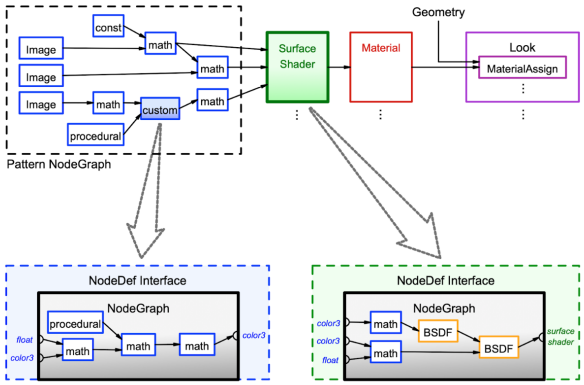
MaterialX Nodes in RealityKit for visionOS

Revision	Date	Changes	Author	Published on
Rev. 0.1	Aug, 2023	• This document summarizes MaterialX Specification v1.38 and MaterialX Supplemental Notes v1.38, and describes the Standard MaterialX Nodes and RealityKit Custom Nodes. • It also shows correspondence between MaterialX Standard Nodes and RealityKit implementation. • Information about RealityKit is based on the visionOS beta. It may change with the release.	@AtarayoSD	GitHub evolution-Metal-ARKit-RealityKit-sheet https://github.com/ynagatomo/evolution-Metal-ARKit-RealityKit-sheet

MaterialX Common Term

		Description	Note
Overview			
Term	**Element**	An Element is a named object within a MaterialX document, which may possess any number of child elements and attributes. An Attribute is a named property of a MaterialX element.	
	Node	A Node is a function that generates or operates upon spatially-varying data. This specification provides a set of standard nodes with precise definitions, and also supports the creation of custom nodes for application-specific uses. The interface for a node's incoming data is declared through Inputs, which may be spatially-varying or uniform, and Tokens, which are string values that can be substituted into filenames declared in node inputs.	
	Pattern	A Pattern is a node that generates or processes simple scalar, vector, and color data, and has access to local properties of any geometry that has been bound.	
	Shader	A Shader is a node that can generate or process arbitrary lighting or BSDF data, and has access to global properties of the scene in which it is evaluated.	
	Material	A Material is a node which internally or externally references one or more shaders with specific data streams or uniform values bound to their inputs.	
	Node Graph	A Node Graph is a directed acyclic graph of nodes, which may be used to define arbitrarily complex generation or processing networks. Common uses of Node Graphs are to describe a network of pattern nodes flowing to a shader input, or to define a complex or layered node in terms of simpler nodes.	
	Stream	A Stream refers to a flow of spatially-varying data from one node to another. A Stream most commonly consists of color, vector, or scalar data, but can transport data of any standard or custom type.	
	Layer	A Layer is a named 1-, 2-, 3- or 4-channel color "plane" within an image file. Image file formats that do not support multiple or named layers within a file should be treated as if the (single) layer was named "rgba".	
	Channel	A Channel is a single float value within a color or vector value, e.g. each layer of an image might have a red Channel, a green Channel, a blue Channel and an alpha Channel.	
	Geometry	A Geometry is any renderable object, while a Partition refers to a specific named renderable subset of a piece of geometry, such as a face set.	
	Collection	A Collection is a recipe for building a list of geometries, which can be used as a shorthand for assigning e.g. a Material to a number of geometries in a Look.	
	Target	A Target is a software environment that interprets MaterialX content to generate images, with common examples being digital content creation tools and 3D renderers.	
Names		All elements in MaterialX (nodes, materials, shaders, etc.) are required to have a name attribute of type "string". The name attribute of a MaterialX element is its unique identifier, and no two elements within the same scope (i.e. elements with the same parent) may share a name. Element names are restricted to upper- and lower-case letters, numbers, and underscores ("_") from the ASCII character set; all other characters and symbols are disallowed. MaterialX names are case-sensitive and are not allowed to begin with a digit.	
