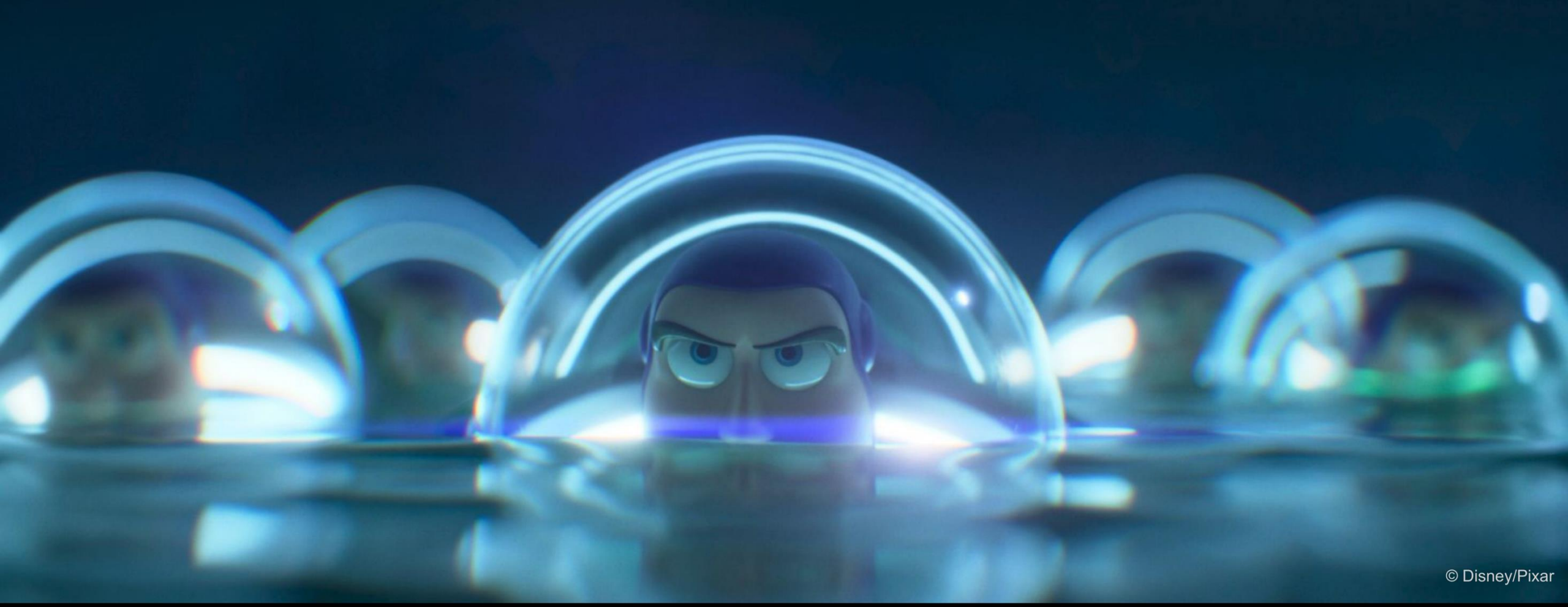


BRINGING ARBITRARY MATERIALX BSDF NETWORKS TO RENDERMAN XPU

JONATHAN STONE | LUCASFILM FRAN GONZALEZ | PIXAR

Content

Motivation
Cross-Disney Studio Collaboration
MaterialX BSDF Networks
Architecture for Arbitrary BSDF Networks
Material Representation
Shim nodes and Closure Generation
From Scene Ingestion to Runtime Evaluation
Results and Validation
Conclusion

Motivation

How it all started?

Shared Goals for MaterialX across Disney Studios

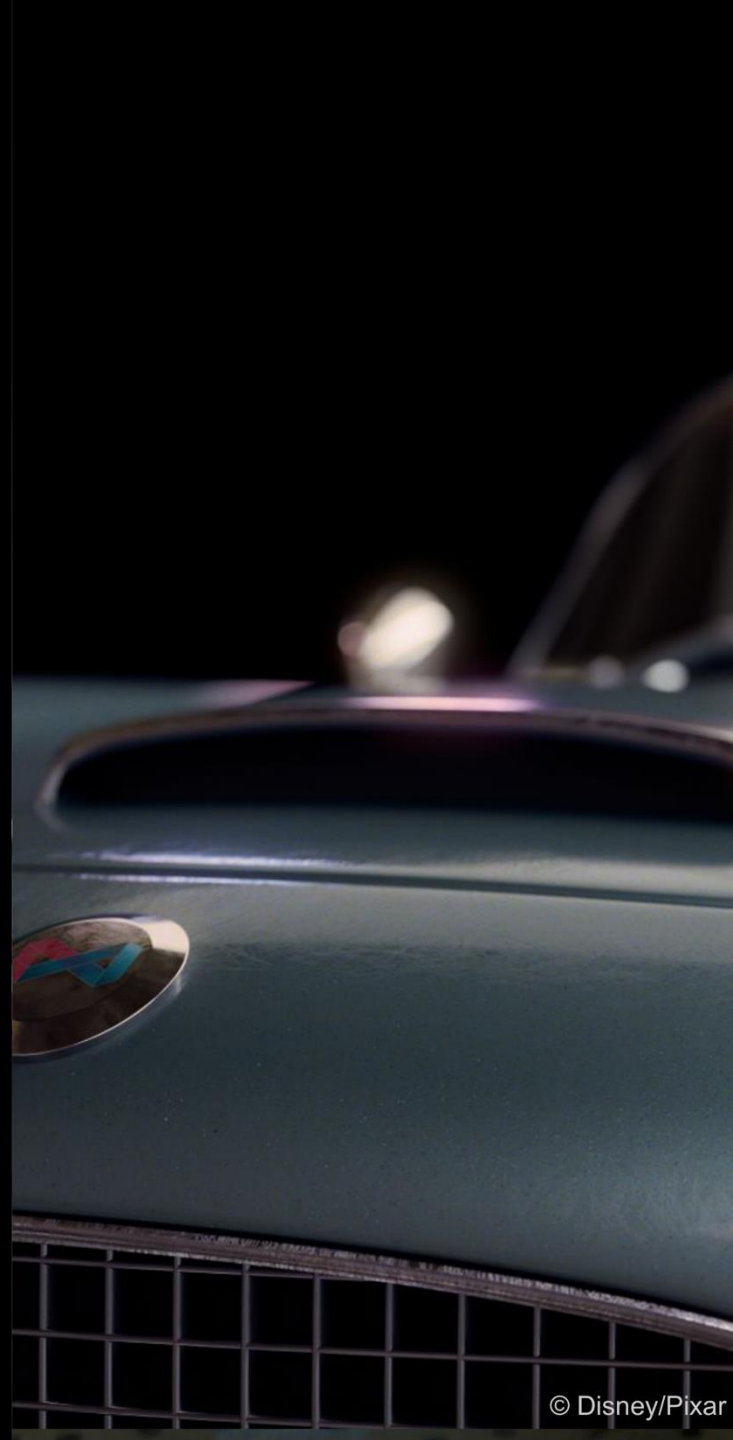
- Final renders for film and streaming
- Accurate artist previews across DCC tools
- Real-time experiences and LED walls
- Digital backlots and asset persistence
- Asset interchange with partner studios

