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

## Presentations

| | |
|---|---|
| Doug Smythe | MaterialX: What's New Since 2016 |
| Jonathan Stone | MaterialX Open Source Project |
| Larry Gritz | OSL Standard Nodes for MaterialX |
| Guido Quaroni | MaterialX Support in USD |
| Niklas Harrysson | MaterialX @ Autodesk |
| Discussion and Q & A | |

## MaterialX Overview

- Schema and File Format used to describe "complete CG object looks":
  - Shading network topology and connections
  - Complex nested materials
  - Texture assignments
  - Geometric assignments
  - Illumination and shadowing assignments for intrinsic asset lights

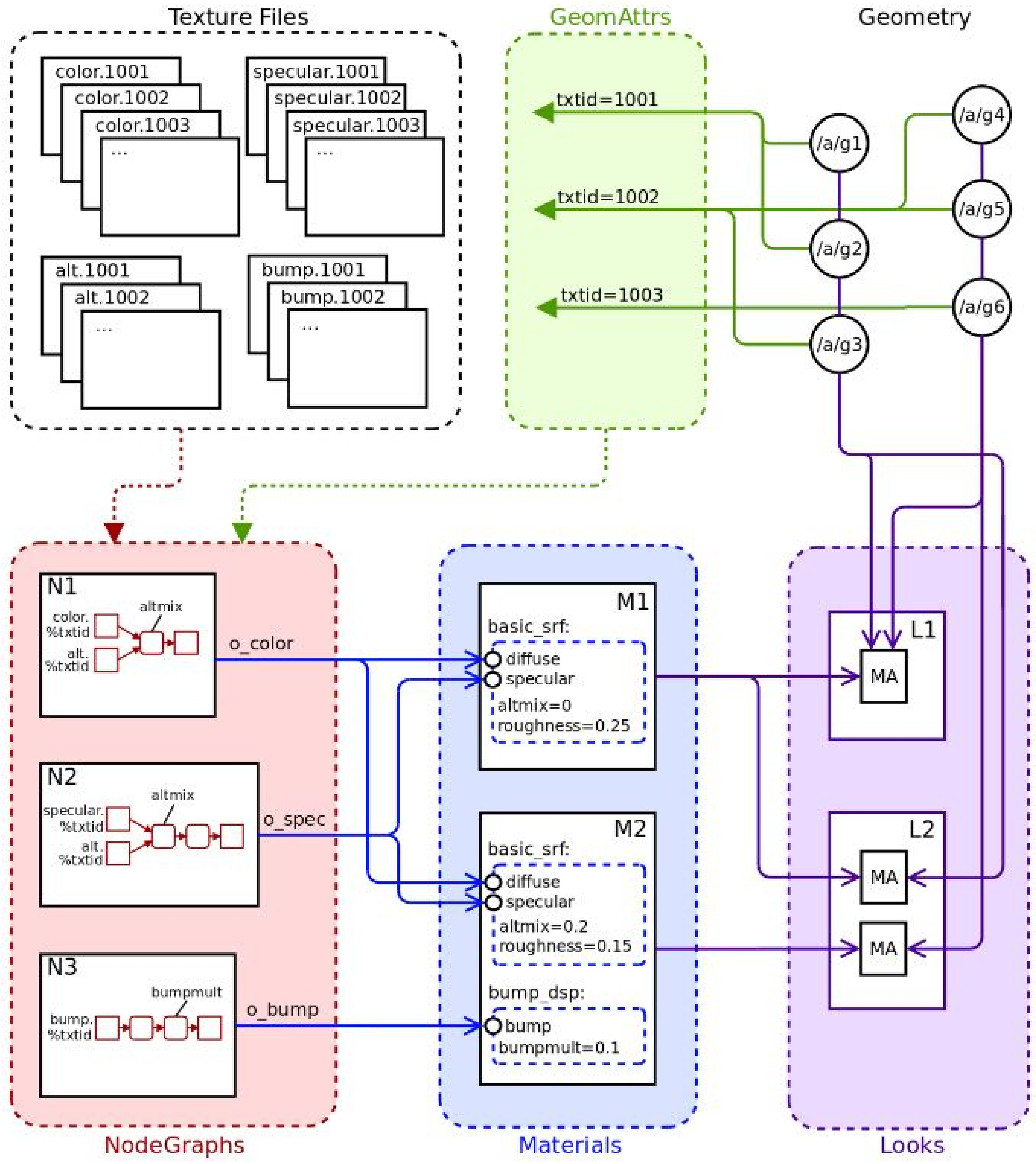
- Specific defined behavior for "Standard Nodes"

## MaterialX Uses

- Transferring looks from one package and/or renderer to another
- Universal material libraries
- Platform-independent asset archiving

## Features of MaterialX

- Node and connection-based, rather than shader parameter list-based
- Strong data typing
- Fully color managed
- Compatible with (but does not require) other open standards
  - OpenColorIO, Alembic, OpenEXR, OSL
- "Live, Not Baked": Setups remain editable after re-import
- Extensible, with support for application- and studio-specific node parameters, user-defined structured data types, and custom operator nodes



## MaterialX Overview

## What's Happened in the Past Year

- Worked with Autodesk to improve MaterialX's capabilities and address various issues discovered during implementation
- Worked with Pixar's USD team to align the data model of USDShade with that of MaterialX
- Three updates to the MaterialX Specification, now at v1.35
- Open Source! https://github.com/materialx/MaterialX
- Full details at materialx.org

## Changes to the Specification

- Unify and simplify how various elements are defined and connected
- Fix things that were unclear or ambiguous
- New capabilities

## Changes to the Specification: Shaders

- Shaders are now simply "nodes", with a node output type declared with a "shader" semantic
  - E.g. nodes that output type "surfaceshader", "displacementshader", etc.
- BxDF shaders can be freely intermixed with regular Pattern and Source nodes within Nodegraphs
- Shaders connect to other shaders using compatible input types rather than specialized "coshader" connections and AOVs/AOVsets.
  - Coshader, aov, and aovset elements no longer needed, and have been removed

# Changes to the Specification: Materials

- Materials can now only bind values or input connections to top-level shaders, not anything feeding into it.

- This leads to a simpler yet more powerful approach to defining network-based shaders, including layered shaders:
  1. Describe connections of multiple nodes and shaders within a nodegraph with (e.g.) output type "surfaceshader"
  2. Use a nodedef to provide an external "shader" interface to it
  3. <Shaderref> that in the material.

## Changes to the Specification: Visibility

- New generalized <visibility> element
  - Defines visibility of a particular category between one or more "viewer geometries" and one or more "other geometries"
  - Standard categories: camera, illumination, shadow, secondary
    - Can define additional arbitrary categories
  - Used within looks

- Replaces standard properties "invisible", "vistocamera", "vistoshadow" and "vistosecondary"

## Changes to the Specification: Nodegraph Nodes

- Unified node connection syntax: Nodes connect to other nodes using <input>s rather than parameters of a special type

- Node inputs can perform inline type conversion, connect to one member within a custom type, or swizzle channels upon input

- New standard source: fractal3d
- New standard operators modulo, smoothstep, floor, dotproduct, crossproduct, scale, rotate2d, compare, switch, swizzle, pack

- Removed operators: reorder, convert (made redundant by swizzle and pack)

## Changes to the Specification: Lights

- Lights are now syntactically treated exactly the same way as any other geometry
  - Refer to a light via its pathed name in the geometry scene graph
  - Assigned to a shader using a <materialassign> in a look
  - No separate <light> element required

- Uses the <visibility> element to define what a light illuminates and what geometry cast shadows from the light
  - <lightillum>, <lightshadow> and <lightassign> elements all removed

## Changes to the Specification: Miscellaneous

- Custom Data Types are now full structs, with named/typed member variables

- Namespaces

- Geometry names can now include wildcard expressions, rather than using separate "regex" attributes

- Various syntax changes for names and parameter values:
  - Hierarchical property names use "_" instead of ":" as group separator
  - interfacename attribute instead of "$paramname" for referencing custom node parameters
  - materialvar attribute instead of "@varname" for materialvar substitutions