Standard Data Types	**Base Types**	`integer, boolean, float, color3, color4, vector2, vector3, vector4, matrix33, matrix44, string, filename, geomname`	
	Array Types	`integerarray, floatarray, color3array, color4array, vector2array, vector3array, vector4array, stringarray, geomnamearray`	
Custom Data Types	**Semantics**	Currently, MaterialX defines three semantics: • "color": the type is interpreted to represent or contain a color, and thus should be color-managed. • "shader": the type is interpreted as a shader output type; nodes or nodegraphs which output a type with a "shader" semantic can be used to define a shader-type node, which can be connected to inputs of "material"-type nodes. • "material": the type is interpreted as a material output type; nodes or nodegraphs which output a type with a "material" semantic can be referenced by a <materialassign> in a <look>. Types not defined with a specific semantic are assumed to have semantic="default".	
	Four Shader-semantic data types	The standard MaterialX distribution includes definitions for four "shader"-semantic data types: surfaceshader, displacementshader, volumeshader, and lightshader.	
MTLX File Format		XML, XML XIncludes, UTF-8	

		Description	Note
Color Spaces and Color Management Systems		MaterialX supports the use of color management systems to associate the RGB components of colors with specific color spaces. MaterialX documents typically specify the working color space of the application that created them, and any image file or color value described in the document can specify the name of the color space it was created in if different from the working color space. MaterialX reserves the color space name "none" to mean no color space conversion should be applied to the images and color values within their scope.	
Units		MaterialX allows floating-point and vector values to be defined in terms of a specific unit and unit type, and can automatically convert values from their specified unit into a scene unit of the same type specified by the application.	
Namespaces		MaterialX supports the specification of “namespaces”, which qualify the MaterialX names of all elements within their scope.	
Geometry Representation		Geometry is referenced by but not specifically defined within MaterialX content. The geometry naming conventions used in the MaterialX specification are designed to be compatible with those used in Alembic (http://www.alembic.io/) and USD (http://graphics.pixar.com/usd). "Geometry" can be any particular geometric object that a host application may support, including but not limited to polygons, meshes, subdivision surfaces, NURBS, implicit surfaces, particle sets, volumes, lights, procedurally-defined objects, etc. The only requirements for MaterialX are that geometries are named using the convention specified below, can be assigned to a material and can be rendered. The naming of geometry should follow a syntax similar to UNIX full paths: /string1/string2/string3/…	
	Lights	MaterialX does not define "light" objects per se, but instead allows referencing externally-defined light objects in the same manner as geometry, via a UNIX-like path. MaterialX does not describe the position, view or shape of a light object: MaterialX presumes that these properties are stored within the external representation. Light object geometries can be turned off (muted) in looks by making the light geometry invisible, assignment of "light"-context shader materials can be done using a <materialassign> within a <look>, and illumination and shadowing assignments can be handled using <visibility> declarations for the light geometry.	
	Geometry Name Expressions	Certain elements in MaterialX files support geometry specification via expressions. The syntax for geometry name expressions in MaterialX largely follows that of “glob” patterns for filenames in Unix environments, with a few extensions for the specific needs of geometry references. Within a single hierarchy level (e.g. between "/"s): • * matches 0 or more characters • ? matches exactly one character • [] are used to match any individual character within the brackets, with "-" meaning match anything between the character preceding and the character following the "-" • {} are used to match any of the comma-separated strings or expressions within the braces	
	Collections	Collections are recipes for building a list of geometries (which can be any path within the geometry hierarchy), which can be used as a shorthand for assignments to a (potentially large) number of geometries at once. Collections can be built up from lists of specific geometries, geometries matching defined geometry name expressions, other collections, or any combination of those.	
