MATERIALX

MaterialX: An Open Standard for Network-Based CG Object Looks

Presentations

. MaterialX: The Year in Review – Doug Smythe, ILM

. MaterialXGenMdl – The New MDL Backend for the MaterialX Shader Generation Framework
– Niklas Harrysson, Autodesk – Lutz Kettner, Jan Jordan, NVidia

. Making the Complex Simple with MaterialX Texture Baking – Madeleine Yip, Lucasfilm

. A Preview of MaterialX v1.38 – Doug Smythe, ILM

. MaterialX in USD/Hydra – Eliot Smyrl, Pixar

. MaterialX and Substance – David Larsson, Adobe

. MaterialX and LookdevX Update – Nikola Milosevic, Bernard Kwok, Autodesk

More Presentations

Course:
Physically Based Shading in Theory and Practice

MaterialX Physically Based Shading Nodes

Jonathan Stone, Lucasfilm Advanced Development Group
Niklas Harrysson, Autodesk

Wednesday Aug 26, 2020
8:30am - 12:00pm

MaterialX Overview

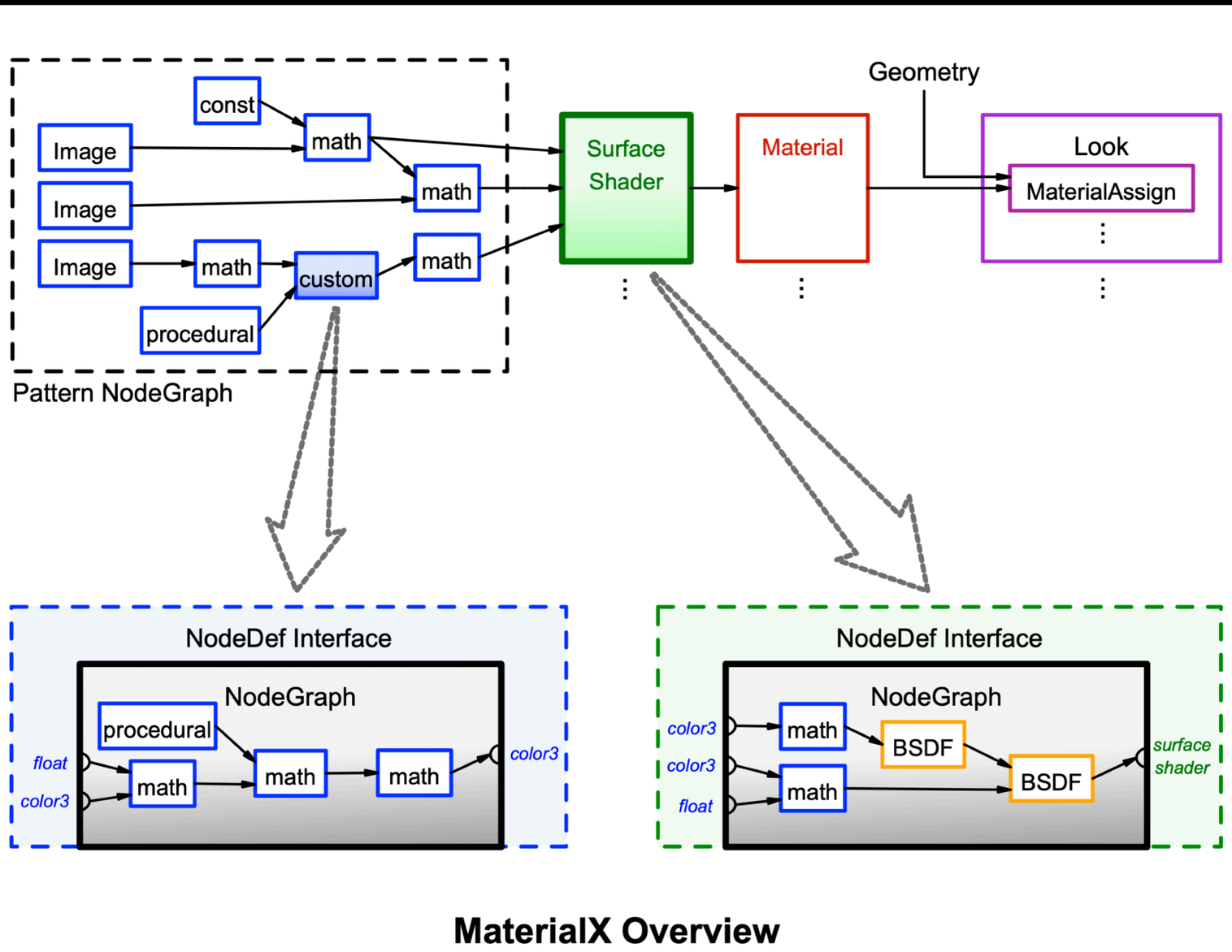
- Schema and File Format used to describe "complete CG object looks":
  - Shading network topology
  - Complex materials with inheritance and multiple rendering targets
  - Texture, Material and Visibility assignments
  - Illumination and shadowing assignments for asset lights
  - Look variants, geometric primitive properties, and much more

- Specific defined behavior for "Standard Nodes":
  - Texture reading
  - Procedural pattern generation
  - Image processing operations
  - Shading operations (BSDFs and layering)

Features of MaterialX

- Strong data typing
- Fully color managed
- Compatible with other open standards
  - e.g. OSL, OpenColorIO, OpenEXR, Alembic, USD

- **Extensible**:
  - User-defined nodes and shaders
  - User-defined data types
  - Application-specific node parameters and attributes



MaterialX Overview

Home Developer Guide Specification Help GITHUB

Introduction to MaterialX

www.materialx.org

MaterialX: The Year In Review

Developments Over the Past Year

- Specification: Definition of Nodes and Elements
- Shader Generation
- Standalone Viewer: MaterialXView
- Open-Source Library (C++ with Python Bindings)
- Build Environment

Specification Updates: Units

```
<unittypedef name="distance"/>
<unitdef name="UD_stdlib_distance" unittype="distance">
  <unit name="micron" scale="0.000001"/>
  <unit name="millimeter" scale="0.001"/>
  <unit name="centimeter" scale="0.01"/>
  <unit name="meter" scale="1.0"/>
  <unit name="kilometer" scale="1000.0"/>
  <unit name="inch" scale="0.0254"/>
  <unit name="foot" scale="0.3048"/>
  <unit name="yard" scale="0.9144"/>
  <unit name="mile" scale="1609.34"/>
</unitdef>
```


Specification Updates: New/Updated Nodes

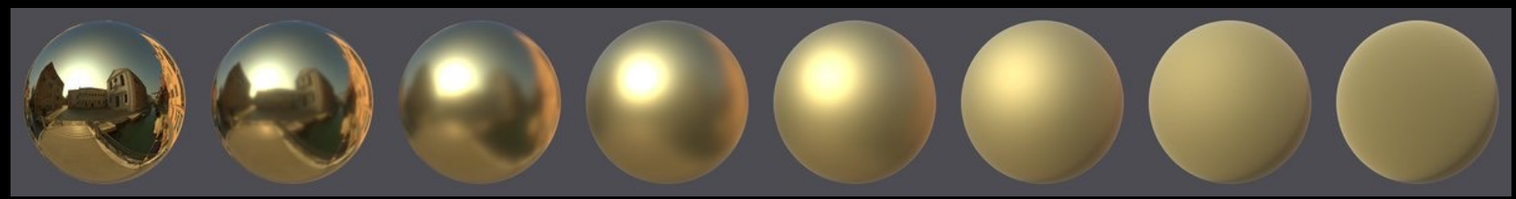
- Multiple signatures for nodes allowed, to support different combinations of types
- Some cases: impossible to resolve exactly which node signature to use

- Affected nodes:
  - invert —> invertmatrix
  - new transformmatrix node
    - transformpoint/transformvector/transformnormal now only between named spaces
  - rotate —> rotate2d, rotate3d
  - combine —> combine2, combine3, combine4
  - separate —> separate2, separate3, separate4

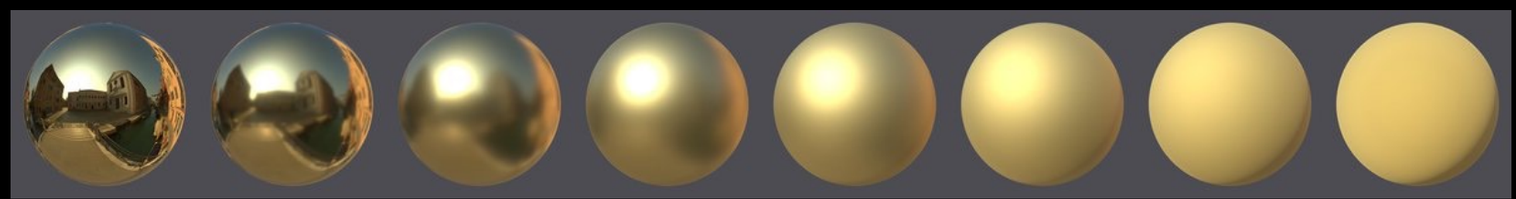
- compare —> ifgreater, ifgreatereq, ifequal

Shader Generation: Energy Compensation

- Before:



- After:

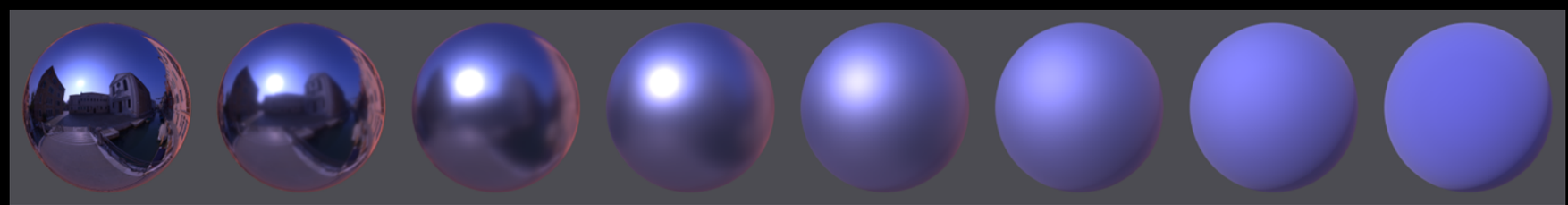


Shader Generation: MDL Backend

https://www.nvidia.com/en-us/gtc/on-demand/?search=s21469

Viewer Updates: "Wedge" Rendering

- Automatically vary-and-render a float shading parameter: "W" hotkey



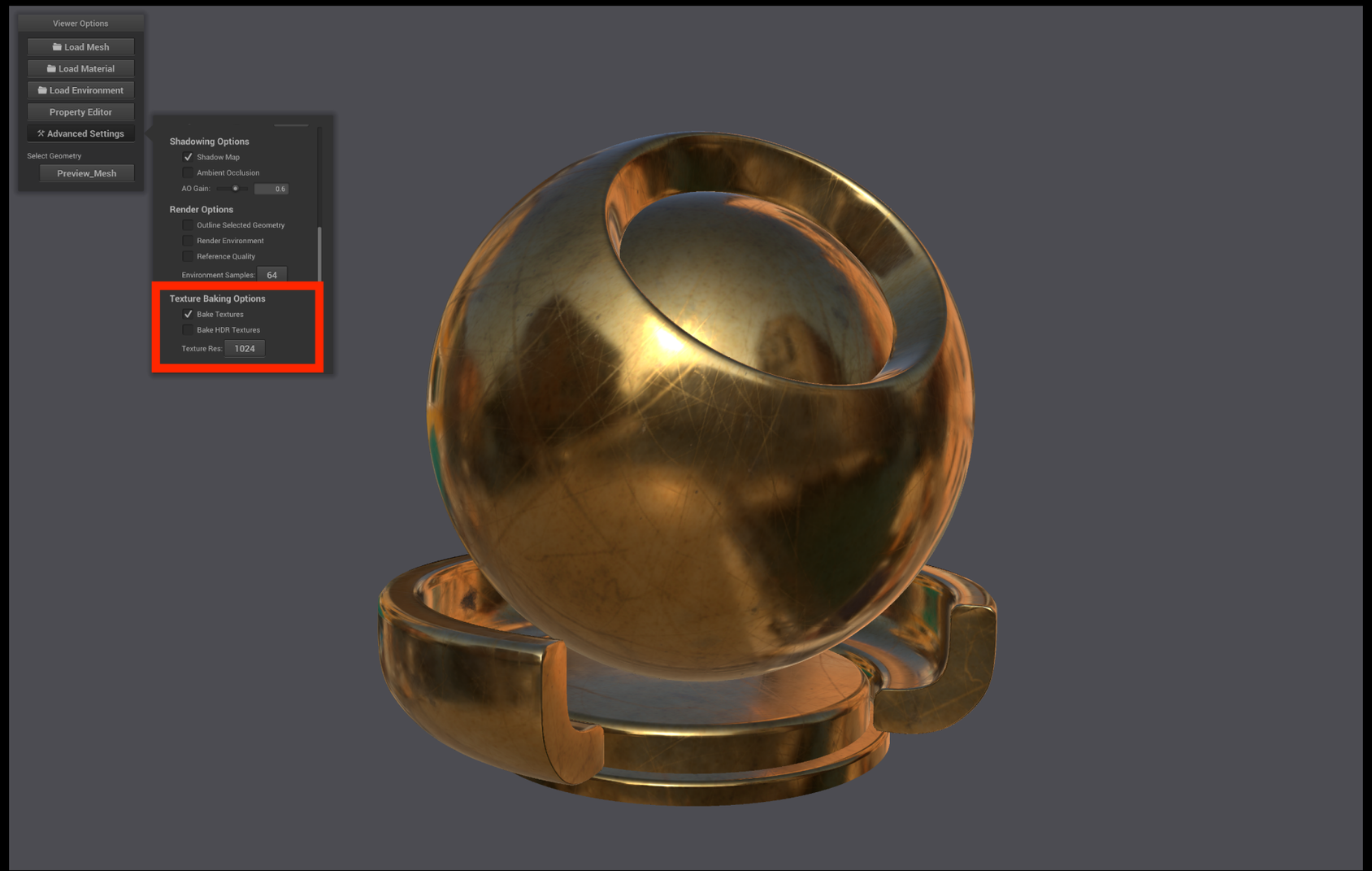
Viewer Updates: Other Changes

- New Light Rotation Slider in Advanced Settings panel:

- New Command line options:
  - Mesh, light and camera transforms
  - Screen dimensions and bg color
- Hotkeys for camera zoom

BB-8 © & ™ Lucasfilm Ltd. Used with permission.

Library Updates: Texture Baking



Library Updates

- Support for Specification v1.37 and 1.37REV2:
  - Lookgroups
  - Child <output>s for Nodedef type declaration
  - uisoftmin/uisoftmax/uistep attributes
  - GeomAttr -> GeomProp
  - Backdrop node -> backdrop element
  - Utility methods to support backdrop elements
  - Upgrade path for 1.36 documents to use new nodes in place of deprecated ones

- New MaterialX::getDefaultSearchPath() method

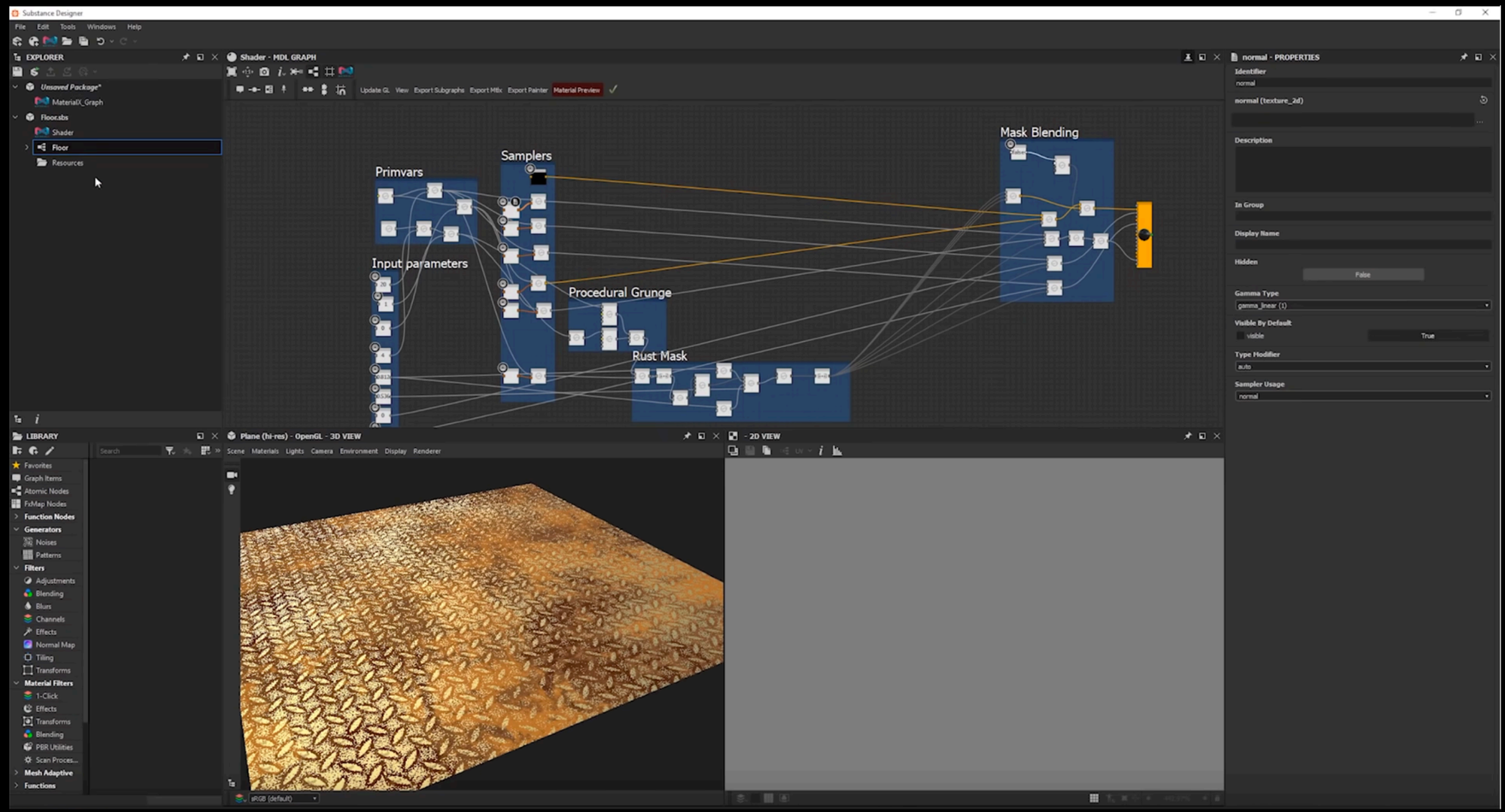
- Matrix types changed to use row vector convention
  - In-memory representation now consistent with Imath, USD and other libraries