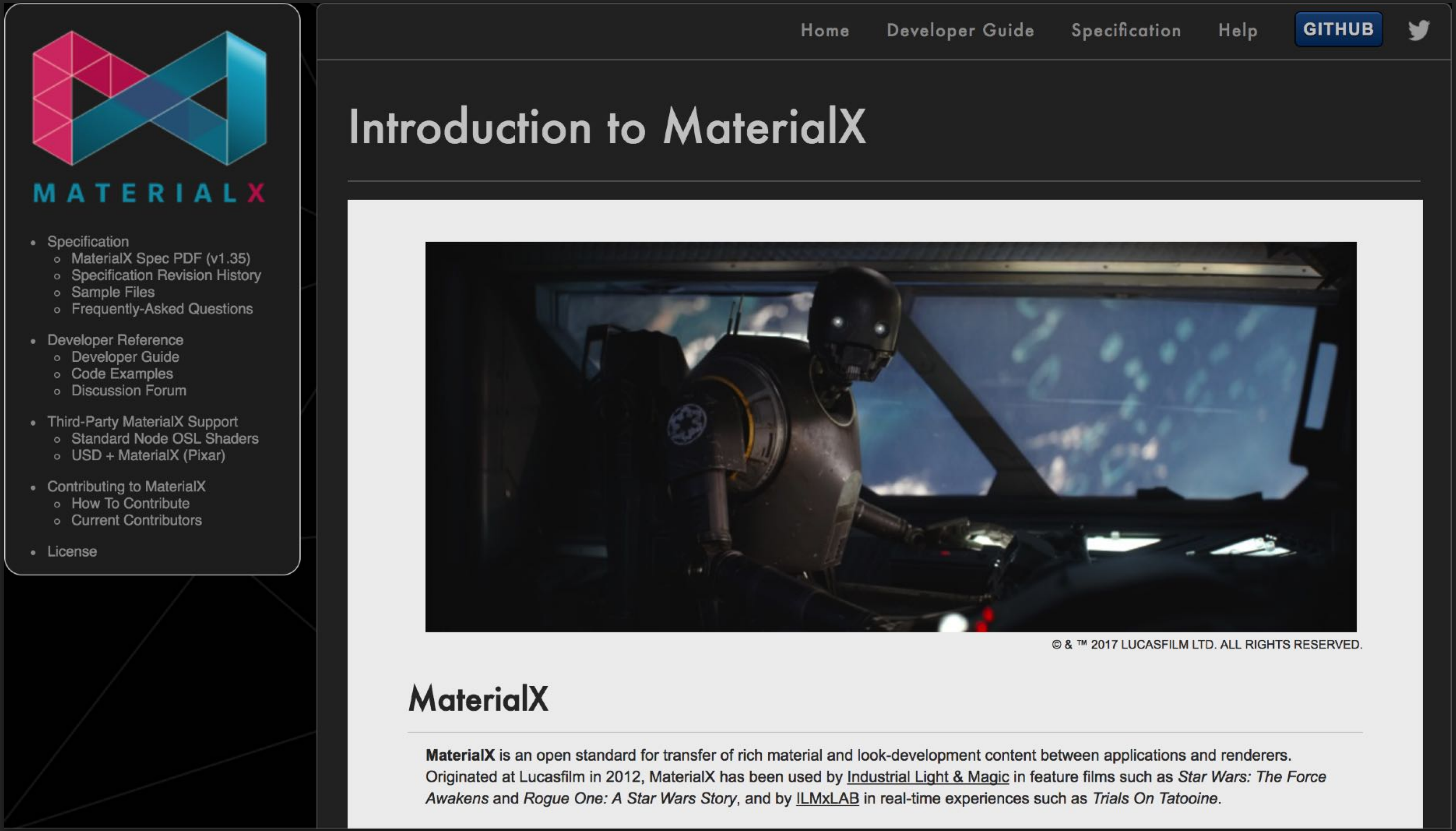
# MaterialX Open Source Project

+ Jonathan Stone, Lucasfilm ADG

## Open Source Launch

- . Open Source Launch
- . Library Introduction
- . Next Steps

# MaterialX Open Source



- Specification
  - MaterialX Spec PDF (v1.35)
  - Specification Revision History
  - Sample Files
  - Frequently-Asked Questions
- Developer Reference
  - Developer Guide
  - Code Examples
  - Discussion Forum
- Third-Party MaterialX Support
  - Standard Node OSL Shaders
  - USD + MaterialX (Pixar)
- Contributing to MaterialX
  - How To Contribute
  - Current Contributors
- License

## Introduction to MaterialX

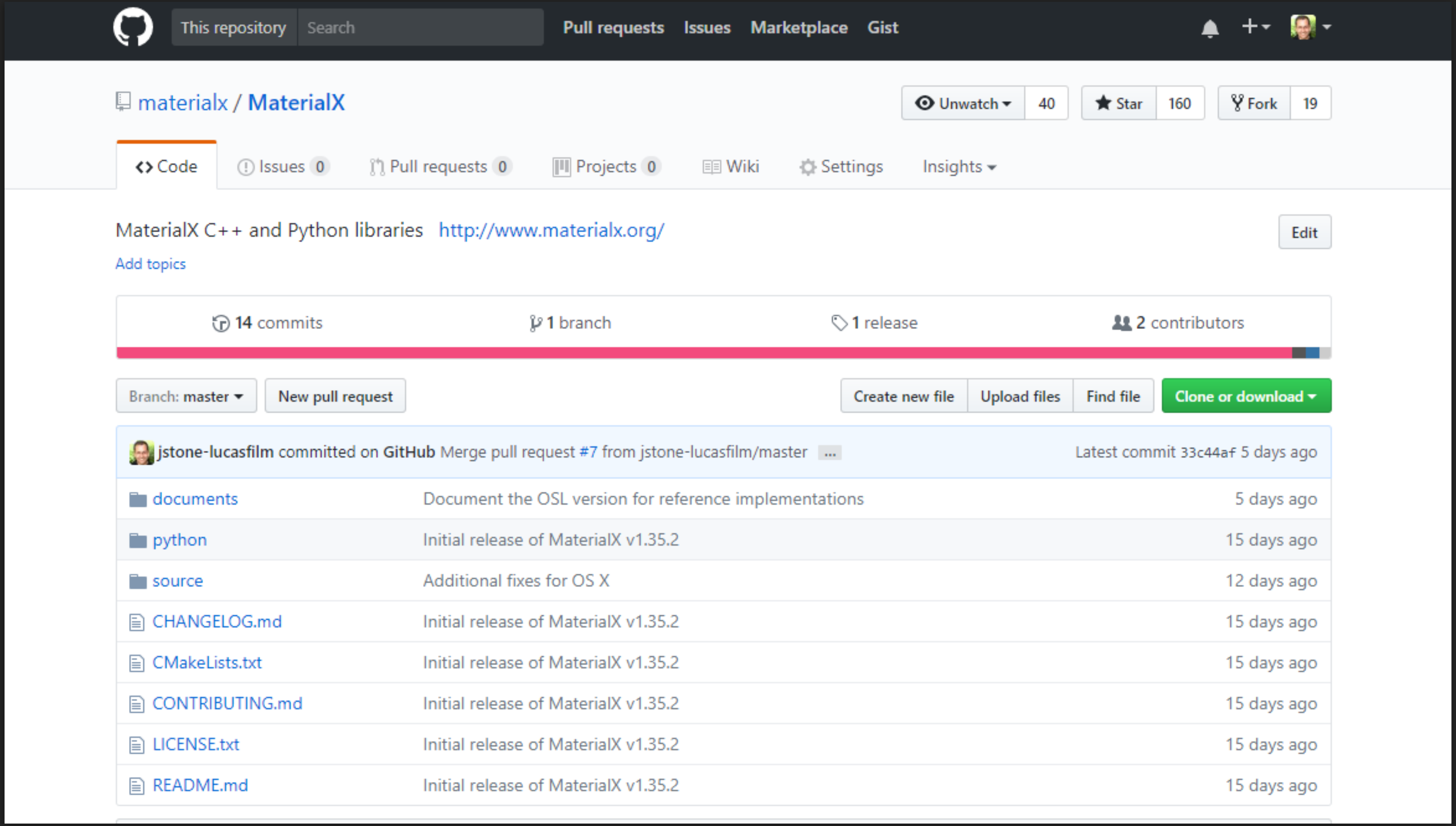
© & ™ 2017 LUCASFILM LTD. ALL RIGHTS RESERVED.

### MaterialX

**MaterialX** is an open standard for transfer of rich material and look-development content between applications and renderers. Originated at Lucasfilm in 2012, MaterialX has been used by Industrial Light & Magic in feature films such as *Star Wars: The Force Awakens* and *Rogue One: A Star Wars Story*, and by ILMxLAB in real-time experiences such as *Trials On Tatooine*.

www.materialx.org

## MaterialX GitHub


materialx / MaterialX

Unwatch 40 | Star 160 | Fork 19

MaterialX C++ and Python libraries http://www.materialx.org/

14 commits | 1 branch | 1 release | 2 contributors

jstone-lucasfilm committed on GitHub Merge pull request #7 from jstone-lucasfilm/master … Latest commit 33c44af 5 days ago

| | | |
|---|---|---|
| documents | Document the OSL version for reference implementations | 5 days ago |
| python | Initial release of MaterialX v1.35.2 | 15 days ago |
| source | Additional fixes for OS X | 12 days ago |
| CHANGELOG.md | Initial release of MaterialX v1.35.2 | 15 days ago |
| CMakeLists.txt | Initial release of MaterialX v1.35.2 | 15 days ago |
| CONTRIBUTING.md | Initial release of MaterialX v1.35.2 | 15 days ago |
| LICENSE.txt | Initial release of MaterialX v1.35.2 | 15 days ago |
| README.md | Initial release of MaterialX v1.35.2 | 15 days ago |

https://github.com/materialx/MaterialX

## Thanks To Our Partner Teams

- Autodesk
  - Niklas Harrysson, Bernard Kwok, Eric Bourque
- OSL team at Sony Pictures Imageworks
  - Larry Gritz, Adam Martinez, Derek Haase
- USD team at Pixar
  - Guido Quaroni, Sebastian Grassia, Davide Pesare