	Geometric Properties	Geometric Properties, or "geomprops", are intrinsic or user-defined surface coordinate properties of geometries referenced in a specific space and/or index, and are functionally equivalent to USD's concept of "primvars". A number of geometric properties are predefined in MaterialX: `position, normal, tangent, bitangent, texcoord, geomcolor`	

MaterialX Standard Nodes

Category	Group	MaterialX Node *1 : MaterialX Supplement Notes	Name in RealityKit * n/a : not implemented	Description	Note
Standard Source Nodes					
	Texture Nodes			Texture nodes are used to read filtered image data from image or texture map files for processing within a node graph.	
		image	**Image**	samples data from a single image, or from a layer within a multi-layer image. When used in the context of rendering a geometry, the image is mapped onto the geometry based on geometry UV coordinates, with the lower-left corner of an image mapping to the (0,0) UV coordinate (or to the fractional (0,0) UV coordinate for tiled images).	
		tiledimage (*1)	**Tiled Image**	samples data from a single image, with provisions for tiling and offsetting the image across uv space.	
		triplanarprojection (*1)	**Triplanar Projection**	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.	
	Procedural Nodes			Procedural nodes are used to generate color data programmatically.	
		constant	**Integer Float, Half Boolean Color3, Color4 Vector2, Vector3, Vector4 Matrix2x2, Matrix3x3, Matrix4x4 String**	a constant value.	* RealityKit: Vector2 (Float)/ Vector2 (Half), Vector3 (Float) / Vector3 (Half), Vector4 (Float) / Vector4 (Half) * RealityKit supports Matrix2x2
		ramplr	**Ramplr Ramp Horizontal**	a left-to-right linear value ramp.	
		ramptb	**Ramptb Ramp Vertical**	a top-to-bottom linear value ramp.	
		ramp4 (*1)	**Ramp4 Ramp 4 Corners**	a 4-corner bilinear value ramp.	
		splitlr	**Splitlr Split Horizontal**	a left-right split matte, split at a specified U value.	
		splittb	**Splittb Split Vertical**	a top-bottom split matte, split at a specified V value.	
		noise2d	**Noise2D Noise 2D**	2D Perlin noise in 1, 2, 3 or 4 channels.	
		noise3d	**Noise3D Noise 3D**	3D Perlin noise in 1, 2, 3 or 4 channels.	
		fractal3d	**Fractal3D Fractal Noise 3D**	Zero-centered 3D Fractal noise in 1, 2, 3 or 4 channels, created by summing several octaves of 3D Perlin noise, increasing the frequency and decreasing the amplitude at each octave.	
		cellnoise2d	**CellNoise2D Cellular Noise 2D**	2D cellular noise, 1 channel (type float).	
		cellnoise3d	**CellNoise3D Cellular Noise 3D**	3D cellular noise, 1 channel (type float).	
		worleynoise2d	**WorleyNoise2D Worley Noise 2D**	2D Worley noise, outputting float (distance to closest feature), vector2 (distances to closest 2 features) or vector3 (distances to closest 3 features).	
		worleynoise3d	**WorleyNoise3D Worley Noise 3D**	3D Worley noise, outputting float (distance to closest feature), vector2 (distances to closest 2 features) or vector3 (distances to closest 3 features).	
	Geometric Nodes			Geometric nodes are used to reference local geometric properties from within a node graph	
		position	**Position**	the coordinates associated with the currently-processed data, as defined in a specific coordinate space. This node must be of type vector3.	* In the specification, “model”, “object”, and “world” are defined. However, Reality Composer Pro (beta) supports an additional “tangent”.
		normal	**Normal**	the geometric normal associated with the currently-processed data, as defined in a specific coordinate space. This node must be of type vector3.	* In the specification, “model”, “object”, and “world” are defined. However, Reality Composer Pro (beta) supports an additional “tangent”.
		tangent	**Tangent**	the geometric tangent vector associated with the currently-processed data, as defined in a specific coordinate space. This node must be of type vector3.	* In the specification, “model”, “object”, and “world” are defined. However, Reality Composer Pro (beta) supports an additional “tangent”.
		bitangent	**Bitangent**	the geometric bitangent vector associated with the currently-processed data, as defined in a specific coordinate space. This node must be of type vector3.	* In the specification, “model”, “object”, and “world” are defined. However, Reality Composer Pro (beta) supports an additional “tangent”.