Build Environment

- Now supported: Visual Studio 2019, Clang 9 and GCC 10

- Future: Support for GitHub Actions
  - Continuous integration, Cloud testing

Third-Party Support: Substance Designer

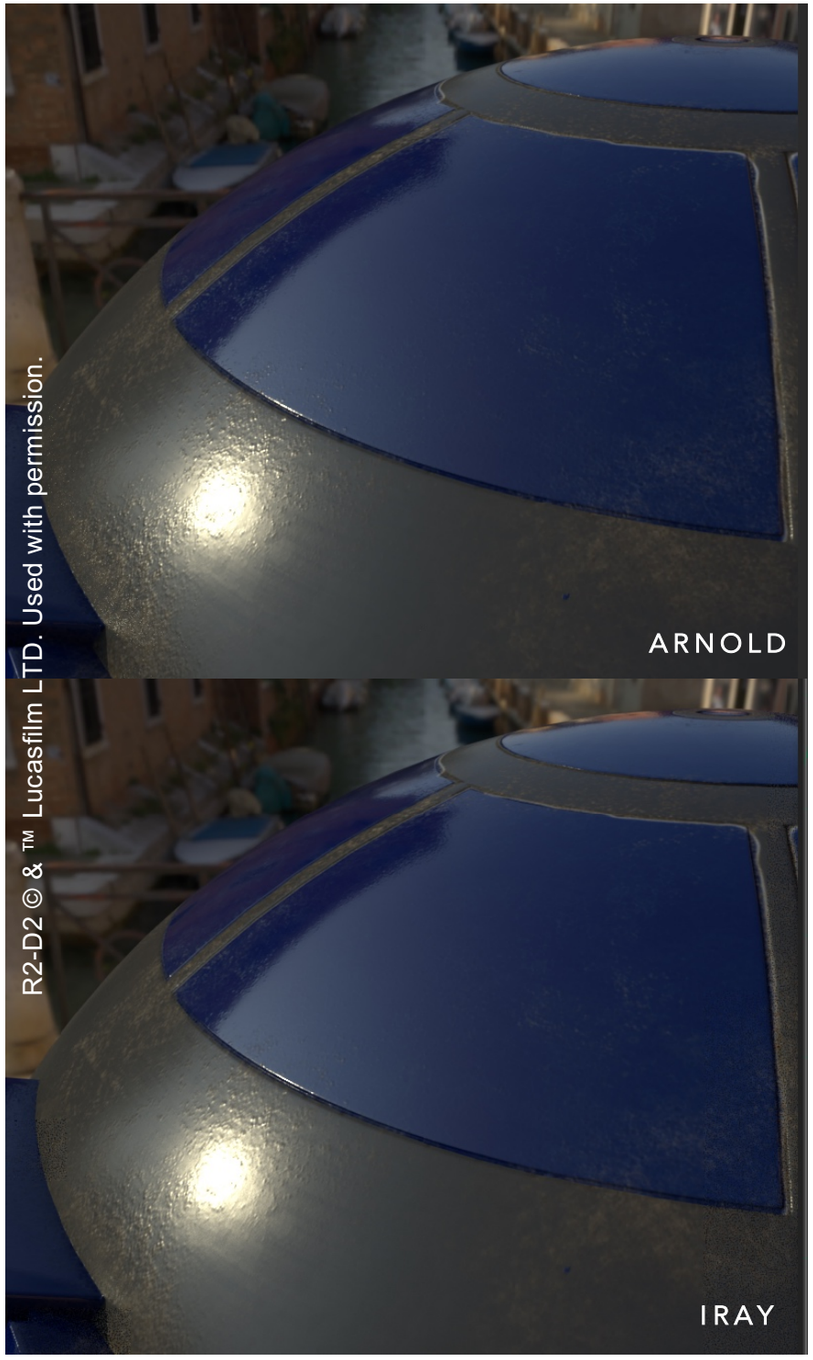


https://share.substance3d.com/libraries/6111

Current Developments

- Support for Displacement Shaders in generated OSL
- Improvements to document importing
- Shader Generation for SPIR-V / SPIRV-Cross
  - Enables generation of shaders for HLSL, Metal SL and WebGL
- Javascript bindings for the MaterialX API

Thanks!



MaterialXGenMdl

The New MDL Backend for the MaterialX Shader Generation Framework

Niklas Harrysson
Principal Engineer
AUTODESK

Jan Jordan
Software Product Manager MDL

Lutz Kettner
Director Rendering and Materials
NVIDIA

Acknowledgements: Bernard Kwok, Nikola Milosevic, Henrik Edstrom, Eric Bourque (Autodesk), Kai Rohmer, Sandra Pappenguth (Nvidia), David Larsson (Adobe), Jonathan Stone (Lucasfilm)

MaterialX and ShaderGen

MaterialX Physically-Based Shading Nodes

Data types, nodes, and node graphs for layered physically-based shading

materialx.org/assets/MaterialX.v1.37REV2.PBRSpec.pdf

See also: Physically Based Shading in Theory and Practice, Wednesday, 26 August 2020, 8:30am - 12:00pm

blog.selfshadow.com/publications/s2020-shading-course

ShaderGen

Transforms the agnostic MaterialX descriptions into executable code for a specific renderer

github.com/materialx/MaterialX/blob/master/documents/DeveloperGuide/ShaderGeneration.md

MaterialX Physically-Based Shading Nodes

Niklas Harrysson - niklas.harrysson@autodesk.com
Doug Smythe - smythe@ilm.com
Jonathan Stone - jstone@lucasfilm.com
July 16, 2019
(Revised October 17, 2019)

Introduction

The MaterialX Specification describes a number of standard nodes that may be used to construct node graphs for the processing of images, procedurally-generated values, coordinates and other data. With the addition of user-defined custom nodes, it is possible to describe complete rendering shaders using node graphs. Up to this point, there has been no standardization of the specific shader-semantic nodes used in these node graph shaders, although with the widespread shift toward physically-based shading, it appears that the industry is settling upon a number of specific BxDF and other functions with standardized parameters and functionality.

This document describes a number of shader-semantic nodes implementing widely-used surface, scattering, emission and volume distribution functions and utility nodes useful is constructing complex layered rendering shaders using node graphs. These nodes in combination with other nodes may be used with the MaterialX shader generation (ShaderGen[1]) system.

Shader Generation

1.1 Scope

A shader generation framework is implemented as part of MaterialX. This can help applications to transform the agnostic MaterialX data description into executable shader code for a specific renderer. A library module named MaterialXGenShader contains the core shader generation features, and support for specific languages resides in separate libraries, e.g. MaterialXGenGlsl, MaterialXGenOsl.

Note that this system has no runtime and the output produced is source code, not binary executable code. The source code produced needs to be compiled by a shading language compiler before being executed by the renderer. See Figure 1 for a high level overview of the system.

ShaderGen

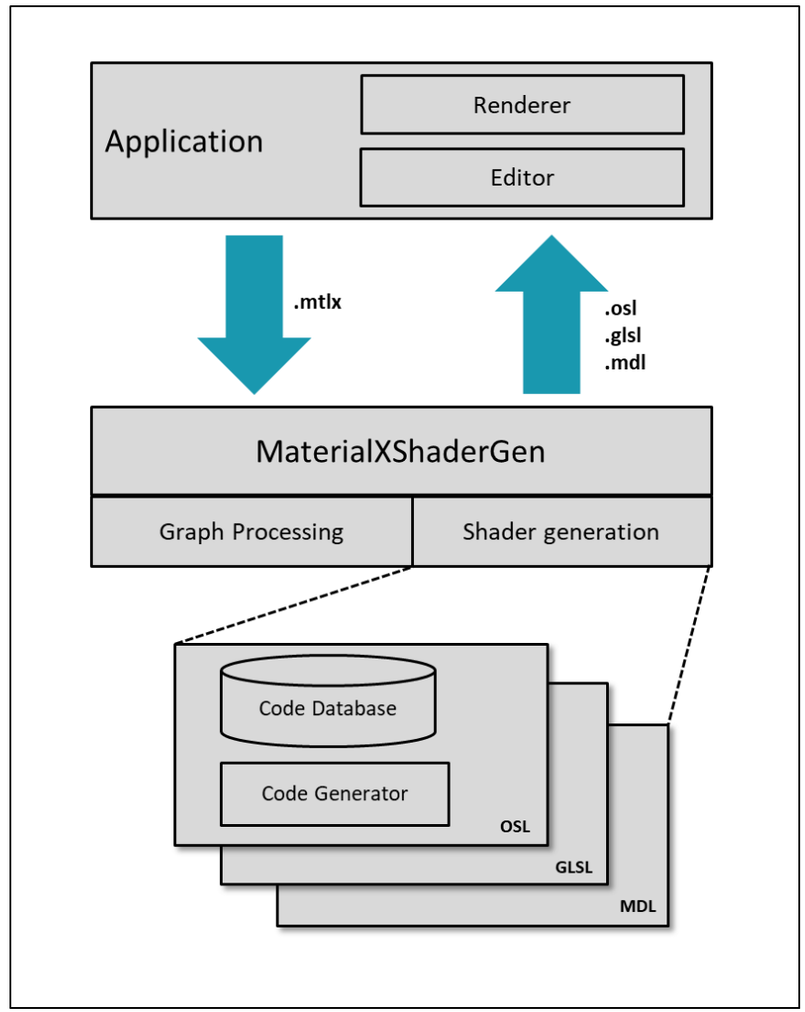
Shader Generation Framework

Generate source code for a target shading language

Extensible to many targets and languages

Supported languages:

- OSL
- GLSL
- WebGL/GLES (WIP)
- SPIR-V/SPIRV-CROSS (WIP)
  - HLSL, Metal SL
- **NVIDIA MDL *NEW***



NVIDIA® MDL

The **NVIDIA Material Definition Language (MDL)**
is technology developed by NVIDIA
to define **physically-based** materials
for physically-based rendering solutions.

- Started in **2011**, now at **MDL 1.6**
- Strong specification for material exchange
- Rendering-algorithm agnostic
- Designed for high-performance on GPUs

Procedural Programming Language

- Texture lookups
- Procedurals
- Uv-transforms
- Projectors
- Noise functions
- Math functions

Declarative Material Definition

- Glossy reflection
- Transparency
- Translucency

Renderer

Rasterizer

Light loops / OIT

Raytracer

Trace N rays

Pathtracer

Ray-marching

MDL SDK 2020

Features

MDL 1.6

DB for MDL definitions

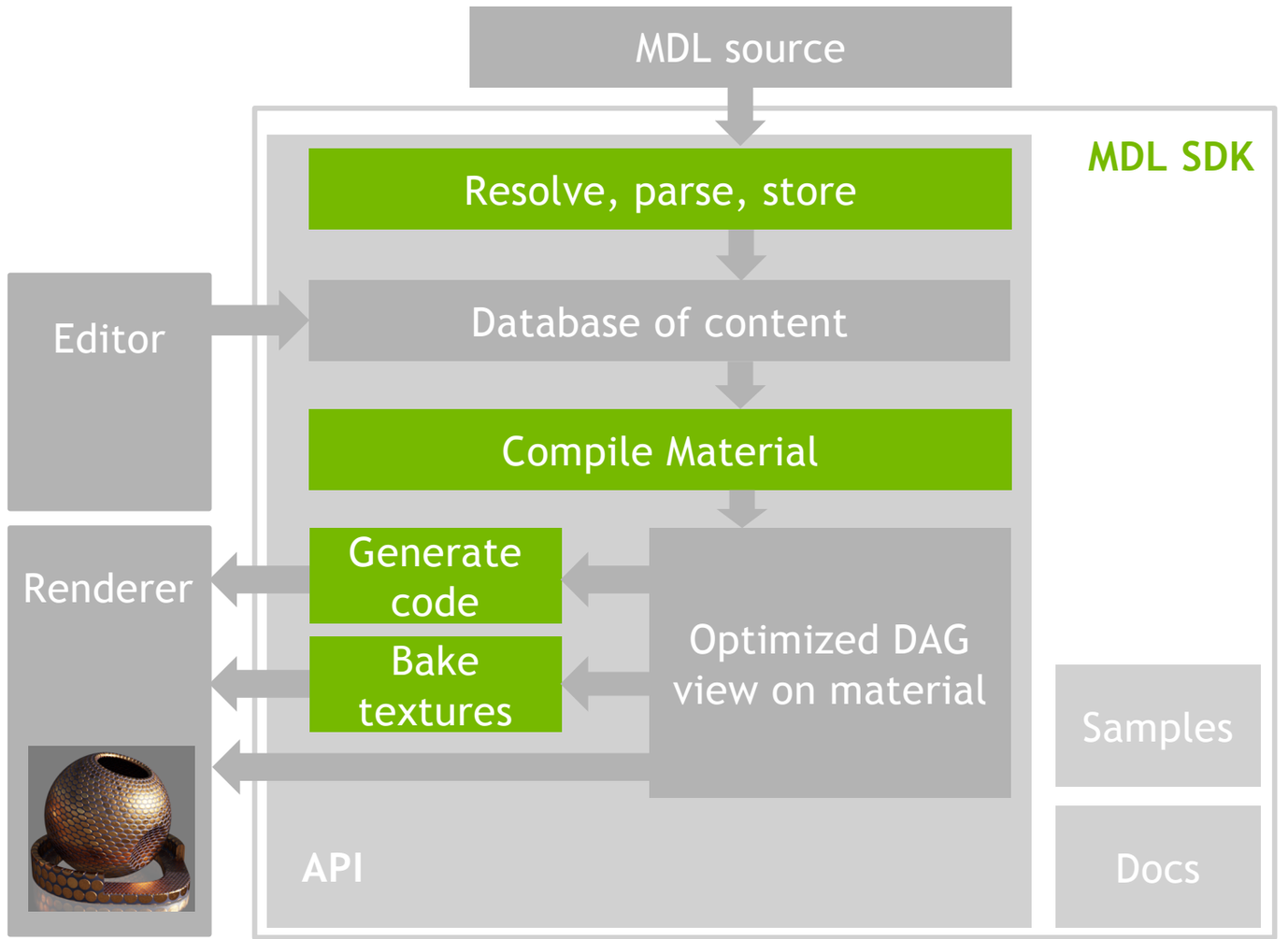
DAG view on materials
several compilation modes

MDL editing

Code generators
PTX, LLVM IR, x86, HLSL

Code examples

Documentation and tutorials



MDL SDK
Open Source Release

https://github.com/NVIDIA/MDL-SDK

12 releases shipped since SIGGRAPH 2018

BSD 3-clause license

Full MDL SDK

- Plus MDL Core API
- Minus distilling, baking, GLSL back-end
- Plus MDL Core Definitions and more

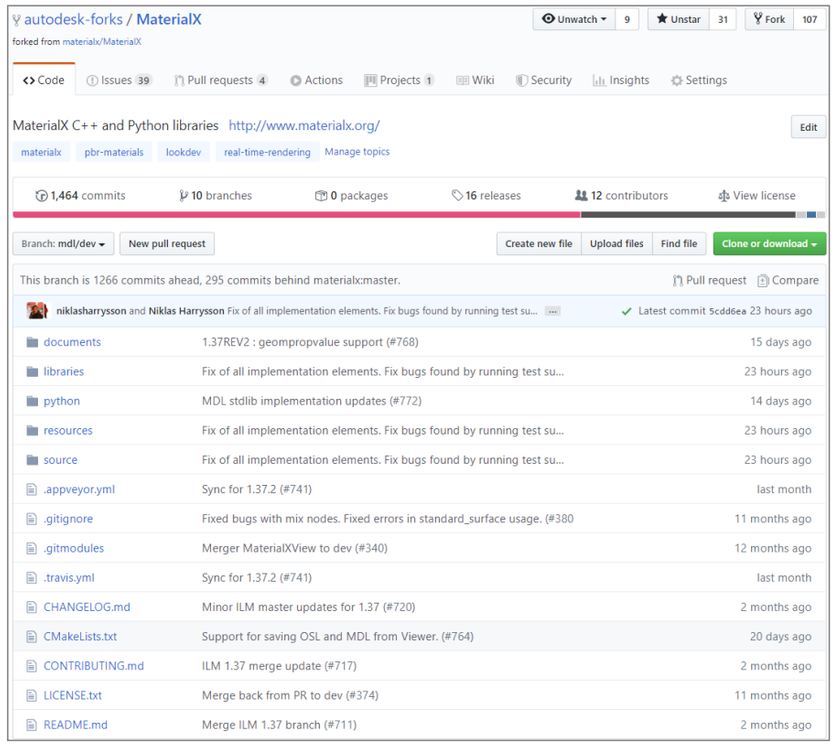
Feature image courtesy of Adobe, created by art director Vladimir Petkovic.

