MaterialX: Supplemental Notes

Doug Smythe - smythe@ilm.com
Jonathan Stone - jstone@lucasfilm.com
July 21, 2018

Introduction

This document details additional information about MaterialX and how it may be incorporated into studio pipelines. The document describes a number of additional Supplemental Nodes providing enhanced functionality over the basic Standard Nodes, as well as a recommended naming convention for node definition elements and a directory structure to define packages of node definitions and implementations from various sources.

Table of Contents

Supplemental Nodes

The MaterialX Specification defines a number of Standard Nodes, which all implementations of MaterialX are expected to support, to the degree their host applications allow. These nodes are the basic "building blocks" upon which more complex node functionality can be built.

This section describes a number of supplemental nodes for MaterialX. These nodes are considered part of "MaterialX", but are typically implemented using nodegraphs of standard MaterialX nodes rather than being implemented in various languages for specific targets. Certain applications may choose to implement these supplemental nodes using native coding languages for efficiency. It is also expected that various applications will choose to extend these supplemental nodes with additional parameters and additional functionality.

Supplemental Texture Nodes

- **`tiledimage`**: samples data from a single image, with provisions for tiling and offsetting the image across uv space. Parameters and inputs:
  - `file` (parameter, filename, required): the URI of an image file. The filename can include one or more substitutions to change the file name (including frame number) that is accessed, as described in **Image Filename Substitutions**.
  - `default` (parameter, float or color*N* or vector*N*, optional): a default value to use if the `file` reference can not be resolved (e.g. if a <geomtoken>, [interfacetoken] or {hostattr} is included in the filename but no substitution value or default is defined, or if the resolved file URI cannot be read), or if the specified `layer` does not exist in the file. The `default` value must be the same type as the `<image>` element itself. If `default` is not defined, the default color value will be 0.0 in all channels.
  - `texcoord` (input, vector2, optional): the name of a vector2-type node specifying the 2D texture coordinate at which the image data is read. Default is to use the current u,v coordinate.
  - `uvtiling` (parameter, vector2, optional): the tiling rate for the given image along the U and V axes. Mathematically equivalent to multiplying the incoming texture coordinates by the reciprocal of the given vector value. Default value is (1.0, 1.0).
  - `uvoffset` (parameter, vector2, optional): the offset for the given image along the U and V axes. Mathematically equivalent to subtracting the given vector value from the incoming texture coordinates. Default value is (0.0, 0.0).
  - `filtertype` (parameter, string, optional): the type of texture filtering to use; standard values include "closest" (nearest-neighbor single-sample), "linear", and "cubic". If not specified, an application may use its own default texture filtering method.

```
<tiledimage name="in3" type="color3">
  <parameter name="file" type="filename" value="textures/mytile.tif"/>
  <parameter name="default" type="color3" value="0.0,0.0,0.0"/>
  <parameter name="uvtiling" type="vector2" value="3.0,3.0"/>
  <parameter name="uvoffset" type="vector2" value="0.5,0.5"/>
</tiledimage>
```


- **`triplanarprojection`**: samples data from three images (or layers within multi-layer images), and projects a tiled representation of the images along each of the three respective coordinate axes, computing a weighted blend of the three samples using the geometric normal. [REQ="geomops"] Parameters and inputs:
  - `filex` (parameter, filename, required): the URI of an image file to be projected in the direction from the +X axis back toward the origin.
  - `filey` (parameter, filename, required): the URI of an image file to be projected in the direction from the +Y axis back toward the origin with the +X axis to the right.
  - `filez` (parameter, filename, required): the URI of an image file to be projected in the direction from the +Z axis back toward the origin.
  - `layerx` (parameter, string, optional): the name of the layer to extract from a multi-layer input file for the x-axis projection. If no value for `layerx` is provided and the input file has multiple layers, then the "default" layer will be used, or "rgba" if there is no "default" layer. Note: the number of channels defined by the `type` of the `<image>` must match the number of channels in the named layer.
  - `layery` (parameter, string, optional): the name of the layer to extract from a multi-layer input file for the y-axis projection.
  - `layerz` (parameter, string, optional): the name of the layer to extract from a multi-layer input file for the z-axis projection.
  - `default` (parameter, float or color*N* or vector*N*, optional): a default value to use if any `fileX` reference can not be resolved (e.g. if a <geomtoken>, [interfacetoken] or {hostattr} is included in the filename but no substitution value or default is defined, or if the resolved file URI cannot be read) The `default` value must be the same type as the `<triplanarprojection>` element itself. If `default` is not defined, the default color value will be 0.0 in all channels.
  - `position` (input, vector3, optional): a spatially-varying input specifying the 3D position at which the projection is evaluated. Default is to use the current 3D object-space coordinate.
  - `normal` (input, vector3, optional): a spatially-varying input specifying the 3D normal vector used for blending. Default is to use the current object-space surface normal.
  - `filtertype` (parameter, string, optional): the type of texture filtering to use; standard values include "closest" (nearest-neighbor single-sample), "linear", and "cubic". If not specified, an application may use its own default texture filtering method.

```
<triplanarprojection name="tri4" type="color3">
  <parameter name="filex" type="filename" value="<colorname>.X.tif"/>
  <parameter name="filey" type="filename" value="<colorname>.Y.tif"/>
  <parameter name="filez" type="filename" value="<colorname>.Z.tif"/>
  <parameter name="default" type="color3" value="0.0,0.0,0.0"/>
</triplanarprojection>
```