MaterialX in RenderMan

- Lucasfilm and Pixar both rely upon RenderMan for final renders
- MaterialX BSDFs in RenderMan would complete the picture

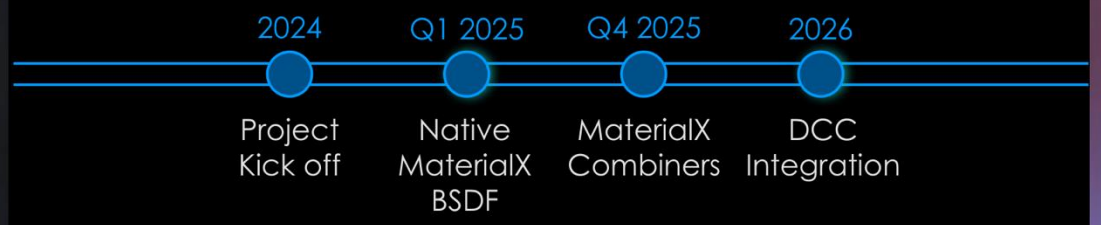
Pixar and Lucasfilm Collaboration

- Introduction of RenderMan XPU opens a window of opportunity
- Formal collaboration launches in Spring 2024



Cross-Disney Studio Collaboration

Pixar & Lucasfilm



- Both studios develop directly within the Pixar RenderMan development environment
- Existing MaterialX Lama infrastructure is extended to support MaterialX BSDF Nodes

PIXAR

MATERIALX

LUCASFILM Ltd

MaterialX BSDF Networks

Base MaterialX BSDFs and EDFs

MaterialX BSDFs

- Support for base BSDFs
  - Burley Diffuse
  - Oren-Nayar Diffuse
  - Conductor
  - Dielectric
  - Generalized Schlick
  - Sheen
  - Subsurface
  - Translucent

MaterialX EDFs

- Emissive Uniform

MaterialX BSDF Networks

Combining Base Nodes to Describe Materials Appearance

MaterialX Combiners

- Support combination of BSDF nodes
  - Add
  - Mix
  - Multiply
  - Layer

SHADING NETWORK

CAR PAINT

Architecture for Arbitrary BSDF Networks

Building a Robust Infrastructure for Hybrid Rendering

Production-Proven Foundation

- Built on the existing MaterialX Lama implementation in XPU.
- Reuses shared RenderMan scattering classes and functions.
- Proven in production at ILM.

Unified CPU/GPU Execution

- OSL emits data into a preallocated closure pool.
- C++/CUDA kernels implement BSDF sampling and evaluation.
- The same kernels compile for CPU and GPU.

Engine-Controlled DAG Traversal

- Combiner traversal is centralized in the engine.
- No recursive execution inside combiner plugins.
- Predictable, GPU-friendly evaluation of arbitrary networks.

Material Representation

Multiple BSDF Representations in XPU

OSL Representation

- Each BSDF is represented by
  - Custom build material closure
  - Shim Node
    - Wires parameters to the material closure
- Generated at build time
  - Regeneration only when BSDF parameters change

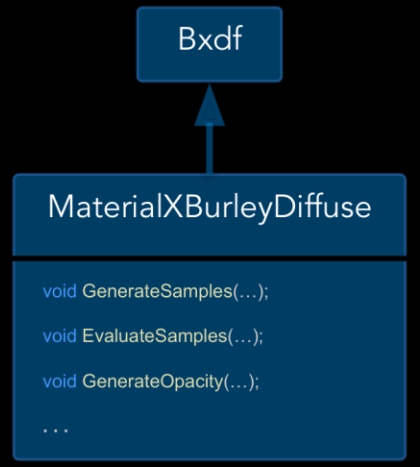
MaterialXBurleyDiffuse.osl

.osl

```
closure color materialx_burleydiffuse(…) [[int builtin = 1]];
shader MaterialXBurleyDiffuse(… parameters …)
{
    if (isconnected(resultBxdf) == 2)
    {
        resultBxdf = materialx_burleydiffuse(…);
    }
    else
    {
        Ci = Ci + materialx_burleydiffuse(…);
    }
}
```

C++ Plugin Representation

- Each BSDF implements a BSDF plugin
  - Inherits from Bxdf.h - Abstract class
  - Implements BSDF kernels
    - *GenerateSamples*, *EvaluateSamples*, etc.
    - Kernels code is shared between CPU and GPU



Shim Nodes and Closure Generation

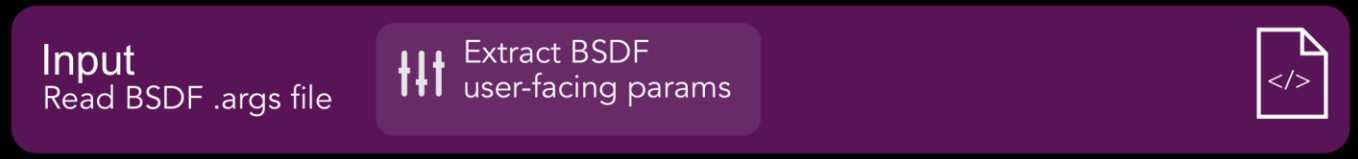
Generating XPU's OSL BSDF Representations