MaterialXGenMdl

New Library for MDL Code Generation

Open Source Release

https://github.com/materialx/MaterialX



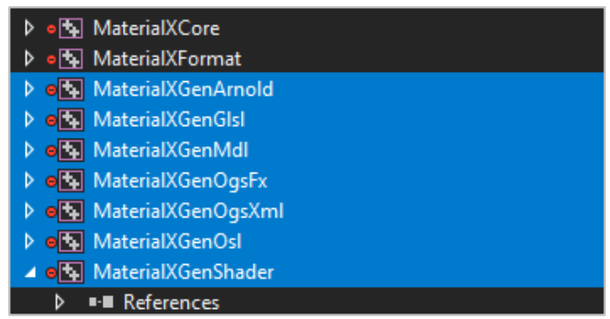
Joint development from

MaterialXGenMdl

New Library for MDL Code Generation

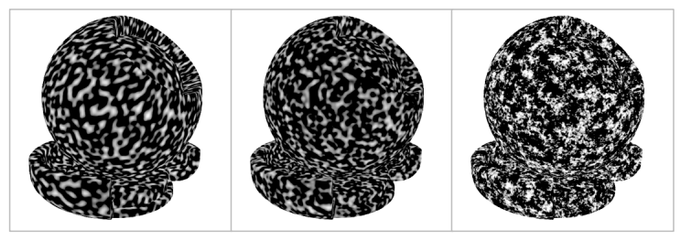
Class specializations to

- Define MDL language syntax
- Define MDL node implementations
- Define MDL shader generator



MDL modules impl. MaterialX's standard libraries

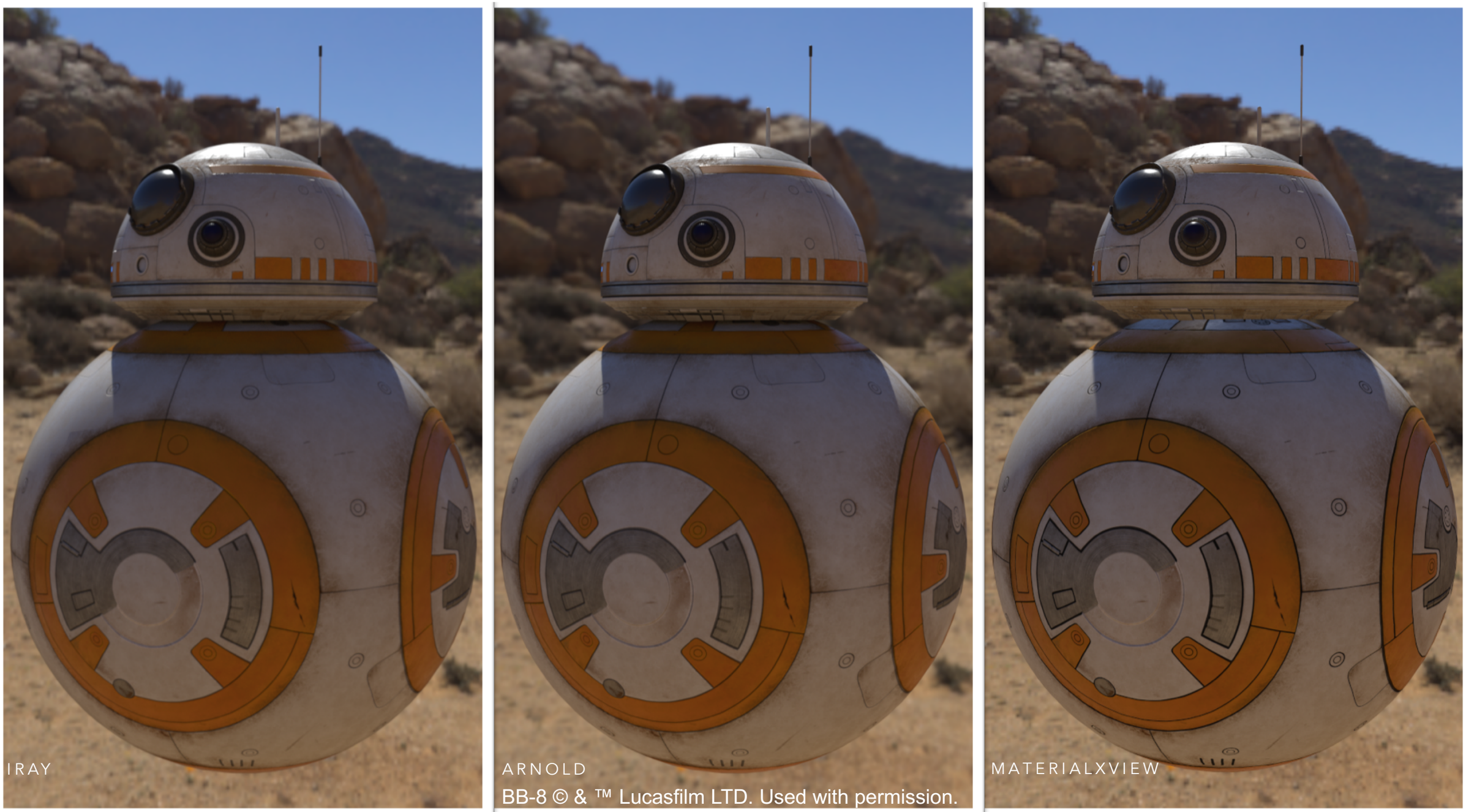
- stdlib.mdl: standard library nodes, incl. Gabor noise
- pbrlib.mdl: PBS nodes, wrapped as MDL materials



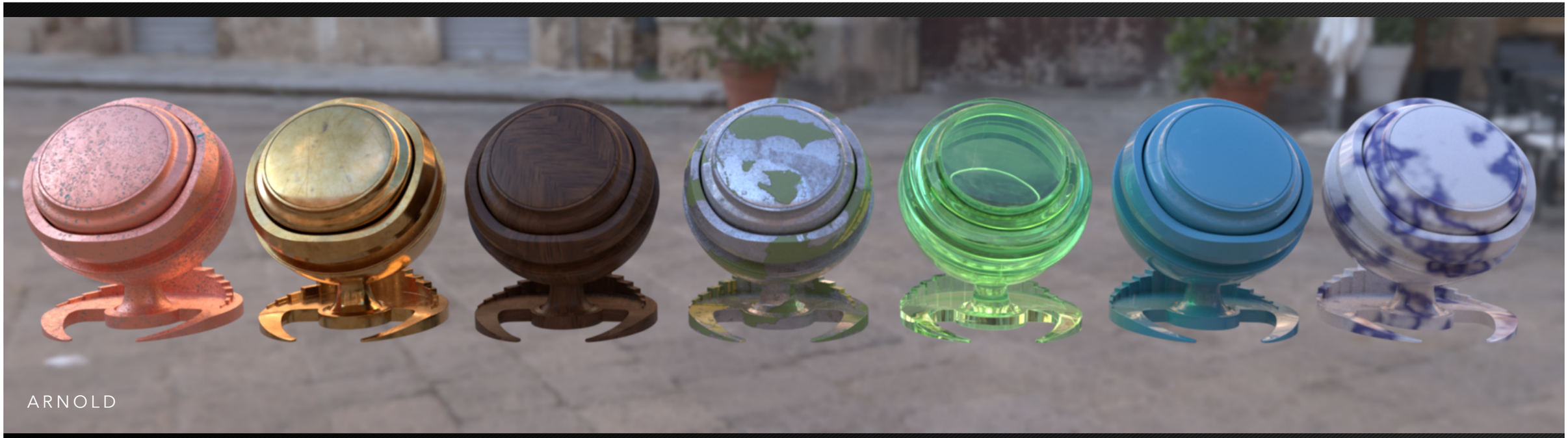
Details in GTC On Demand Talk S21469:

Material Interoperability Using MaterialX, Standard Surface, and MDL

www.nvidia.com/en-us/gtc/on-demand/?search=s21469

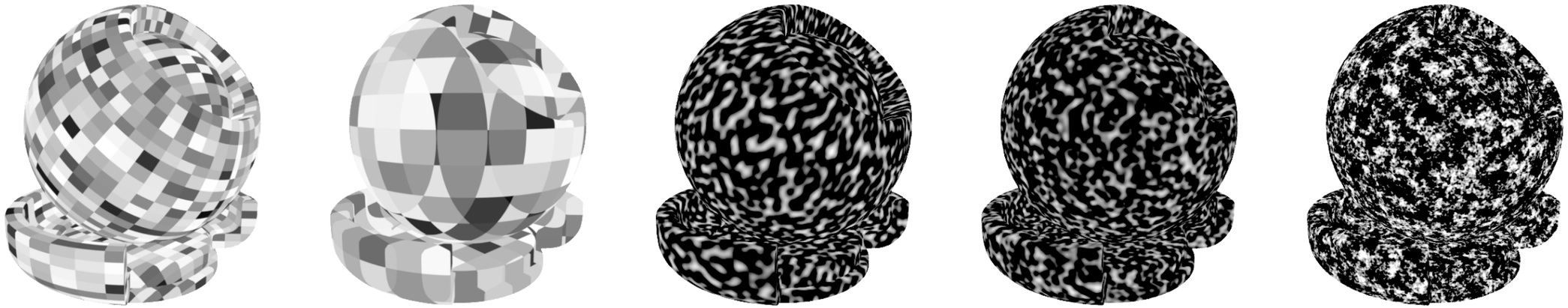


BB-8 © & ™ Lucasfilm LTD. Used with permission.

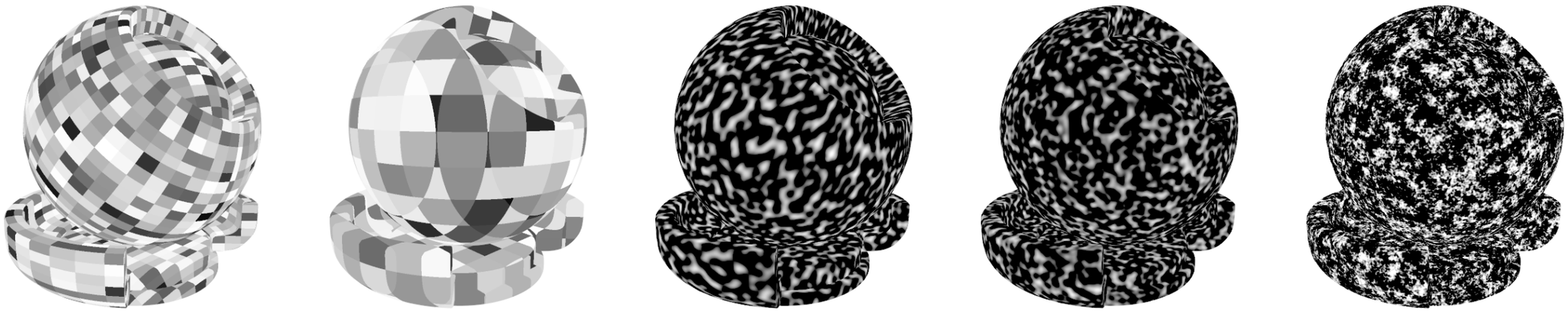

ARNOLD


IRAY

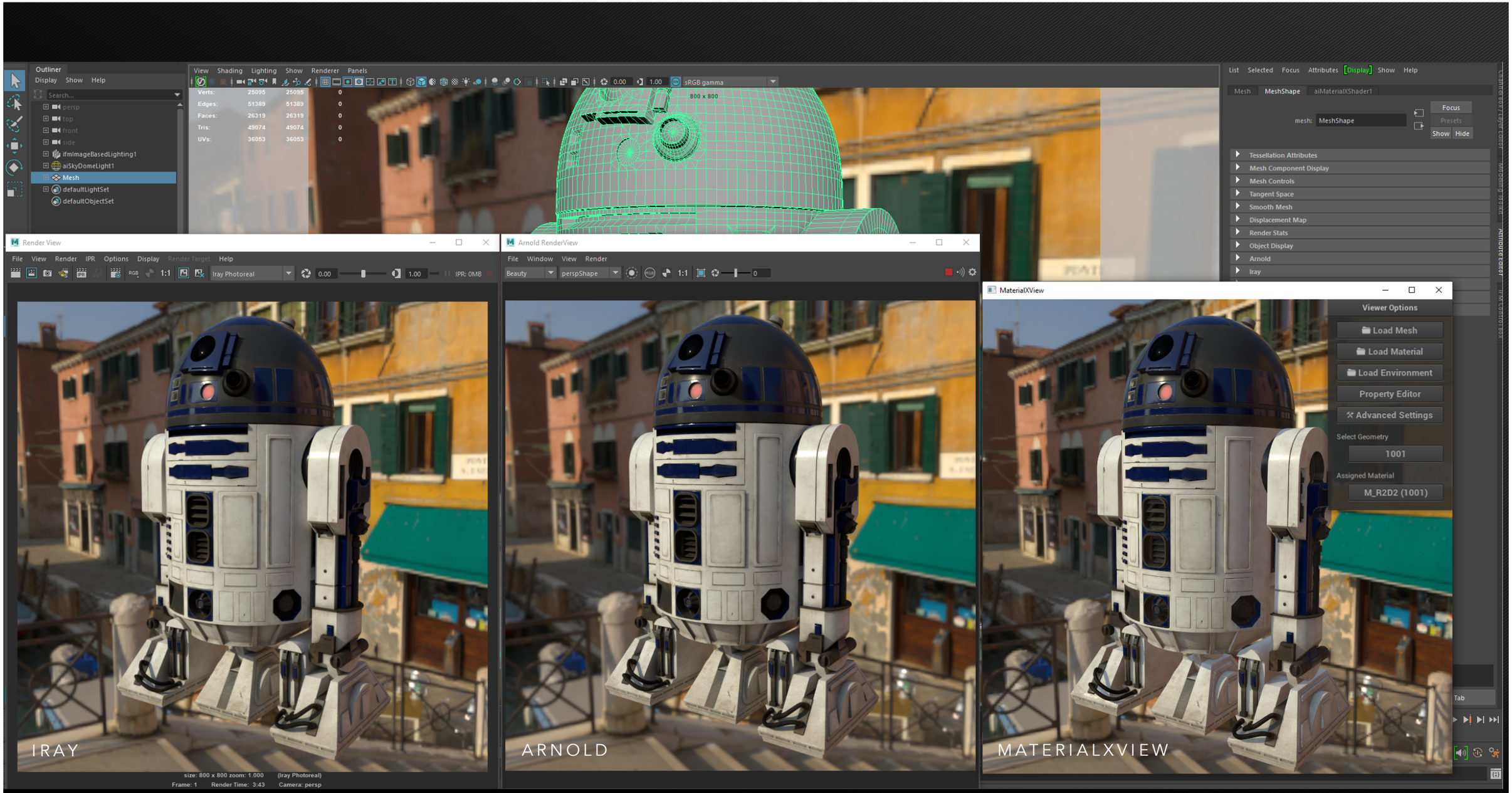
Noise Nodes



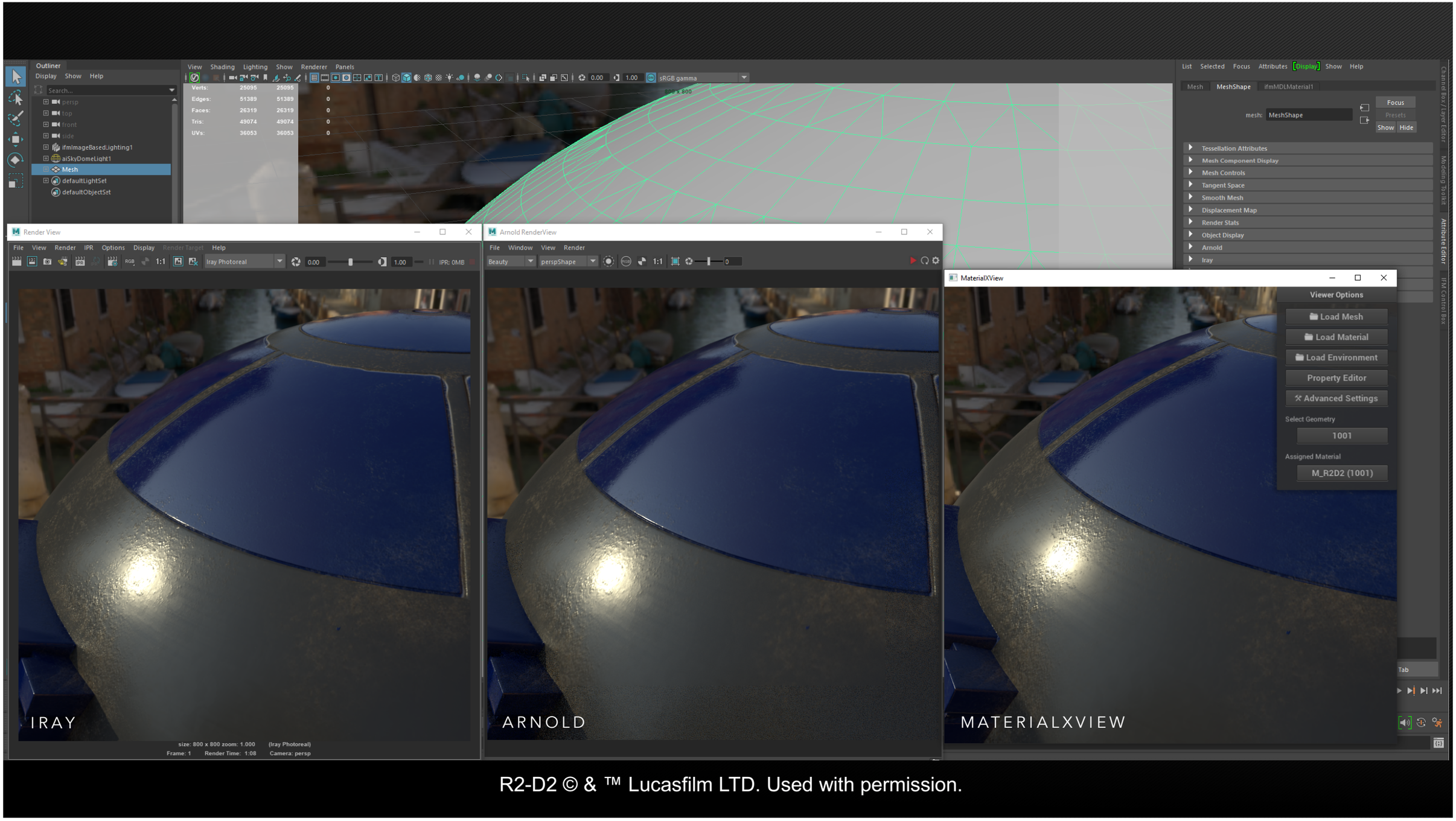
MDL in Iray



GLSL in MaterialXView



R2-D2 © & ™ Lucasfilm LTD. Used with permission.



R2-D2 © & ™ Lucasfilm LTD. Used with permission.