## Library Introduction

. Open Source Launch

. Library Introduction

. Next Steps

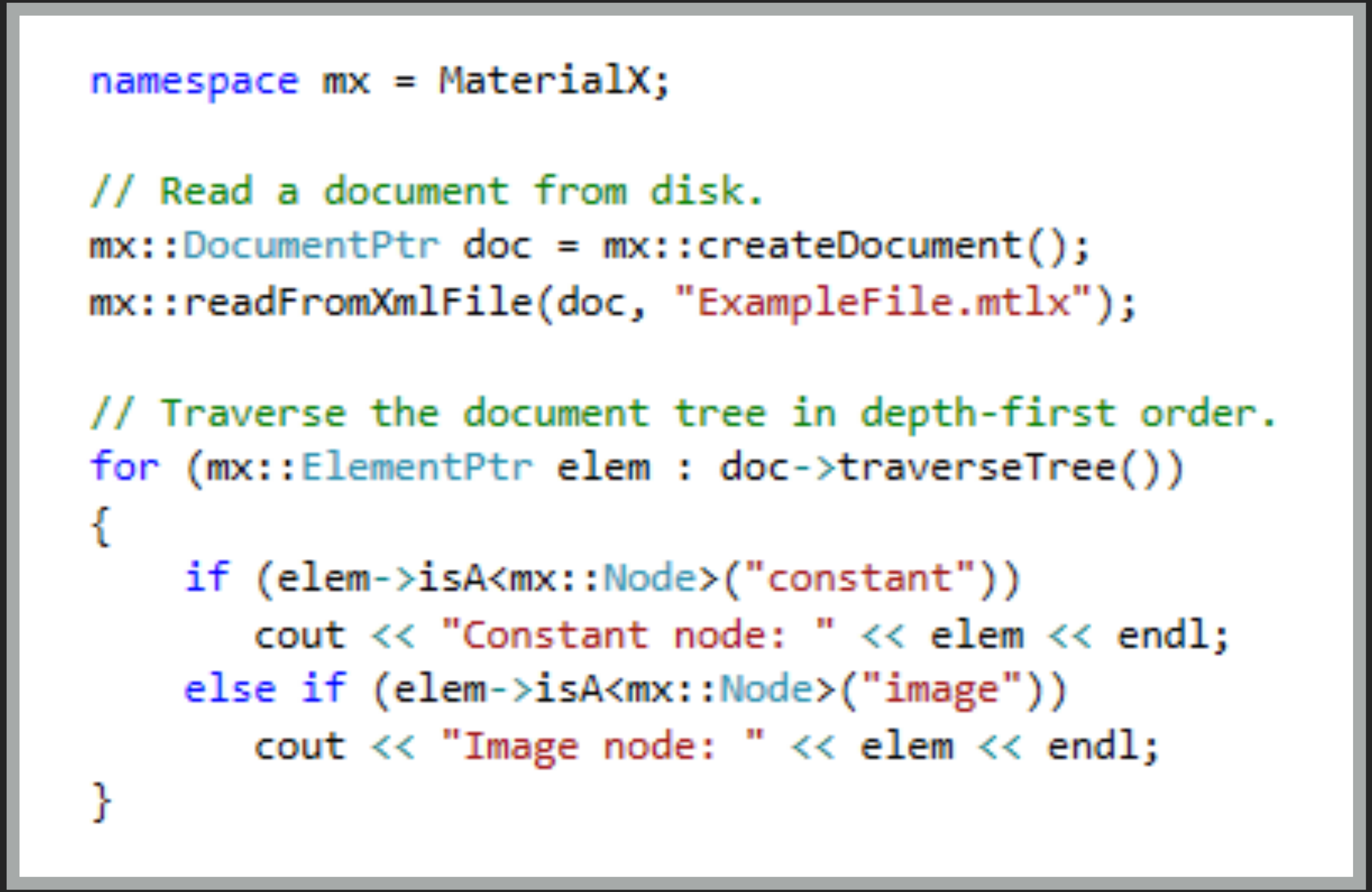
## What’s In The Open Source Library?

- C++11 codebase with Python bindings
  - CMake build system
  - PyBind11 bindings
  - Very lightweight, with no external code dependencies
- Developer guide
  - C++ API documentation (Doxygen)
  - Simple examples of C++/Python code and MaterialX documents
- OSL definitions for the standard nodes
  - Snapshot of shader generators maintained by the OSL team

## MaterialX C++ Modules

- **MaterialXCore**
  - Creation, editing, and traversal of documents and graphs
- **MaterialXFormat**
  - Reading and writing reference MTLX files
- **MaterialXTest**
  - Unit tests (Catch)
- **PyMaterialX**
  - Python bindings (PyBind11)

## MaterialX Code Example

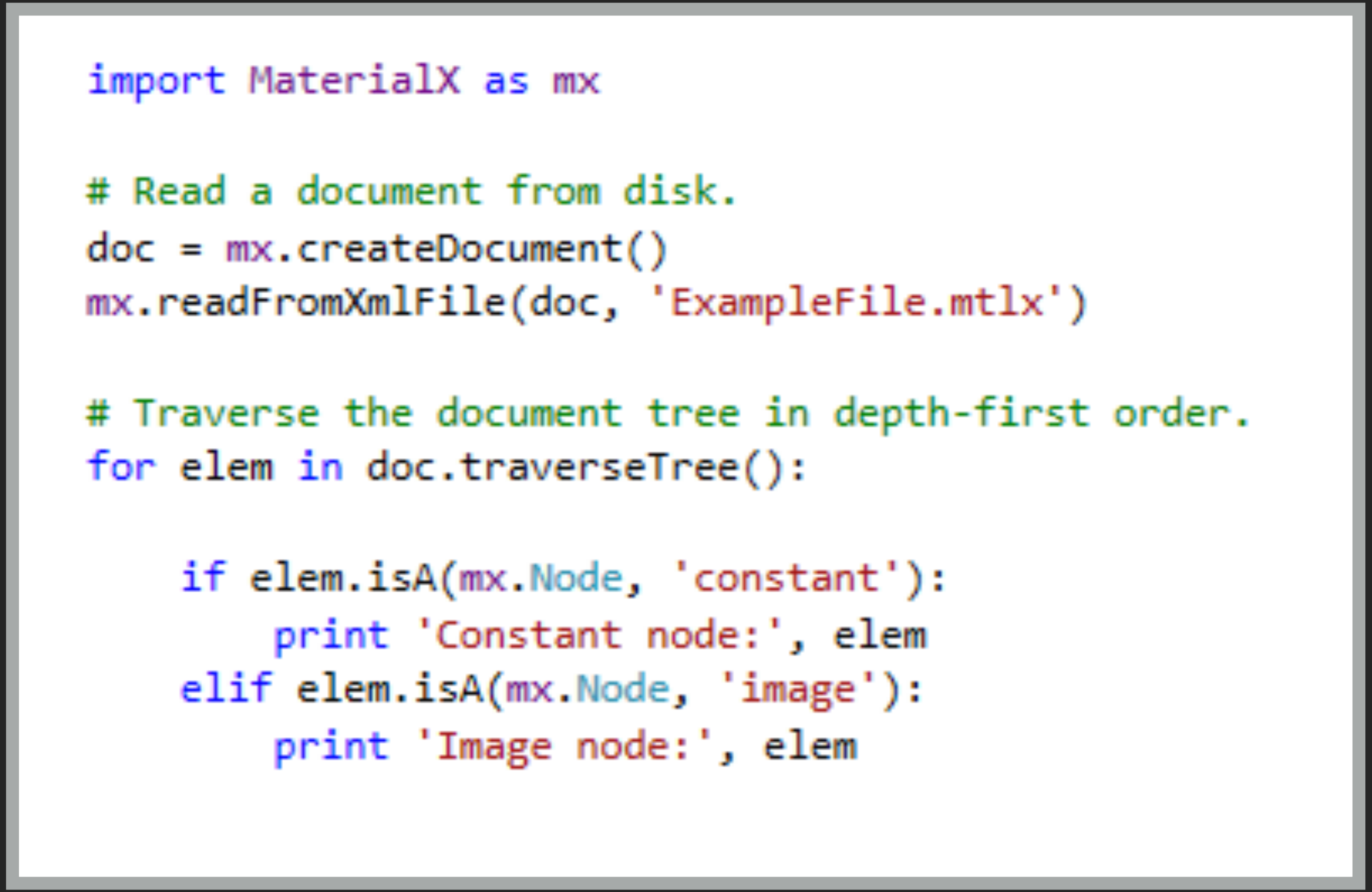


```cpp
namespace mx = MaterialX;

// Read a document from disk.
mx::DocumentPtr doc = mx::createDocument();
mx::readFromXmlFile(doc, "ExampleFile.mtlx");

// Traverse the document tree in depth-first order.
for (mx::ElementPtr elem : doc->traverseTree())
{
    if (elem->isA<mx::Node>("constant"))
       cout << "Constant node: " << elem << endl;
    else if (elem->isA<mx::Node>("image"))
       cout << "Image node: " << elem << endl;
}
```

C++



```python
import MaterialX as mx

# Read a document from disk.
doc = mx.createDocument()
mx.readFromXmlFile(doc, 'ExampleFile.mtlx')

# Traverse the document tree in depth-first order.
for elem in doc.traverseTree():

    if elem.isA(mx.Node, 'constant'):
        print 'Constant node:', elem
    elif elem.isA(mx.Node, 'image'):
        print 'Image node:', elem
```

Python

http://www.materialx.org/docs/api/codeexamples.html

## Next Steps

- . Open Source Launch
- . Library Introduction
- . Next Steps

## Improved High-Level API

- Allowing common material operations to be performed in a single call, e.g.
  - Iterating through the inputs of a material’s primary shader
  - Checking whether a shader input is uniform or spatially-varying
- Some specific improvements are driven by ILM production
- Interested in additional ideas from the community

