ASWF OPEN SOURCE DAYS 2021

NEW DEVELOPMENTS IN MATERIALX

INTRODUCTION

MATERIALX AT THE ASWF

- MaterialX joined the Academy Software Foundation this year
- Opportunities for additional teams to collaborate on its development
- Steering meetings are open to the entire community

ASWF /* ACADEMY SOFTWARE FOUNDATION

MATERIALX SESSION SCHEDULE

MaterialX in Hydra - Karen Lucknavalai (Pixar)

Shader Translation Graphs - Jonathan Stone (ILM)

MaterialX Shader Generation - Bernard Kwok & Ashwin Bhat (Autodesk)

MaterialX in MayaUSD - Krystian Ligenza (Autodesk)

MaterialX in Houdini - Mark Elendt (SideFX)

The Adobe Standard Material Model - Paul Edmondson (Adobe)

Adding MaterialX Closures to OSL – Chris Kulla (Epic Games)

Open Source Days 2021

HOSTED BY /* ACADEMY SOFTWARE FOUNDATION

MaterialX in Hydra

Karen Lucknavalai - Pixar

MaterialX in Hydra

How to use MaterialX within Hydra?

1. Attach a MaterialX file to a USD object through the materialBinding
2. Run a MaterialX file through usdcat to generate a USD version of the mtlx file and use as usual

Both these methods translate the MaterialX network into UsdShade

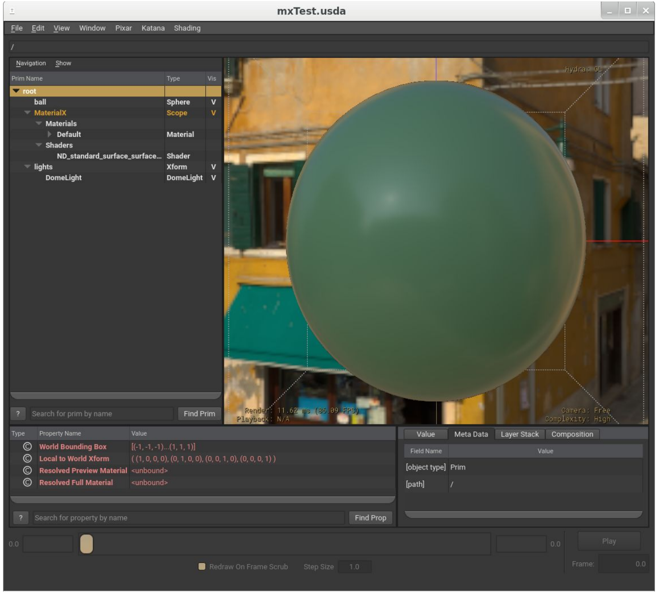
```
#usda 1.0

def Sphere "sphere" {
    rel material:binding = </MaterialX/Materials/USD_Default>
}

def Scope "MaterialX" (
    references = [
        @./usd_preview_surface_default.mtlx@</MaterialX>,
    ]
)
{
}
```

MaterialX in Hydra - Lights

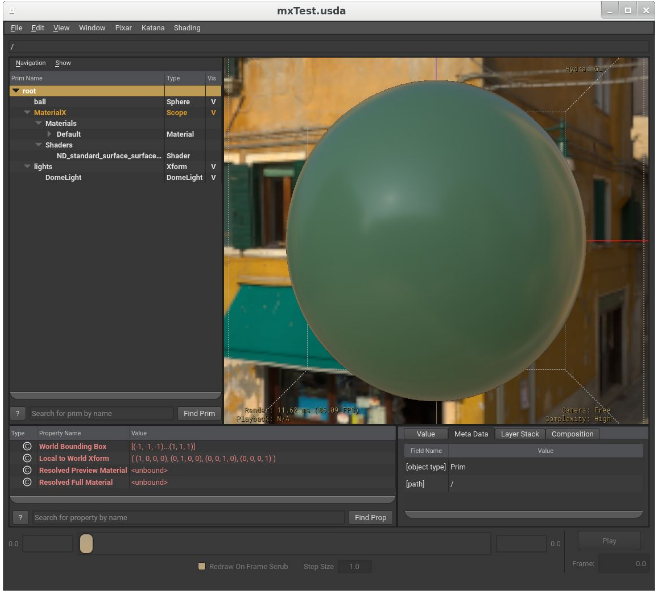
Just add lights to your USD file as usual!

Support for:

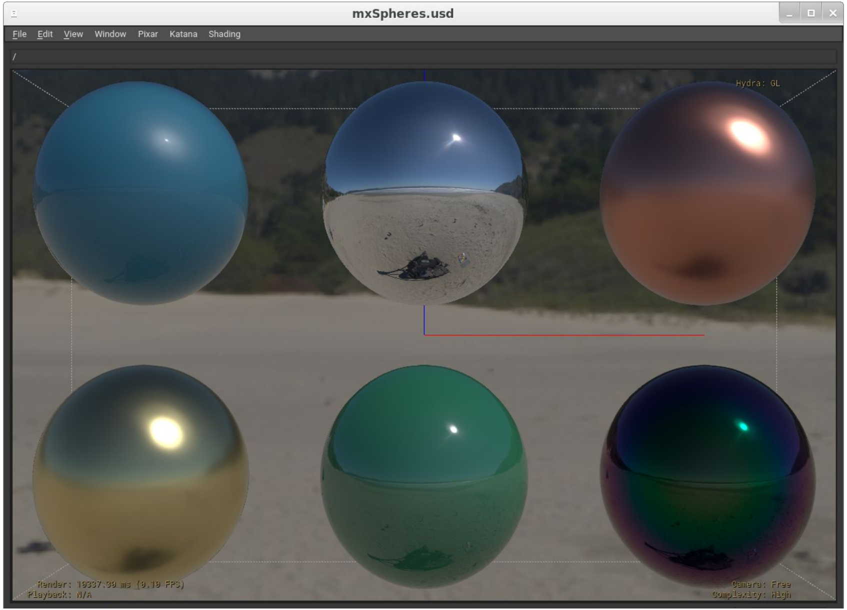
- Indirect lights →environment lights
- Direct lights →point lights



MaterialX in Hydra - Lights



MaterialX in Hydra



MaterialX in Hydra - Textures

```xml
<!-- NodeGraph using a geompropvalue for the primvar name -->
<nodegraph name="NG_Brass">
  <geompropvalue name="stcoords" type="vector2">
    <input name="geomprop" type="string" value="st" />
  </geompropvalue>
  <tiledimage name="image_color" type="color3">
    <input name="file" type="filename" value="brass_color.jpg" />
    <input name="uvtiling" type="vector2" value="1.0, 1.0" />
    <input name="texcoord" type="vector2" nodename="stcoords"/>
  </tiledimage>
  <tiledimage name="image_roughness" type="float">
    <input name="file" type="filename" value="brass_roughness.jpg" />
    <input name="uvtiling" type="vector2" value="1.0, 1.0" />
    <input name="texcoord" type="vector2" nodename="stcoords"/>
  </tiledimage>
  <output name="out_color" type="color3" nodename="image_color" />
  <output name="out_roughness" type="float" nodename="image_roughness" />
</nodegraph>
```

standard_surface_brass_tiled.mtlx

MaterialX in Hydra - Textures

```xml
<!-- NodeGraph using a geompropvalue for the primvar name -->
<nodegraph name="NG_Brass">
  <geompropvalue name="stcoords" type="vector2">
    <input name="geomprop" type="string" value="st" />
  </geompropvalue>
  <tiledimage name="image_color" type="color3">
    <input name="file" type="filename" value="brass_color.jpg" />
    <input name="uvtiling" type="vector2" value="1.0, 1.0" />
    <input name="texcoord" type="vector2" nodename="stcoords"/>
  </tiledimage>
  <tiledimage name="image_roughness" type="float">
    <input name="file" type="filename" value="brass_roughness.jpg" />
    <input name="uvtiling" type="vector2" value="1.0, 1.0" />
    <input name="texcoord" type="vector2" nodename="stcoords"/>
  </tiledimage>
  <output name="out_color" type="color3" nodename="image_color" />
  <output name="out_roughness" type="float" nodename="image_roughness" />
</nodegraph>
```

standard_surface_brass_tiled.mtlx

MaterialX in Hydra - Textures

```
<!-- NodeGraph using a geompropvalue for the primvar name -->
<nodegraph name="NG_Brass">
  <geompropvalue name="stcoords" type="vector2">
    <input name="geomprop" type="string" value="st" />
  </geompropvalue>
  <tiledimage name="image_color" type="color3">
    <input name="file" type="filename" value="brass_color.jpg" />
    <input name="uvtiling" type="vector2" value="1.0, 1.0" />
    <input name="texcoord" type="vector2" nodename="stcoords"/>
  </tiledimage>
  <tiledimage name="image_roughness" type="float">
    <input name="file" type="filename" value="brass_roughness.jpg" />
    <input name="uvtiling" type="vector2" value="1.0, 1.0" />
    <input name="texcoord" type="vector2" nodename="stcoords"/>
  </tiledimage>
  <output name="out_color" type="color3" nodename="image_color" />
  <output name="out_roughness" type="float" nodename="image_roughness" />
</nodegraph>
```

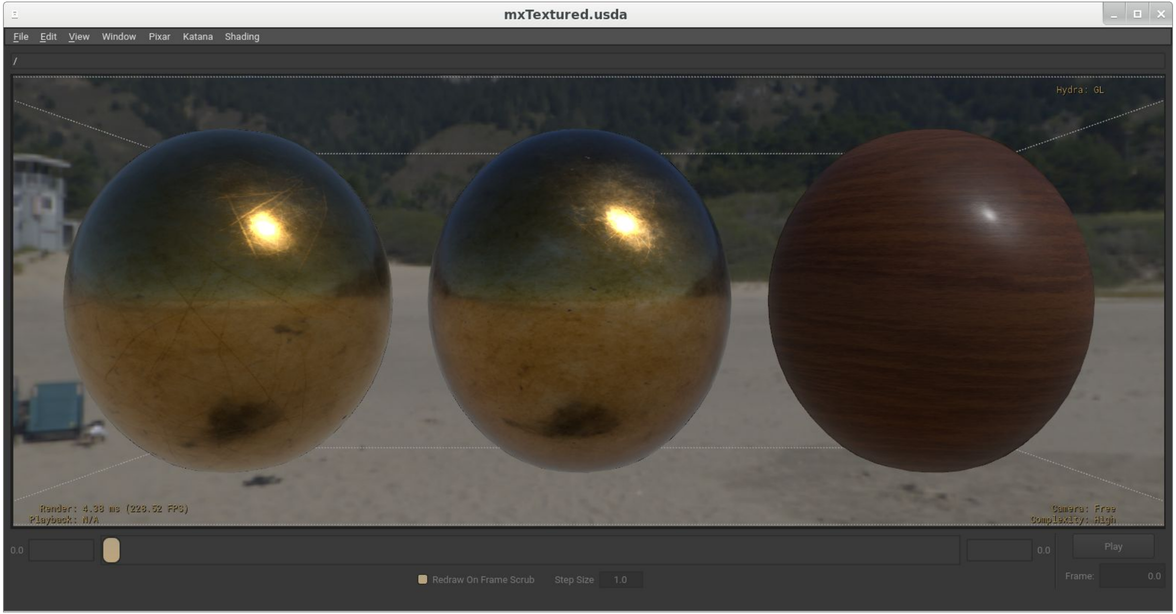
standard_surface_brass_tiled.mtlx

MaterialX in Hydra - Textures

```
<!-- NodeGraph using a geompropvalue for the primvar name -->
<nodegraph name="NG_Brass">
  <geompropvalue name="stcoords" type="vector2">
    <input name="geomprop" type="string" value="st" />
  </geompropvalue>
  <tiledimage name="image_color" type="color3">
    <input name="file" type="filename" value="brass_color.jpg" />
    <input name="uvtiling" type="vector2" value="1.0, 1.0" />
    <input name="texcoord" type="vector2" nodename="stcoords"/>
  </tiledimage>
  <tiledimage name="image_roughness" type="float">
    <input name="file" type="filename" value="brass_roughness.jpg" />
    <input name="uvtiling" type="vector2" value="1.0, 1.0" />
    <input name="texcoord" type="vector2" nodename="stcoords"/>
  </tiledimage>
  <output name="out_color" type="color3" nodename="image_color" />
  <output name="out_roughness" type="float" nodename="image_roughness" />
</nodegraph>
```