Limitations and Future Work

Limited support

- blur
- heighttonormal

Not yet implemented

- displacement
- subsurface_bsdf
- backfacing

Unsupported

- viewdirection
- fresnel
- volume edf
- add(bsdf/edf/vdf)
- sheen_brdf

Improvements with **MDL 1.7**

→ through code transforms of common patterns to MDL BRDF nodes

→ used for under-clearcoat emission → directional_factor for EDF

→ (partial) volume emission intensity, but no EDF

→ non-clamped mixer, varying weights for EDF mixer

→ non-diffuse base

Our Contributions

MDL Backend for MaterialX

MaterialXGenMdl

Open Source in MaterialX repo on

GitHub

Reference Implementations

Standard Library

PBS Nodes

Material Interoperability

MaterialX & Autodesk Standard Surface

↓

MDL eco-system

USD

NVIDIA Omniverse™ RTX Renderer

NVIDIA Iray®

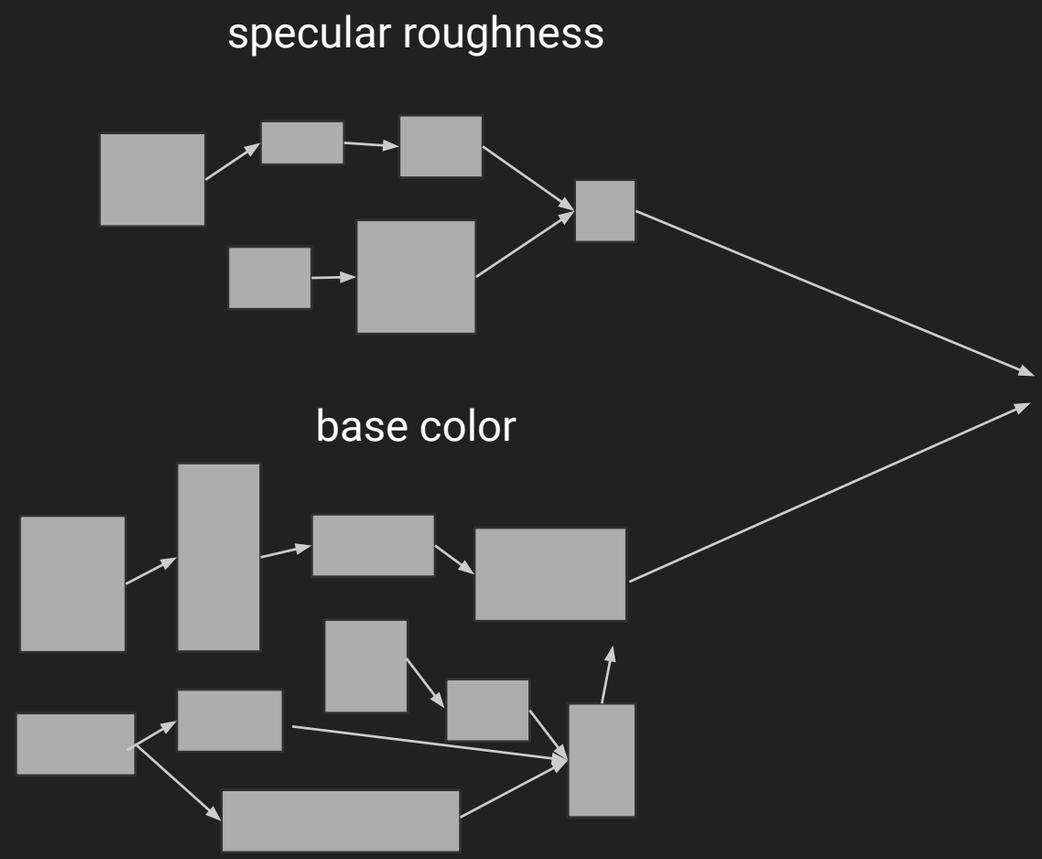
...

Fini

Making the Complex Simple with MaterialX Texture Baking

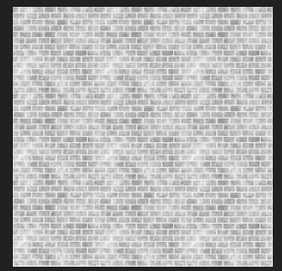
Madeleine Yip
Rendering Engineer Intern

A Complex Material…..

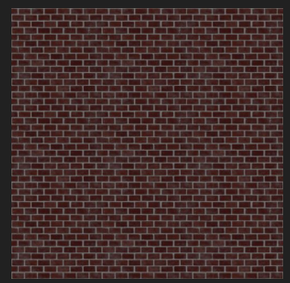


.....Made Simple

specular roughness



base color



TextureBaker class

- Accessed through MaterialXView window
- Multiple UDIM support when baking textures
- Python bindings for command-line use
  - Refer to MaterialX/python/Scripts/baketextures.py in the Github

Texture Baker Implementation Details

- Traces and rewires nodegraphs back to the output ports
- Points ShaderGenerator to the output ports to generate GLSL code
- Uses MaterialX GlslRenderer class to render to texture space

Reference BB8

Texture Baked BB8

4k image textures for comparison

Future for MaterialX Texture Baking

- Extend to OSL
- Data for potential inverse rendering projects
- Translation between shading models
  - Come to the upcoming Physically Based Shading course tomorrow for more details!
- Integrate into game engines
  - Translation from film quality assets in game
  - More seamless content pipeline
- Providing access to MaterialX content in tools without full support for graphs

A Preview of MaterialX v1.38

Material Nodes

Previously:

. Materials use special <material> elements, with <shaderref>s and <bindinput>s

```
<material name="gold">
  <shaderref name="SR_gold" node="standard_surface">
    <bindinput name="base" type="float" value="1"/>
    <bindinput name="base_color" type="color3" value="0.944, 0.776, 0.373"/>
    <bindinput name="specular" type="float" value="1"/>
    <bindinput name="specular_color" type="color3" value="0.998, 0.981, 0.751"/>
  </shaderref>
</material>
```


Material Nodes

v1.38:

. Materials are just regular nodes with regular inputs connecting to shader nodes

```
<nodedef name="ND_surfacemtl" node="surfacematerial"
  <input name="surfaceshader" type="surfaceshader" value=""/>
  <input name="displacementshader" type="displacementshader" value=""/>
  <output name="out" type="material"/>
</nodedef>


<surfacematerial name="gold">
  <input name="surfaceshader" type="surfaceshader" nodename="goldsrf"/>
  <input name="displacementshader" type="displacementshader" nodename="dsp"/>
</surfacematerial>
```


Material Nodes

Previously:

```
<material name="gold">
  <shaderref name="SR_gold" node="standard_surface">
    <bindinput name="base" type="float" value="1"/>
    <bindinput name="base_color" type="color3" value="0.944, 0.776, 0.373"/>
    <bindinput name="specular" type="float" value="1"/>
    <bindinput name="specular_color" type="color3" value="0.998, 0.981, 0.751"/>
  </shaderref>
</material>
```

v1.38:

```
<standard_surface name="goldsrf">
  <input name="base" type="float" value="1"/>
  <input name="base_color" type="color3" value="0.944, 0.776, 0.373"/>
  <input name="specular" type="float" value="1"/>
  <input name="specular_color" type="color3" value="0.998, 0.981, 0.751"/>
</standard_surface>

<surfacematerial name="gold">
  <input name="surfaceshader" type="surfaceshader" nodename="goldsrf"/>
</surfacematerial>
```


Material Nodes

Advantages:

- Use standard API to connect nodes and inputs and "walk the nodegraph"
  - No special getShaderRefs(), getPrimaryShaderXX() or getBindParams() / getBindInputs() functions required!

Material Nodes

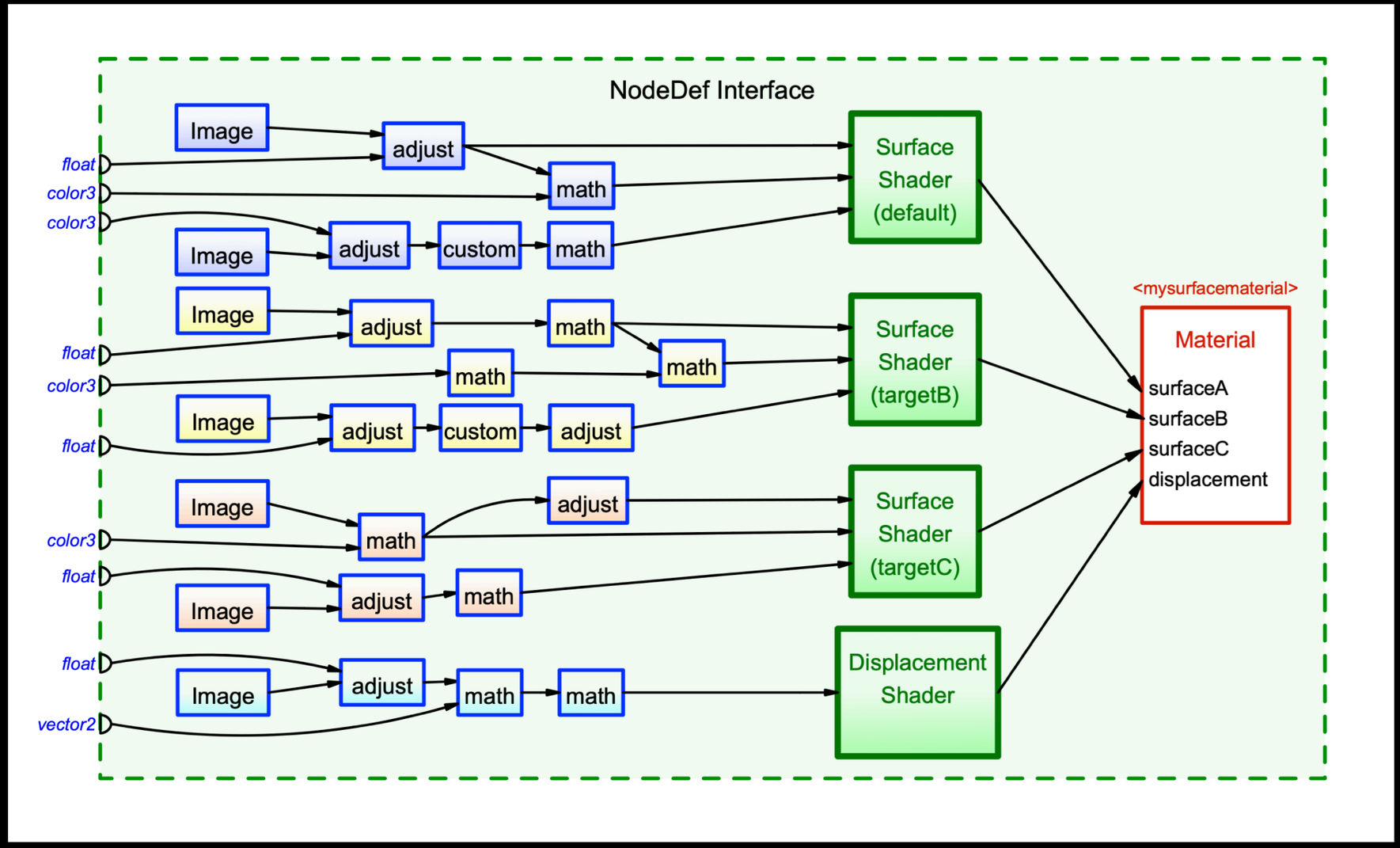
Advantages:

. Use standard API to connect nodes and inputs and "walk the nodegraph"

. Can define arbitrary "material"-output nodes

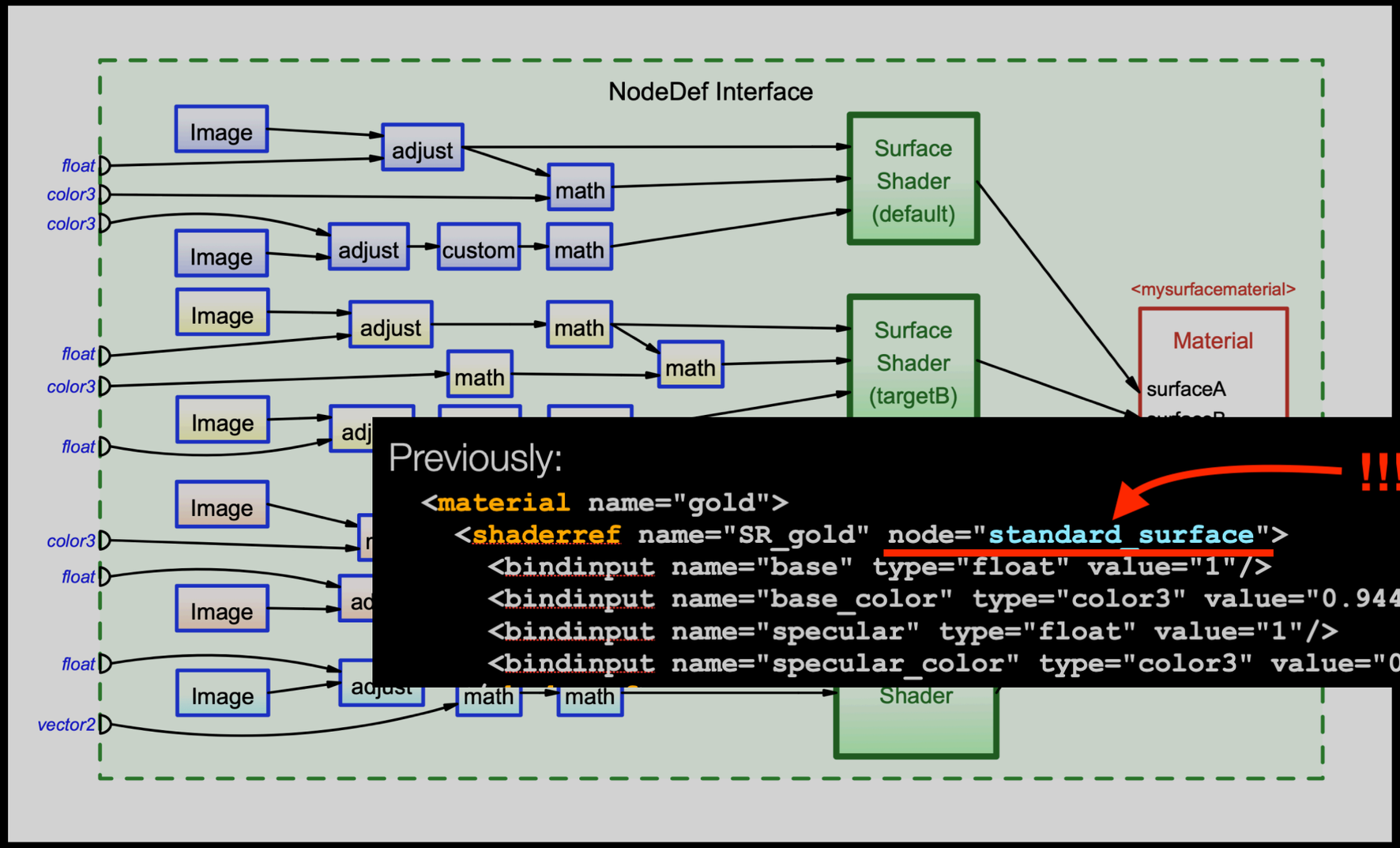
```
<nodedef name="ND_mysurfacemtl" node="mysurfacematerial"
  <input name="surfaceshader" type="surfaceshader" value=""/>
  <input name="rmansurfaceshader" type="surfaceshader" value="" target="rman"/>
  <input name="previewsurfaceshader" type="surfaceshader" value="" target="glsl"/>
  <input name="displacementshader" type="displacementshader" value=""/>
  <output name="out" type="material"/>
</nodedef>

<mysurfacematerial name="gold">
  <input name="surfaceshader" type="surfaceshader" nodename="goldsrf"/>
  <input name="rmansurfaceshader" type="surfaceshader" nodename="rmangoldsrf"/>
</mysurfacematerial>
```


Material Nodes



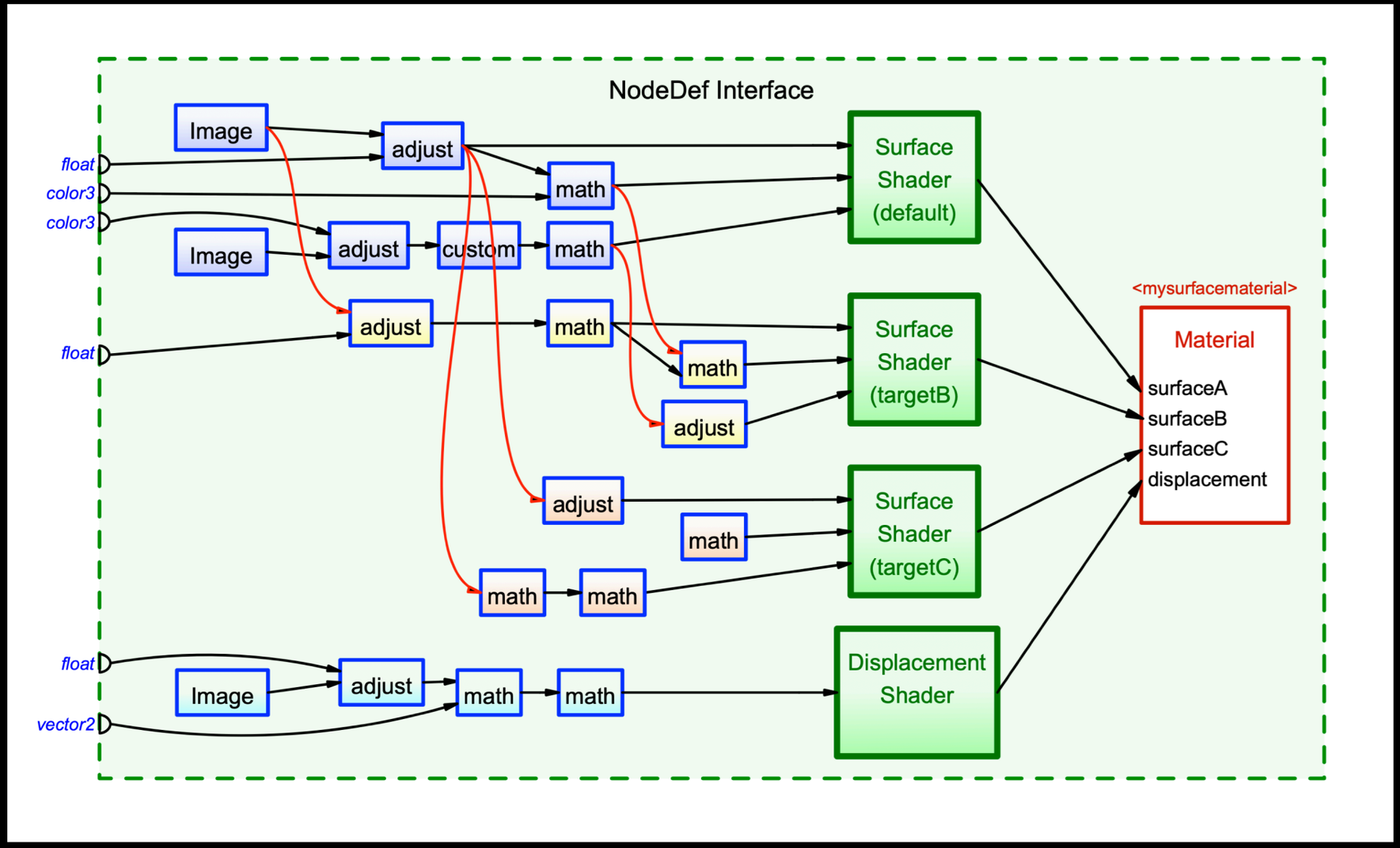
All-in-one "Super Material": one thing, can be instanced and given values directly and used in a Look