OSL

C++ Plugin

Shim Nodes and Closure Generation

Generating XPU's OSL BSDF Representations

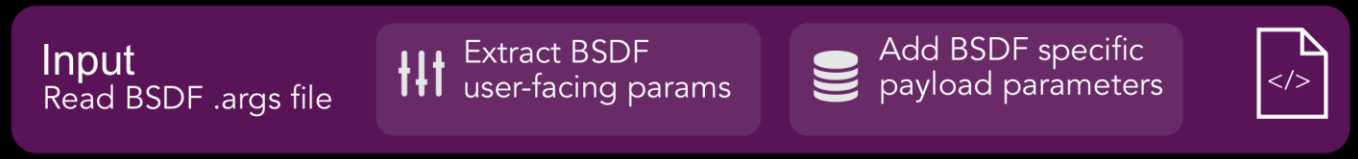


OSL

C++ Plugin

Shim Nodes and Closure Generation

Generating XPU's OSL BSDF Representations

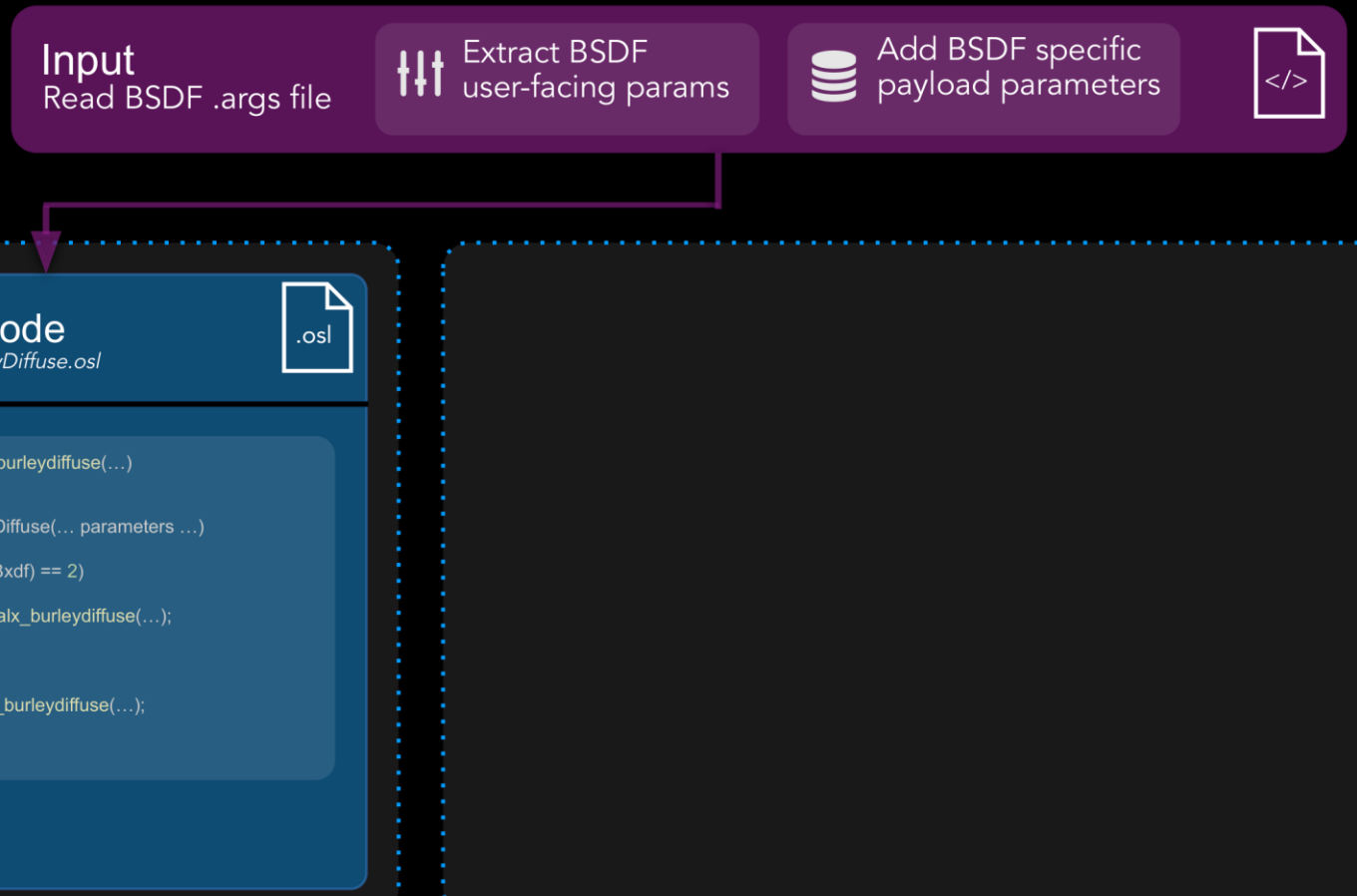


OSL

C++ Plugin

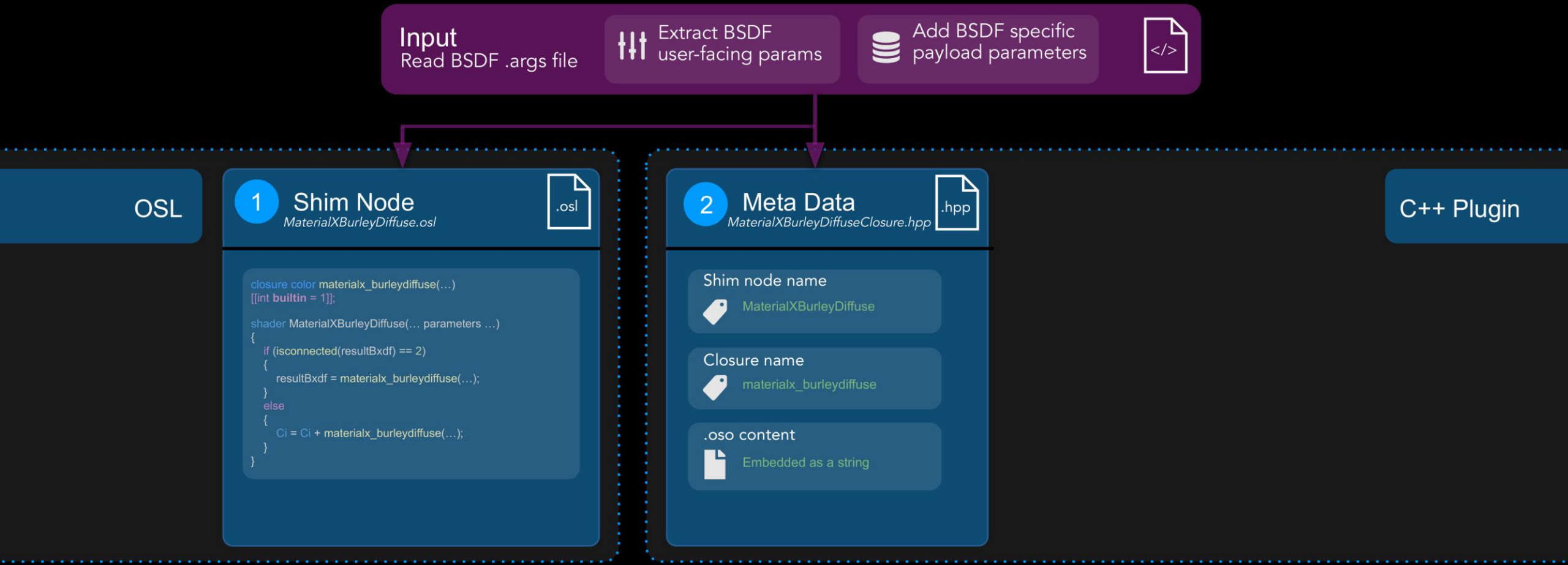
Shim Nodes and Closure Generation

Generating XPU's OSL BSDF Representations



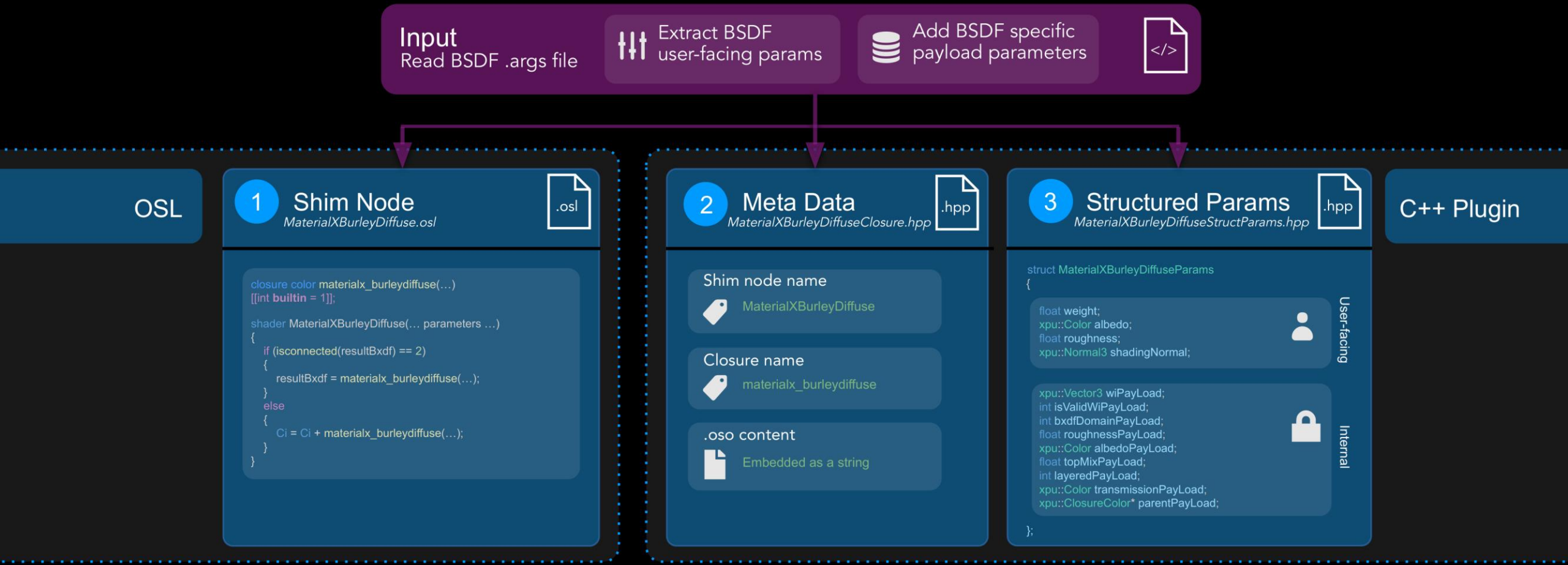
Shim Nodes and Closure Generation

Generating XPU's OSL BSDF Representations



Shim Nodes and Closure Generation

Generating XPU's OSL BSDF Representations



Shim Nodes and Closure Generation

Generating XPU's OSL BSDF Representations

Input
Read BSDF .args file

Extract BSDF user-facing params

Add BSDF specific payload parameters

</>

OSL

1 Shim Node
MaterialXBurleyDiffuse.osl

.osl

```
closure color materialx_burleydiffuse(…)
[[int builtin = 1]];

shader MaterialXBurleyDiffuse(… parameters …)
{
    if (isconnected(resultBxdf) == 2)
    {
        resultBxdf = materialx_burleydiffuse(…);
    }
    else
    {