## Shared BxDF Shaders

- With interfaces and behaviors that are open and well-defined
- Two potential approaches:
  - MaterialX interfaces for canonical shader implementations
  - Graphs of BxDF nodes
- Additional ideas from the community are welcomed

## Standalone MaterialX Visualizers

- Key in establishing a ground truth for MaterialX content
- Allow visual validation of new importers and exporters
- MaterialX shader generators are a very promising approach
    - See upcoming presentation from Niklas

## Thanks to Our Lucasfilm Colleagues

- Including Rob Bredow, David Brickhill, David Bullock, Francois Chardavoine, Roger Cordes, Laura Evangelista, John Gaeta, Andrew Grant, Ben Grimes, Ed Hanway, Naty Hoffman, Polly Ing, Yoon Bae Kim, Matt Koehler, Hilmar Koch, David Meny, Hayden Landis, Matthew Lausch, Andrew Meshekoff, Robert Molholm, Bekah Noulles, Rachel Rose, Kirk Shimano, and Masuo Suzuki
- We're hiring!
    - Send resumes and demo reels to siggraph2017@ilm.com

## MaterialX
## OpenShadingLanguage
## Reference Implementation

MATERIALX

OSL

## Cast

Adam Martinez and Derek Haase - Mercenary shader writers with over 30 years combined experience in production rendering

Larry Gritz - OSL project architect

Doug Smythe and Jonathan Stone - Creators of the MaterialX specification and API

MATERIALX

http://www.materialx.org

The specification was developed by Doug Smythe and Jonathan Stone at Industrial Light and Magic and was released at Siggraph 2016.

A standardized way to specify the look of cg objects built using shader networks so that these looks (or subcomponents of looks) can be passed from one software package to another.

A strict set of rules for transferring the recipe used to create a cg object from package to package.

The MaterialX - OSL Reference Implementation is a library of atomic shader nodes that adhere to the nodes described in the MaterialX specification v1.35 (pp. 25-40)

Caveats:  blur and ambient occlusion nodes are in the specification, but not in the reference implementation.

## Motivations

Larry Gritz - “OSL should come with some basic shaders.”

Doug Smythe and Jonathan Stone - “MaterialX needs a reference implementation.”

Adam Martinez and Derek Haase - “We like working on shader libraries”

## Benefits

Benefits to OSL:

OSL was lacking a structured shader library, so this is a good fit!

Benefits to MaterialX:

OSL is becoming the defacto shading language, so this is a good fit!

## What this library is and is not

Provides a set of shader nodes to work with any of the MaterialX types: float, color, color2, color4, vector, vector2 and vector4

Provides a high degree of granularity to create more complex, and transferrable, shading networks.

*Does not* provide illumination models or general surface shaders

*Does not* provide complex procedural patterns or effects

## Think of it this way:

The MaterialX OSL reference implementation gives you the building blocks to make complicated, transferrable, shading-networks to describe a chain of operations as input to a surface, displacement or light shader.

## OSL Enhancements

Operator Overloading - allows for the implementation of math operators on user data types:

```
color2 __operator__add__(color2 a, color2 b)
{
   return color2(a.r + a.r, b.a + b.a);
}
```

Constructors - Dramatically simplify code by allowing in-line type declarations:

```
return color2(a.r + a.r, b.a + b.a);
```

New Data Types -  MaterialX-inspired types color2, color4 vector2 and vector4 have been added to OSL via the headers in src/shaders

## Reference Implementation Specifics

Every shader is described by *type-agnostic* `.mx` file in src/shaders/MaterialX

At build time, string substitution generates a *type-specific* flavor of .osl

```
shader mx_max_TYPE_SUFFIX
(
    TYPE in1 = TYPE_DEFAULT_IN,
    TYPE in2 = TYPE_DEFAULT_IN,
    output TYPE out = TYPE_DEFAULT_OUT
)
{
    out = max(in1, in2);
}
```

```
shader mx_max_color4
(
    color4 in1 = {color(0,0,0), 0},
    color4 in2 = {color(0,0,0), 0},
    output color4 out = {color(0,0,0), 0}
)
{
    out = max(in1, in2);
}
```

## Reference Implementation Specifics

The reference implementation builds 475 distinct OSL shaders from 97 .mx definitions

The default OSL library build process generates both readable .osl and bytecode .oso files

Individual shaders, or shaders for specific types, can be built using the build_materialx_osl.py script in src/shaders/MaterialX

The MaterialX distribution includes the .osl shaders as reference.

## Next Steps

The OSL library is still under active development!  As the spec changes, so will the reference implementation.

We would like to continue our work on MaterialX and OSL, with potential Katana development, GLSL implementations and Game Engine integration.

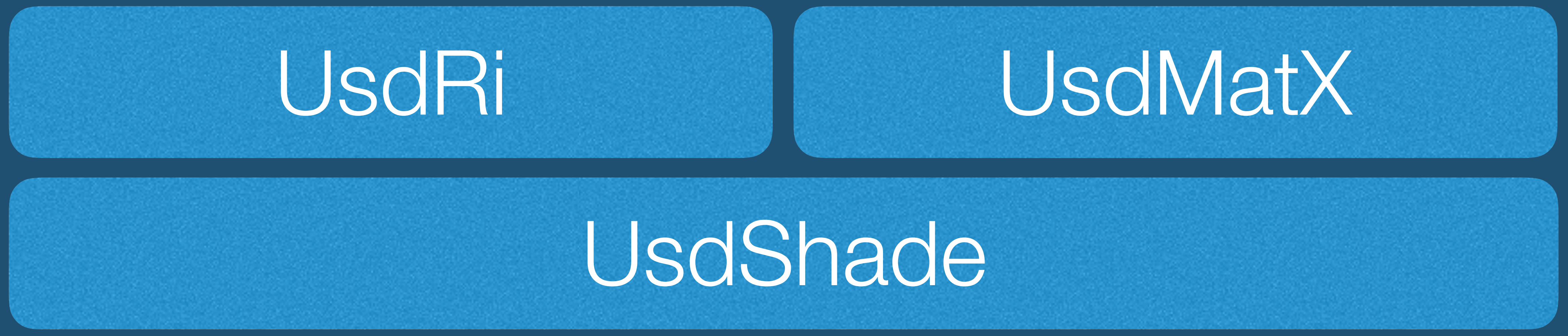
https://github.com/imageworks/OpenShadingLanguage     src/shaders/MaterialX

https://github.com/materialx/MaterialX/     documents/Libraries/Source/OSL

# MaterialX support in USD

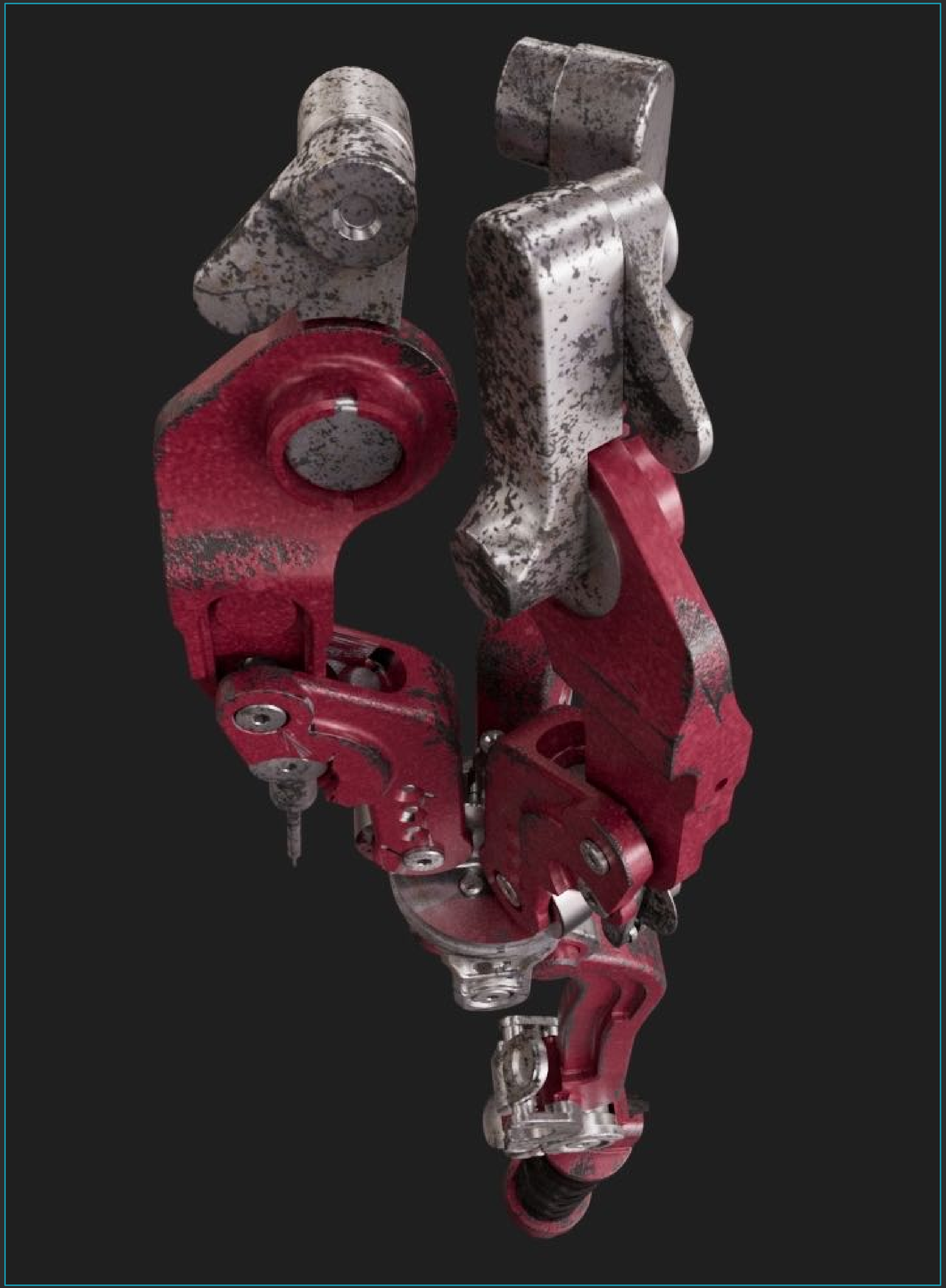
# MaterialX support in USD

- UsdShade core additions:
  - Colorspace, color configuration encoding
  - Collection based material assignment
  - Core schema nodes for UV textures, primvar reader and swizzle
- MaterialX fully encodable in UsdShade via a core UsdMatX extension (Fall 2017)