Material Nodes



Unlike the previous shaderref method, instances don't need to know internal shader node names

Material Nodes



Use nodes to convert shared inputs for different shader parameterizations

Formalized Target Definition

```
<input name="previewsurfaceshader> type="surfaceshader"
       value="" target="glsl"/>
```

1.38:

```
<targetdef name="glsl"/>
<targetdef name="oslpattern"/>
<targetdef name="osl" inherit="oslpattern"/>
```

. Inheritance allows separation of OSL closures and patterns

Physically Based Shader Layering

MaterialX Physically-Based Shading Nodes

Niklas Harrysson - niklas.harrysson@autodesk.com
Doug Smythe - smythe@ilm.com
Jonathan Stone - jstone@lucasfilm.com
July 16, 2019
(Revised October 17, 2019)

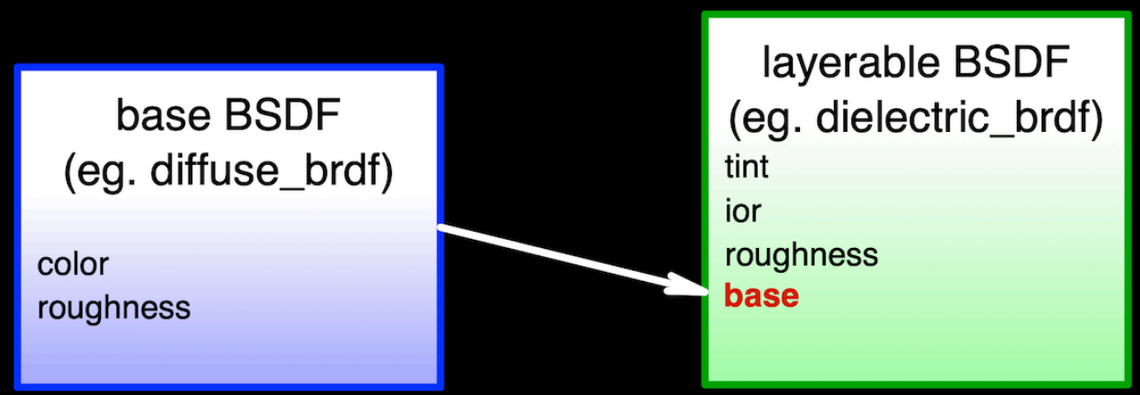
Introduction

The MaterialX Specification describes a number of standard nodes that may be used to construct node graphs for the processing of images, procedurally-generated values, coordinates and other data. With the addition of user-defined custom nodes, it is possible to describe complete rendering shaders using node graphs. Up to this point, there has been no standardization of the specific shader-semantic nodes used in these node graph shaders, although with the widespread shift toward physically-based shading, it appears that the industry is settling upon a number of specific BxDF and other functions with standardized parameters and functionality.

This document describes a number of shader-semantic nodes implementing widely-used surface, scattering, emission and volume distribution functions and utility nodes useful is constructing complex layered rendering shaders using node graphs. These nodes in combination with other nodes may be used with the MaterialX shader generation (ShaderGen[1]) system.

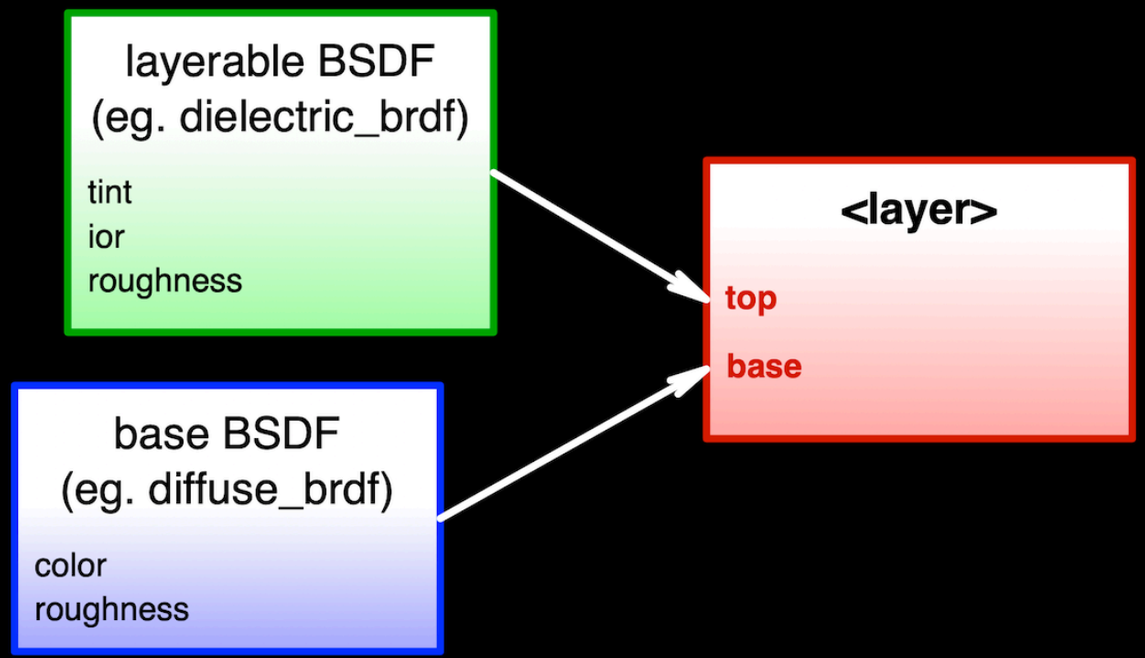
Physically Based Shader Layering

- Original setup: "base" input for vertically-layered "top" layer nodes



Physically Based Shader Layering

. v1.38 setup: new <layer> node with "base" and "top" inputs



Node Parameters Are Now Inputs

. Before: many float/color/vector node arguments were unconnectable <parameter>s

. v1.38: Almost all float/color/vector node arguments are now <input>s

Moreover…

. Having separate parameters and inputs is annoying, leads to code duplication

Node Parameters Are Now Uniform Inputs

- Before: many float/color/vector node arguments were unconnectable <parameter>s
- v1.38: Almost all float/color/vector node arguments are now <input>s
- Having separate parameters and inputs is annoying, leads to code duplication

- v1.38: The <parameter> element itself is now removed
  - Replaced by <input> with a new **`uniform`** attribute in the nodedef:

```
<nodedef name="ND_image_float" node="image">
  <input name="file" type="filename" uniform="true"/>
  ...
```


Color2 Type

- Gray scale + mask
- No one uses this type, burden to support

New Nodes

<updirection>	world-space "up vector"
<round>	round to nearest integer
<safepower>	"power" functions that work with negative values
<curvelookup>	bridge to supporting "ramp" type inputs

New Nodes

<updirection>
<round>
<safepower>
<curvelookup>

<atan2>	inputs now called "iny" and "inx"
<switch>	can now take 10 inputs
<normalmap>	"scale" input can be float or vector2

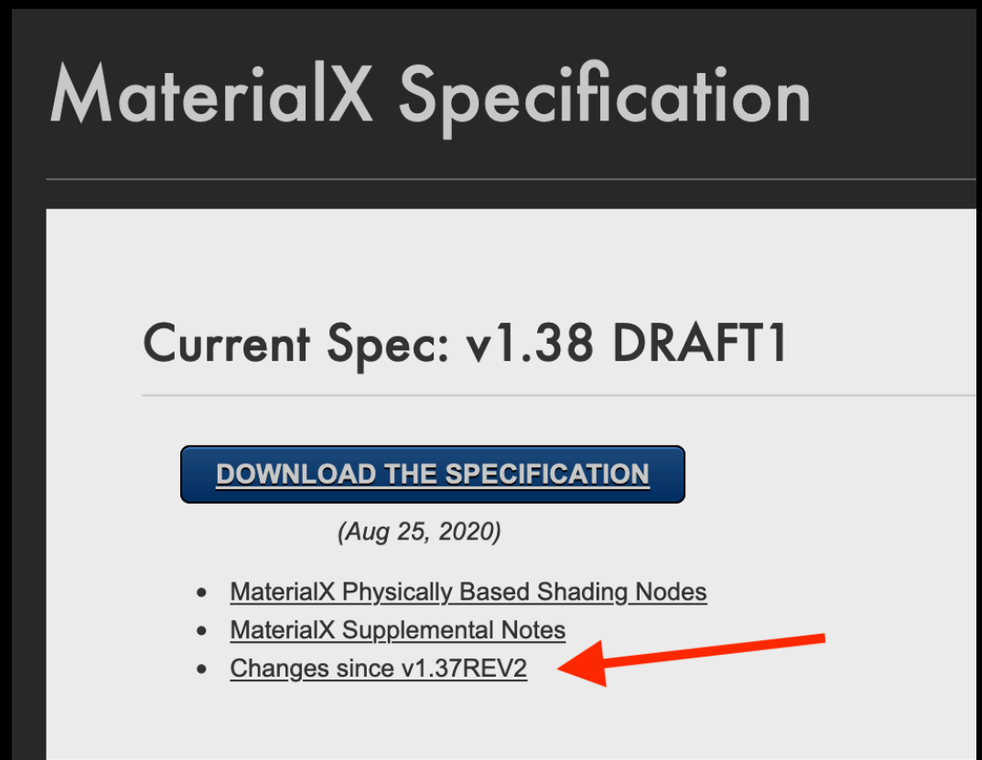
Organizational Elements and Attributes

```
<nodedef name="ND_multinoise" node="multinoise">
  <uifolder name="ui_noise" uifolder="Noise" doc="Noise Controls">
  <uifolder name="ui_noiselg" uifolder="Noise/Large" doc="Large Scale Noise">
  ...
```

- All nodes may now specify width and height attributes
- <backdrop> : new minimized attribute
- Standalone <nodegraph> can now have <input>s

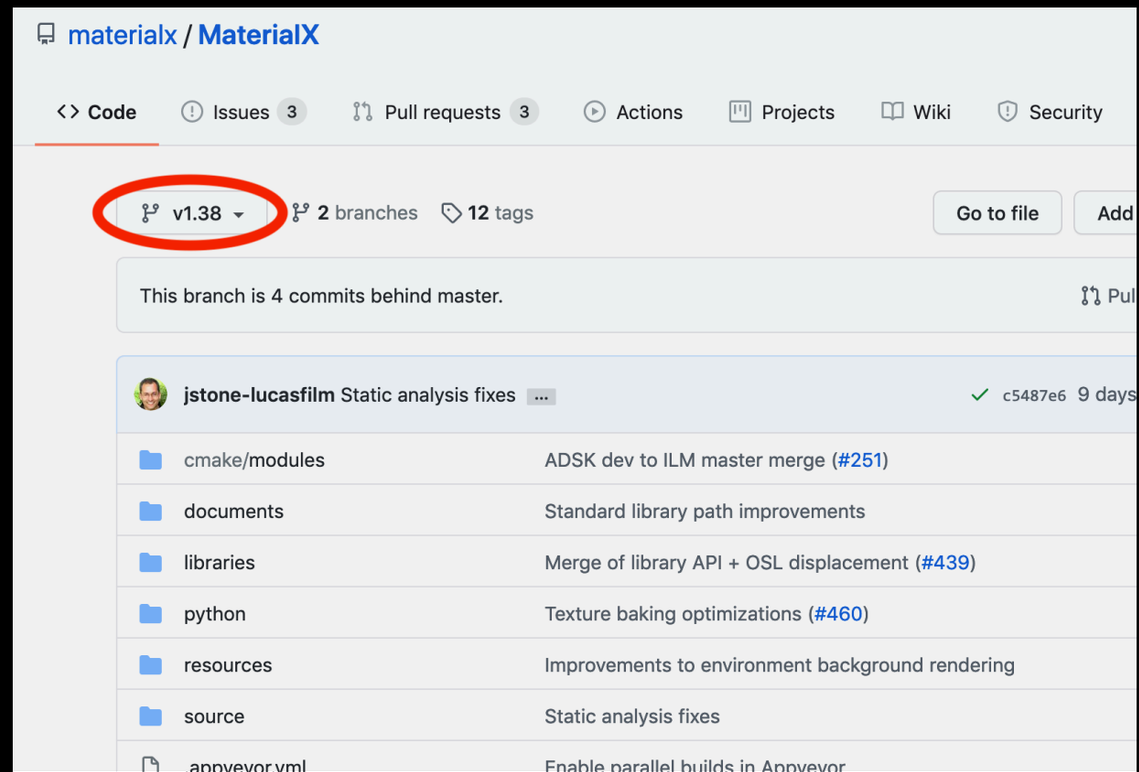
Other Changes

- New **`sourcecode`** attribute for <implementation>
- **`language`** <implementation> attribute removed, new **`format`** attribute added
- Custom attributes can now be exported as metadata in generated shaders
- Additional misc changes



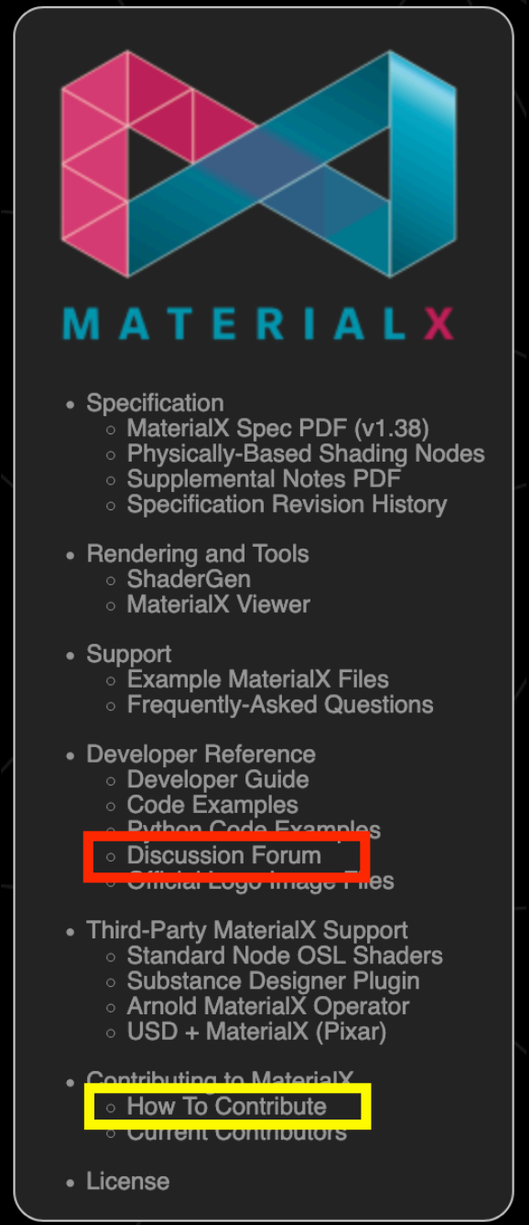
The 1.38 Library

- Still in progress, no firm release date
- Development in separate 1.38 branch
- Possible minor changes to Specification

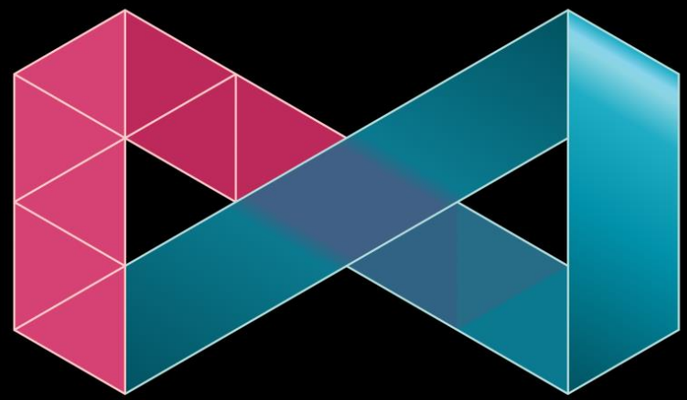


Feedback and Contribution

- Join the Discussion Forum through materialx.org and tell us what you'd like to see added
- Outside developers – we'd love your help!



Thanks!