        Ci = Ci + materialx_burleydiffuse(…);
    }
}
```

2 Meta Data
MaterialXBurleyDiffuseClosure.hpp

.hpp

Shim node name
MaterialXBurleyDiffuse

Closure name
materialx_burleydiffuse

.oso content
Embedded as a string

3 Structured Params
MaterialXBurleyDiffuseStructParams.hpp

.hpp

```
struct MaterialXBurleyDiffuseParams
{
    float weight;
    xpu::Color albedo;
    float roughness;
    xpu::Normal3 shadingNormal;

    xpu::Vector3 wiPayLoad;
    int isValidWiPayLoad;
    int bxdfDomainPayLoad;
    float roughnessPayLoad;
    xpu::Color albedoPayLoad;
    float topMixPayLoad;
    int layeredPayLoad;
    xpu::Color transmissionPayLoad;
    xpu::ClosureColor* parentPayLoad;
};
```

User-facing

Internal

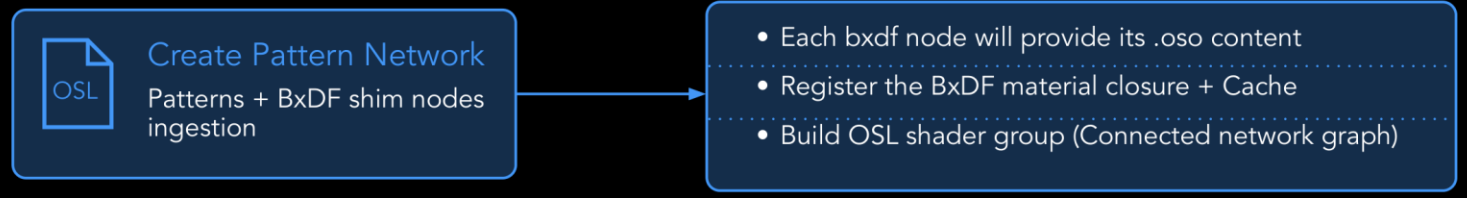
C++ Plugin

Purpose: Build the runtime pieces XPU needs: Shim Node / Material Closure, Meta data and strongly typed parameter structures

From Scene Ingestion to Runtime Evaluation

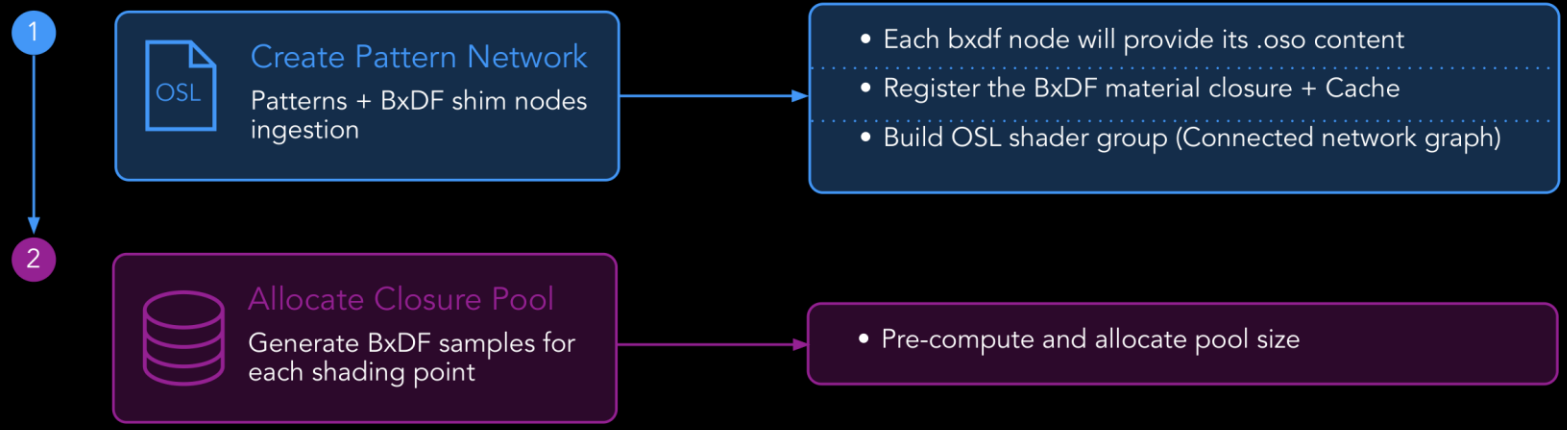
Scene Ingestion

1



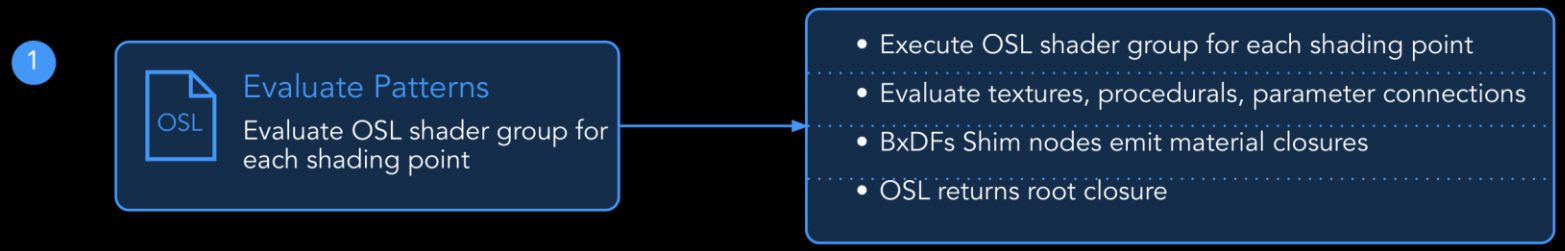
From Scene Ingestion to Runtime Evaluation

