Override."

```
rel material:slotBinding:_050:_:_000 = </PlasticOverride>
```

B.2 Custom Schema Usage Example

At the end of the day, a user just wants an easy way to author and read slot bindings inside of a USD scene. MaterialSlotBindingAPI handles the slot encoding, allowing users to deal only with canonical slot name:

Listing 6: "Slot Binding Made Easy."

```
prim     = stage.DefinePrim('/Set', 'Xform')
material = UsdShade.Material.Define(stage, '/Set/Mat')

# Create an instance of the schema attached to your prim
bindingAPI = schemas.MaterialSlotBindingAPI(prim)

# Create a binding between 'prim' and 'material' on slot '000'
bindingAPI.Bind(material, '000') # create relationship named '↩
      material:slotBinding:_000'

# Get slot bindings expressed on 'prim'
bindingAPI.GetBindings()[0].getSlot()          # return '000'
bindingAPI.GetBindings()[0].GetMaterialPath() # return '/Set/Mat↩
      '
bindingAPI.GetBindings()[0].GetMaterial()      # return ↩
      UsdShaderMaterial 'material'
```

B.3 Blocking Relationship

It is possible to remove a slot binding for a given a primitive. This affects the binding resolution on this primitive (and all descendants) by forcing all inherited bindings (on this slot) to be ignored – unless redefined.

MaterialSlotBindingAPI encodes these bindings using Attribute Block. To obtain such a result, bind an invalid UsdMaterial:

Listing 7: "Blocking Relationship."

```
bindingAPI = schemas.MaterialSlotBindingAPI(prim)
bindingAPI.Bind(UsdShade.Material(), '000')
```

This will generate the following relationship as shown in Listing 8.

Listing 8: "USD Blocked Slot Binding."

```
  rel material:slotBinding:_000 = None
```