MaterialX in USD/Hydra

Eliot Smyrl
Pixar Animation Studios

Existing MaterialX support in USD

- MaterialX graphs can be expressed in UsdShade schema
- UsdMtlx file format plugin can reference mtlx files directly from USD; internally translated to UsdShade
- USD is thus completely usable as interchange medium for MaterialX

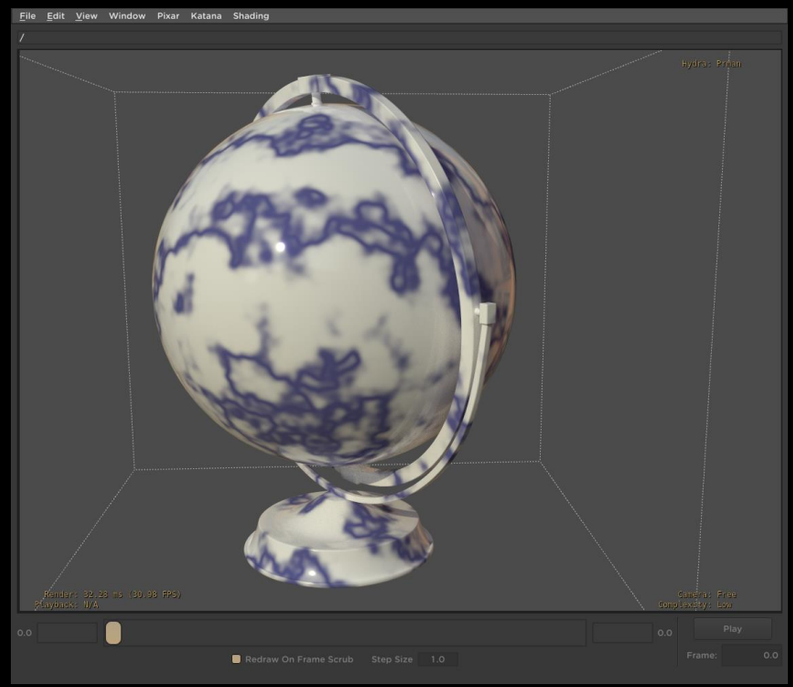
New project: MaterialX in USD/Hydra

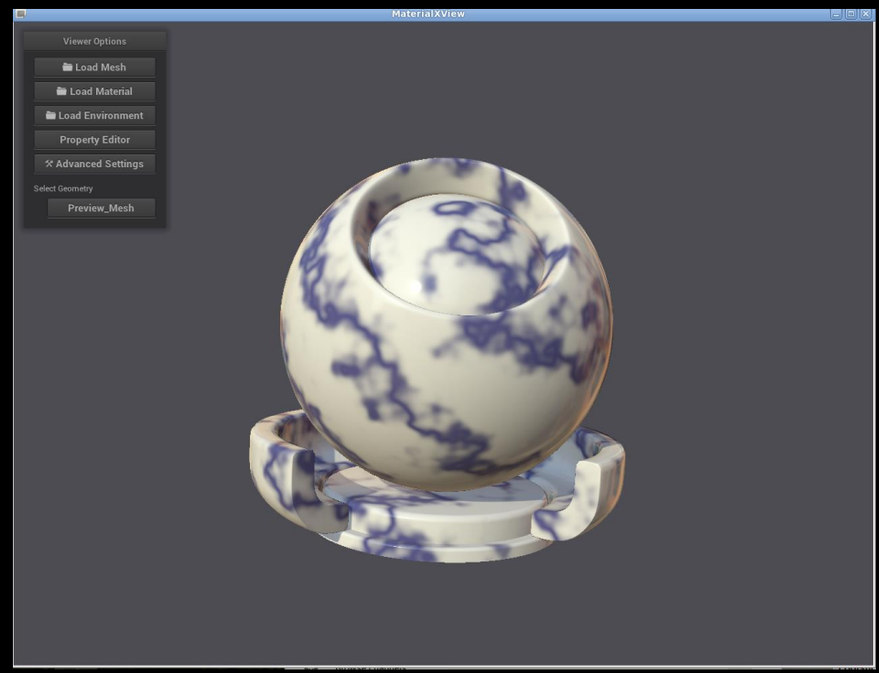
- Funding support provided by Epic MegaGrants

- Hydra: imaging framework shipping with USD distribution
- Multiple frontends (consuming scene data), multiple backends (driving renderers)
- Backends – hdStorm (GL renderer), hdEmbree, hdPrman, …

New project: MaterialX in USD/Hydra

- First phase: render in hdPrman backend using MaterialX shading





hdPrman MaterialX implementation

- Traverse incoming USD shading graph
- Recognize MaterialX shading networks
- Use MaterialXGenShader to generate one OSL node for each pattern network
- But prman/RIS only supports C++ for terminal nodes, so:
- Recognize specific MaterialX terminals (UsdPreviewSurface, StandardSurface) and replace them with PxrSurface-plus-adapter

Features

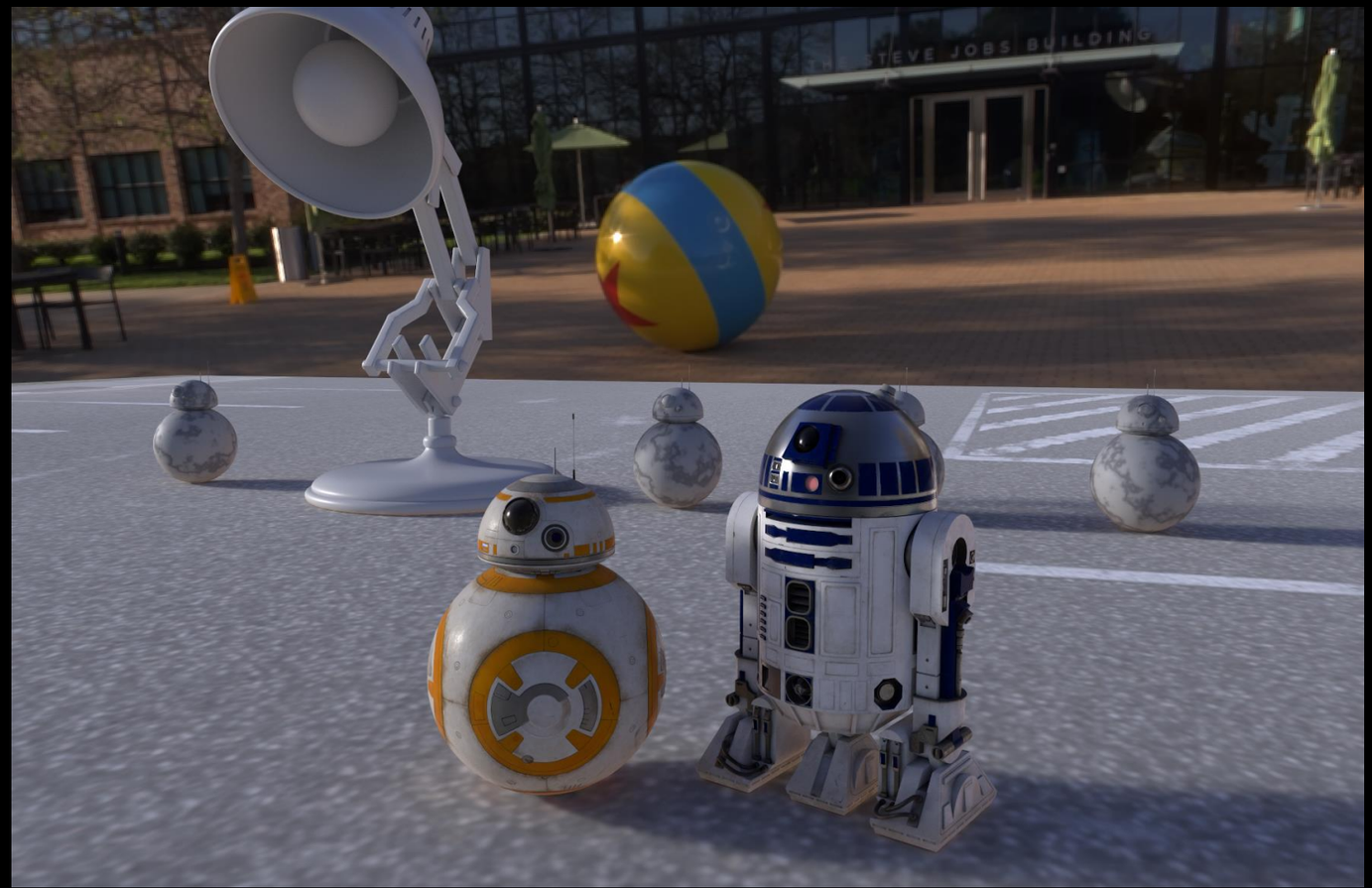
- Supports arbitrary pattern graphs
- Graphs can be authored as either USD or mtlx files

Limitations

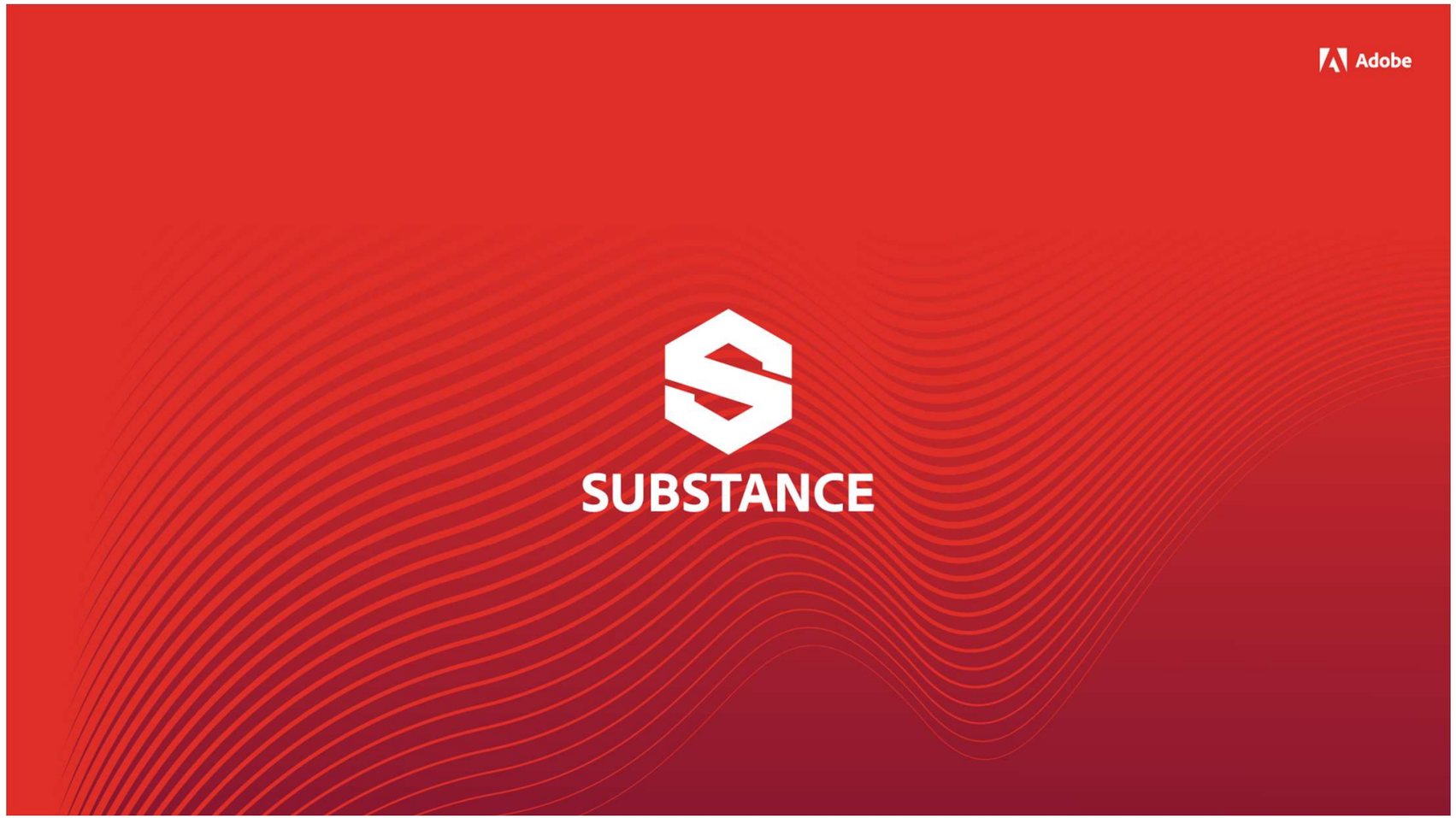
- Each terminal must be handled as a special case
- Does not support graphs of PBS lobe nodes to build up custom BXDFs

Future Work

- Handle custom nodedefs, displacement, volumes
- MaterialX support in hdStorm (GL renderer)
- Pattern baking into UV or ptex textures from OSL generated code



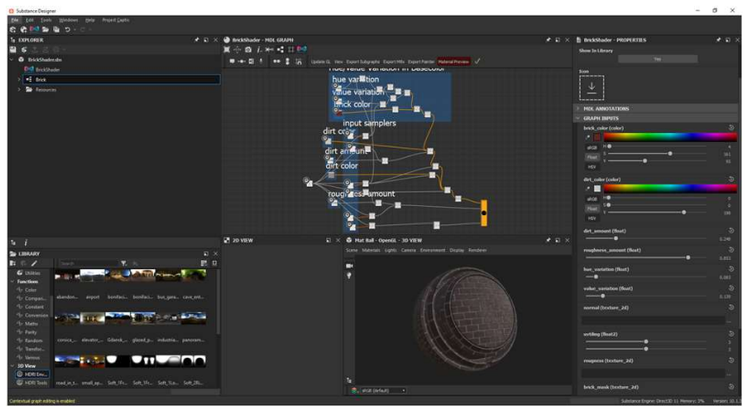
BB-8 and R2-D2 © & TM Lucasfilm Ltd. Used with Permission.


Adobe
SUBSTANCE

MaterialX and Substance

SUBSTANCE

Adobe



MaterialX Plugin for Substance Designer

SUBSTANCE

Adobe

Overview

- Presented at last SIGGRAPH
- Released on Substance Share in April
- Still in Beta

SUBSTANCE

Adobe

Shaders

- Build shaders using most of nodes in MaterialX Standard Library
- iRay and OpenGL Viewport Support
- Limitation: No support for BSDF nodes (Standard Surface only)

SUBSTANCE

Adobe

Integration with texture workflows

- Bind procedural textures from Designer
- Texture and shader procedurals work together

SUBSTANCE

Adobe

Substance Painter Export

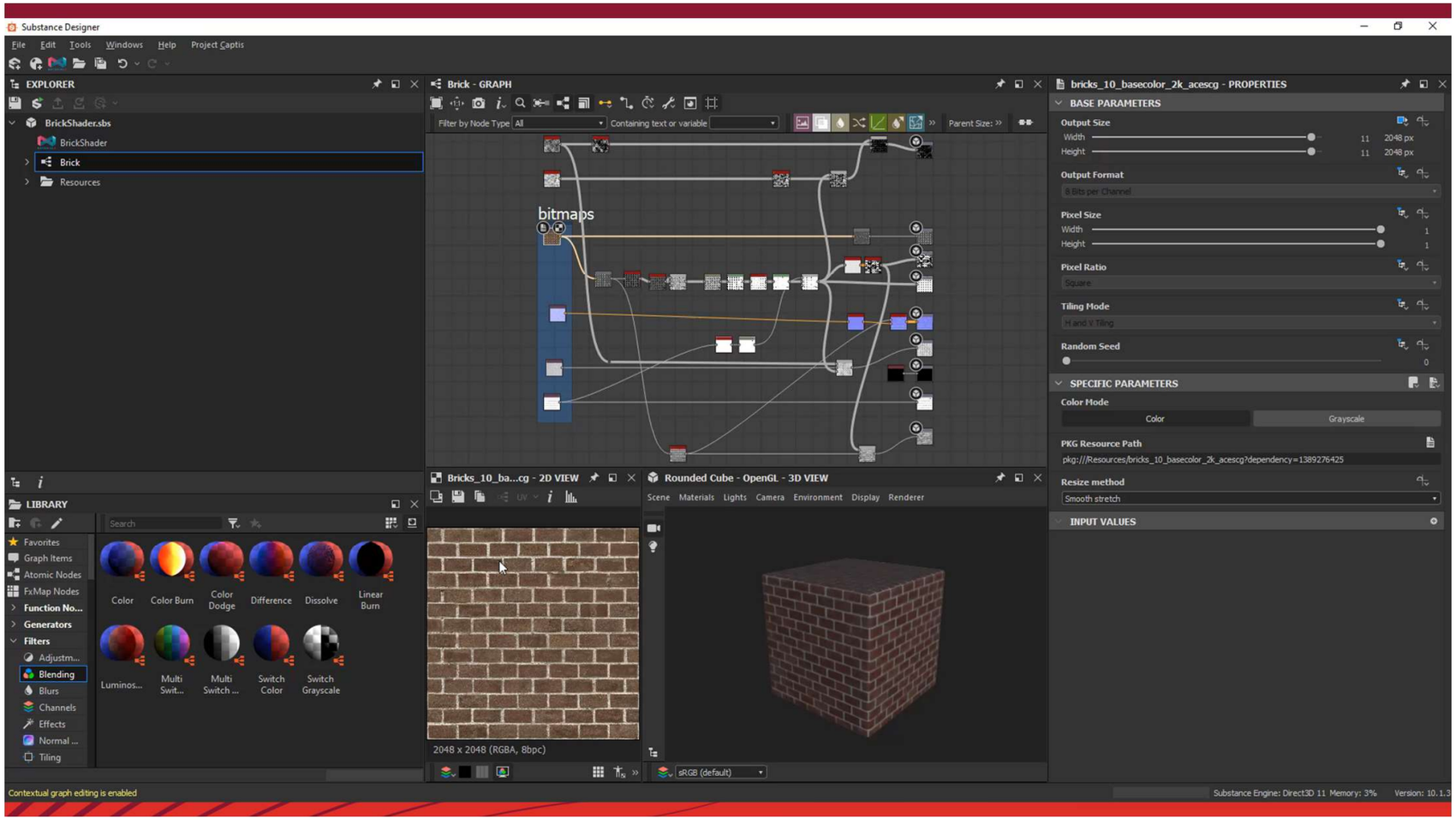
- Generates shaders compatible with painter
- Access textures from the layer stack
- Access textures and parameters specific to the shader

SUBSTANCE

Adobe

MaterialX Export

- Export MaterialX documents from shaders
- Export bound textures from Designer texture graphs


Substance Designer
File Edit Tools Windows Help Project Captis
EXPLORER
BrickShader.sbs
BrickShader
Brick
Resources
Brick - GRAPH
Filter by Node Type All
Containing text or variable
Parent Size:
bitmaps
bricks_10_basecolor_2k_acescg - PROPERTIES
BASE PARAMETERS
Output Size
Width 11 2048 px
Height 11 2048 px
Output Format
8 Bits per Channel
Pixel Size
Width 1
Height 1
Pixel Ratio
Square
Tiling Mode
H and V Tiling
Random Seed
0
SPECIFIC PARAMETERS
Color Mode
Color
Grayscale
PKG Resource Path
pkg:///Resources/bricks_10_basecolor_2k_acescg?dependency=1389276425
Resize method
Smooth stretch
INPUT VALUES
LIBRARY
Search
Favorites
Graph Items
Atomic Nodes
FxMap Nodes
Function No...
Generators
Filters
Adjustm...
Blending
Blurs
Channels
Effects
Normal ...
Tiling
Color
Color Burn
Color Dodge
Difference
Dissolve
Linear Burn
Luminos...
Multi Swit...
Multi Switch ...
Switch Color
Switch Grayscale
Bricks_10_ba...cg - 2D VIEW
2048 x 2048 (RGBA, 8bpc)
Rounded Cube - OpenGL - 3D VIEW
Scene Materials Lights Camera Environment Display Renderer
sRGB (default)
Contextual graph editing is enabled
Substance Engine: Direct3D 11 Memory: 3%
Version: 10.1.3