Scene Ingestion



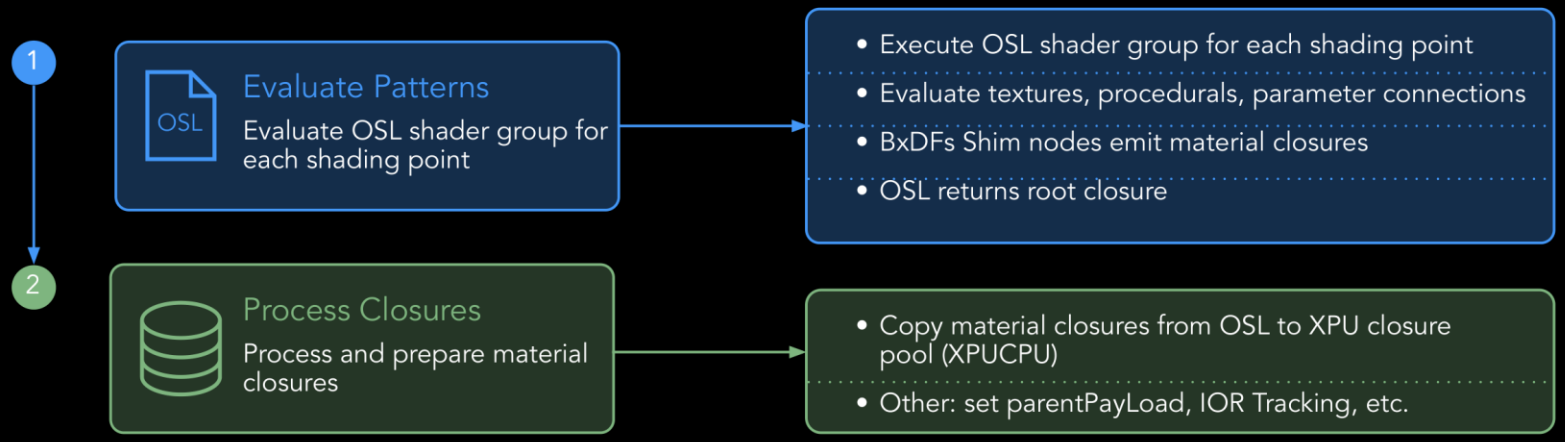
From Scene Ingestion to Runtime Evaluation

Runtime Evaluation



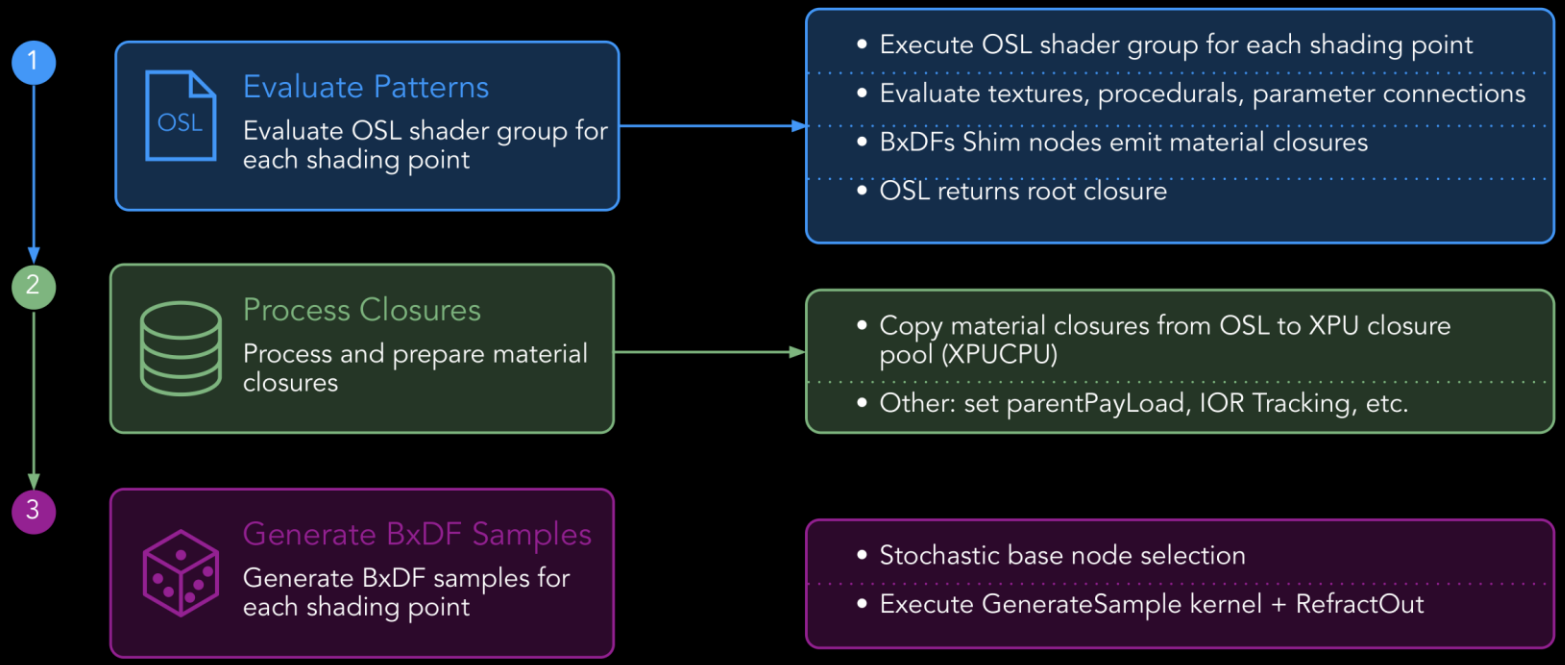
From Scene Ingestion to Runtime Evaluation

Runtime Evaluation



From Scene Ingestion to Runtime Evaluation

Runtime Evaluation



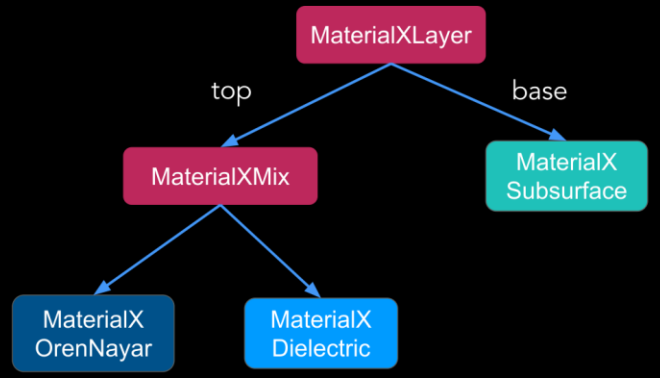
From Scene Ingestion to Runtime Evaluation

Runtime Evaluation

1. **Evaluate Patterns**
   Evaluate OSL shader group for each shading point
   - Execute OSL shader group for each shading point
   - Evaluate textures, procedurals, parameter connections
   - BxDFs Shim nodes emit material closures
   - OSL returns root closure

2. **Process Closures**
   Process and prepare material closures
   - Copy material closures from OSL to XPU closure pool (XPUCPU)
   - Other: set parentPayLoad, IOR Tracking, etc.