standard_surface_brass_tiled.mtlx

MaterialX in Hydra - Textures



MaterialX in Hydra - Storm and Prman

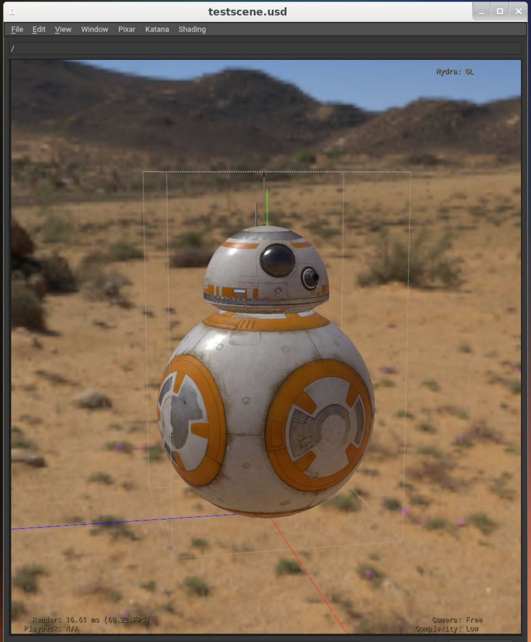
Open Source Days 2021

HOSTED BY /* ACADEMY SOFTWARE FOUNDATION

Thank you

More information at the USD, Hydra BOF at SIGGRAPH:

Wed August 11, 2pm-4pm



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

JONATHAN STONE – INDUSTRIAL LIGHT & MAGIC

SHADER TRANSLATION GRAPHS

BACKGROUND

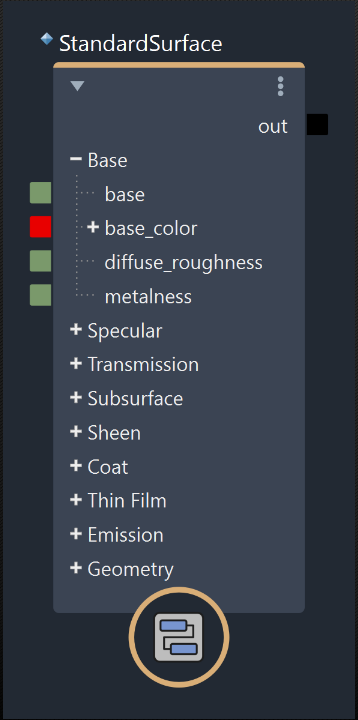
PHYSICALLY BASED SHADING NODES

- Standard building blocks for composing shading models
- Existing graph definitions for Autodesk Standard Surface and UsdPreviewSurface
- New graph definitions for the MaterialX Lama nodes

BACKGROUND

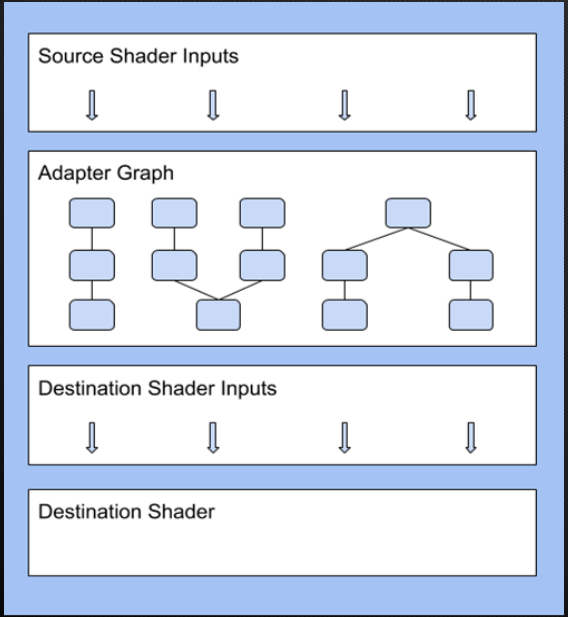
GRAPH BASED SHADING MODELS

- High-level definition of shading model behavior
- Maintains independence from renderer-specific choices
- Allows more natural comparison of differences between models



SHADER TRANSLATION GRAPHS

- Graph based definitions of translations between shading models
- MaterialX shader generation can be applied to both content and translation
- Translations can remain “live” as graphs or be baked to flat textures



NEW DEVELOPMENTS

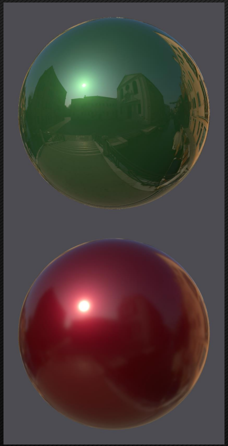
BB-8 AT SIGGRAPH 2019

ILM UNIFIED

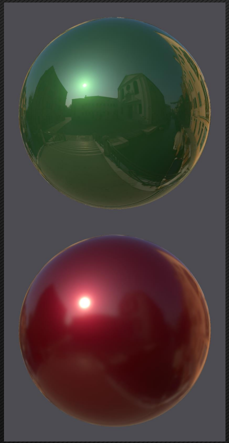
STANDARD SURFACE

ILM PRODUCTION TESTS

- ILM began refining the technique for use in production
- Translation becomes data-driven and automated
- Extended to include dual specular lobes, anisotropy, and other techniques



ILM UNIFIED



STANDARD SURFACE

NEW DEVELOPMENTS

ILM PRODUCTION TESTS

ILM UNIFIED

STANDARD SURFACE

NEW DEVELOPMENTS

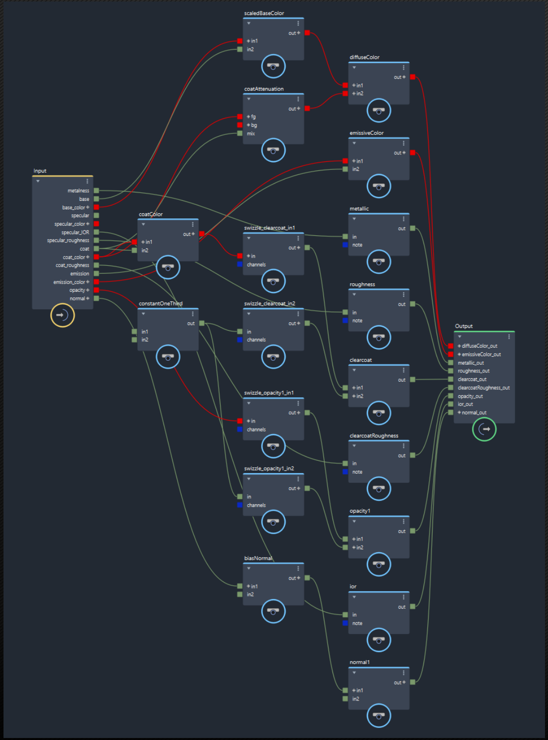
ILM PRODUCTION TESTS

ILM UNIFIED

STANDARD SURFACE

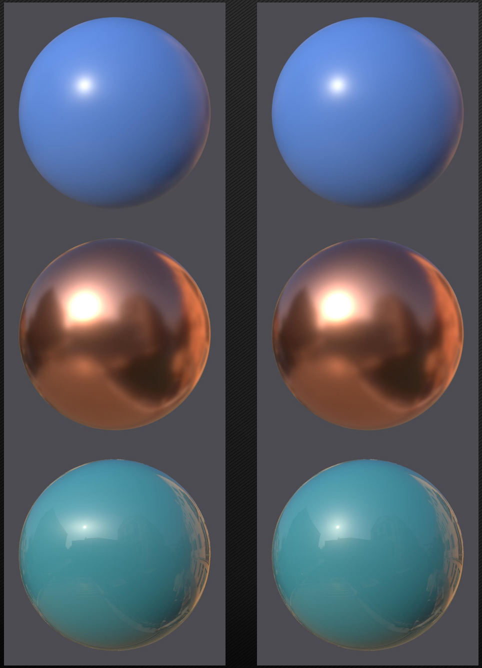
EXAMPLE TRANSLATION GRAPH

- A first example translation graph has been added to MaterialX
- Translates from Standard Surface to UsdPreviewSurface



EXAMPLE TRANSLATION GRAPH

- UsdPreviewSurface has a smaller feature set, so some techniques are omitted
- Anisotropic roughness is averaged
- Sheen, thin film, and subsurface effects are ignored



STANDARD SURFACE

USD PREVIEW SURFACE

EXAMPLE TRANSLATION GRAPH

- For a Python example, see translateshader.py in the Scripts folder
- For a C++ example, see Viewer.cpp in the MaterialXView project
- Shader translation has been added to render tests in GitHub Actions

ACKNOWLEDGEMENTS

THANKS TO…

Doug Smythe
Madeleine Yip
André Mazzone
Karen Lucknavalai
Bernard Kwok
Krystian Ligenza
Paul Edmondson
Mark Elendt
Chris Kulla
Eoghan Cunneen
Emma Holthouser
David Meny
George ElKoura
Ashwin Bhat
Zap Andersson
David Larsson
Lee Kerley
Adrien Herubel
Roger Cordes
François Chardavoine
Rob Bredow
Nick Porcino
Niklas Harrysson
Eric Bourque
Guido Quaroni
Mark Tucker
Larry Gritz

MaterialX Shader Generation

Bernard Kwok and Ashwin Bhat

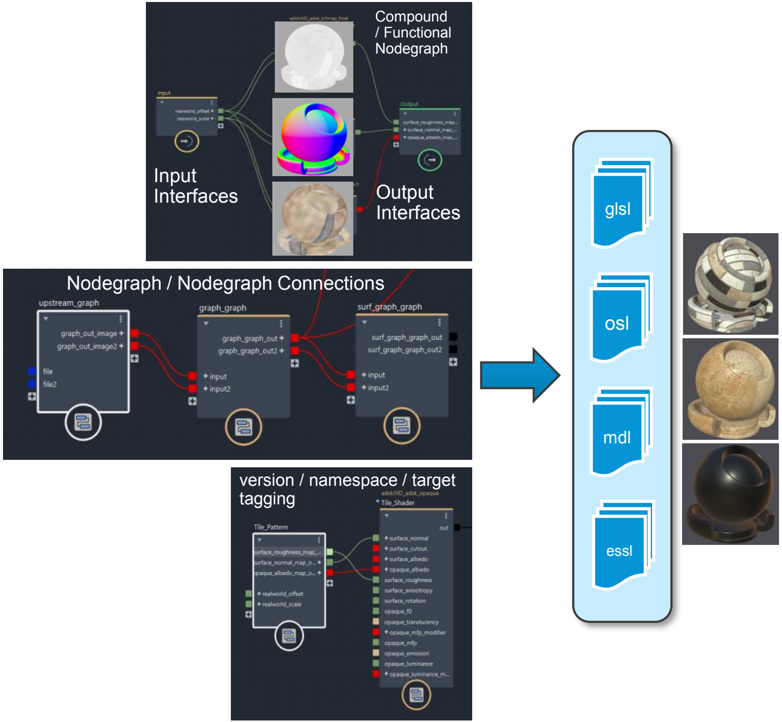
bernard.kwok@autodesk.com
ashwin.bhat@autodesk.com

AUTODESK.