Supplemental Source Nodes

- **`ramp4`**: a 4-corner bilinear value ramp. Parameters and inputs:
  - `valuetl` (parameter, float or color*N* or vector*N*, required): the value at the top-left (U0V1) corner
  - `valuetr` (parameter, float or color*N* or vector*N*, required): the value at the top-right (U1V1) corner
  - `valuebl` (parameter, float or color*N* or vector*N*, required): the value at the bottom-left (U0V0) corner
  - `valuebr` (parameter, float or color*N* or vector*N*, required): the value at the bottom-right (U1V0) corner
  - `texcoord` (input, vector2, optional): the name of a vector2-type node specifying the 2D texture coordinate at which the ramp interpolation is evaluated. Default is to use the first set of texture coordinates.

Supplemental Adjustment Nodes

- **`contrast`**: increase or decrease contrast of incoming float/color values using a linear slope multiplier. Parameters and inputs:
  - `in` (input, float or color*N* or vector*N*, required): the name of the node to connect to the input
  - `amount` (parameter, same type as `in` or float, required): slope multiplier for contrast adjustment, 0.0 to infinity range. Values greater than 1.0 increase contrast, values between 0.0 and 1.0 reduce contrast
  - `pivot` (parameter, same type as `in` or float, optional): center pivot value of contrast adjustment; this is the value that will not change as contrast is adjusted (default value=0.5)

- **`range`**: remap incoming values from one range of float/color/vector values to another, optionally applying a gamma correction "in the middle". Input values below `inlow` or above `outhigh` are extrapolated unless `doclamp` is true. Parameters and inputs:
  - `in` (input, float or color*N* or vector*N*, required): the name of the node to connect to the input
  - `inlow` (parameter, same type as `in` or float, optional): low value for input range (default value=0)
  - `inhigh` (parameter, same type as `in` or float, optional): high value for input range (default value=1)
  - `gamma` (parameter, same type as `in` or float, optional): inverse exponent applied to input value after first transforming from `inlow..inhigh` to 0..1; `gamma` values greater than 1.0 make midtones brighter. (default value=1)
  - `outlow` (parameter, same type as `in` or float, optional): low value for output range (default value=0)
  - `outhigh` (parameter, same type as `in` or float, optional): high value for output range (default value=1)
  - `doclamp` (parameter, boolean, optional): If true, the output is clamped to the range `outlow..outhigh` (default is false)

- **`hsvadjust`**: adjust the hue, saturation and value of an RGB color by converting the input color to HSV, adding amount.x to the hue, multiplying the saturation by amount.y, multiplying the

value by amount.z, then converting back to RGB. A positive "amount.x" rotates hue in the "red to green to blue" direction, with amount of 1.0 being the equivalent to a 360 degree (e.g. no-op) rotation. Negative or greater-than-1.0 hue adjustment values are allowed, wrapping at the 0-1 boundaries. For color4 inputs, the alpha value is unchanged. Parameters and inputs:
  - `in` (input, color3 or color4, required): the name of the node to connect to the input
  - `amount` (parameter, vector3, required): the HSV adjustment; a value of (0, 1, 1) is "no change".