3. **Generate BxDF Samples**
   Generate BxDF samples for each shading point
   - Stochastic base node selection
   - Execute GenerateSample kernel + RefractOut



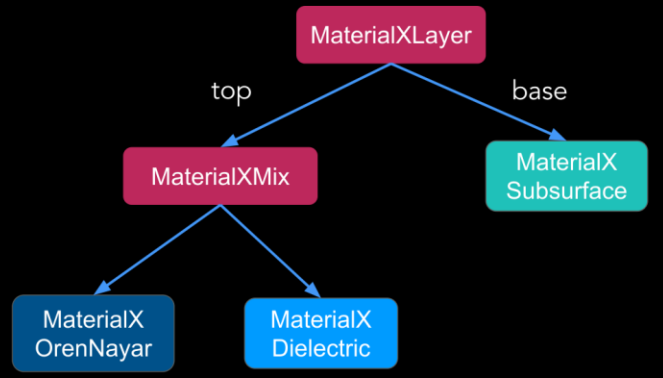
From Scene Ingestion to Runtime Evaluation

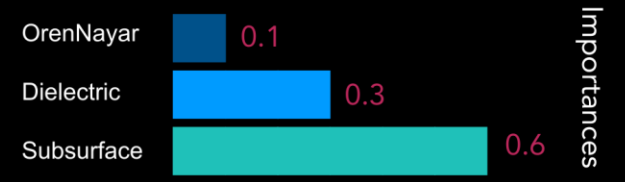
Runtime Evaluation

1. **Evaluate Patterns**
   Evaluate OSL shader group for each shading point
   - Execute OSL shader group for each shading point
   - Evaluate textures, procedurals, parameter connections
   - BxDFs Shim nodes emit material closures
   - OSL returns root closure

2. **Process Closures**
   Process and prepare material closures
   - Copy material closures from OSL to XPU closure pool (XPUCPU)
   - Other: set parentPayLoad, IOR Tracking, etc.

3. **Generate BxDF Samples**
   Generate BxDF samples for each shading point
   - Stochastic base node selection
   - Execute GenerateSample kernel + RefractOut





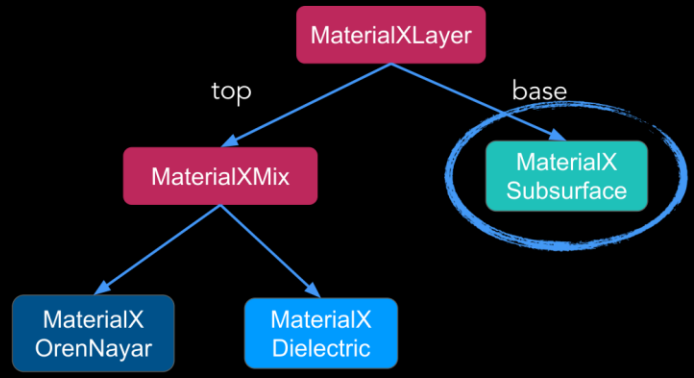
From Scene Ingestion to Runtime Evaluation

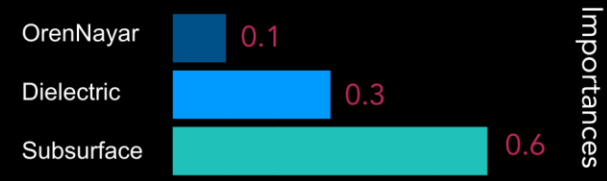
Runtime Evaluation

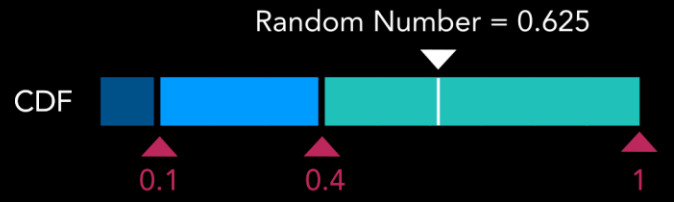
1. **Evaluate Patterns**
   Evaluate OSL shader group for each shading point
   - Execute OSL shader group for each shading point
   - Evaluate textures, procedurals, parameter connections
   - BxDFs Shim nodes emit material closures
   - OSL returns root closure

2. **Process Closures**
   Process and prepare material closures
   - Copy material closures from OSL to XPU closure pool (XPUCPU)
   - Other: set parentPayLoad, IOR Tracking, etc.

3. **Generate BxDF Samples**
   Generate BxDF samples for each shading point
   - Stochastic base node selection
   - Execute GenerateSample kernel + RefractOut







From Scene Ingestion to Runtime Evaluation

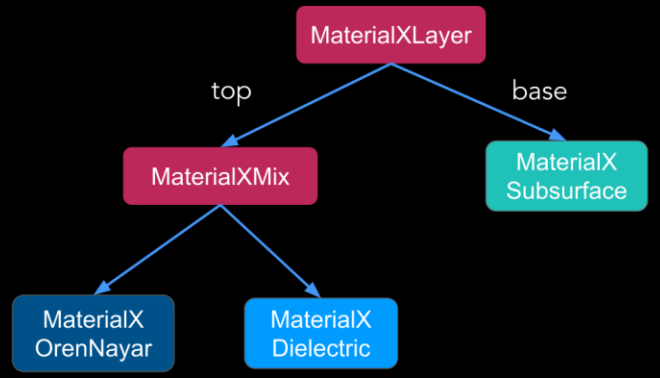
Runtime Evaluation

1. **Evaluate Patterns**
   Evaluate OSL shader group for each shading point
   - Execute OSL shader group for each shading point
   - Evaluate textures, procedurals, parameter connections
   - BxDFs Shim nodes emit material closures
   - OSL returns root closure

2. **Process Closures**
   Process and prepare material closures
   - Copy material closures from OSL to XPU closure pool (XPUCPU)
   - Other: set parentPayLoad, IOR Tracking, etc.