1.38.x Updates

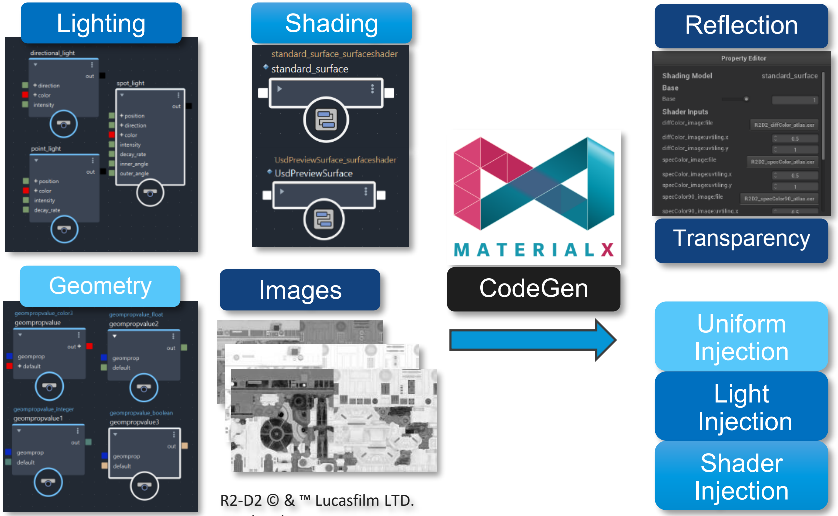
Shading Graph Configurability

- Consistent and robust compound and functional graph support
- Improved traversal logic for node and graph interface connections
- New: Nodegraph-to-nodegraph connections, Translation graph support.
- Improved namespace, version, target support
- Improved input value resolution to handle: inheritance, interface connections, geometry and filenames (incl. tokens)
- Improved ability to code generate for individual nodes, and sub-graphs.



Code Generation Configurability

- Improved light injection and geometry stream bindings
- Improved uniform injection including layout support
- Improved reflection for resource binding and transparency heuristics
- Improved image format and texturing support

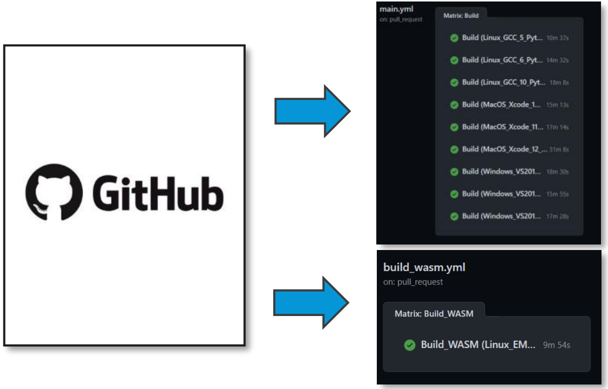


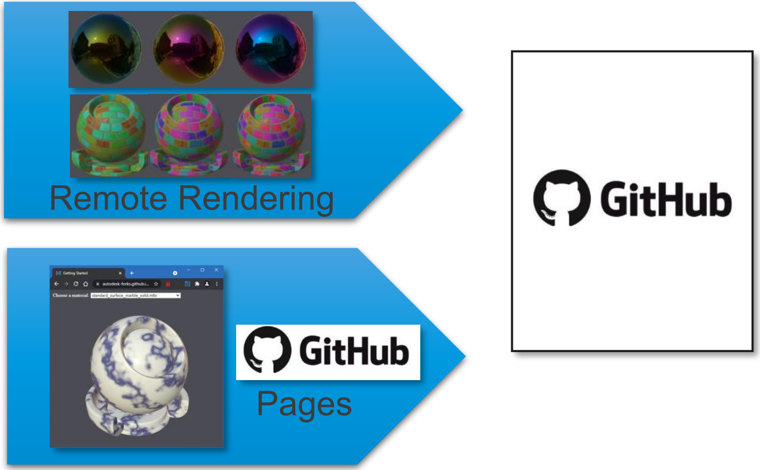
R2-D2 © & ™ Lucasfilm LTD.
Used with permission

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

Infrastructure

Github Actions Migration



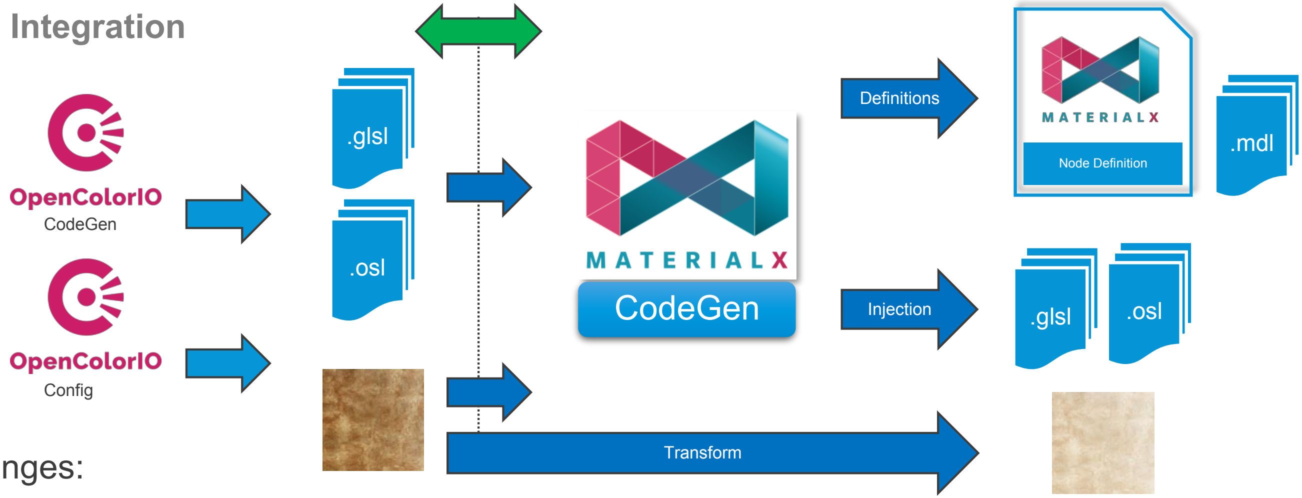


- Rendering validation (Intel OpenSWR)
- MaterialX Web: WASM generation and Github pages hosting.
- Goal: support fully automated code generation / rendering validation

Initiatives In Progress

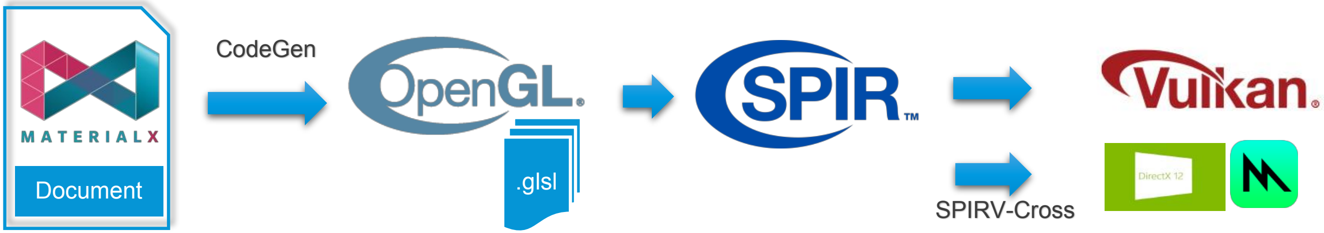
Color Management

OCIO v2 Integration



- Challenges:
  - ACEScg color space naming consistency
  - Code generation targets: GLSL, OSL, MDL, ESSL
  - Deployment flexibility: pre-compute, function generation, full shader, reference definition
  - OCIO enhancements for uniform injection / format control

SPIRV Code Generation Overview



- Use `mx::GlslResourceBindingContext`
- Generate SPIRV compatible GLSL.
  E.g., use `#extension GL_ARB_shading_language_420pack`
- Demonstrated feasibility of Cross Compilation during SIGGRAPH 2020 Autodesk Vision Series demo.
- Explore and improve **KhronosGroup/SPIRV-Tools** to provide per target Shader Reflection.

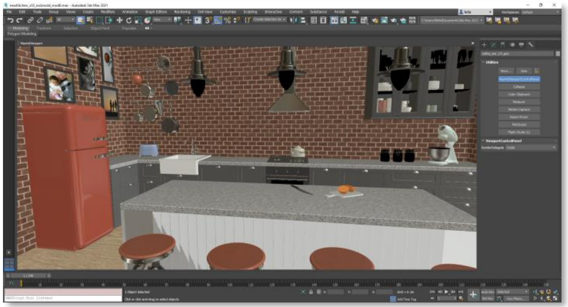
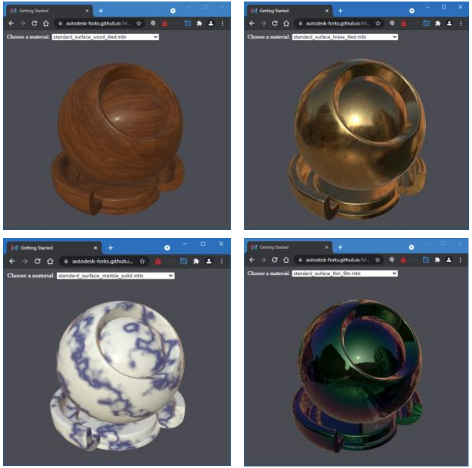


Image credit:
3ds Max: Open Standards & Next Generation Viewport Framework (SIGGRAPH 2020 Autodesk Vision Series)

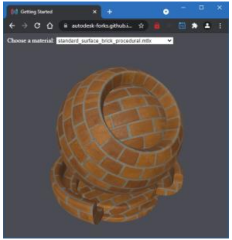
MaterialX for Web

MaterialX JavaScript library

- In progress project for upcoming release.
- Components:
  - JavaScript Bindings + Web Assembly.
  - CodeGen for OpenGL ES 3.0.
  - Web Viewer Sample Application https://autodesk-forks.github.io/MaterialX/
- Fully compatible with current GLSL implementation.
- Supported Browsers Chrome, Firefox, Edge, Safari*
- Supports material shading graphs and pattern graphs (textures, procedurals)
- Framework agnostic.