		texcoord	**Texcoord Texture Coordinates**	the 2D or 3D texture coordinates associated with the currently-processed data. This node must be of type vector2 or vector3.	
		geomcolor	**Geomcolor Geometry Color**	the color associated with the current geometry at the current position, generally bound via per-vertex color values. Can be of type float, color3 or color4, and must match the type of the "color" bound to the geometry.	
		geompropvalue	**GeomPropValue Geometic Property**	the value of the specified geometric property (defined using <geompropdef>) of the currently-bound geometry. This node's type must match that of the referenced geomprop.	
	Global Nodes			Global nodes generate color data using non-local geometric context, requiring access to geometric features beyond the surface point being processed. This non-local context can be provided by tracing rays into the scene, rasterizing scene geometry, or any other appropriate method.	
		ambientocclusion	**n/a**	Compute the ambient occlusion at the current surface point, returning a scalar value between 0 and 1. Ambient occlusion represents the accessibility of each surface point to ambient lighting, with larger values representing greater accessibility to light. This node must be of type float.	
	Application Nodes			Application nodes are used to reference application-defined properties within a node graph, and have no inputs	
		frame	**n/a**	the current frame number as defined by the host environment. This node must be of type float. Applications may use whatever method is appropriate to communicate the current frame number to the <frame> node's implementation, whether via an internal state variable, a custom input, or other method.	
		time	**Time Time (float)**	the current time in seconds, as defined by the host environment. This node must be of type float.	* In the Reality Composer Pro, it outputs the current time in seconds.

Category	Group	MaterialX Node *1 : MaterialX Supplement Notes	Name in RealityKit * n/a : not implemented	Description	Note
		updirection	**UpDirection** **Up Direction**	the current scene "up vector" direction, as defined by the shading environment. This node must be of type vector3.	* In the specification, “model”, “object”, and “world” are defined. However, Reality Composer Pro (beta) supports an additional “tangent”.
Standard Operator Nodes				Operator nodes process one or more required input streams to form an output. Like other nodes, each operator must define its output type, which in most cases also determines the type(s) of the required input streams. The inputs of compositing operators are called "fg" and "bg" (plus "alpha" for float and color3 variants, and "mix" for all variants of the mix operator), while the inputs of other operators are called "in" if there is exactly one input, or "in1", "in2" etc. if there are more than one input. If an implementation does not support a particular operator, it should pass through the "bg", "in" or "in1" input unchanged.	
	Math Nodes			Math nodes have one or two spatially-varying inputs, and are used to perform a math operation on values in one spatially-varying input stream, or to combine two spatially-varying input streams using a specified math operation. The given math operation is performed for each channel of the input stream(s), and the data type of each input must either match that of the input stream(s), or be a float value that will be applied to each channel separately.	
		add	**Add**	add a value to the incoming float/color/vector/matrix.	
		subtract	**Subtract**	subtract a value from the incoming float/color/vector/matrix, outputting "in1-in2".	
		multiply	**Multiply**	multiply an incoming float/color/vector/matrix by a value. Multiplication of two vectors is interpreted as a component-wise vector multiplication, while multiplication of two matrices is interpreted as a standard matrix product.	
		divide	**Divide**	divide an incoming float/color/vector/matrix by a value; dividing a channel value by 0 results in floating-point "NaN". Division of two vectors is interpreted as a component-wise division of the first vector by the second, while division of two matrices is interpreted as a standard matrix product of the in1 matrix and the inverse of the in2 matrix.	
		modulo	**Modulo**	the remaining fraction after dividing an incoming float/color/vector by a value and subtracting the integer portion. Modulo always returns a non-negative result, matching the interpretation of the GLSL and OSL mod() function (not fmod()).	
		absval	**Absval** **Abs**	the per-channel absolute value of the incoming float/color/vector.	
		sign	**Sign**	the per-channel sign of the incoming float/color/vector value: -1 for negative, +1 for positive, or 0 for zero.	