- **`saturate`**: (color3 or color4 only) adjust the saturation of a color; the alpha channel will be unchanged if present. Note that this operation is **not** equivalent to the "amount.y" saturation adjustment of `hsvadjust`, as that operator does not take the working or any other colorspace into account. Parameters and inputs:
  - `in` (input, float or color*N* or vector*N*, required): the name of the node to connect to the input
  - `amount` (parameter, float, required): a multiplier for saturation; the saturate operator performs a linear interpolation between the luminance of the incoming color value (copied to all three color channels) and the incoming color value itself. Note that setting amount to 0 will result in an R=G=B gray value equal to the value that the `luminance` node (below) returns.
  - `lumacoeffs` (parameter, color3, optional): the luma coefficients of the current working color space; if no specific color space can be determined, the ACEScg (ap1) luma coefficients [0.272287, 0.6740818, 0.0536895] will be used. Applications which support color management systems may choose to retrieve this value from the CMS to pass to the <saturate> node's implementation directly, rather than exposing it to the user.

Supplemental Channel Nodes

- **`extract`**: generate a float stream from one channel of a color*N* or vector*N* stream. Parameters and inputs:
  - `in` (input, color*N* or vector*N*, required): the name of the node to connect to the input
  - `index` (parameter, integer, required): the channel number of the input stream to extract, in the range from 0 to 3.

- **`separate`**: output each of the channels of a color*N* or vector*N* as a separate float output. [REQ="multioutput"]
  - `in` (input, color*N* or vector*N*, required): the name of the node to connect to the input
  - `outr`/`outx` (output, float): the value of the red (for color*N* streams) or x (for vector*N* streams) channel.
  - `outg`/`outy` (output, float): the value of the red (for color*N* streams) or x (for vector*N* streams) channel.
  - `outb`/`outz` (output, float): For 3 or 4 channel inputs, the value of the red (for color*N* streams) or x (for vector*N* streams) channel.
  - `outa`/`outw` (output, float): For color4/vector4 inputs, the value of the alpha (for color4 streams) or w (for vector4 streams) channel.

Recommended Element Naming Conventions

While MaterialX elements can be given any valid name as described in the MaterialX Names section of the main specification, adhering to the following recommended naming conventions or slight variations thereof will help reduce the possibility of elements from different sources having the same name.

Nodedef: "ND_*nodename_outputtype*[_*target*][_*version*]", or for nodes with multiple input types for a given output type (e.g. <swizzle>), "ND_*nodename_inputtype_outputtype*[_*target*][_*version*]".

Implementation: "IM_*nodename*[_*inputtype*]_*outputtype*[_*language*][_*target*][_*version*]".

Nodegraph, as an implementation for a node:
"NG_*nodename*[_*inputtype*]_*outputtype*[_*target*][_*version*]".

Node Definition File Structure

As studios develop and incorporate more MaterialX node types and implementations from various library sources for various targets, it becomes beneficial to have a consistent, logical organization for the files on disk that make up these libraries. We propose the following organization for files defining <nodedef>s, <implementation>s, and actual shader source code.

It should be noted that this file structure is only recommended but is not required, and that it is only used to help sort out which files are part of which libraries, in which languages, for which targets. The actual specification of which library/language/target/version/etc. of any node definition element is set by the contents of the node element itself, e.g. what values its `target`, `language` and `version` attributes have.

Legend for various components within the folder structure:

libname	The name of the library; the MaterialX Standard nodes are the "stdlib" library. It is recommended that libraries also declare themselves to be in the *libname* namespace, though this is not required.
defsfile	A .mtlx file containing <nodedef> elements
nodegrfile	A .mtlx file containing <nodegraph> implementations for nodes
implfile	A .mtlx file containing <implementation> elements
language	The implementation language, e.g. "osl", "glsl", "cpp", etc. See

	note below.
target	The target for this implementation, e.g. "maya" for "glsl" language, or "arnold" or "vray" for "osl" language.
sourcefiles	Source files (including includes and makefiles) for the language/target, in whatever format and structure the applicable build system requires.

Here are the suggested locations and naming for the various files making up a MaterialX node definition setup. Italicized terms should be replaced with actual values, while boldface terms should appear verbatim. $MXROOT is a placeholder for one of any number of studio-dependent root paths for MaterialX library content.

```
$MXROOT/libname[_*]_defs.mtlx                              (1)
$MXROOT/libname_language[_*]_impl.mtlx                     (2)
$MXROOT/libname_language_target[_*]_impl.mtlx              (3)
$MXROOT/libname[_*]_ng.mtlx                                (4)
$MXROOT/libname/language/sourcefiles                       (5)
$MXROOT/libname/language/target/sourcefiles                (6)
```

(1) Nodedefs for targetless or target-specific nodes for library *libname*.
(2) Target-independent implementation elements for *libname* in language *language*.
(3) Implementation elements for *libname* in language *language* for target *target*.
(4) Nodegraph implementations for nodes in library *libname*.
(5) Source code files for target-independent implementations in a specific language.
(6) Source code files for target-specific implementations in a specific language.

