MaterialX: Supplemental Notes

Version 1.38

Doug Smythe - Industrial Light & Magic
Jonathan Stone - Lucasfilm Advanced Development Group

February 2, 2021

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 graphs of standard MaterialX nodes rather than being implemented 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.
  - `file` (uniform filename): 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 **Filename Substitutions** in the main Specification document.
  - `default` (float or color*N* or vector*N*): 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` (vector2): 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` (vector2): the tiling rate for the given image along the U and V axes. Mathematically equivalent to multiplying the incoming texture coordinates by the given vector value. Default value is (1.0, 1.0).
  - `uvoffset` (vector2): 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).
  - `realworldimagesize` (vector2): the real-world size represented by the `file` image, with unittype "distance". A `unit` attribute may be provided to indicate the units that `realworldimagesize` is expressed in.
  - `realworldtilesize` (vector2): the real-world size of a single square 0-1 UV tile, with unittype "distance". A `unit` attribute may be provided to indicate the units that `realworldtilesize` is expressed in.
  - `filtertype` (uniform string): 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">
  <input name="file" type="filename" value="textures/mytile.tif"/>
  <input name="default" type="color3" value="0.0,0.0,0.0"/>
  <input name="uvtiling" type="vector2" value="3.0,3.0"/>
  <input 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.
  - `filex` (uniform filename): the URI of an image file to be projected in the direction from the +X axis back toward the origin.
  - `filey` (uniform filename): 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` (uniform filename): the URI of an image file to be projected in the direction from the +Z axis back toward the origin.
  - `layerx` (uniform string): 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` (uniform string): the name of the layer to extract from a multi-layer input file for the y-axis projection.
  - `layerz` (uniform string): the name of the layer to extract from a multi-layer input file for the z-axis projection.
  - `default` (float or color*N* or vector*N*): a default value to use if any `file`*`X`* 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` (vector3): 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` (vector3): a spatially-varying input specifying the 3D normal vector used for blending. Default is to use the current object-space surface normal.
  - `filtertype` (uniform string): 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">
  <input name="filex" type="filename" value="<colorname>.X.tif"/>
  <input name="filey" type="filename" value="<colorname>.Y.tif"/>
  <input name="filez" type="filename" value="<colorname>.Z.tif"/>
  <input name="default" type="color3" value="0.0,0.0,0.0"/>
</triplanarprojection>
```

Supplemental Source Nodes

- **`ramp4`**: a 4-corner bilinear value ramp.
  - `valuetl` (float or color*N* or vector*N*): the value at the top-left (U0V1) corner
  - `valuetr` (float or color*N* or vector*N*): the value at the top-right (U1V1) corner
  - `valuebl` (float or color*N* or vector*N*): the value at the bottom-left (U0V0) corner
  - `valuebr` (float or color*N* or vector*N*): the value at the bottom-right (U1V0) corner
  - `texcoord` (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 Math Nodes

- **`place2d`**: transform incoming UV texture coordinates for 2D texture placement.
  - `texcoord` (vector2): the input UV coordinate to transform; defaults to the current surface index=0 uv coordinate.
  - `pivot` (vector2): the pivot coordinate for scale and rotate: this is subtracted from u,v before applying scale/rotate, then added back after. Default is (0,0).
  - `scale` (vector2): divide the u,v coord (after subtracting `pivot`) by this, so a scale (2,2) makes the texture image appear twice as big. Negative values can be used to flip or flop the texture space. Default is (1,1).
  - `rotate` (float): rotate u,v coord (after subtracting pivot) by this amount in degrees, so a positive value rotates UV coords counter-clockwise, and the image clockwise. Default is 0.
  - `offset` (vector2): subtract this amount from the scaled/rotated/"pivot added back" UV coordinate; since U0,V0 is typically the lower left corner, a positive offset moves the texture image up and right. Default is (0,0).

- **`safepower`**: raise incoming float/color values to the specified exponent. Unlike the standard <power> node, negative `in1` values for <safepower> will result in negative output values, e.g. `out = sign(in1)*pow(abs(in1),in2).`
  - `in1` (float or color*N* or vector*N*): the value or nodename for the primary input
  - `in2` (same type as `in` or float): exponent value or nodename; default is 1.0 in all channels

Supplemental Adjustment Nodes

- **`contrast`**: increase or decrease contrast of incoming float/color values using a linear slope multiplier.
  - `in` (float or color*N* or vector*N*): the input value or nodename
  - `amount` (same type as `in` or float): 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. Default is 1.0 in all channels.
  - `pivot` (same type as `in` or float): center pivot value of contrast adjustment; this is the value that will not change as contrast is adjusted. Default is 0.5 in all channels.

- **`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.
  - `in` (float or color*N* or vector*N*): the input value or nodename
  - `inlow` (same type as `in` or float): low value for input range. Default is 0.0 in all channels.
  - `inhigh` (same type as `in` or float): high value for input range. Default is 1.0 in all channels.
  - `gamma` (same type as `in` or float): 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 is 1.0 in all channels.
  - `outlow` (same type as `in` or float): low value for output range. Default is 0.0 in all channels.
  - `outhigh` (same type as `in` or float): high value for output range. Default is 1.0 in all channels.
  - `doclamp` (boolean): 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.
  - `in` (color3 or color4): the input value or nodename
  - `amount` (vector3): the HSV adjustment; a value of (0, 1, 1) is "no change" and is the default.

- **`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.
  - `in` (float or color*N* or vector*N*): the input value or nodename
  - `amount` (float): 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. Default is 1.0.
  - `lumacoeffs` (uniform color3): 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.
  - `in` (color*N* or vector*N*): the input value or nodename
  - `index` (uniform integer): the channel number of the input stream to extract, in the range from 0 to 3. Default is 0.

- **`separate2`**: output each of the channels of a vector2 as a separate float output.
  - `in` (vector2): the input value or nodename
  - `outx` (**output**, float): the value of x channel.
  - `outy` (**output**, float): the value of y channel.

- **`separate3`**: output each of the channels of a color3 or vector3 as a separate float output.
  - `in` (color3 or vector3): the input value or nodename
  - `outr/outx` (**output**, float): the value of the red (for color3 streams) or x (for vector3 streams) channel.
  - `outg/outy` (**output**, float): the value of the green (for color3 streams) or y (for vector3 streams) channel.
  - `outb/outz` (**output**, float): the value of the blue (for color3 streams) or z (for vector3 streams) channel.

- **`separate4`**: output each of the channels of a color4 or vector4 as a separate float output.
  - `in` (color4 or vector4): the input value or nodename
  - `outr/outx` (**output**, float): the value of the red (for color4 streams) or x (for vector4 streams) channel.
  - `outg/outy` (**output**, float): the value of the green (for color4 streams) or y (for vector4 streams) channel.
  - `outb/outz` (**output**, float): the value of the blue (for color4 streams) or z (for vector4 streams) channel.
  - `outa/outw` (**output**, float): 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 will make it easier to predict the name of a nodedef for use in implementation and nodegraph elements as well as 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*[_*target*][_*version*]".

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

Material and Node Library File Structure

As studios and vendors develop libraries of shared definitions and implementations of MaterialX materials and nodes for various targets, it becomes beneficial to have a consistent, logical organizational structure for the files on disk that make up these libraries. In this section, we propose a structure for files defining libraries of material nodes, <nodedef>s, nodegraph implementations and actual target-specific native source code, as well as a mechanism for applications and MaterialX content to find and reference files within these libraries.

Legend for various components within folder hierarchies:

libname	The name of the library; the MaterialX Standard nodes are the "stdlib" library. Libraries may choose to declare themselves to be in the *libname* namespace, although this is not required.
target	The target for an implementation, e.g. "glsl", "oslpattern", "osl" or "mdl".
sourcefiles	Source files (including includes and makefiles) for the target, in whatever format and structure the applicable build system requires.

Here is the suggested structure and naming for the various files making up a MaterialX material or node definition library setup. Italicized terms should be replaced with appropriate values, while boldface terms should appear verbatim. The optional "_*" component of the filename can be any useful descriptor of the file's contents, e.g. "_ng" for nodegraphs or "_mtls" for materials.