		floor	**Floor**	the per-channel nearest integer value less than or equal to the incoming float/color/vector; the output remains in floating point per-channel, i.e. the same type as the input.	
		ceil	**Ceil** **Ceilling**	the per-channel nearest integer value greater than or equal to the incoming float/color/vector; the output remains in floating point per-channel, i.e. the same type as the input.	
		round	**Round**	round each channel of the incoming float/color/vector values to the nearest integer value, e.g "floor(in+0.5)".	
		power	**Power**	raise incoming float/color values to the specified exponent, commonly used for "gamma" adjustment.	
		safepower (*1)	**SafePower** **Safe Power**	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).	
		sin	**Sin**	the sine of the incoming value, which is expected to be expressed in radians.	
		cos	**Cos**	the cosine of the incoming value, which is expected to be expressed in radians.	
		tan	**Tangent**	the tangent of the incoming value, which is expected to be expressed in radians.	
		asin	**Asin**	the arcsine of the incoming value; the output will be expressed in radians.	
		acos	**Acos**	the arccosine of the incoming value; the output will be expressed in radians.	
		atan2	**Atan2**	the arctangent of the expression (iny/inx); the output will be expressed in radians. If both in1 and in2 are provided, they must be the same type.	
		sqrt	**Sqrt** **Square Root**	the square root of the incoming value.	
		ln	**Ln** **Natural Log**	the natural log of the incoming value.	
		exp	**Exp**	"e" to the power of the incoming value.	
		clamp	**Clamp**	clamp incoming values per-channel to a specified range of float/color/vector values.	
		min	**Min**	select the minimum of the two incoming values	
		max	**Max**	select the maximum of the two incoming values	
		normalize	**Normalize**	output the normalized vectorN from the incoming vectorN stream; cannot be used on float or color N streams. Note: the fourth channel in vector4 streams is not treated any differently, e.g. not as a homogeneous "w" value.	
		magnitude	**Magnitude**	output the float magnitude (vector length) of the incoming vectorN stream; cannot be used on float or colorN streams. Note: the fourth channel in vector4 streams is not treated any differently, e.g. not as a homogeneous "w" value.	
		dotproduct	**DotProduct** **Dot Product**	output the (float) dot product of two incoming vectorN streams; cannot be used on float or colorN streams.	
		crossproduct	**CrossProduct** **Cross Product**	output the (vector3) cross product of two incoming vector3 streams; cannot be used on any other stream type. A disabled crossproduct node passes through the value of in1 unchanged.	
		transformpoint	**TransformPoint** **Transform Point**	transform the incoming vector3 coordinate from one specified space to another; cannot be used on any other stream type.	
		transformvector	**TransformVector** **Transform Vector**	transform the incoming vector3 vector from one specified space to another; cannot be used on any other stream type.	
		transformnormal	**TransformNormal** **Transform Normal**	transform the incoming vector3 normal from one specified space to another; cannot be used on any other stream type.	
		transformmatrix	**TransformMatrix** **Transform Matrix**	transform the incoming vectorN coordinate by the specified matrix. ○ in (vectorN) : the input vector. If needed, an additional 1.0 component will be temporarily appended to the in vector to make it match the dimension of the transforming mat matrix, then removed after transformation.	
		normalmap	**NormalMap** **Normal Map**	transform a normal vector from object or tangent space into "world" space.	
		place2d (*1)	**Place2D** **Place 2D**	transform incoming UV texture coordinates for 2D texture placement.	
		transpose	**Transpose**	output the transpose of the incoming matrix.	
		determinant	**Determinant**	output the float determinant of the incoming matrixNN stream.	