3. **Generate BxDF Samples**
   Generate BxDF samples for each shading point
   - Stochastic base node selection
   - Execute GenerateSample kernel + RefractOut

4. **Evaluate BxDF Samples**
   Evaluate all BxDFs
   - Evaluate all base nodes
   - Execute EvaluateSample kernel + RefractIn



From Scene Ingestion to Runtime Evaluation

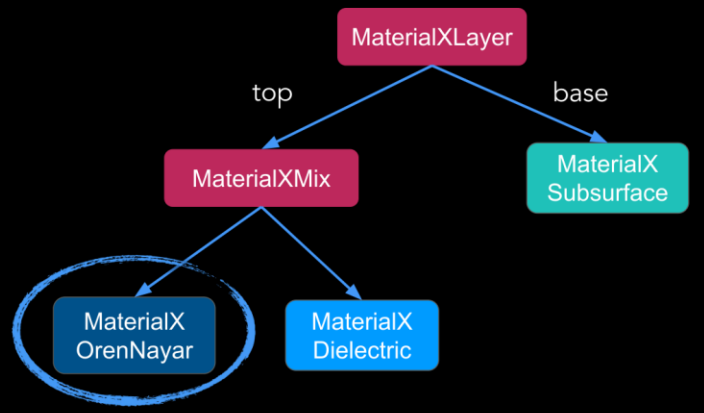
Runtime Evaluation

1. **Evaluate Patterns**
   Evaluate OSL shader group for each shading point
   - Execute OSL shader group for each shading point
   - Evaluate textures, procedurals, parameter connections
   - BxDFs Shim nodes emit material closures
   - OSL returns root closure

2. **Process Closures**
   Process and prepare material closures
   - Copy material closures from OSL to XPU closure pool (XPUCPU)
   - Other: set parentPayLoad, IOR Tracking, etc.

3. **Generate BxDF Samples**
   Generate BxDF samples for each shading point
   - Stochastic base node selection
   - Execute GenerateSample kernel + RefractOut

4. **Evaluate BxDF Samples**
   Evaluate all BxDFs
   - Evaluate all base nodes
   - Execute EvaluateSample kernel + RefractIn



Node Evaluation ▸ MaterialX OrenNayar

From Scene Ingestion to Runtime Evaluation

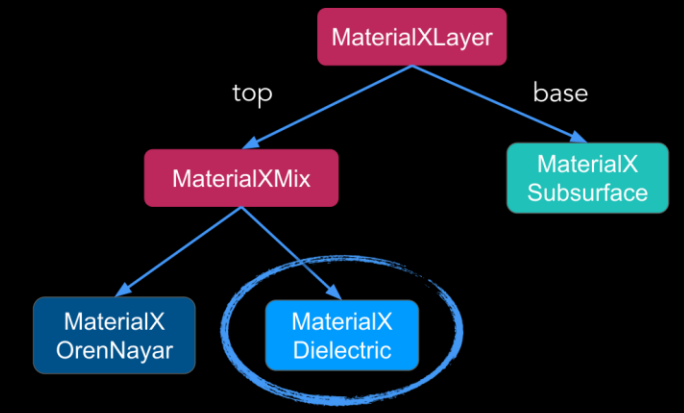
Runtime Evaluation

1. **Evaluate Patterns**
   Evaluate OSL shader group for each shading point
   - Execute OSL shader group for each shading point
   - Evaluate textures, procedurals, parameter connections
   - BxDFs Shim nodes emit material closures
   - OSL returns root closure

2. **Process Closures**
   Process and prepare material closures
   - Copy material closures from OSL to XPU closure pool (XPUCPU)
   - Other: set parentPayLoad, IOR Tracking, etc.

3. **Generate BxDF Samples**
   Generate BxDF samples for each shading point
   - Stochastic base node selection
   - Execute GenerateSample kernel + RefractOut

4. **Evaluate BxDF Samples**
   Evaluate all BxDFs
   - Evaluate all base nodes
   - Execute EvaluateSample kernel + RefractIn



Node Evaluation ▸ MaterialX OrenNayar | MaterialX Dielectric

From Scene Ingestion to Runtime Evaluation

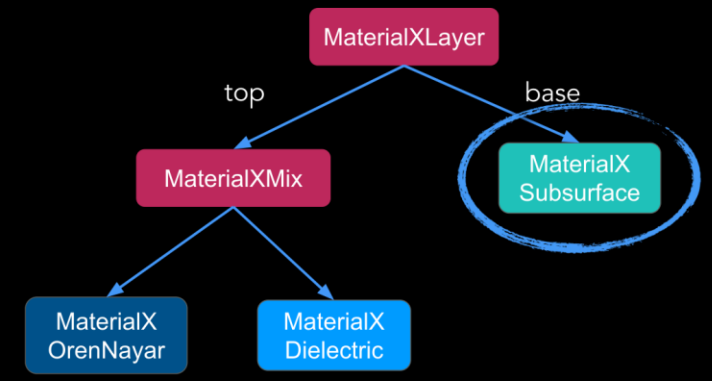
Runtime Evaluation

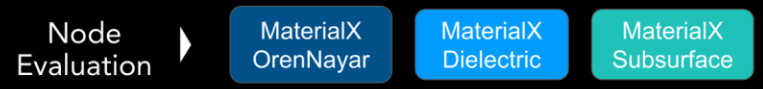
1. **Evaluate Patterns**
   Evaluate OSL shader group for each shading point
   - Execute OSL shader group for each shading point
   - Evaluate textures, procedurals, parameter connections
   - BxDFs Shim nodes emit material closures
   - OSL returns root closure

2. **Process Closures**
   Process and prepare material closures
   - Copy material closures from OSL to XPU closure pool (XPUCPU)
   - Other: set parentPayLoad, IOR Tracking, etc.

3. **Generate BxDF Samples**
   Generate BxDF samples for each shading point
   - Stochastic base node selection
   - Execute GenerateSample kernel + RefractOut

4. **Evaluate BxDF Samples**
   Evaluate all BxDFs
   - Evaluate all base nodes
   - Execute EvaluateSample kernel + RefractIn



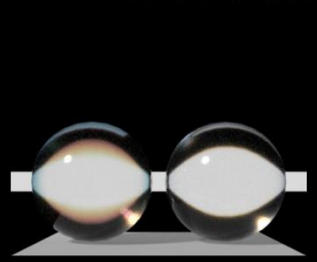
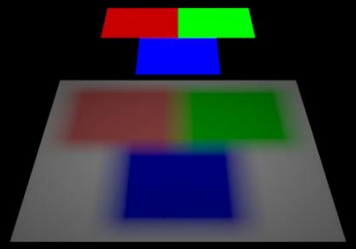
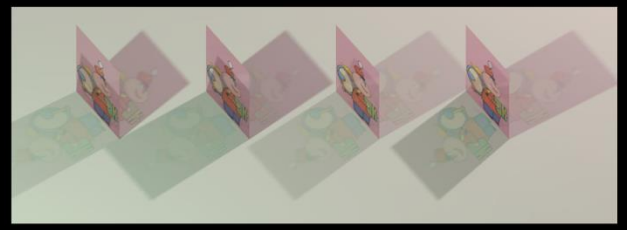