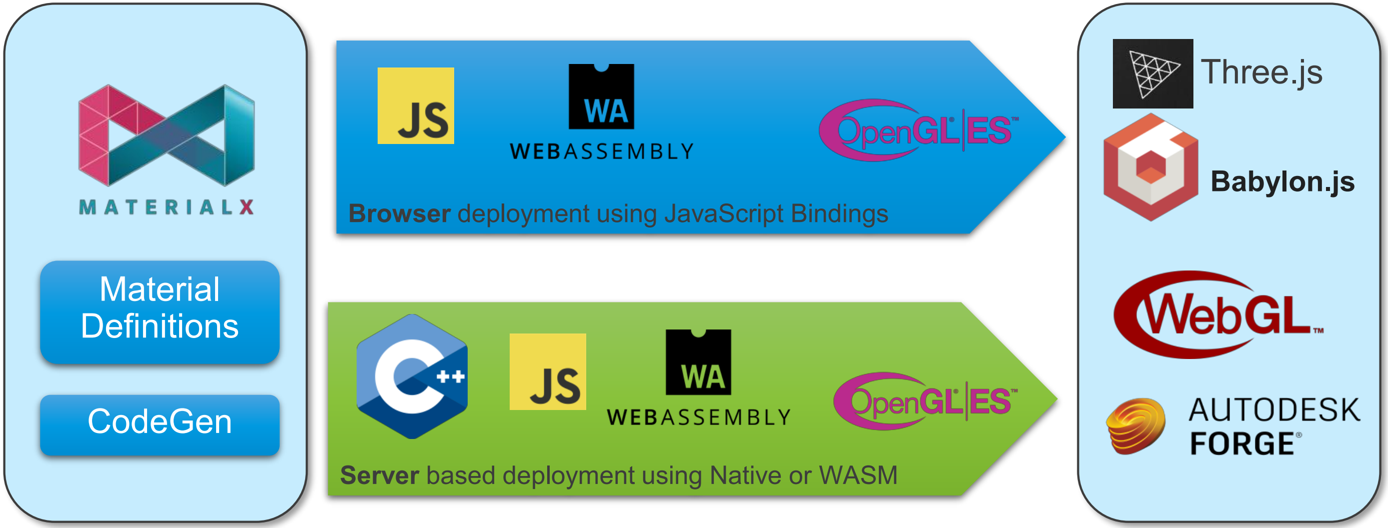


Above: Examples from MaterialX distribution using Standard Surface in Google Chrome.
Below: Example procedural material from **Adobe Substance** as MaterialX in Google Chrome.

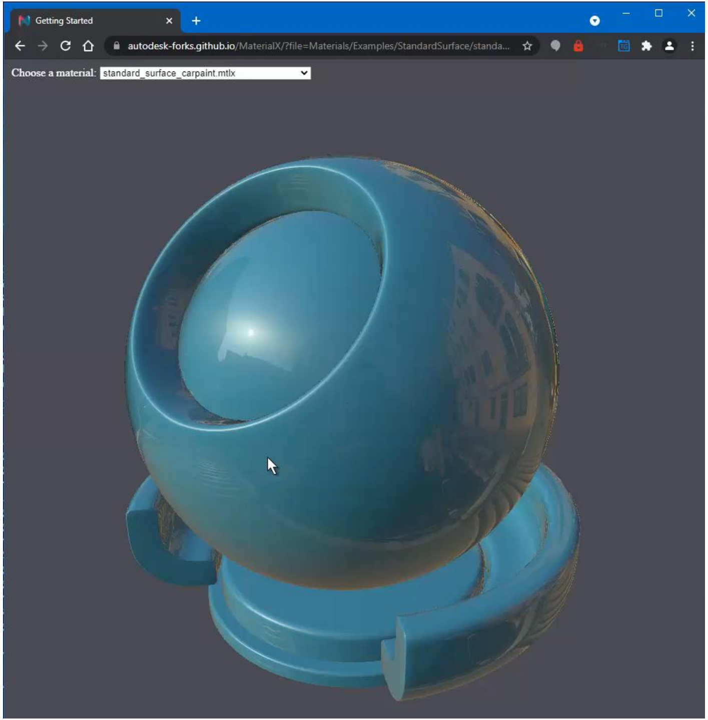


MaterialX for Web

Deployment options (framework agnostic)



MaterialX WebGL (in Google Chrome)



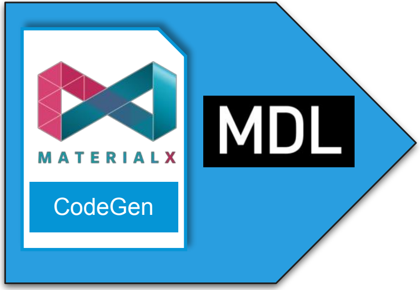
MaterialX API in JavaScript, using GL ES Shader Generator

```
let gen = new mx.EsslShaderGenerator();
let genContext = new mx.GenContext(gen);
let stdlib = mx.loadStandardLibraries(genContext);
doc.importLibrary(stdlib);

// Load material
if (mtlxMaterial)
    await mx.readFromXmlString(doc, mtlxMaterial);
else
    fallbackMaterial(doc);

let elem = mx.findRenderableElement(doc);

// Handle transparent materials
const isTransparent = mx.isTransparentSurface(elem, gen.getTarget());
genContext.getOptions().hwTransparency = isTransparent;

// Load lighting setup into document
const lightRigDoc = mx.createDocument();
await mx.readFromXmlString(lightRigDoc, loadedLightSetup);
doc.importLibrary(lightRigDoc);

// Register lights with generation context
const lights = (0,_helper_js__WEBPACK_IMPORTED_MODULE_0__.findLights)(do
const lightData = (0,_helper_js__WEBPACK_IMPORTED_MODULE_0__.registerLig

let shader = gen.generate(elem.getNamePath(), elem, genContext);

// Get GL ES shaders and uniform values
let vShader = shader.getSourceCode("vertex");
let fShader = shader.getSourceCode("pixel");
```

Future Work

NVIDIA MDL Updates

- Forthcoming MDL 1.7 release will have better alignment with MaterialX (e.g., sheen layer, unbound mixer nodes)
- End of year target to have MaterialX import for Omniverse
  - Background improvements in MDL generation and consumption (E.g., resource path handling)
- See SIGGRAPH 2021 updates from NVIDIA.

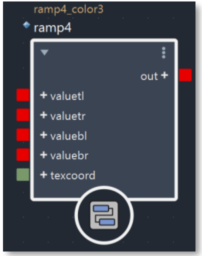


Generation Configurability

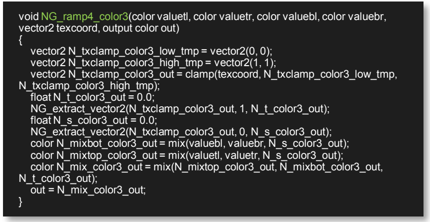
- Fragment / Function Export
  vs new generator derivation

- Uniform format control / reflection

- Sub-graph / node export as graphs
  or images



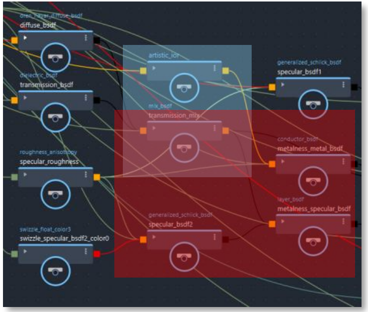
Fragment



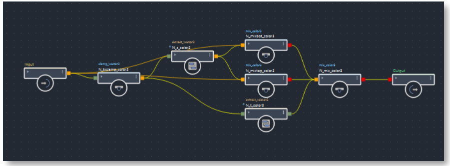
```
void NG_ramp4_color3(color valuetl, color valuetr, color valuebl, color valuebr,
vector2 texcoord, output color out)
{
    vector2 N_txclamp_color3_low_tmp = vector2(0, 0);
    vector2 N_txclamp_color3_high_tmp = vector2(1, 1);
    vector2 N_txclamp_color3_out = clamp(texcoord, N_txclamp_color3_low_tmp,
N_txclamp_color3_high_tmp);
    float N_t_color3_out = 0.0;
    NG_extract_vector2(N_txclamp_color3_out, 1, N_t_color3_out);
    float N_s_color3_out = 0.0;
    NG_extract_vector2(N_txclamp_color3_out, 0, N_s_color3_out);
    color N_mixbot_color3_out = mix(valuebl, valuebr, N_s_color3_out);
    color N_mixtop_color3_out = mix(valuetl, valuetr, N_s_color3_out);
    color N_mix_color3_out = mix(N_mixtop_color3_out, N_mixbot_color3_out,
N_t_color3_out);
    out = N_mix_color3_out;
}
```

Uniform Format / Reflection

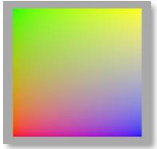
Valuetl	1	0	0
Valuetr	0	0	1
Valuebl	0	1	0
Valuebr	1	1	0
Texcoord	0	0	



Export



Bake



Generation Optimization

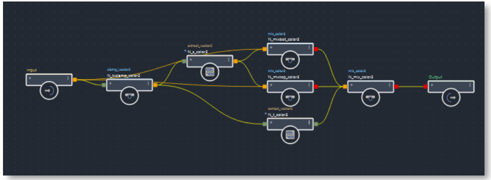
- Performance optimizations for language / platform / workflow

```
void NG_ramp4_color3_slow(color valuetl, color valuetr, color valuebl,
color valuebr, vector2 texcoord, output color out)
{
   vector2 N_txclamp_color3_low_tmp = vector2(0, 0);
   vector2 N_txclamp_color3_high_tmp = vector2(1, 1);
   vector2 N_txclamp_color3_out = clamp(texcoord,
N_txclamp_color3_low_tmp, N_txclamp_color3_high_tmp);
   float N_t_color3_out = 0.0;
   NG_extract_vector2(N_txclamp_color3_out, 1, N_t_color3_out);
   float N_s_color3_out = 0.0;
   NG_extract_vector2(N_txclamp_color3_out, 0, N_s_color3_out);
   color N_mixbot_color3_out = mix(valuebl, valuebr, N_s_color3_out);
   color N_mixtop_color3_out = mix(valuetl, valuetr, N_s_color3_out);
   color N_mix_color3_out = mix(N_mixtop_color3_out,
N_mixbot_color3_out, N_t_color3_out);
   out = N_mix_color3_out;
}
```

Optimize

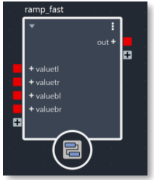
```
void NG_ramp4_color3_fast(color valuetl, color
valuetr, color valuebl, color valuebr, vector2
texcoord, output color out)
{
   out = fast_code;
}
```

- Optimize at code, node, and/or definition level

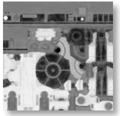


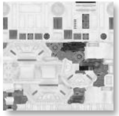
Optimize

Reparametrize

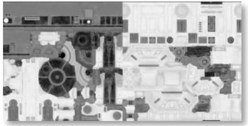


- Repackaging of resources: baking, packing, access atlas / arrays (e.g. UDIMs), alternate formats (e.g. IBL cubemaps)