```
libname/libname_defs.mtlx                                    (1)
libname/libname_*.mtlx                                       (2)
libname/target/libname_target[_*]_impl.mtlx                 (3)
libname/target/sourcefiles                                   (4)
```

(1) Nodedefs and other definitions in library *libname*.
(2) Additional elements (e.g. nodegraph implementations for nodes, materials, etc.) in library *libname*.
(3) Implementation elements for *libname* specific to target *target*.
(4) Source code files for *libname* implementations specific to target *target*.

Note that nodedef files and nodegraph-implementation files go at the top `libname` level, while <implementation> element files go under the corresponding `libname/target` level, next to their source code files. This is so that studios may easily install only the implementations that are relevant to them, and applications can easily locate the implementations of nodes for specific desired targets. Libraries are free to add additional arbitrarily-named folders for related content, such as an "images" subfolder for material library textures.

The *libname*_defs.mtlx file typically contains nodedefs for the library, but may also contain other node types such as implementation nodegraphs, materials, looks, and any other element types. The use of additional *libname*_*.mtlx files is optional, but those files should be Xincluded by the *libname*_defs.mtlx file.

A file referenced by a MaterialX document or tool (e.g. XInclude files, filenames in <image> or other MaterialX nodes, or command-line arguments in MaterialX tools) can be specified using either a relative or a fully-qualified absolute filesystem path. A relative path is interpreted to be relative to either the location of the referencing MaterialX document itself, or relative to a location found within the current MaterialX search path: this path may be specified via an application setting (e.g. the "--path" option in MaterialXView) or globally using the MATERIALX_SEARCH_PATH environment variable. These search paths are used for both XIncluded definitions and filename input values (e.g. images for nodes or source code for <implementation>s), and applications may choose to define different search paths for different contexts if desired, e.g. for document processing vs. rendering.

The standard libraries `stdlib` and `pbrlib` are typically included *automatically* by MaterialX applications, rather than through explicit XInclude directives within .mtlx files. Non-standard libraries are included into MaterialX documents by XIncluding the top-level `libname/libname_defs.mtlx` file, which is expected to in turn XInclude any additional .mtlx files needed by the library.

Examples

In the examples below, MXROOT is a placeholder for one of the root paths defined in the current MaterialX search path.

A library of studio-custom material shading networks and example library materials:

```
MXROOT/mtllib/mtllib_defs.mtlx                 (material nodedefs and nodegraphs)
MXROOT/mtllib/mtllib_mtls.mtlx                 (library of materials using mtllib_defs)
MXROOT/mtllib/images/*.tif                     (texture files used by mtllib_mtls <image> nodes)
```

Documents may include the above library using

```
<xi:include href="mtllib/mtllib_defs.mtlx"/>
```

and that file would XInclude `mtllib_mtls.mtlx`. <Image> nodes within `mtllib_mtls.mtlx` would use `file` input values such as "images/bronze_color.tif", e.g. relative to the path of the `mtllib_mtls.mtlx` file itself.

Standard node definitions and reference OSL implementation:

```
MXROOT/stdlib/stdlib_defs.mtlx                     (standard library node definitions)
MXROOT/stdlib/stdlib_ng.mtlx                       (supplemental library node nodegraphs)
MXROOT/stdlib/osl/stdlib_osl_impl.mtlx             (stdlib OSL implementation elem file)
MXROOT/stdlib/osl/*.{h,osl} (etc)                  (stdlib OSL source files)
```

Layout for "genglsl" and "genosl" implementations of "stdlib" for MaterialX's shadergen component, referencing the above standard `stdlib_defs.mtlx` file:

```
# Generated-GLSL implementations
MXROOT/stdlib/genglsl/stdlib_genglsl_impl.mtlx     (stdlib genGLSL implementation file)
MXROOT/stdlib/genglsl/stdlib_genglsl_cm_impl.mtlx  (stdlib genGLSL color-mgmt impl. file)
MXROOT/stdlib/genglsl/*.{inline,glsl}              (stdlib common genGLSL code)

# Generated-OSL implementations
MXROOT/stdlib/genosl/stdlib_genosl_impl.mtlx       (stdlib genOSL implementation file)
MXROOT/stdlib/genosl/stdlib_genosl_cm_impl.mtlx    (stdlib genOSL color-mgmt impl. file)
MXROOT/stdlib/genosl/*.{inline,osl}                (stdlib common genOSL code)
```

Layout for the shadergen PBR shader library ("pbrlib") with implementations for "genglsl" and "genosl" (generated GLSL and OSL, respectively) targets:

```
MXROOT/pbrlib/pbrlib_defs.mtlx                     (PBR library definitions)
MXROOT/pbrlib/pbrlib_ng.mtlx                       (PBR library nodegraphs)
MXROOT/pbrlib/genglsl/pbrlib_genglsl_impl.mtlx     (pbr impl file referencing genGLSL source)
MXROOT/pbrlib/genglsl/*.{inline,glsl}              (pbr common genGLSL code)
MXROOT/pbrlib/genosl/pbrlib_genosl_impl.mtlx       (pbr impl file referencing genOSL source)
MXROOT/pbrlib/genosl/*.{inline,osl}                (pbr common genOSL code)
```