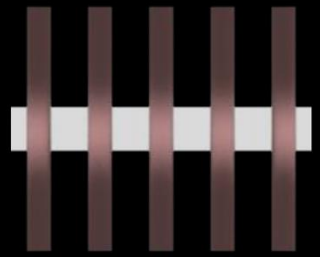
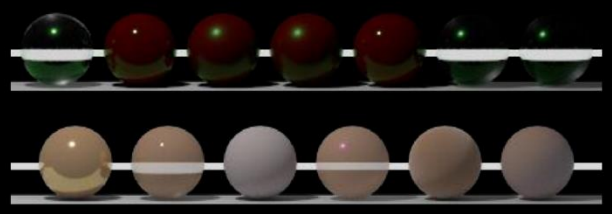
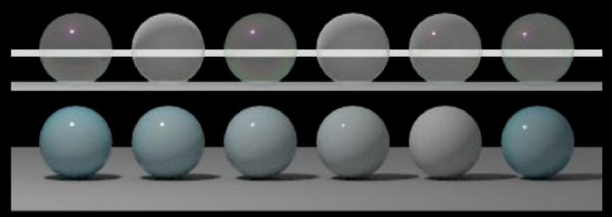
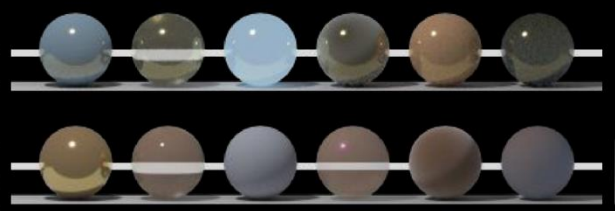
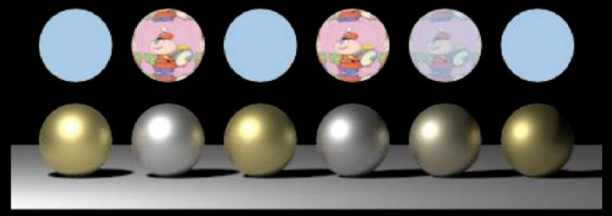
Results and Validation

MaterialX Testing

- Developed a comprehensive MaterialX regression suite
  - Validates both BSDF nodes and combiner nodes
  - Across multiple configurations - XPUCPU, XPUGPU and XPU

- Executed automatically every night
  - Detect regressions
  - Ensure consistency

Results and Validation

MaterialX Stirling

Artist validation using MaterialX in XPU - Leif Pedersen *(Sr Technical Product Marketing Manager)*


© Disney/Pixar

Results and Validation

MaterialX Stirling

Artist validation using MaterialX in XPU - Leif Pedersen *(Sr Technical Product Marketing Manager)*

Results and Validation

MaterialX Stirling

Artist validation using MaterialX in XPU - Leif Pedersen *(Sr Technical Product Marketing Manager)*

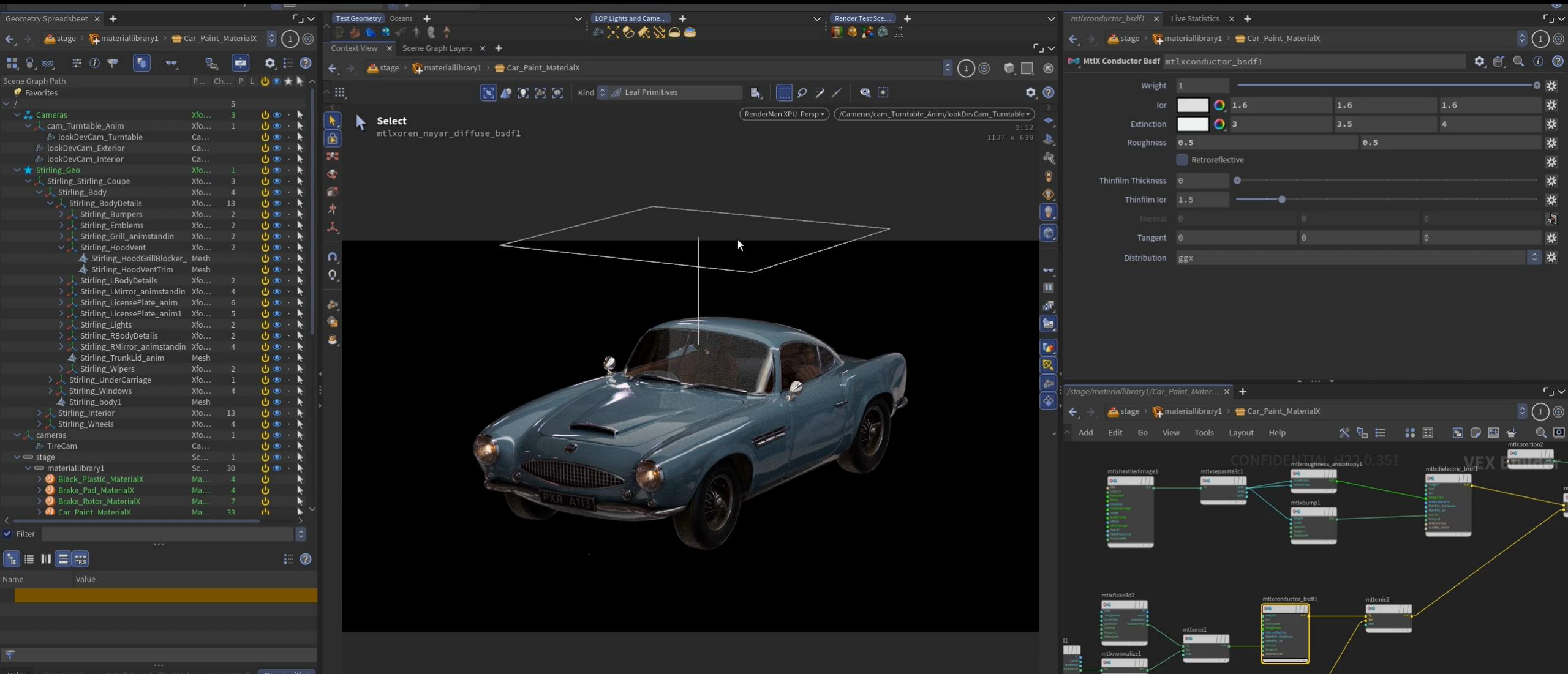
Results and Validation

MaterialX Stirling

Artist validation using MaterialX in XPU - Leif Pedersen *(Sr Technical Product Marketing Manager)*

Results and Validation

Artist-driven Integration



Conclusion

Key Takeaways

Cross-Studio Collaboration

- A shared codebase and open standards enabled long-term collaboration across Disney studios.

Unified Material Architecture

- Shared material system and kernel logic across CPU and GPU.

Efficient Evaluation of Arbitrary Networks

- GPU-friendly closure pools and DAG traversal eliminate recursion and virtual dispatch during shading.

Scalable and Extensible Design

- Modular BxDF representation and automatic shim/closure generation simplify integration of new materials.

Production Ready

- Support for complex material networks.
- Successfully exercised on real production assets.

THANK YOU

Per Christensen, Akshay Shah, Julian Fong, Katrin Bratland, Leif Pedersen, Mark Manka, Susan Salituro and Stephen Friedman

Eoghan Cunneen, Roger Cordes, Stephen Hill, André Mazzone, Alain Hostettler, David Meny, Mark Westbury, and Erin Young

BRINGING ARBITRARY MATERIALX BSDF NETWORKS TO RENDERMAN XPU

JONATHAN STONE | LUCASFILM FRAN GONZALEZ | PIXAR

One More Thing!!!

And now what?

MaterialX Stirling Contribution to the Public

Stay tuned for more information about Release Date

MaterialX Stirling Contribution to the Public

Stay tuned for more information about Release Date