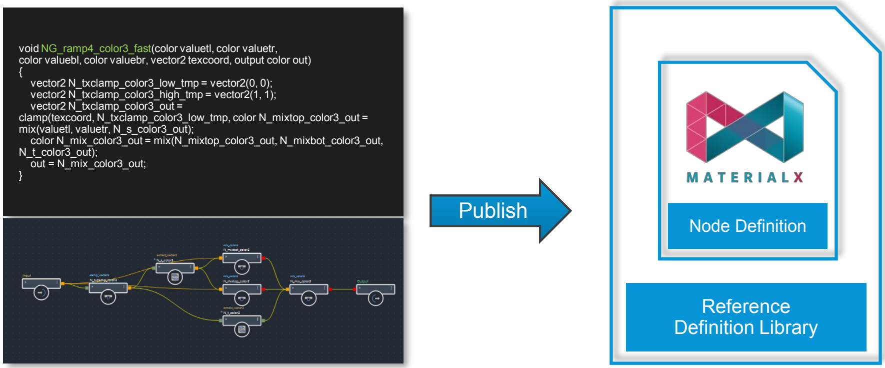


Format / Pack

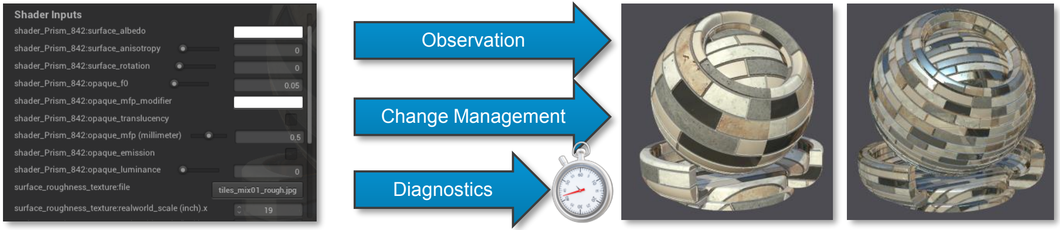


Generation Deployment

- Publishing for reuse, produce reference libraries (e.g. OSL reference library)



- Realtime Updates:
  - Observability
  - Change management
  - Diagnostics / Feedback



Credits

Adam Felt
Aura Munoz
Brent Scannell
Cedrick Muenstermann
David Larsson
Doug Smythe
Doug Walker
Dusan Kovic
Eric Bourque

Fedor Nikolayev
Gareth Morgan
Guillaume Laforge
Harv Saund
Henrik Edstrom
Jan Jordan
Jerran Schmidt
Jerry Gamache
Jonathan Stone

Kai Rohmer
Kevin Zhang
Krishna Kalvai
Krishnan Chunangad Ramachandran
Krystian Ligenza
Lutz Kettner
Mauricio Vives
Nicolas Savva
Niklas Harrysson

Nikola Milosevic
Patrick Hodoul
Philippe Frericks
Phenix Xu
Rishabh Bisht
Roberto Ziche
Sankar Ganesh
Sebastian Dunkel
Toni Qin

Wayne Catalfano
Will Telford
Zap Andersson

AUTODESK®

Make anything™

MaterialX in MayaUSD and ArnoldUSD

Krystian Ligenza
Software Architect | krystian.ligenza@autodesk.com

AUTODESK

Safe Harbor Statement

During the course of this presentation, we may make statements regarding future events and/or statements regarding planned or future development efforts for our existing or new products and services.

We wish to caution you that such statements reflect our current expectations, estimates and assumptions based on factors currently known to us and that actual events or results could differ materially. Also, these statements are not intended to be a promise or guarantee of future delivery of products, services or features but merely reflect our current plans, which may change.

Purchasing decisions should not be made based upon reliance on these statements. The statements made in this presentation are being made as of the time and date of its live presentation. We do not assume any obligation to update any statements we make to reflect events that occur or circumstances that exist after the date of this presentation.

Level-set on **MayaUSD**

MayaUSD | Repository

Closed pull requests:
1086

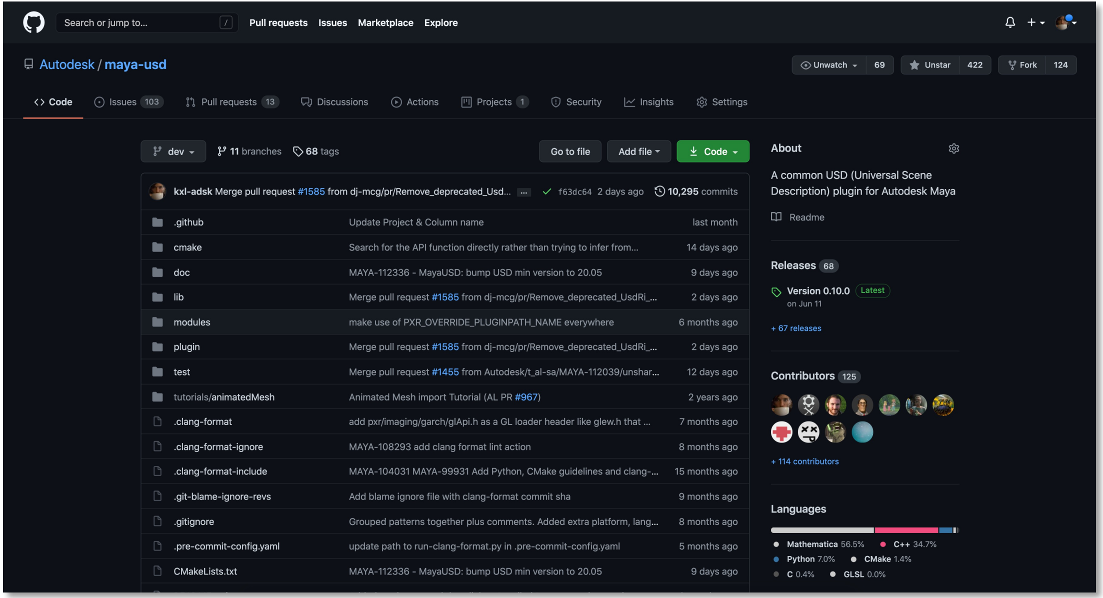
Closed issues:
266 (customer reported)

Stars:
422

Forks:
124

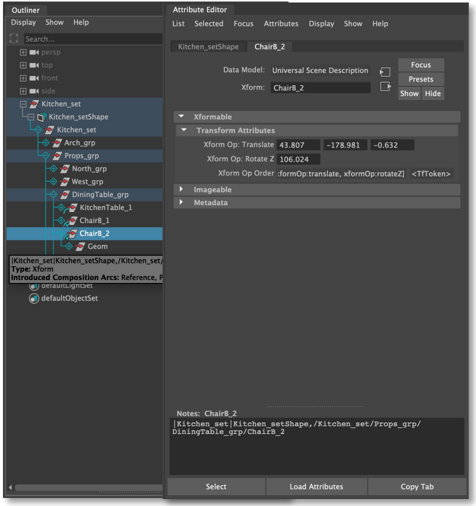
Releases:
11

Total downloads:
13 877



MayaUSD | Workflows

Import/Export

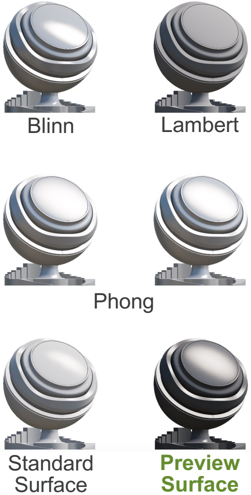


Direct Editing

Visualization

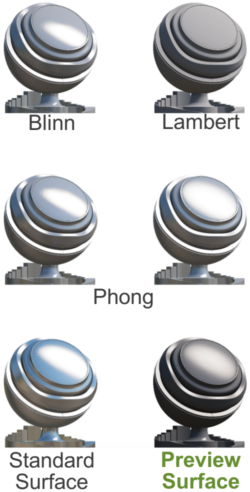
MayaUSD | Interop via UsdShade + PreviewSurface

Asset source: https://developer.apple.com/augmented-reality/quick-look/



Export

Import



USD Preview Surface

Custom translation with plugin architecture

MaterialX in MayaUSD

MaterialX | Phase1 Workflows

Import/Export

Visualization

MaterialX | Interop via UsdShade + MaterialX

Asset source: https://github.com/KhronosGroup/glTF-Sample-Models

Export

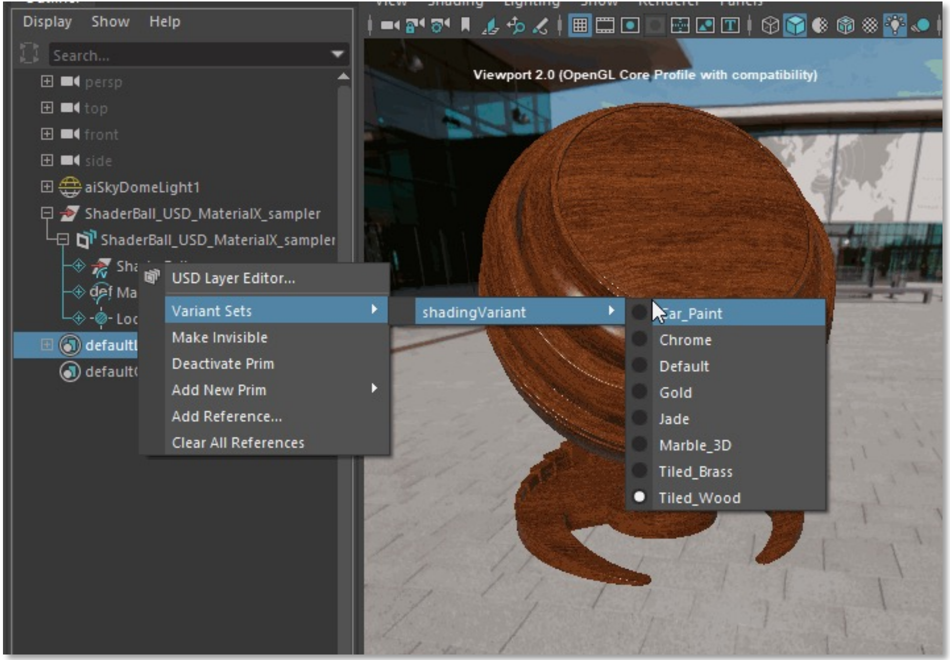
Import

MATERIALX

Standard surface
Preview surface

MaterialX | Visualization in Maya's Viewport

We are using the hdMtlx translation framework, which is also in use for hdStorm and hdPrman, and using the same GLSL code generator as hdStorm.