## MaterialX support in USD

Open Source at Pixar: USD and OpenSubdiv
Tomorrow (Tuesday 8/1) at 4PM. Room 501C



# MaterialX @ Autodesk

Niklas Harrysson

Principal Engineer

Autodesk

# MATERIALX

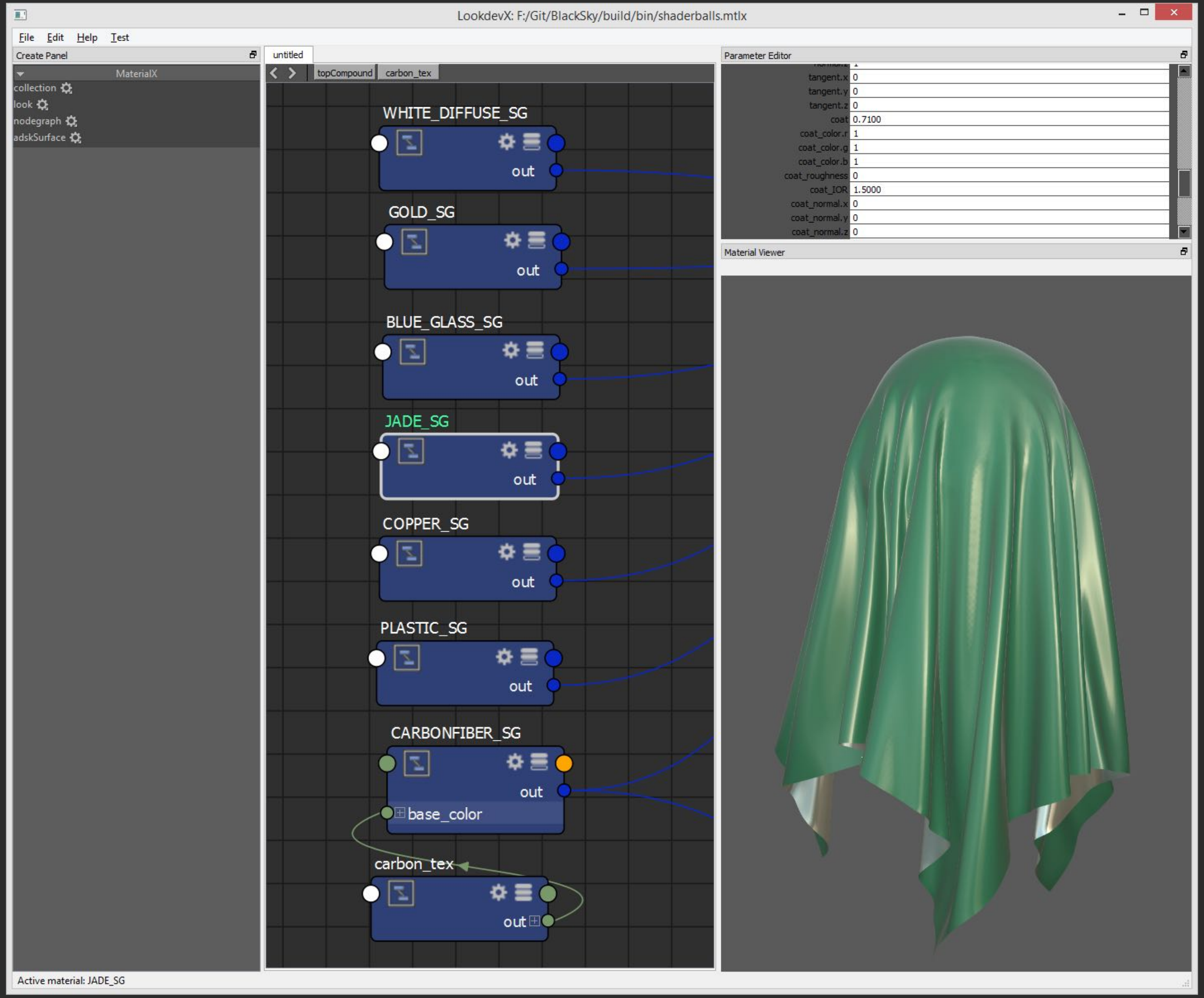
Games
MAYA
MAYA
PlayStation
MAX
PIXAR's RenderMan
iPad
PORTABLE
ART DIRECT-ABLE
360
Autodesk BIFROST
SUITE
MAXWELL RENDER
MAX
Film & TV
Design

## Why MaterialX?

- Application agnostic
- A baseline stdlib of shader nodes
- Extensible
- Not only materials, complete looks!
- Lightweight SDK
- No dependencies
- Open Source - with leading industry partners

## Collaboration | Autodesk's focus

1. In app representation
   - A data model not only for transportation
   - Editable
   - Observable
   - For shader libraries
   - Compounds / sub-graphs

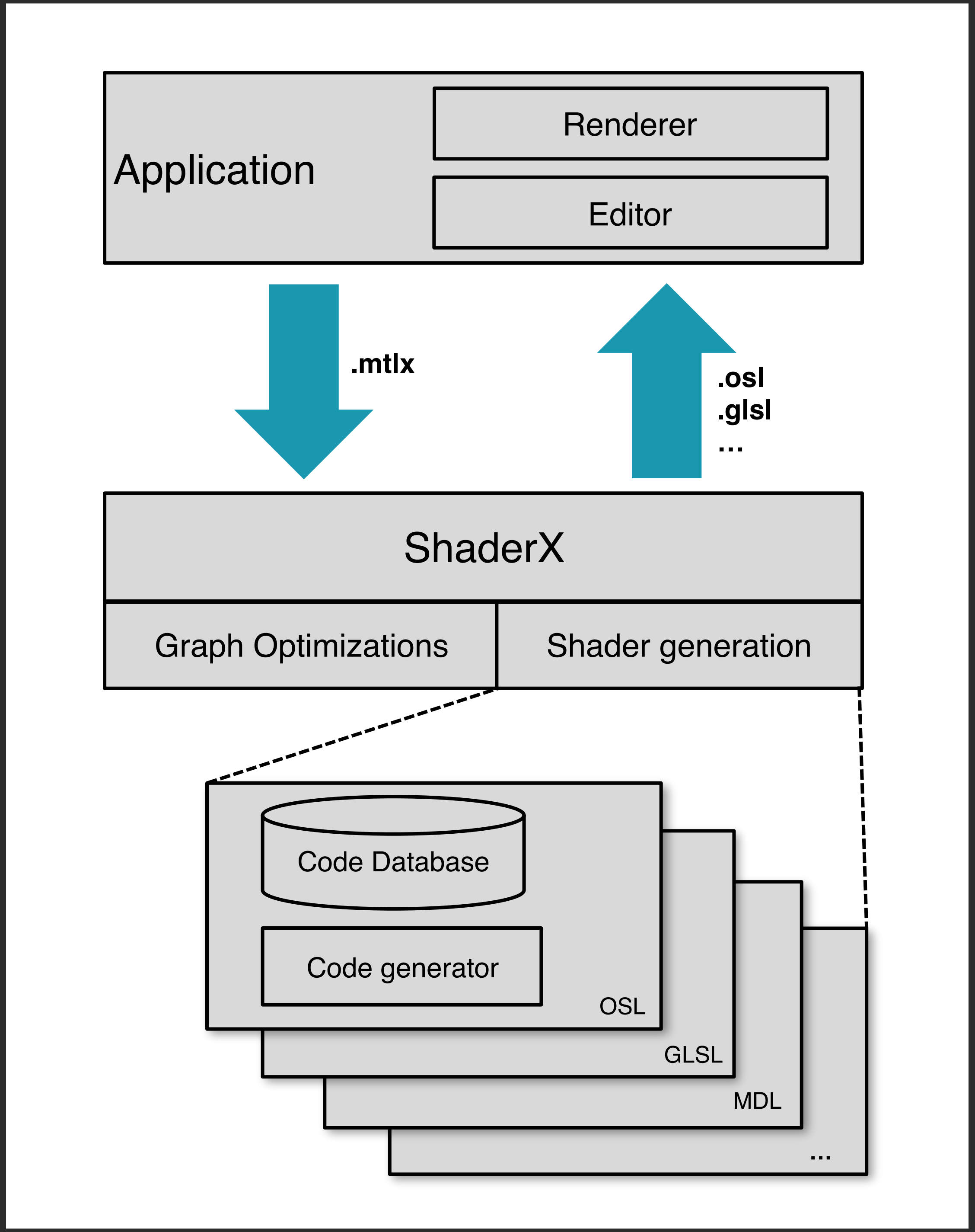

LookdevX: F:/Git/BlackSky/build/bin/shaderballs.mtlx

| | |
|---|---|
| tangent.x | 0 |
| tangent.y | 0 |
| tangent.z | 0 |
| coat | 0.7100 |
| coat_color.r | 1 |
| coat_color.g | 1 |
| coat_color.b | 1 |
| coat_roughness | 0 |
| coat_IOR | 1.5000 |
| coat_normal.x | 0 |
| coat_normal.y | 0 |
| coat_normal.z | 0 |