Category	Group	MaterialX Node *1 : MaterialX Supplement Notes	Name in RealityKit * n/a : not implemented	Description	Note
		invertmatrix	**InvertMatrix** **Invert Matrix**	output the inverse of the incoming matrix; if the input matrix is not invertible, the output matrix will consist of all floating-point "NaN" values.	
		rotate2d	**Rotate2D** **Rotate 2D**	rotate a vector2 value about the origin in 2D.	
		rotate3d	**Rotate 3D** **Rotate3D**	rotate a vector3 value about a specified unit axis vector.	
		arrayappend	**n/a**	create a two-element array [in1, in2] from two base-type values, or append the in2 value to the end of the in1 array of the same type.	
		dot	**Dot**	a no-op, passes its input through to its output unchanged. Users can use dot nodes to shape edge connection paths or provide documentation checkpoints in node graph layout UI's.	
	Adjustment Nodes			Adjustment nodes have one input named "in", and apply a specified function to values in the incoming stream.	
		remap	**Remap**	linearly remap incoming values from one range of float/color/vector values to another.	
		smoothstep	**SmoothStep** **Smooth Step**	output a smooth (hermite-interpolated) remapping of input values from low-high to output 0-1.	
		curveadjust	**n/a**	output a smooth remapping of input values using the centripetal Catmull-Rom cubic spline curve defined by specified knot values, using an inverse spline lookup on input knot values and a forward spline through output knot values. All channels of the input will be remapped using the same curve.	
		curvelookup	**n/a**	output a float, colorN or vectorN value smoothly interpolated between a number of knotvalue values, using the position of in within knots as the knotvalues interpolant.	
		luminance	**Luminance**	(color3 or color4 only) output a grayscale value containing the luminance of the incoming RGB color in all color channels, computed using the dot product of the incoming color with the luma coefficients of the active CMS configuration; the alpha channel is left unchanged if present.	
		rgbtohsv	**RGBToHSV** **RGB to HSV**	(color3 or color4 only) convert an incoming color from RGB to HSV space (with H and S ranging from 0 to 1); the alpha channel is left unchanged if present. This conversion is not affected by the current color space or CMS configuration.	
		hsvtorgb	**HSVToRGB** **HSV to RGB**	(color3 or color4 only) convert an incoming color from HSV to RGB space; the alpha channel is left unchanged if present. This conversion is not affected by the current color space or CMS configuration.	
		contrast (*1)	**Contrast**	increase or decrease contrast of incoming float/color values using a linear slope multiplier.	
		range (*1)	**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.	
		saturate (*1)	**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.	
		hsvadjust (*1)	**HSVAdjust** **HSV Adjust**	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.	
	Compositing Nodes			Compositing nodes have two (required) inputs named fg and bg, and apply a function to combine them.	
	Premult			Premult nodes operate on 4-channel (color4) inputs/outputs, have one input named in, and either apply or unapply the alpha to the float or RGB color.	
		premult	**Premult** **Premultiply**	multiply the RGB channels of the input by the Alpha channel of the input. ○ in (color4): the input value or nodename; default is (0,0,0,1).	
		unpremult	**Unpremult** **Unpremultiply**	divide the RGB channels of the input by the Alpha channel of the input. If the Alpha value is zero, the original color4 input value is passed through unchanged.	
	Blend			Blend nodes take two 1-4 channel inputs and apply the same operator to all channels (the math for alpha is the same as for R or RGB). In the Blend Operator table, "F" and "B" refer to any individual channel of the fg and bg inputs respectively. Blend nodes support an optional float input mix, which can be used to mix the original bg value (mix=0) with the result of the blend operation (mix=1, the default).	
		plus	**Plus** **Additive Mix**	B+F	
		minus	**Minus** **Subtractive Mix**	B-F	
		difference	**Difference**	abs(B-F)	
		burn	**Burn**	1-(1-B)/F	
		dodge	**Dodge**	B/(1-F)	
		screen	**Screen**	1-(1-F)(1-B)	
		overlay	**Overlay**	2FB if F<0.5 1-(1-F)(1-B) if F>=0.5	
	Merge			Merge nodes take two 4-channel (color4) inputs and use the built-in alpha channel(s) to control the compositing of the fg and bg inputs. In the Merge Operator table, "F" and "B" refer to the non-alpha channels of the fg and bg inputs respectively, and "f" and "b" refer to the alpha channels of the fg and bg inputs. Merge nodes are not defined for 1-channel or 3-channel inputs, and cannot be used on vectorN streams. Merge nodes support an optional float input mix, which can be used to mix the original bg value (mix=0) with Merge Operator the result of the blend operation (mix=1, the default).	