MaterialX | Visualization in Maya's Viewport

Problem: USD refers to texture coordinates by name, but MaterialX defaults to index
Solution: Modify MaterialX GLSL codegen for geompropvalue to emit varying inputs
Modify MaterialX document in-flight to remap indexed UV streams to USD named streams

```xml
<!-- Direct connection by name -->
<image name="BB8_color" type="color3">
  <input name="file" type="filename" value="BB8_color.png" />
  <input name="texcoord" type="vector2" nodename="st_dirtmap" />
</image>
<geompropvalue name="st_dirtmap" type="vector2">
  <input name="geomprop" type="string" value="dirtmap" />
</geompropvalue>

<!-- Implicit connection to UV0 -->
<image name="BB8_roughness" type="float">
  <input name="file" type="filename" value="BB8_roughness.jpg" />
</image>          ?

<!-- Explicit connection to UV1 -->
<image name="BB8_normals" type="vector3">
  <input name="file" type="filename" value="BB8_normals.jpg" />
  <input name="texcoord" type="vector2" nodename="st_UV1" />
</image>
<texcoord name="st_UV1" type="vector2">
  <input name="index" type="integer" value="1" />
</texcoord>
```

```
def Mesh "BB8" ()
{
    rel material:binding = </Looks/BB8Surface>
    texCoord2f[] primvars:st = [...]
    texCoord2f[] primvars:st1 = [...]
    texCoord2f[] primvars:dirtmap = [...]
}
```

```xml
<!-- Automatically added for UV0 -->
<geompropvalue name="auto_UV0" type="vector2">
  <input name="geomprop" type="string" value="st" />
</geompropvalue>

<!-- Automatically added for UV1 -->
<geompropvalue name="auto_UV1" type="vector2">
  <input name="geomprop" type="string" value="st1" />
</geompropvalue>
```

MaterialX | Visualization in Maya's Viewport

Problem: MaterialX did not interact with Maya's scene lights
Solution: Modify the Maya light code generator to provide GLSL entry points to query Maya light information directly from MaterialX light loop

MaterialX | Visualization in Maya's Viewport

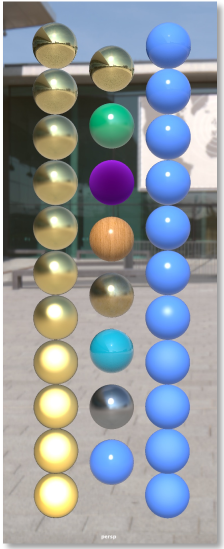
Problem: A single instance of a standard surface material can take seconds, and each material generated its own shader

Solution: Computing the minimal topologically equivalent Hydra material

```xml
<?xml version="1.0"?>
<materialx version="1.38" colorspace="lin_rec709">
  <standard_surface name="SR_chrome" type="surfaceshader">
    <input name="base" type="float" value="1" />
    <input name="base_color" type="color3" value="1.0, 1.0, 1.0" />
    <input name="specular" type="float" value="1" />
    <input name="specular_color" type="color3" value="1.0, 1.0, 1.0" />
    <input name="specular_roughness" type="float" value="0" />
    <input name="metalness" type="float" value="1" />
  </standard_surface>
  <surfacematerial name="Chrome" type="material">
    <input name="surfaceshader" type="surfaceshader" nodename="SR_chrome" />
  </surfacematerial>
</materialx>
```

```xml
<?xml version="1.0"?>
<materialx version="1.38" colorspace="lin_rec709">
<standard_surface name="N1" type="surfaceshader">
</standard_surface>
<surfacematerial name="M1" type="material">
    <input name="surfaceshader" type="surfaceshader" nodename="N1" />
</surfacematerial>
</materialx>
```

```xml
<?xml version="1.0"?>
<materialx version="1.38" colorspace="lin_rec709">
  <standard_surface name="SR_gold" type="surfaceshader">
    <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" />
    <input name="specular_roughness" type="float" value="0.02" />
    <input name="metalness" type="float" value="1" />
  </standard_surface>
  <surfacematerial name="Gold" type="material">
    <input name="surfaceshader" type="surfaceshader" nodename="SR_gold" />
  </surfacematerial>
</materialx>
```



Opportunities & References

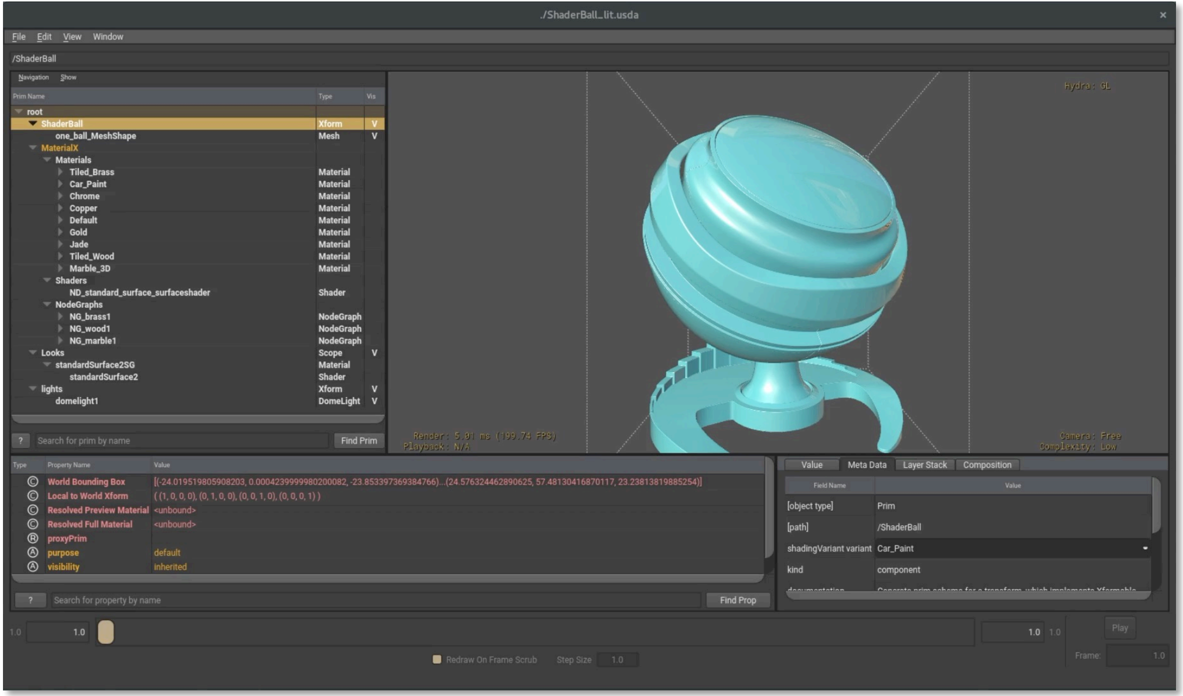
Opportunities / future investigations

- Render context
- More material translators
- Direct material binding and graph & parameters authoring
- Color Management
- ArnoldUSD

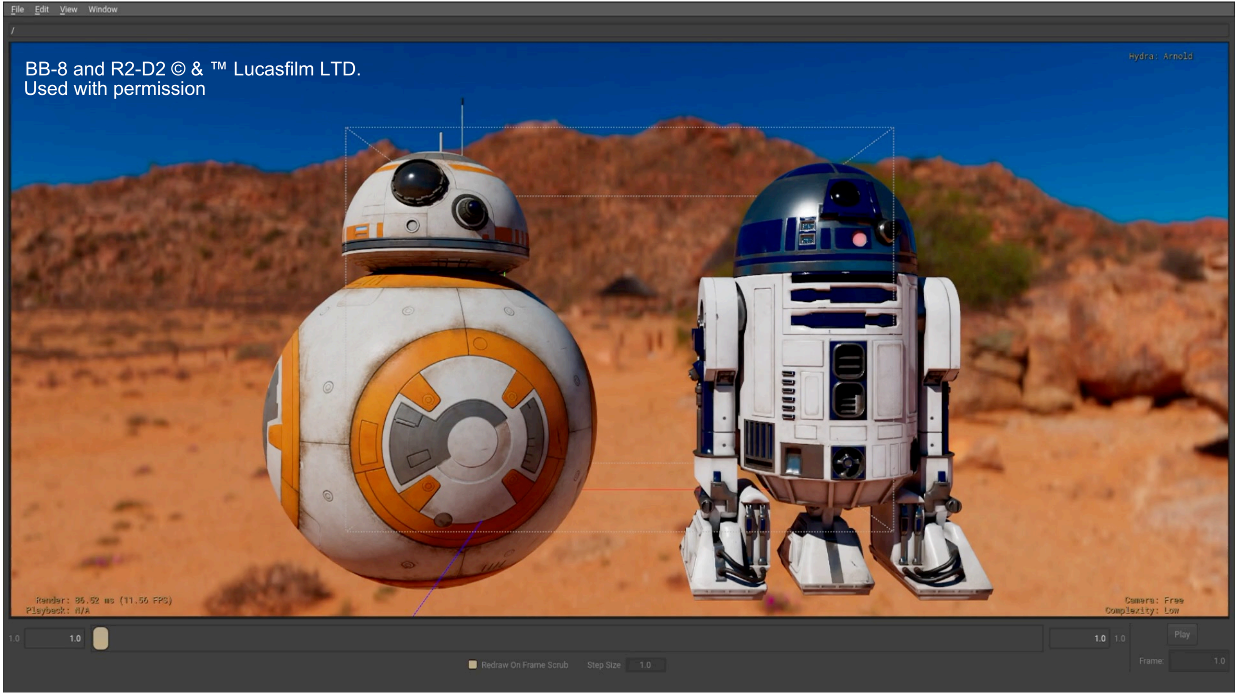
Useful links:

- https://github.com/Autodesk/maya-usd/blob/dev/doc/MaterialX.md
- https://github.com/autodesk-forks/MaterialX/tree/adsk_contrib/dev/source/MaterialXGenOgsXml#superior-environmental-lighting
- https://github.com/autodesk-forks/MaterialX
- https://github.com/Autodesk/maya-usd/
- https://github.com/Autodesk/maya-usd/discussions

Sneak-peek: MaterialX in ArnoldUSD



Sneak-peek: MaterialX in ArnoldUSD



Credits

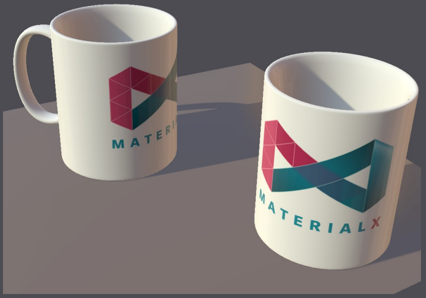
Adam Felt
Ashwin Bhat
Aura Munoz
Brent Scannell
Bernard Kwok
Cedrick Muenstermann
David Larsson
Doug Smythe
Doug Walker
Dusan Kovic

Eric Bourque
Fedor Nikolayev
Gareth Morgan
Guillaume Laforge
Harv Saund
Henrik Edstrom
Jan Jordan
Jerran Schmidt
Jerry Gamache
Jonathan Stone

Kai Rohmer
Kevin Zhang
Krishna Kalvai
Krishnan Chunangad Ramachandran
Krystian Ligenza
Lutz Kettner
Mauricio Vives
Nicolas Savva
Niklas Harrysson
Nikola Milosevic

Patrick Hodoul
Pal Mezei
Philippe Frericks
Phenix Xu
Rishabh Bisht
Roberto Ziche
Sankar Ganesh
Sebastian Dunkel
Toni Qin
Wayne Catalfano

Will Telford
Zap Andersson



AUTODESK®

Make anything™

Open Source Days 2021

HOSTED BY /* ACADEMY SOFTWARE FOUNDATION

MaterialX in Houdini

Mark Elendt

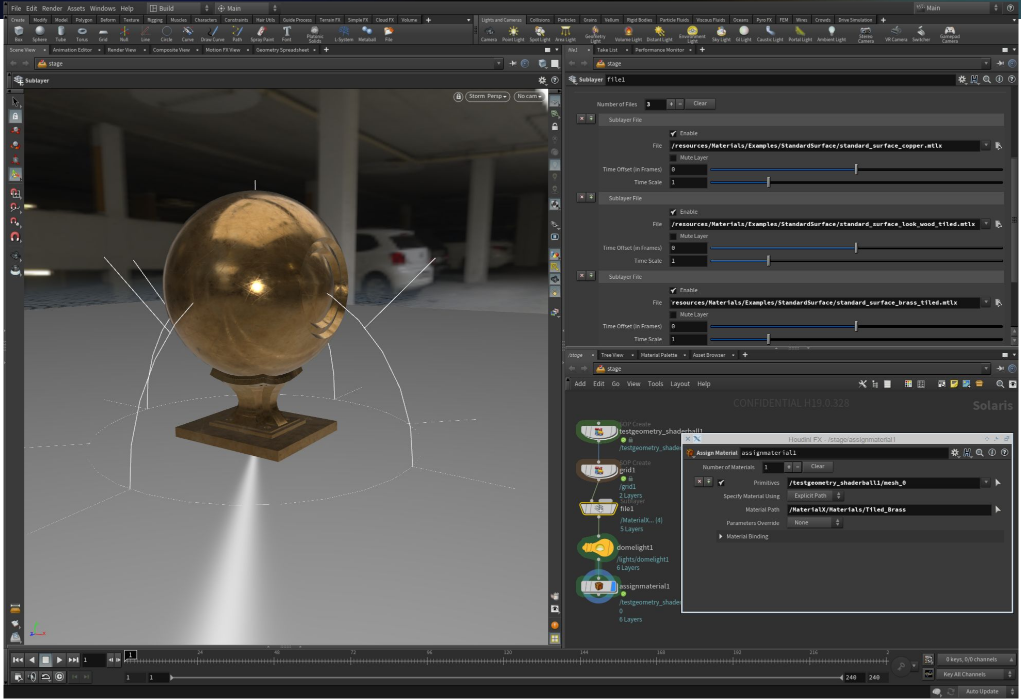
Usd 21.05/Houdini ??.?