Active material: JADE_SG

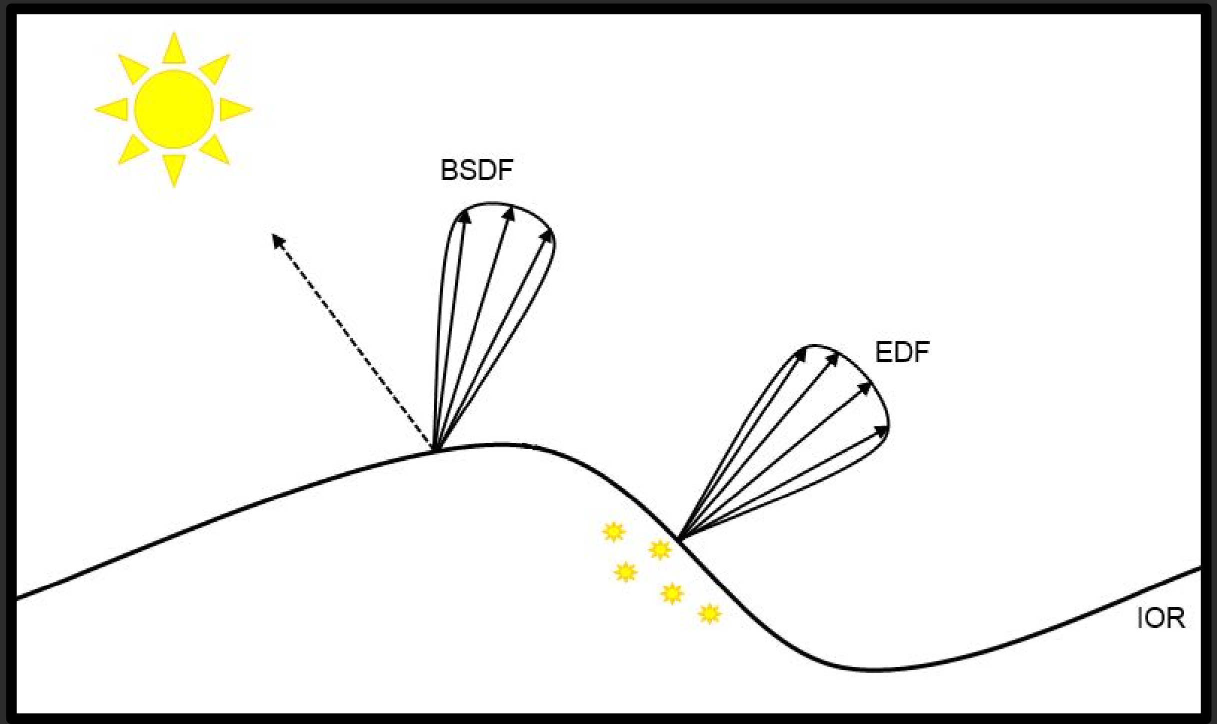
## Collaboration | Autodesk's focus

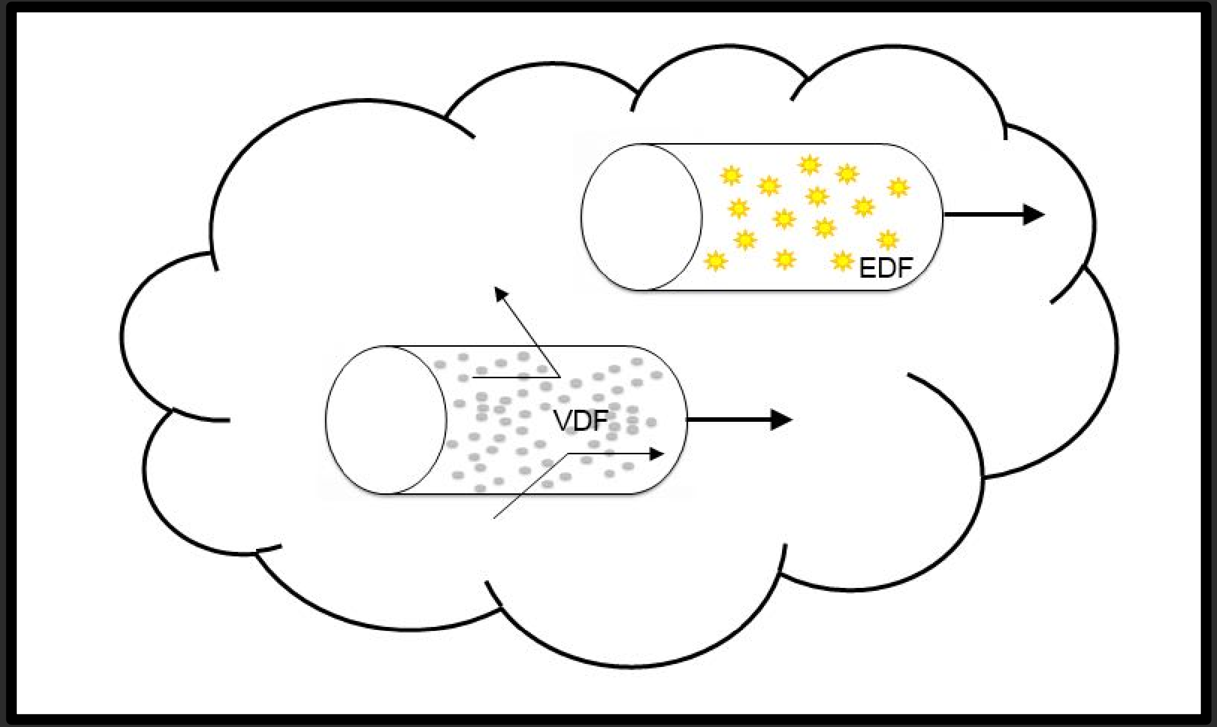
2. Suitable for shader generation
   - Translation to executable shader code
   - Graph traversal
   - Flattening sub-graphs
   - Topological sorting
   - Node implementation descriptions



## ShaderX | Adding a custom node library

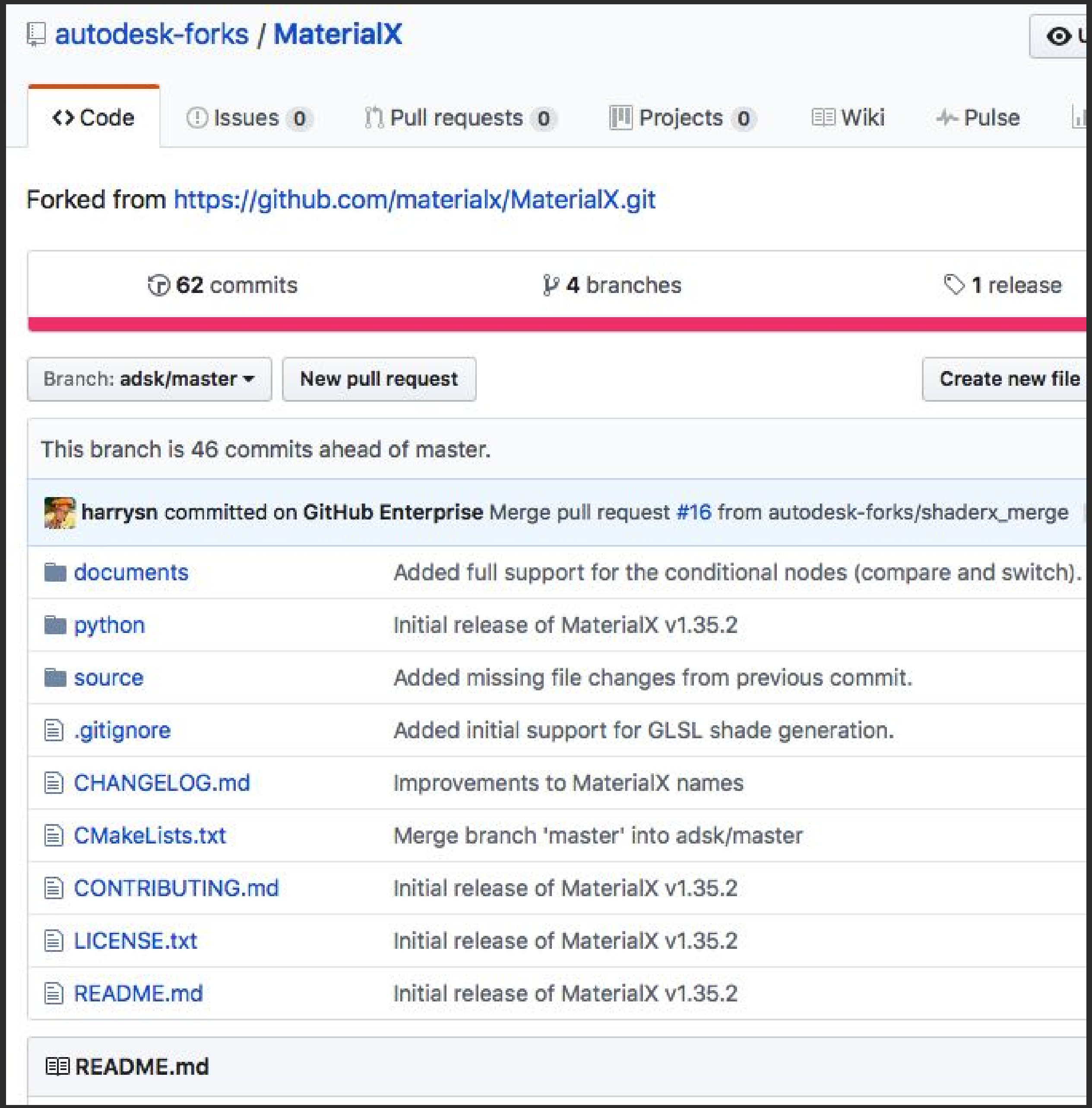
- A library of nodes for physically based shading
- A standard “uber” surface shader
- Nodes for BxDF based shader construction