Examples

Standard node definitions and reference OSL implementation:

```
$MXROOT/stdlib_defs.mtlx                    (standard library definitions)
$MXROOT/stdlib_ng.mtlx                      (supplemental library node nodegraphs)
$MXROOT/stdlib_osl_impl.mtlx                (standard library OSL implementation impl file)
$MXROOT/stdlib/osl/*                        (standard library OSL source files)
```

Layout for a ShaderX implementation of "stdlib", referencing the above standard `stdlib_defs.mtlx` file. ShaderX is an on-the-fly shader build system making use of GLSL and OSL shader code snippets, so the "language" portion of the paths has an added "sx-" prefix to denote its specific adaptation of those languages. Target-specific implementations have subfolders to store additional code specific to each target (e.g. Maya, Arnold, Vray and hardware OgsFx language additions); otherwise the target will use the common implementation.

```
# GLSL language implementations
$MXROOT/stdlib_sx-glsl_impl.mtlx            (stdlib sx-GLSL implementation file)
$MXROOT/stdlib_sx-glsl_ogsfx_impl.mtlx      (ogsfx Effects-specific implementations)
```



```
$MXROOT/stdlib_sx-glsl_maya_impl.mtlx          (Maya-specific implementations)
$MXROOT/stdlib/sx-glsl/*                       (stdlib common sx-GLSL code)
$MXROOT/stdlib/sx-glsl/ogsfx/*                 (ogsfx Effects-specific sx-GLSL code)
$MXROOT/stdlib/sx-glsl/maya/*                  (Maya-specific sx-GLSL code)

# OSL language implementations
$MXROOT/stdlib_sx-osl_impl.mtlx                (stdlib shaderx sx-OSL implementation file)
$MXROOT/stdlib_sx-osl_arnold_impl.mtlx         (Arnold-specific implementations)
$MXROOT/stdlib_sx-osl_vray_impl.mtlx           (VRay-specific implementations)
$MXROOT/stdlib/sx-osl/*                        (stdlib shaderx common sx-OSL code)
$MXROOT/stdlib/sx-osl/arnold/*                 (Arnold-specific sx-OSL code)
$MXROOT/stdlib/sx-osl/vray/*                   (VRay-specific sx-OSL code)
```

Layout for a supplemental ShaderX PBR ("sxpbrlib") shader library with implementations in "sx-GLSL" and "sx-OSL" languages:

```
$MXROOT/sxpbrlib_defs.mtlx                     (ShaderX PBR library definitions)
$MXROOT/sxpbrlib_sx-glsl_impl.mtlx             (sxpbr impl file referencing sx-GLSL source)
$MXROOT/sxpbrlib_sx-glsl_ogsfx_impl.mtlx       (ogsfx Effects-specific sxpbr implementations)
$MXROOT/sxpbrlib_sx-glsl_maya_impl.mtlx        (Maya-specific sxpbr implementations)
$MXROOT/sxpbrlib/sx-glsl/*                     (sxpbr common sx-GLSL code)
$MXROOT/sxpbrlib/sx-glsl/ogsfx/*               (ogsfx Effects-specific sxpbr sx-GLSL code)
$MXROOT/sxpbrlib/sx-glsl/maya/*                (Maya-specific sxpbr sx-GLSL code)

$MXROOT/sxpbrlib_sx-osl_impl.mtlx              (sxpbr impl file referencing OSL source)
$MXROOT/sxpbrlib_sx-osl_arnold_impl.mtlx       (Arnold-specific sxpbr implementations)
$MXROOT/sxpbrlib_sx-osl_vray_impl.mtlx         (VRay-specific sxpbr implementations)
$MXROOT/sxpbrlib/sx-osl/*                      (sxpbr common sx-OSL code)
$MXROOT/sxpbrlib/sx-osl/arnold/*               (Arnold-specific sxpbr sx-OSL code)
$MXROOT/sxpbrlib/sx-osl/vray/*                 (VRay-specific sxpbr sx-OSL code)
```