- Usd 21.05 added support for MaterialX 1.38
  - MaterialX networks can be loaded as Usd Shade nodes
  - Shade nodes are passed through Hydra (available to all render delegates)

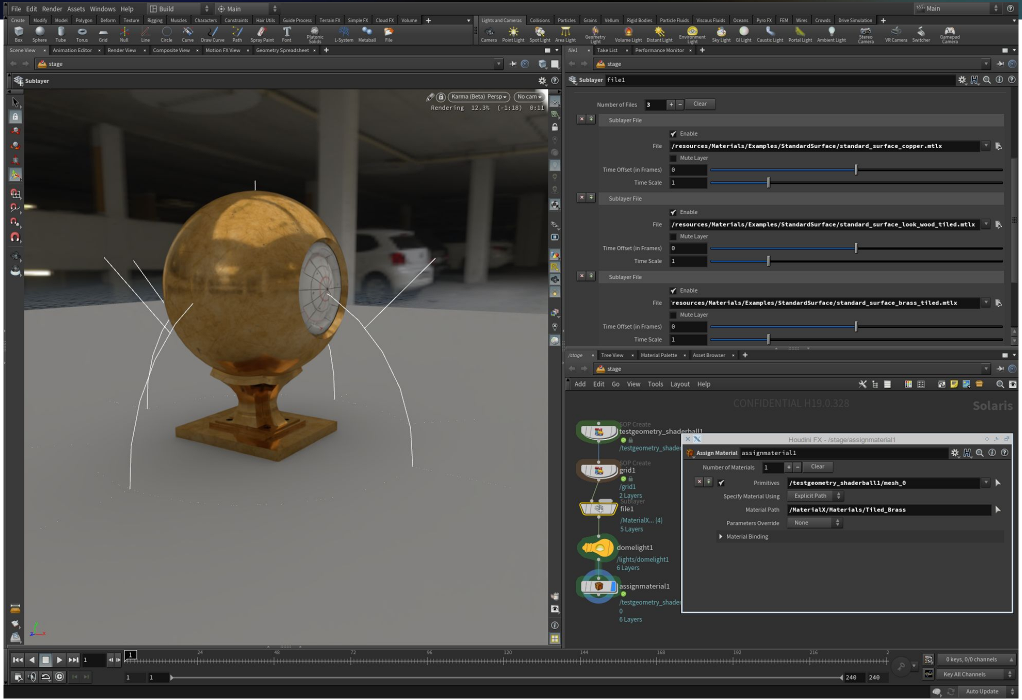
- Houdini/Solaris
  - Read .mtlx directly into LOPs (free with Usd 21.05)
  - New set of MaterialX nodes in the shader editor
  - Usd tools/workflows/edits
  - Houdini tools/workflows/edits

standard_surface_brass_tiled.mtlx: Storm

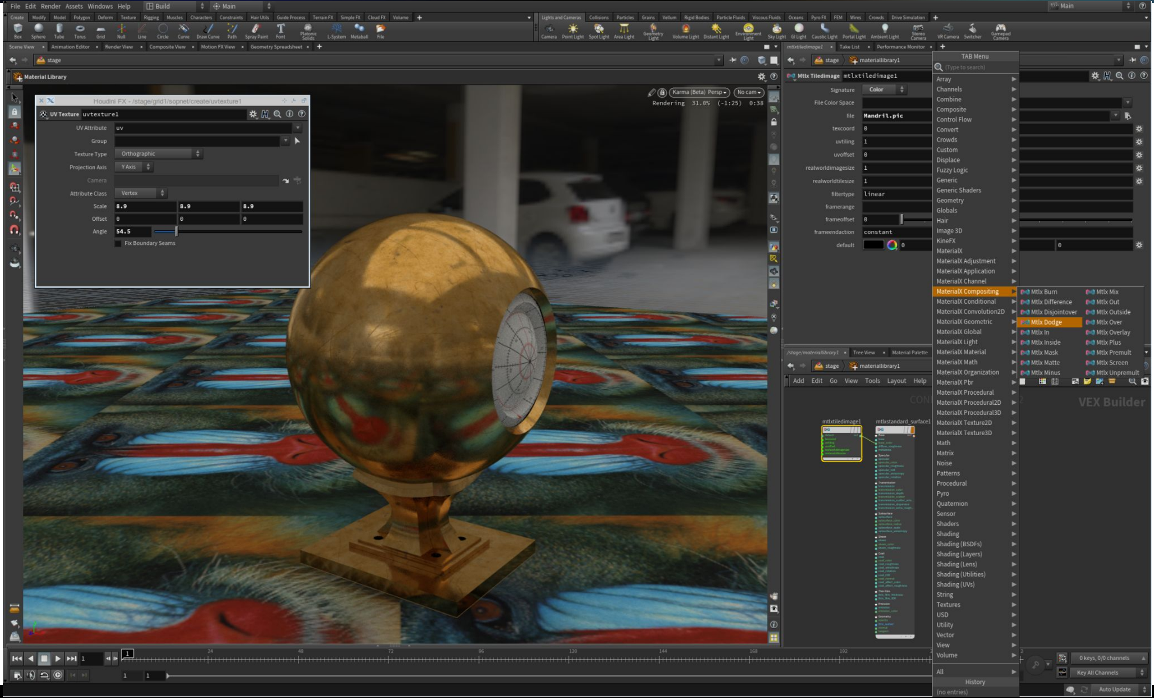
Open Source Days 2021



standard_surface_brass_tiled.mtlx: Karma



Building shader networks



Usd Workflow

BB-8 and R2-D2 © & ™ Lucasfilm LTD. Used with permission.

Thank You

Open Source Days 2021

HOSTED BY /* ACADEMY SOFTWARE FOUNDATION

MaterialX and the Adobe Standard Material Model

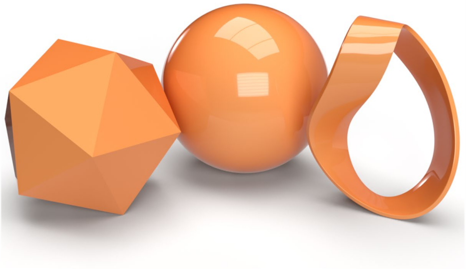
Paul Edmondson
Senior Graphics Engineer, Adobe

Adobe and 3D

- Adobe has a long history of forays into 3D
  - Photoshop, After Effects, Dimension(s), Aero
- Allegorithmic and Medium teams joined Adobe in 2019
- Substance 3D Collection released in 2021:
  - Designer
  - Painter
  - Sampler
  - Stager
  - Modeler (beta)

The Adobe Standard Material

- A common, unified basis for material interchange between tools
- Not a shader to end all shaders, but a model for data-driven look alignment
- Goals:
  - Support for use by raster-based and traced renderers
  - Backward compatibility with materials in existing tools
  - Maximal interoperability between apps for most materials
  - Emphasis on art-directability and ease of use where possible
  - Not bound to any single language or file format
  - Don’t reinvent the wheel

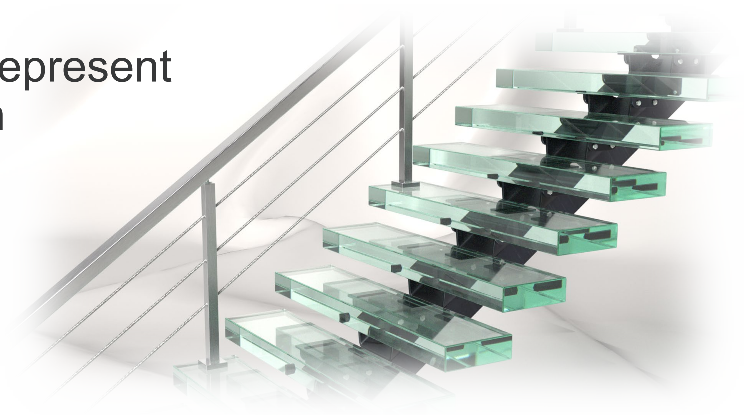


Version 4 Features

- Expanded metal/roughness PBR model
- Native material model for Substance 3D Stager via MDL and SBSAR
- Compatibility with Substance 3D Assets, Painter, and Designer
- Technical specification to be posted in coming weeks
- Translatable with minimal loss to/from Disney PBR, glTF, USD Preview Surface, Autodesk Standard Surface, etc. for supported features

Why we didn't use ____________

- The Adobe Standard Material (ASM) model is not meant to supplant all other material models
- The Substance 3D tools continue to support many alternate models
- We needed something that could represent the intersection of our toolsets with minimal loss of information
- We also wanted to support interchange with external DCC tools