SUBSTANCE

Adobe

Future Work

SUBSTANCE

Adobe

Shader Generation using BSDF nodes

- Use any MaterialX shader in Designer and Painter
- Currently uses custom GLSL generation for Painter/Designer
- Use the new MDL generator for iRay/MDL editor integration

SUBSTANCE

Adobe

Embed MaterialX in Substance Materials

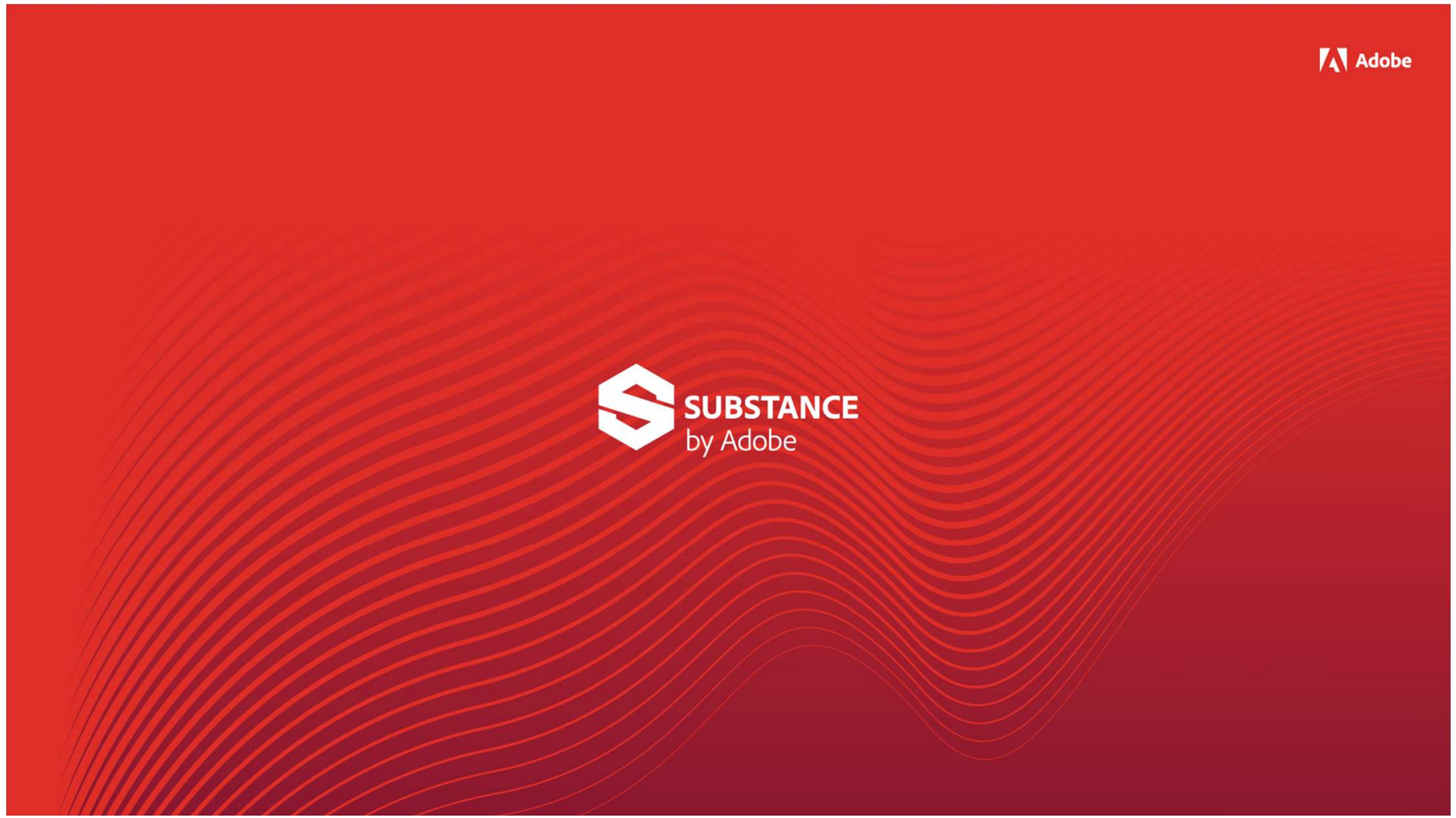
- Provide a full shader specification inside SBSAR
- Encode as metadata

SUBSTANCE

Adobe

Bind Substance Materials in MaterialX Graphs

- Allow using SBSAR's in MaterialX Graph
- Extract nodedef from SBSAR interface
- Set procedural parameters
- Bake to texture for compatibility with applications not familiar with SBSAR nodes


Adobe
SUBSTANCE
by Adobe

MaterialX and LookdevX Update

Nikola Milosevic, Bernard Kwok

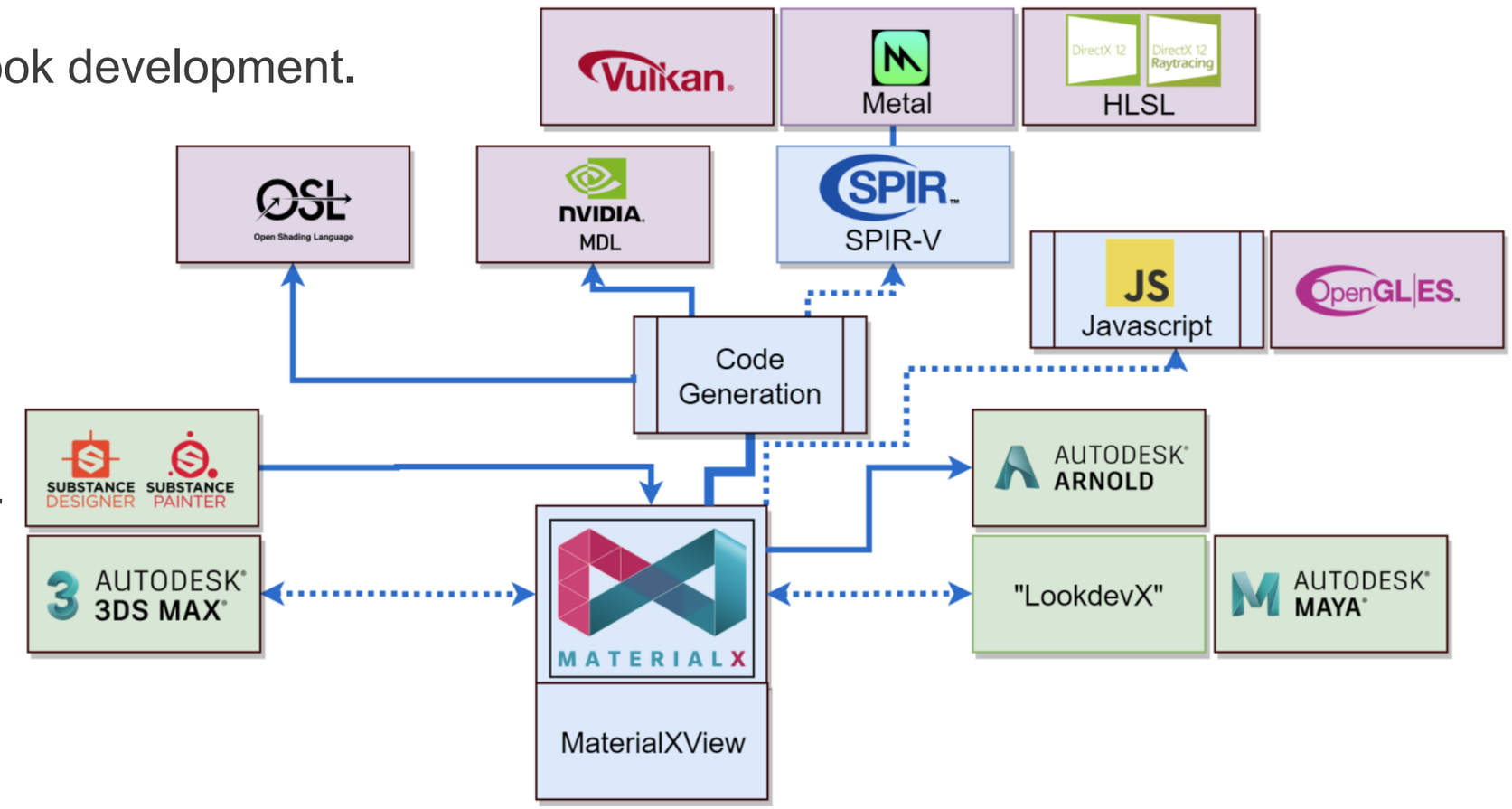
Principal Product Designer, Principal Engineer

AUTODESK.

Overview

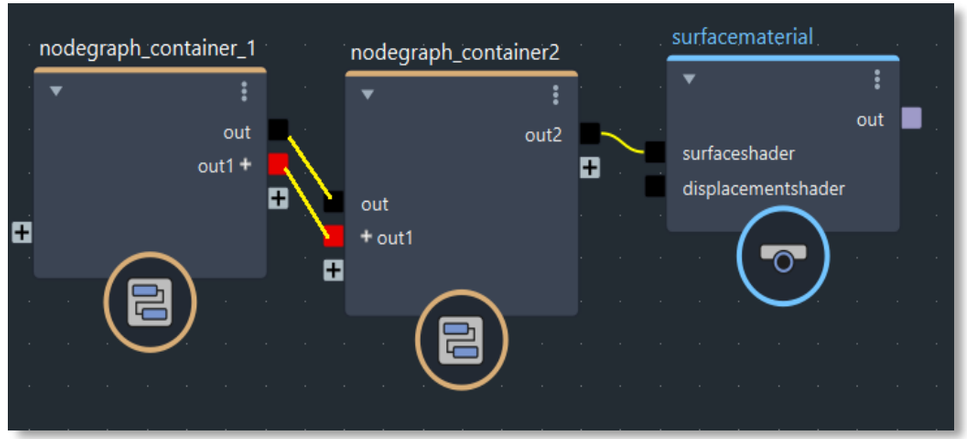
Focus

1. MaterialX Core features.
2. Node editing and look development.
3. Tools and renderer interoperability.
4. MaterialX shader generation, and real-time rendering.

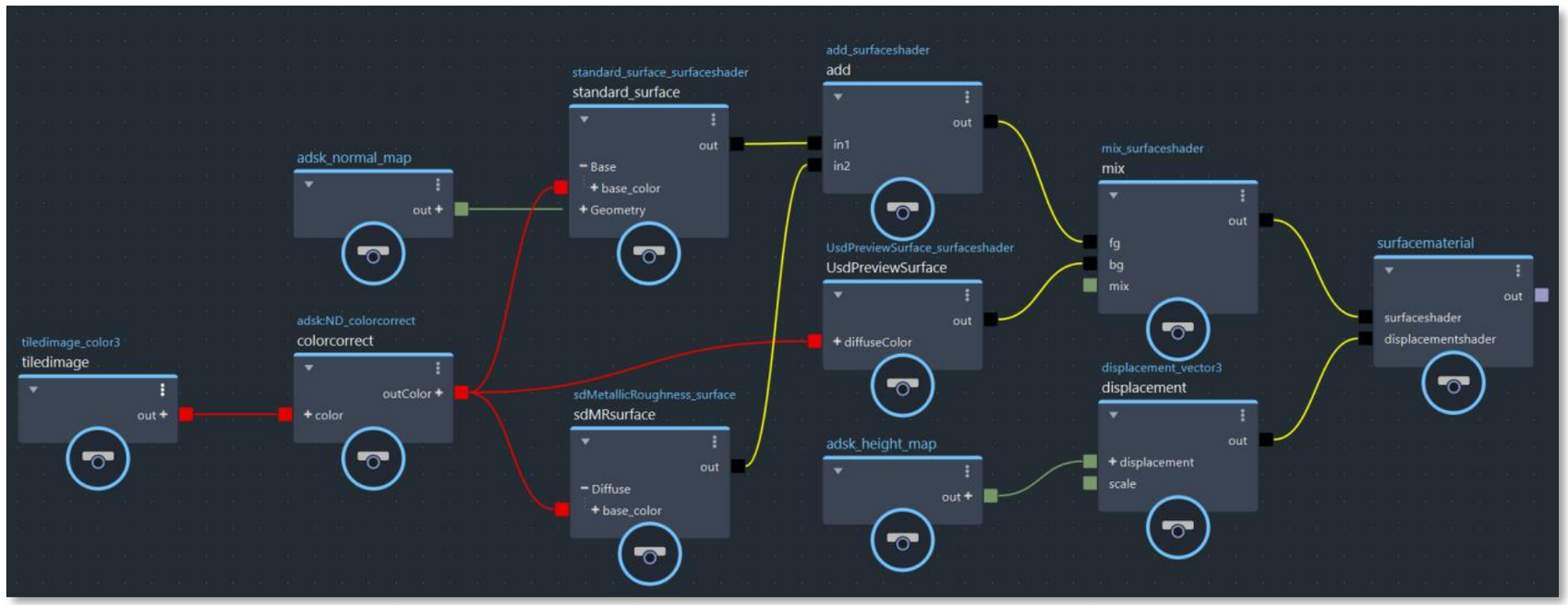


Runtime Data Model

- 1.37.1 / 1.38 updates for runtime shader graphs.
- Material nodes, nodegraphs as containers, <nodedef> publishing, code generation.
- Core libraries simplification:
  - stdlib, pbrlib, bxdf
- Uniform multiple library support:
  - Autodesk (adsk)
  - Others

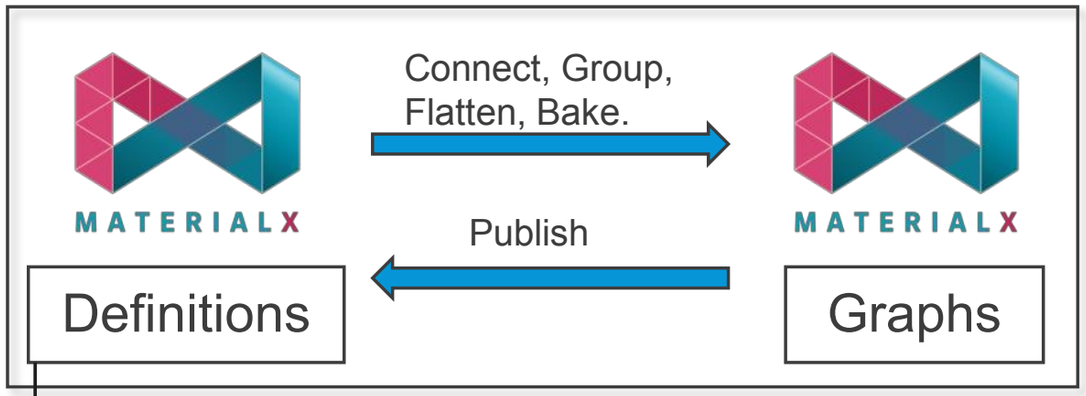


1.38 <nodegraph> pipeline configuration into a final surfacematerial..



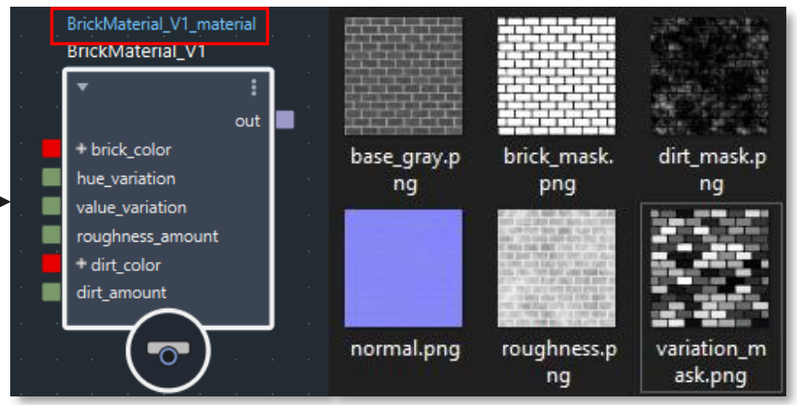
Sample shader graph composed of core, Autodesk, and Substance Designer MaterialX Plug-In definitions.

Managing Complexity

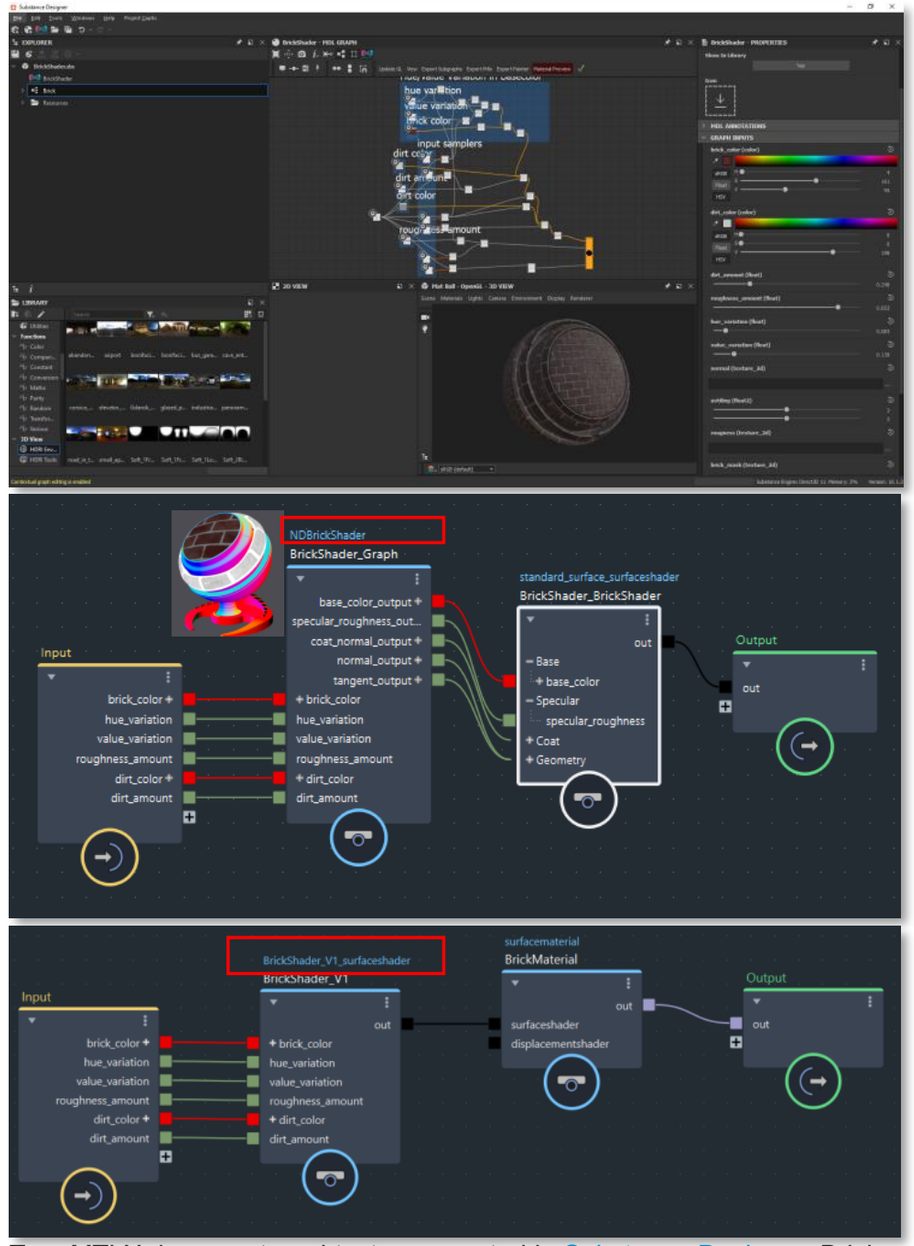


<nodedef>, <nodegraph> characteristics:

- Interface exposure: <input>, <output>.
- “version” / “namespace” / “target” tags
- Units / color management / user meta-data.