## ShaderX | A MaterialX fork

- ShaderX now being integrated in a MaterialX fork
- Some work already pushed back to MaterialX master
- Autodesk's fork will be made public when practical
- Continue to push work to MaterialX master
- Remove fork when fully integrated



Forked from https://github.com/materialx/MaterialX.git

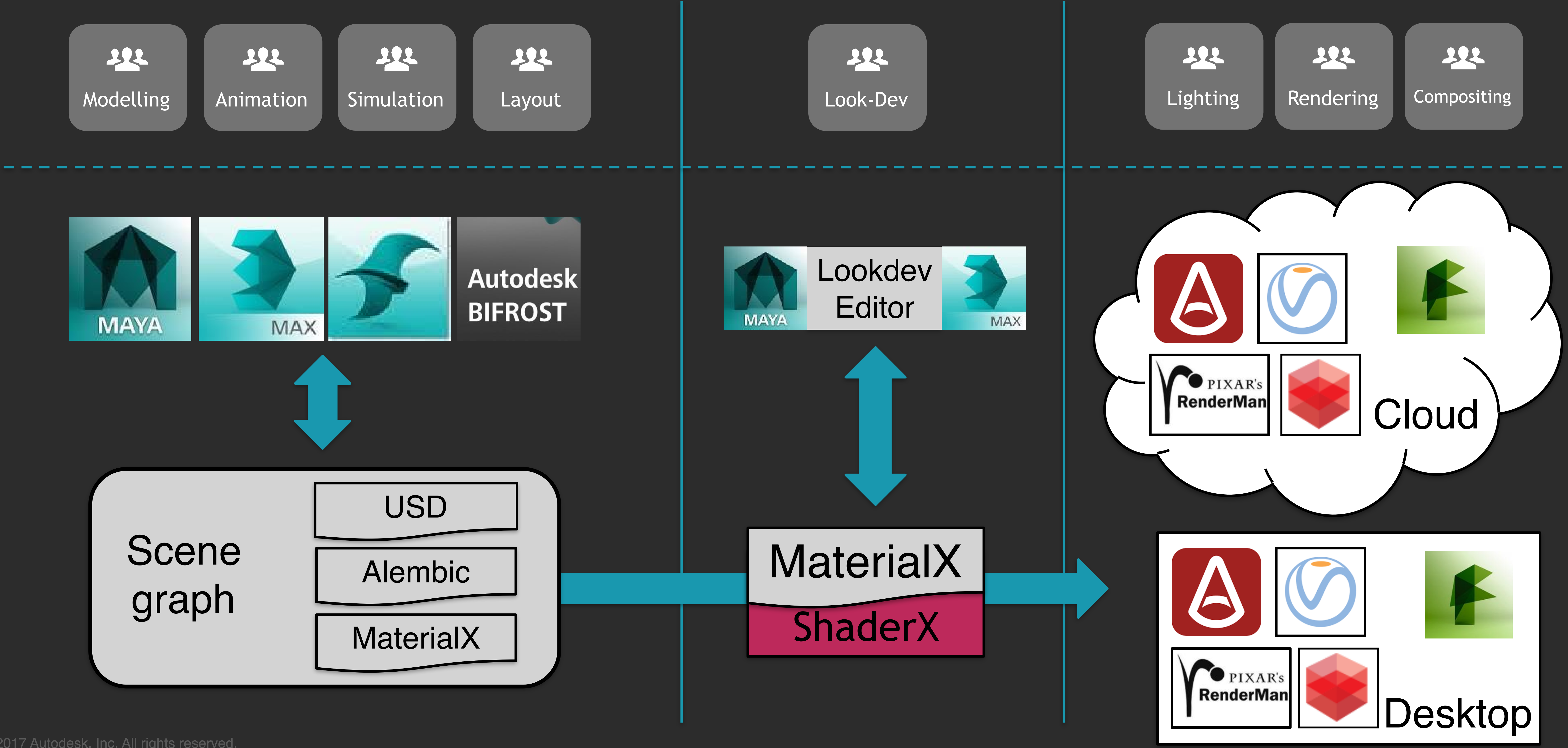
This branch is 46 commits ahead of master.

harrysn committed on GitHub Enterprise Merge pull request #16 from autodesk-forks/shaderx_merge

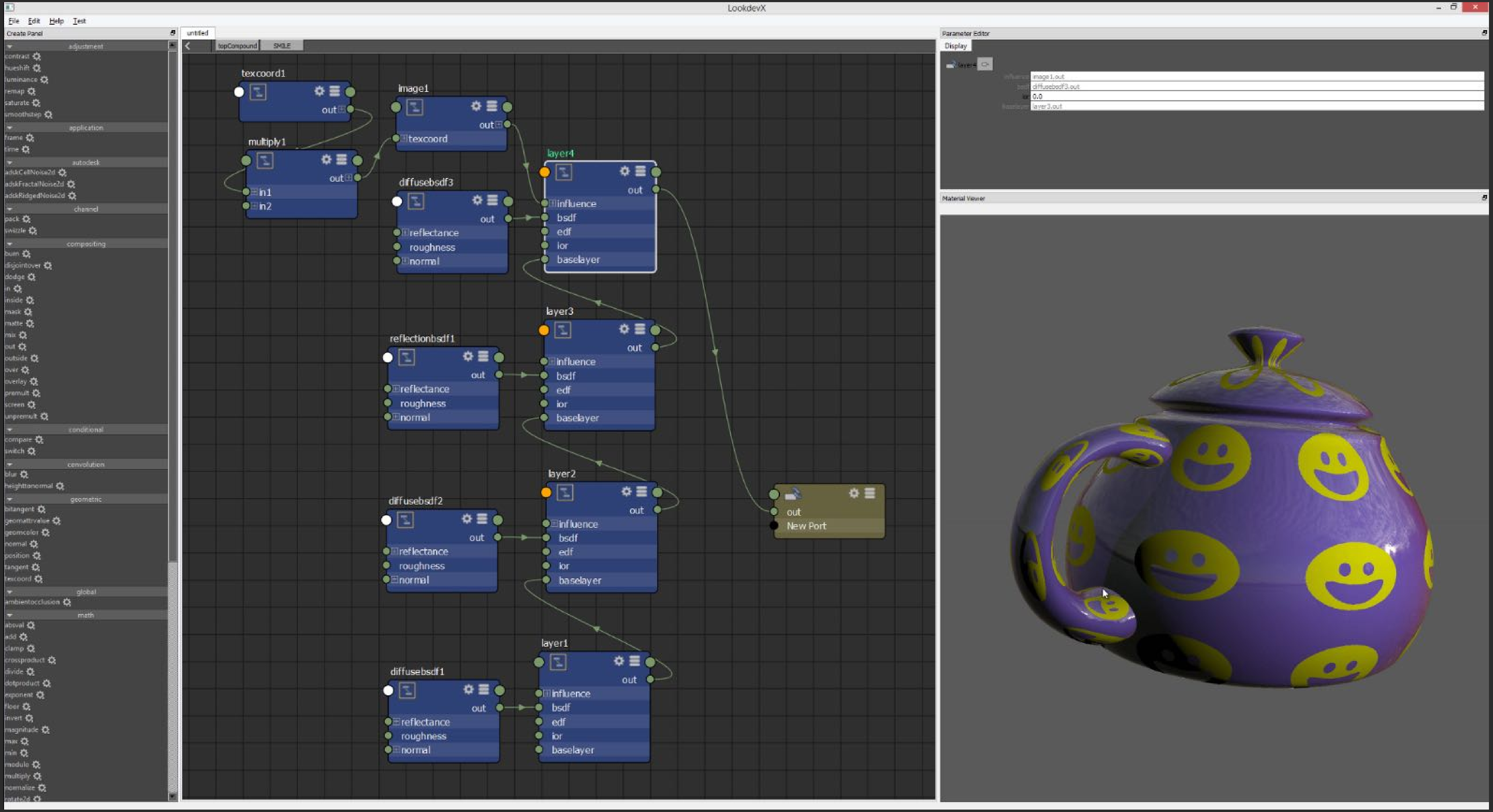
| | |
|---|---|
| documents | Added full support for the conditional nodes (compare and switch). |
| python | Initial release of MaterialX v1.35.2 |
| source | Added missing file changes from previous commit. |
| .gitignore | Added initial support for GLSL shade generation. |
| CHANGELOG.md | Improvements to MaterialX names |
| CMakeLists.txt | Merge branch 'master' into adsk/master |
| CONTRIBUTING.md | Initial release of MaterialX v1.35.2 |
| LICENSE.txt | Initial release of MaterialX v1.35.2 |
| README.md | Initial release of MaterialX v1.35.2 |