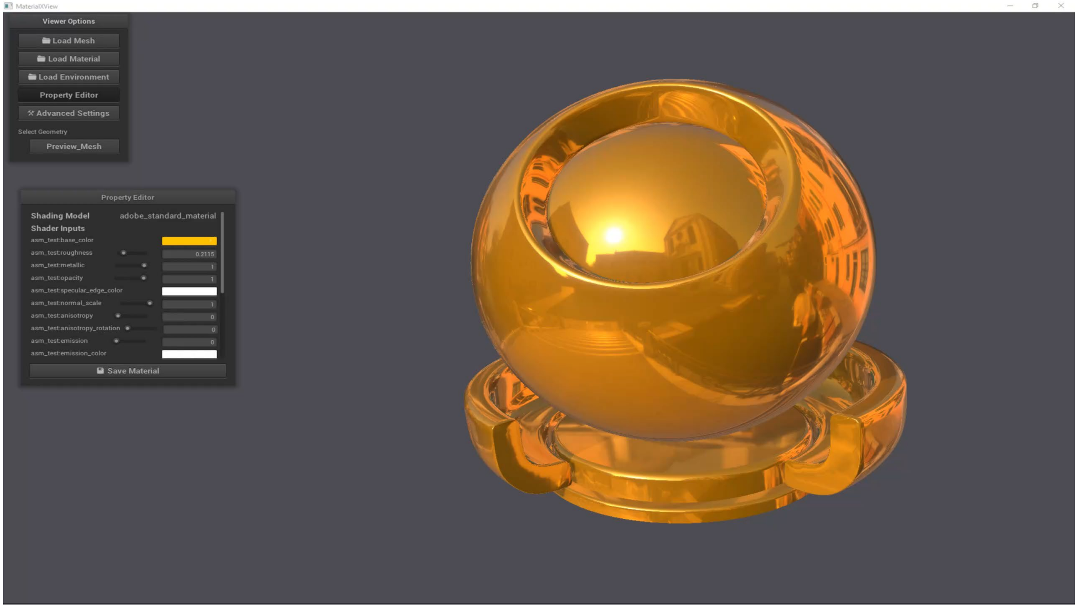


MaterialX Prototype

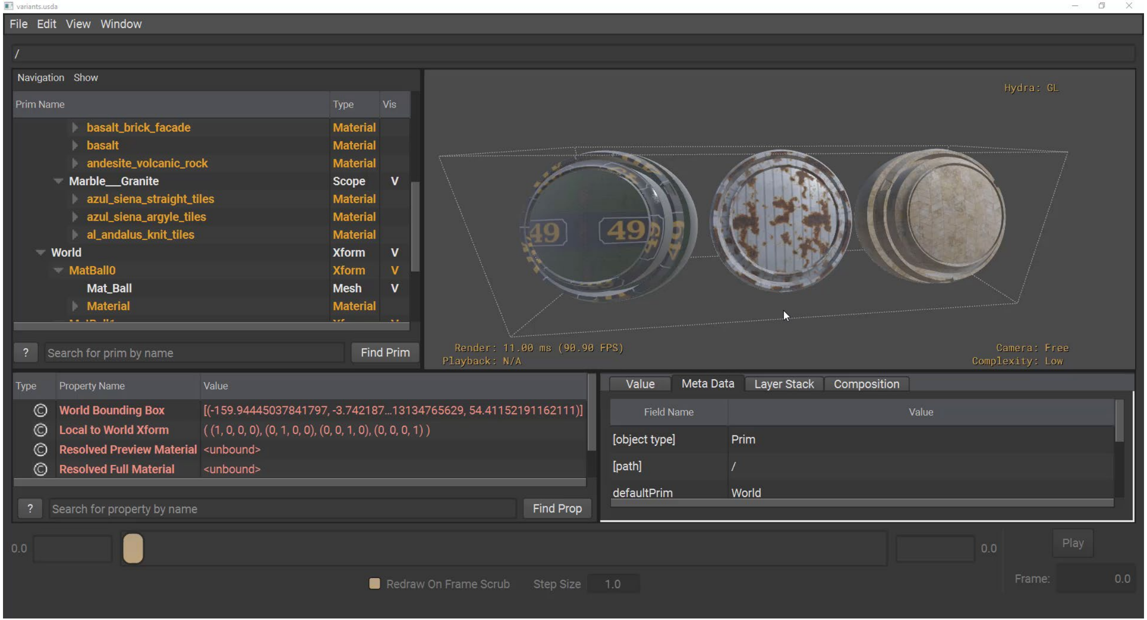
- Eventual support of MaterialX was always planned for
- Only prototype stage at present, but with eye toward future
- Relatively easy to get reasonable results with standard PBR nodes
- Some BxDFs will require customization down the line for full visual fidelity

```
input name="base_color"                   type="color3"  value="0.5, 0.5, 0.5" uniform="fa
input name="roughness"                    type="float"   value="0.5"           uniform="fals
input name="metallic"                     type="float"   value="0"             uniform="fals
input name="opacity"                      type="float"   value="1"             uniform="fals
input name="ambient_occlusion"            type="float"   value="1"             uniform="fals
input name="specular_level"               type="float"   value="0.5"           uniform="fals
input name="specular_edge_color"          type="color3"  value="1, 1, 1"       uniform="fals
input name="normal"                       type="vector3"                       uniform="fals
input name="tangent"                      type="vector3"                       uniform="fals
input name="normal_scale"                 type="float"   value="1"             uniform="true
input name="combine_normal_and_height"    type="boolean" value="false"         uniform="true
input name="height"                       type="float"   value="0.5"           uniform="fals
input name="height_scale"                 type="float"   value="1"             uniform="true
input name="height_level"                 type="float"   value="0.5"           uniform="true
input name="anisotropy"                   type="float"   value="0"             uniform="fals
input name="anisotropy_rotation"          type="float"   value="0"             uniform="fals
input name="emission"                     type="float"   value="0"             uniform="true
input name="emission_color"               type="color3"  value="1, 1, 1"       uniform="fals
input name="sheen"                        type="float"   value="0"             uniform="fals
input name="sheen_color"                  type="color3"  value="1, 1, 1"       uniform="fals
input name="sheen_roughness"              type="float"   value="0.5"           uniform="fals
input name="translucency"                 type="float"   value="0"             uniform="fals
input name="thin_walled"                  type="boolean" value="false"         uniform="true
input name="absorption_color"             type="color3"  value="1, 1, 1"       uniform="fals
input name="absorption_distance"          type="float"   value="0"             uniform="true
input name="specular_ior"                 type="float"   value="1.5"           uniform="true
input name="dispersion"                   type="float"   value="0"             uniform="true
input name="scatter"                      type="boolean" value="false"         uniform="true
input name="scatter_color"                type="color3"  value="1, 1, 1"       uniform="fals
input name="scatter_distance"             type="float"   value="1"             uniform="true
input name="scatter_distance_scale"       type="color3"  value="1, 1, 1"       uniform="fals
input name="scatter_red_shift"            type="float"   value="0"             uniform="true
input name="scatter_rayleigh"             type="float"   value="0"             uniform="true
input name="scatter_anisotropy"           type="float"   value="0"             uniform="true
input name="volume_thickness"             type="float"   value="1"             uniform="fals
input name="volume_thickness_scale"       type="float"   value="1"             uniform="true
input name="coat"                         type="float"   value="0"             uniform="fals
input name="coat_color"                   type="color3"  value="1, 1, 1"       uniform="fals
input name="coat_roughness"               type="float"   value="0"             uniform="fals
input name="coat_ior"                     type="float"   value="1.6"           uniform="true
input name="coat_specular_level"          type="float"   value="0.5"           uniform="fals
input name="coat_normal"                  type="vector3"                       uniform="fa
```

ASM in MaterialXView

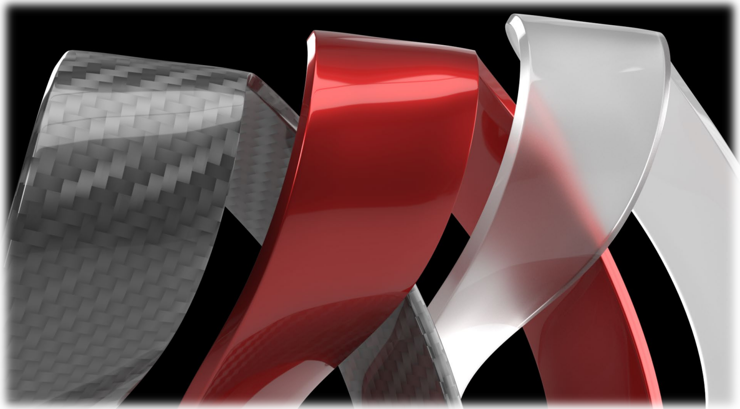


ASM in MaterialX + USD



Next Steps

- Finish prototype with standard node support of all properties
- Translation support to other common models like Standard Surface
- Support of additional targets
- Share!



Project Collaborators:

- David Larsson (dlarsson@adobe.com)
- Andréa Machizaud (machizau@adobe.com)
- Paul Edmondson (paule@adobe.com)

Open Source Days 2021

HOSTED BY /* ACADEMY SOFTWARE FOUNDATION

MaterialX ❤ OpenShadingLanguage

Adding MaterialX closures to OSL

Chris Kulla (OSL TSC Chair)

About Me

- Chris Kulla
  - Principal Rendering Engineer at Epic Games
  - Previously at Sony Imageworks
  - Chair of the Technical Steering Committee for OpenShadingLanguage
- OSL TSC meets every other week to discuss the evolution of the open source library

How are MaterialX and OSL related?

- MaterialX
  - Represents shading networks without being tied to a particular renderer
  - Can *generate* OSL code directly (among other backends)
- OpenShadingLanguage
  - A programming language for shading calculations
  - A compiler and execution framework to run efficiently on CPU & GPU
  - Primarily used by production path tracers
- MaterialX sits **outside** the renderer, OSL lives **inside** the renderer

What is missing from OSL backend in MaterialX?

- MaterialX has supported OSL for a long time already
  - But recent addition of PBR nodes could not be expressed directly
- OSL thinks about surface/light interaction in abstract terms to provide flexibility to the implementation
  - Details of how light paths are sampled and traced left up to renderer
  - All BxDF,EDF,VDF definitions abstracted as *closures* to be defined
  - OSL specification had an outdated set of recommended standard closures (most implementations defined their own)
- We have decided to adopt MaterialX's PBR nodes as the canonical set of OSL closures!

What does this mean concretely?

1. Update OSL spec to refer to MaterialX's PBR shading nodes
2. Ship a header definition of the expected MaterialX closures
3. Add a reference implementation to OSL's testrender
4. Integrate with MaterialX unit tests

What have we done so far?

1. ☐ Update OSL spec to refer to MaterialX's PBR shading nodes
2. ☑️ Ship a header definition of the expected MaterialX closures
3. ☐ Add a reference implementation to OSL's testrender
4. ☐ Integrate with MaterialX unit tests

What do the closures look like?

```
// Constructs a diffuse reflection BSDF based on the Oren-Nayar reflectance model.
//
//  \param  N           Normal vector of the surface point beeing shaded.
//  \param  albedo      Surface albedo.
//  \param  roughness   Surface roughness [0,1]. A value of 0.0 gives Lambertian reflectance.
//  \param  label       Optional string parameter to name this component. For use in AOVs / LPEs.
//
closure color oren_nayar_diffuse_bsdf(normal N, color albedo, float roughness) BUILTIN;
```

What about layering?

```
// Vertically layer a layerable BSDF such as dielectric_bsdf, generalized_schlick_bsdf or
// sheen_bsdf over a BSDF or VDF. The implementation is target specific, but a standard way
// of handling this is by albedo scaling, using "base*(1-reflectance(top)) + top", where
// reflectance() calculates the directional albedo of a given top BSDF.
//
//  \param  top   Closure defining the top layer.
//  \param  base  Closure defining the base layer.
closure color layer(closure color top, closure color base) BUILTIN;
```

Conclusion

- Provide a specification for OSL to express the same set of material properties as MaterialX documents
- Provide a reference implementation of these ideas
- Iterate with the community on the more subtle details of efficient layering, IOR and medium tracking, and other conventions and best practices

Open Source Days

2021

HOSTED BY /* ACADEMY SOFTWARE FOUNDATION

Thank you!

Questions?

Open Source Days

2021

HOSTED BY /* ACADEMY SOFTWARE FOUNDATION