		disjointover	**DisjointOver** **Disjoint Over**	RGB output : F+B if f+b<=1; F+B(1-f)/b if f+b>1 Alpha output: min(f+b,1)	
		in	**In**	RGB output: Fb Alpha output: fb	
		mask	**Mask**	RGB output: Bf Alpha output: bf	
		matte	**Matte**	RGB output: Ff+B(1-f) Alpha output: f+b(1-f)	
		out	**Out**	RGB output: F(1-b) Alpha output: f(1-b)	
		over	**Over**	RGB output: F+B(1-f) Alpha output: f+b(1-f)	

Category	Group	MaterialX Node *1 : MaterialX Supplement Notes	Name in RealityKit * n/a : not implemented	Description	Note
	Masking			Masking nodes take one 1-4 channel input in plus a separate float mask input and apply the same operator to all channels (if present, the math for alpha is the same as for R or RGB). The default value for the mask input is 1.0 for the inside operator, and 0.0 for the outside operator In the Masking Operator table, "F" refers to any individual channel of the in input. Note: for all types, inside is equivalent to the multiply node: both operators exist to provide companion functions for other data types or their respective inverse or complementary operations.	
		inside	**Inside**	Fm	
		outside	**Outside**	F(1-m)	
	Mix			The Mix node takes two 1-4 channel inputs fg and bg plus a separate 1-channel mix input and mixes the fg and bg according to the mix value. The equation for "mix" is as follows, with "F" and "B" referring to any channel of the fg and bg inputs respectively (which can be float, colorN or vectorN but must match), and "m" referring to the float mix input value (which has a default value of 0):	
		mix	**Mix**	Fm+B(1-m)	
	Conditional Nodes			Conditional nodes are used to compare values of two streams, or to select a value from one of several streams.	
		ifgreater	**Ifgreater** **If Greater**	output the value of the in1 or in2 stream depending on whether the value of one test input is greater than the value of another. Ifgreater nodes can be of output type float, colorN or vectorN.	
		ifgreatereq	**IfGreaterOrEqual** **If Greater Or Equal**	output the value of the in1 or in2 stream depending on whether the value of one test input is greater than or equal to the value of another. Ifgreatereq nodes can be of output type float, colorN or vector N.	
		ifequal	**IfEqual** **If Equal**	output the value of the in1 or in2 stream depending on whether the value of two test inputs are equal or not. Ifequal nodes can be of output type float, colorN or vectorN.	
		switch	**Switch**	output the value of one of up to ten input streams, according to the value of a selector input which. Switch nodes can be of output type float, colorN or vectorN, and have five inputs, in1 through in10 (not all of which must be connected), which must match the output type.	
	Channel Nodes			Channel nodes are used to perform channel manipulations and data type conversions on float, colorN, and vectorN streams, allowing the order and number of channels to be modified, and the data types of streams to be altered.	
		convert	**Convert**	convert a stream from one data type to another. Only certain unambiguous and commonly-needed conversions are supported	
		swizzle	**Swizzle**	perform an arbitrary permutation of the channels of the input stream, returning a new stream of the specified type. Individual channels may be replicated or omitted, and the output stream may have a different number of channels than the input.	
		combine2 **combine3** **combine4**	**Combine2** **Combine3** **Combine4**	combine the channels from two, three or four streams into the same total number of channels of a single output stream of a specified compatible type. For color output types, no colorspace conversion will take place; the channels are simply copied as-is.	
		extract (*1)	**Extract**	generate a float stream from one channel of a colorN o r vectorN stream.	
		separateN (*1)	**Separate2** **Separate3** **Separate4**	separate2: output each of the channels of a vector2 as a separate float output. separate3: output each of the channels of a color3 or vector3 as a separate float output. separate4: output each of the channels of a color4 or vector4 as a separate float output.	