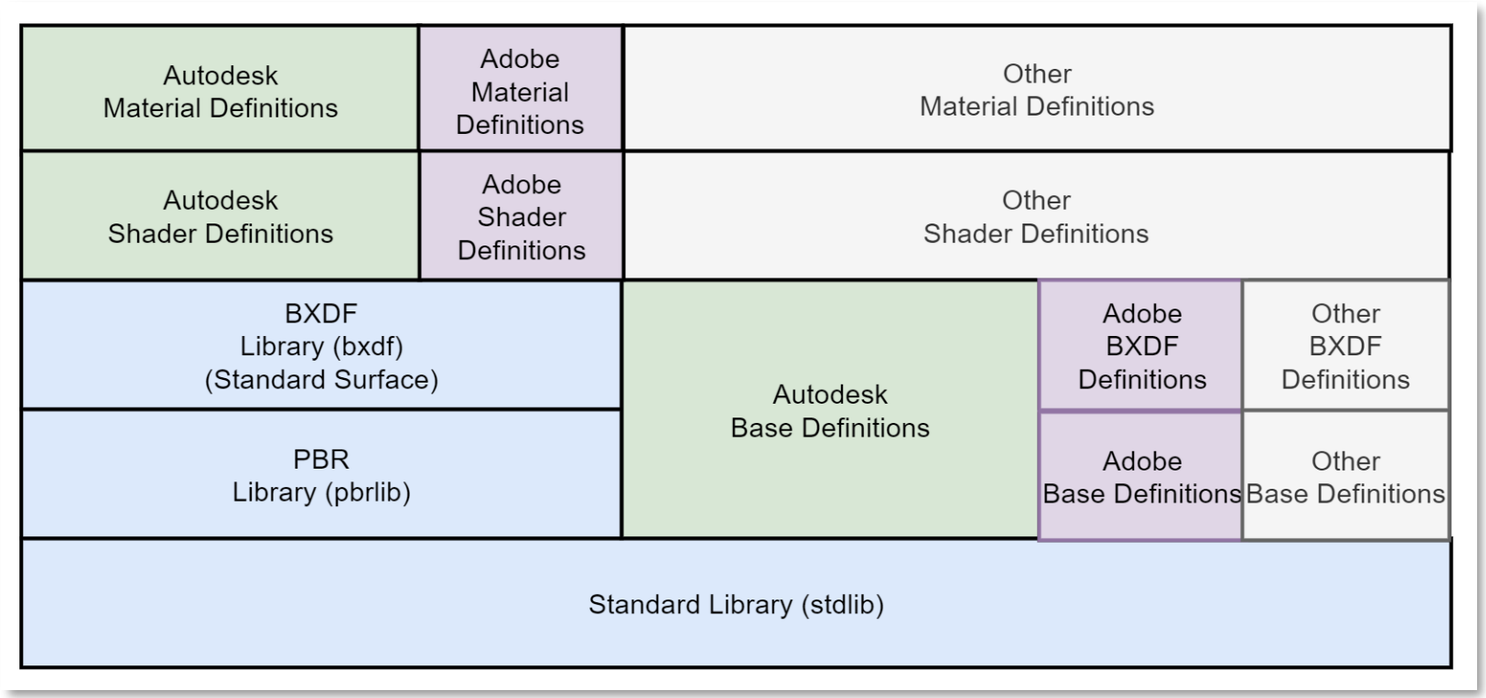


“Published” brick instance with associated Substance Designer textures.

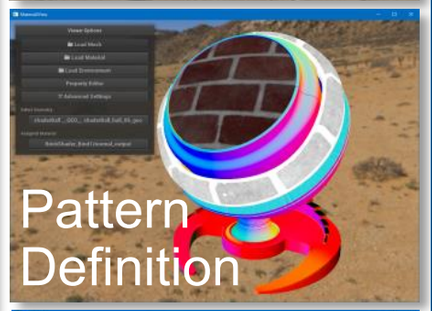


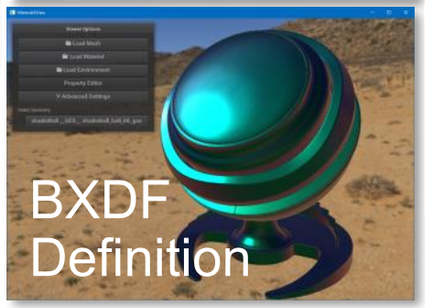
Top: MTLX document, and textures created in Substance Designer. Brick shader (middle), and Brick material (bottom) definitions created in LookdevX.

Asset Organization



- Plan to Open Source some examples.





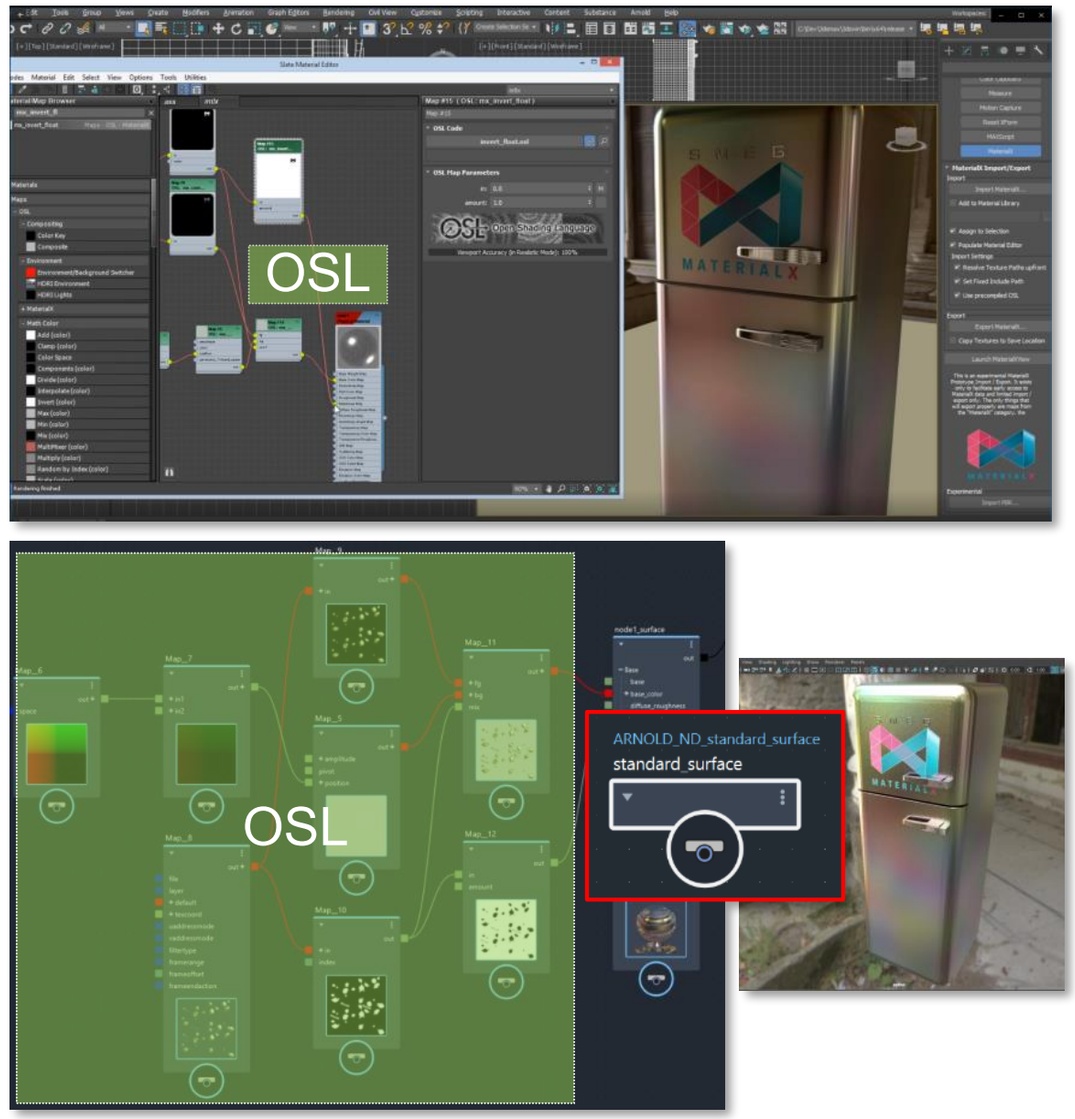

```
adsklib
    adsklib_defs.mtlx
    adsklib_ng.mtlx

    genglsl
    genmdl
    genosl
lights
    lights_defs.mtlx

    genglsl
        lights_genglsl_impl.mtlx

materials
    brick
        brick_02
            ND_brick_02_material.mtlx

    wood
        parquet_03
            ND_parquet_03_material.mtlx

procedurals
    ND_adsk_colorcorrect.mtlx
    NG_adsk_colorcorrect.mtlx

shaders
    brick
        ND_brick_02_shader.mtlx
        ND_brick_10_shader .mtlx
        NG_brick_02_shader.mtlx
        NG_brick_10_shader.mtlx

    plastic
        ND_plastic08_red_surfaceshader.mtlx
        NG_plastic08_red_surfaceshader.mtlx

    wood
        parquet_3
            ND_parquet_03_shader.mtlx
            NG_parquet_03_shader.mtlx

textures
    README.md

    bricks_02
        2k
            bricks_02_basecolor_2k_acescg.exr
            bricks_02_height_2k_raw.exr
            bricks_02_normal_2k_raw.exr
            bricks_02_roughness_2k_raw.exr
```

Images shown in MaterialXView (top to bottom): Autodesk brick shader, derived Substance Designer brick shader and supporting graph definition, Thin-film Standard Surface example. Shader ball asset created in Maya.

Application Integration

Strategies for consuming assets

- Consistent rendering via common tools:
  - Document merge, flattening / baking
  - Input bindings exposure.
- “Fridge” Shader Example:
  - 3ds Max:
    - <nodegraph> flattening + code gen per node.
  - Arnold:
    - <nodegraph> pattern code generation plus built-in terminal nodes.
    - Substance / hdPRman approach similar.
    - Displacement / nodedef support in-progress.
  - Swatch: Upstream sub-graph bake.



OSL rendering in 3ds Max (top). OSL rendering using Arnold in Maya. Swatch rendering using MaterialX GLSL renderer. (Fridge shader created in 3ds Max.)

Real Time Viewports

Extending Code Generation

- Make SPIR-V an official target
- Cross-compile to new targets
  - HLSL for DX12
  - Vulkan
  - MetalSL
  - DXR and Vulkan RT
- Plan to make available as Open Source.


```
void mx_multiply_bsdf_color_indirect(vec3 V, vec3 in1, v
{
    result = in1 * clamp(in2, 0.0, 1.0);
}

void IMPL_standard_surface_surfaceshader(float base, vec
{
    vec3 emission_weight_out = emission_color * emission
    vec3 metal_reflectivity_out = base_color * base;
    const float coat_tangent_rotate_degree_in2_tmp = 360
    float coat_tangent_rotate_degree_out = coat_rotation
    vec2 coat_roughness2_out = vec2(0.0);
    mx_roughness_anisotropy(coat_roughness, coat_anisotr
    vec3 coat_attenuation_bg_cm_out = vec3(0.0);
    mx_acescg_to_linear_color3(vec3(1.000000, 1.000000,
    vec3 coat_emission_attenuation_bg_cm_out = vec3(0.0)
    mx_acescg_to_linear_color3(vec3(1.000000, 1.000000,
    vec3 opacity_luminance_out = vec3(0.0);
```

GLSL - CodeGen

```
void mx_multiply_bsdf_color_indirect(float3 V, float3
{
    result = in1 * clamp(in2, 0.0f.xxx, 1.0f.xxx);
}

void IMPL_standard_surface_surfaceshader(float base, fl
{
    float3 emission_weight_out = emission_color * emiss
    float3 metal_reflectivity_out = base_color * base;
    float coat_tangent_rotate_degree_out = coat_rotatio
    float2 coat_roughness2_out = 0.0f.xx;
    float param = coat_roughness;
    float param_1 = coat_anisotropy;
    float2 param_2;
    mx_roughness_anisotropy(param, param_1, param_2);
    coat_roughness2_out = param_2;
    float3 coat_attenuation_bg_cm_out = 0.0f.xxx;
    float3 param_3 = 1.0f.xxx;
    float3 param_4;
```

HLSL – via SPIR-V

GLSL to HLSL cross compilation via SPIR-V helps maintain MaterialX node implementation portability.

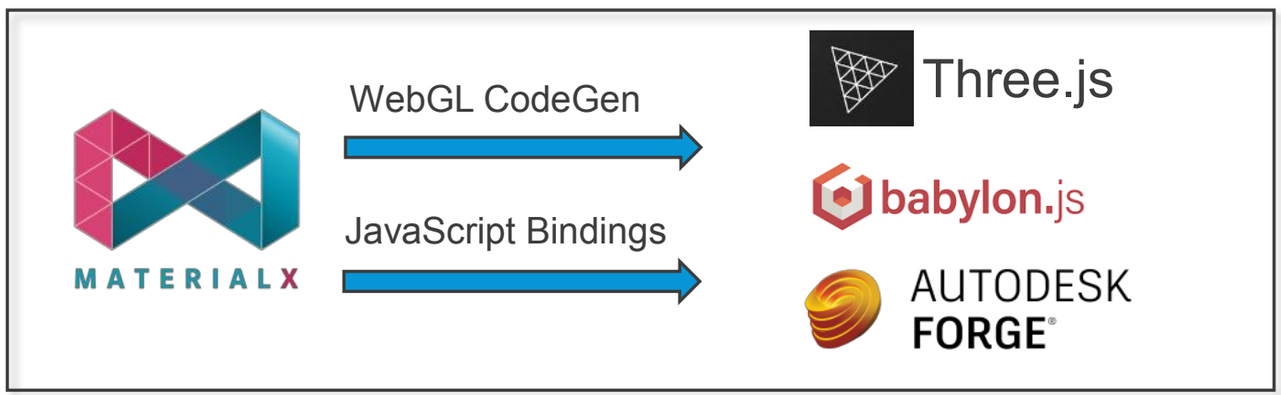
Web Browser

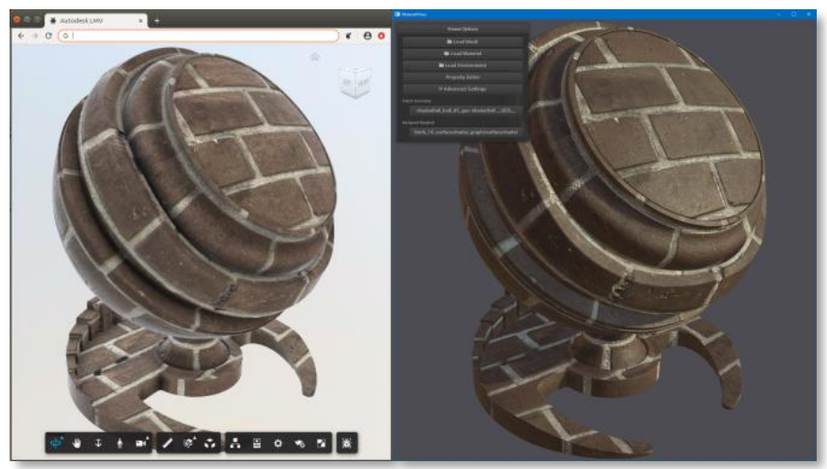
MaterialX for the Web

- Javascript bindings for MaterialX
  - Goal is to match Python bindings
  - Using Web Assembly (WASM)
  - Open Source support in progress: GitHub
    - *JsMaterialXCore*, *JsMaterialXFormat*

```
import Module from 'JsMaterialX.js';

let mx, doc;
mx = await initiMaterialX();
doc = mx.createDocument();

let nodeGraph;
nodeGraph = doc.addNodeGraph();

let output1, output2, constant, image;
constant = nodeGraph.addNode('constant');
image = nodeGraph.addNode('image');
output = nodeGraph.addOutput();
output.setConnectedNode(image);
```

- Autodesk web solution: Autodesk Forge Viewer





Left: Autodesk Forge Viewer: Standard Surface through 3dsMax “Shared Views”.
Right: The same shader rendered in MaterialXView.

Acknowledgements / Links

- MaterialX/ShaderX/LookdevX : Niklas Harrysson, Ashwin Bhat, Jonathan Feldstein, Nicolas Savva, Fedor Nikolayev, Tom Varik, Henrik Edström, Eric Bourque
- 3ds Max: Zap Andersson, Neil Hazzard
- Arnold : Orn Gunnarsson, Krishnan Chunangad Ramachandran
- Substance Designer / Painter: David Larsson
- Lucasfilm: Doug Smythe, Jonathan Stone
- Assets: Arvid Shneider and Dušan Ković (Autodesk), Justin Patton (Adobe), Matz models, (Turbosquid)

Autodesk Vision Series

- *3ds Max: Open Standards & Next Generation Viewport Framework*
- *The Source Awakens: USD, MaterialX & OpenColorIO*

LookdevX Demo