README.md

# MaterialX | Product interop.

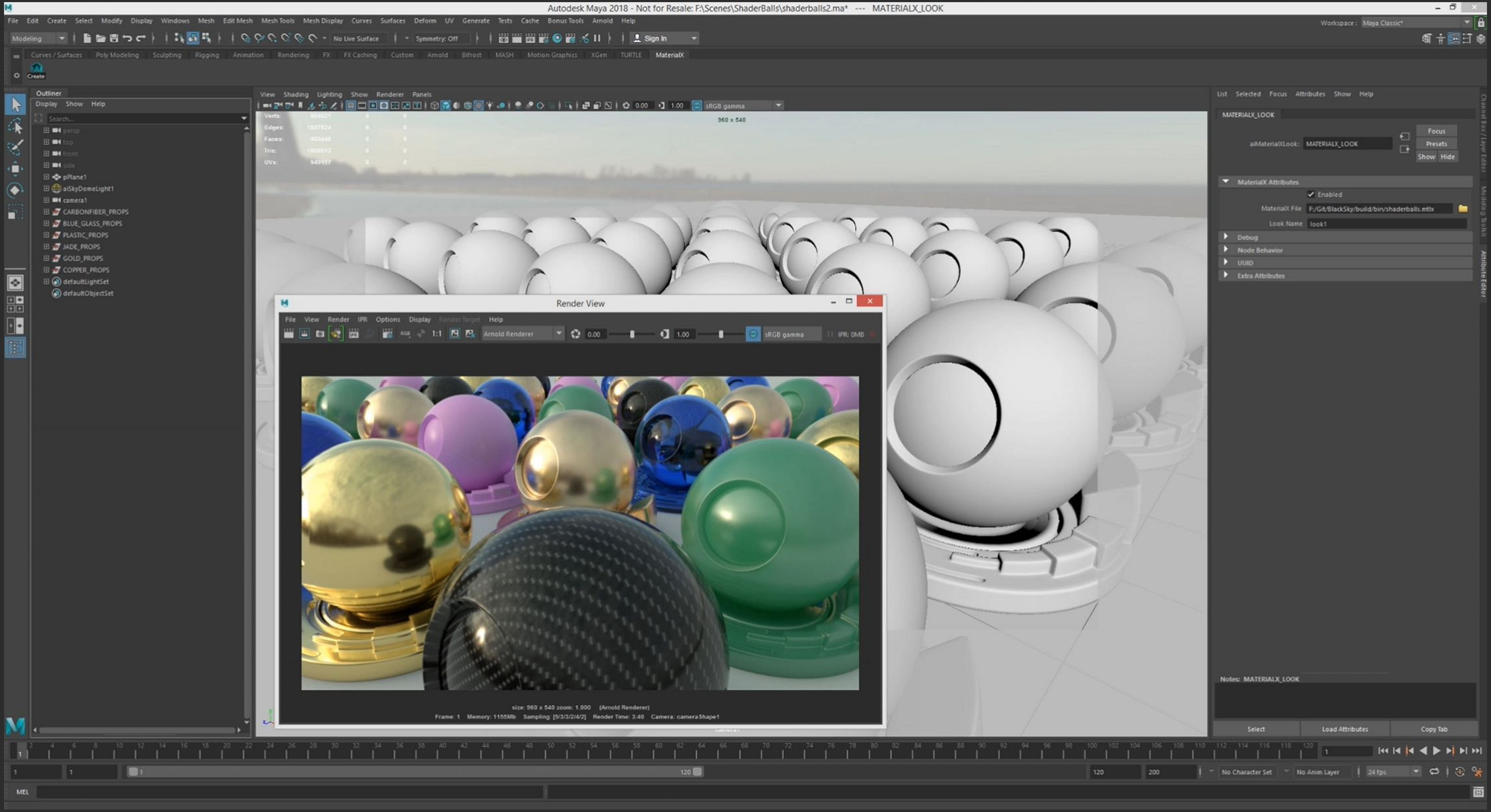


# Demo | LookdevX



influence image1.out
bsdf diffusebsdf3.out
ior 0.0
baselayer layer3.out

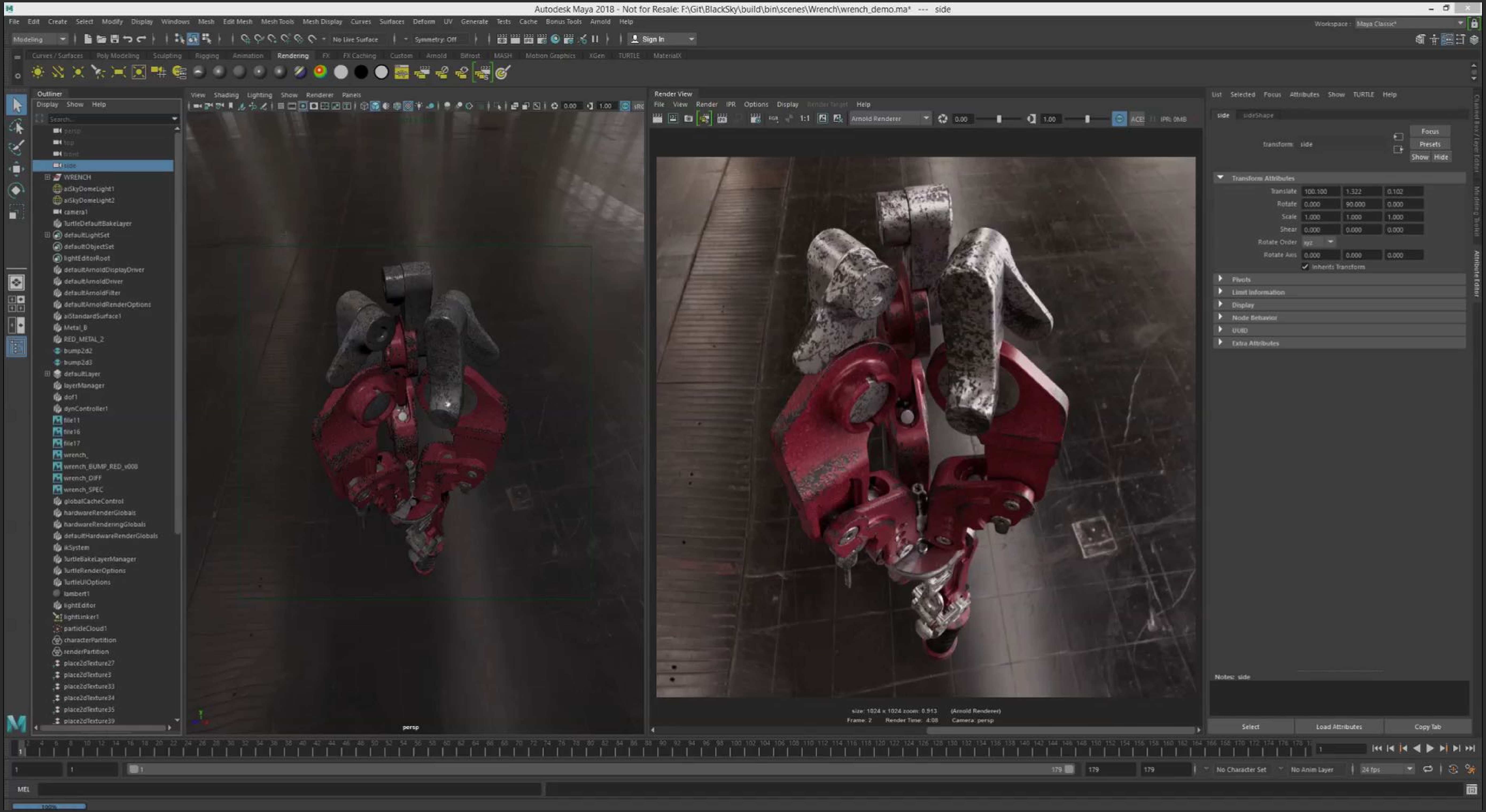
# Demo | MaterialX to Arnold


Autodesk Maya 2018 - Not for Resale: F:\Scenes\ShaderBalls\shaderballs2.ma* --- MATERIALX_LOOK

size: 960 x 540 zoom: 1.000 (Arnold Renderer)
Frame: 1 Memory: 1155Mb Sampling: [5/3/3/2/4/2] Render Time: 3:40 Camera: cameraShape1

MATERIALX_LOOK
aiMaterialXLook: MATERIALX_LOOK
MaterialX Attributes
✓ Enabled
MaterialX File: F:/Git/BlackSky/build/bin/shaderballs.mtlx
Look Name: look1

# Demo | Maya Exporter


Autodesk Maya 2018 - Not for Resale: F:\Git\BlackSky\build\bin\scenes\Wrench\wrench_demo.ma* --- side
Outliner
Render View
Arnold Renderer
Transform Attributes
size: 1024 x 1024 zoom: 0.913 (Arnold Renderer)
Frame: 2 Render Time: 4:08 Camera: persp