	Convolution Nodes			Convolution nodes have one input named "in", and apply a defined convolution function on the input stream. Some of these nodes may not be implementable in ray tracing applications; they are provided for the benefit of purely 2D image processing applications.	
		blur	**n/a**	a convolution blur.	
		heighttonormal	**n/a**	convert a scalar height map to a normal map of type vector3.	
Standard Shader-Semantic Operator Nodes				The Standard MaterialX Library defines the following node variants operating on "shader"-semantic types.	
		add	**n/a**	add two surface/displacement/volumeshader closures.	
		multiply	**n/a**	multiply a surface/displacement/volumeshader closure by a float or color3/vector3 value: surfaceshaders and volumeshaders may be multiplied by a float or color3, while displacementshaders may be multiplied by a float or vector3.	
		mix	**n/a**	linear blend between two surface/displacement/volumeshader closures.	

Custom Nodes in RealityKit

Group	Node	Description	Note
RealityKit			
	Camera Index Switch	Render different results for each eye in a stereoscopic render.	
	Camera Position	The position of the camera in the scene.	
	Cube Image	A cube map in .ktx format, corresponding to Metal texture `typeCube`. Can be used for skybox or environment maps.	
	Geometry Modifier	A function that manipulates the location of a model's vertices, run once per vertex.	
	Geometry Modifier Model To View		
	Geometry Modifier Model To World		
	Geometry Modifier Normal To World		
	Geometry Modifier Projection To View		
	Geometry Modifier Vertex ID		
	Geometry Modifier View To Projection		
	Geometry Modifier World To Model		
	RealityKit Environment Radiance		
	Occlusion Surface	A surface shader that defines properties for a RealityKit Occlusion material that does not receive dynamic lighting.	
	PBR Surface	A surface shader that defines properties for a RealityKit Physically Based Rendering material.	
	Shadow Receiver Surface	A surface shader that defines properties for a RealityKit Occlusion material that receives dynamic lighting.	
	Surface Custom Attribute **Surface Custom Attribute 0** **Surface Custom Attribute 1** **Surface Custom Attribute 2** **Surface Custom Attribute 3**	Surface Custom Attribute Surface Custom Attribute 0 (vector2h) / (vector4h) Surface Custom Attribute 1 (vector2h) / (vector4h) Surface Custom Attribute 2 (vector4h) Surface Custom Attribute 3 (vector4h)	
	Surface Model To View		
	Surface Model To World		
	Surface Projection To View		
	Surface Screen Position		
	Surface View Direction		
	Surface View To Projection		
	Surface World To View		
	Unlit Surface	A surface shader that defines properties for a RealityKit Unlit material.	
	View Direction	A vector from a position in the scene to the view reference point.	
Adjustment			
	Step	Step	
Geometric			
	Reflect	Reflects a vector about another vector.	
	Refract	Refracts a vector using a given normal and index of refraction (eta).	
Logic			
	And	Boolean operation in1 & in2	
	Not	Returns !input.	
	Or	Boolean operation in1 || in2	
	XOR	Returns true if only one of the inputs is true.	
Math			
	Fractional	Returns the fractional part of a floating point number.	
	One Minus	One Minus	
2D Texture			
	Transform 2D	A node that applies an affine transformation to a 2d input.	
	UV Texture	A MaterialX version of USD UV Texture reader.	
Data			
	Primvar Reader (bool) **Primvar Reader (float)** **Primvar Reader (integer)** **Primvar Reader (vector2f)** **Primvar Reader (vector3f)** **Primvar Reader (vector4f)**	A node that provides the ability for shading network to consume data defined on geometry.	
Material			
	NodeGraph	A node that can contain shading nodes and other node graphs.	
PBR			
	Preview Surface	A MaterialX version of USD Preview Surface.	