glTF Texture Procedurals

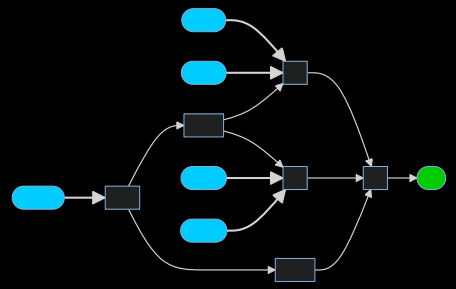
December 2024 Update

Bernard Kwok & Ashwin Bhat

Introduction

What

Runtime representation of “procedural” graphs



Where

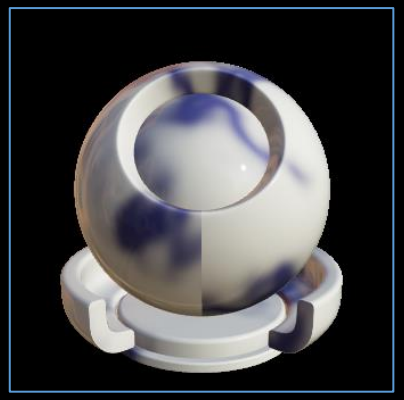
Khronos Extensions:

KHR_texture_procedurals

EXT_texture_procedurals_mx_1_39

Why

Give alternative to “distillation” (e.g “baking”)



When

Specification for 2025

Target: MaterialX 1.39

Objectives

Interoperability	Fidelity	Editability	Validation
Adheres “1:1” to a “standard” schema 	Allow for non-lossy bi - direction translation of graphs	Exposes explicit logic and interfaces	Validate against KHR specification schema
But uses runtime friendly format		Reusable and extensible	No  runtime dependencies
		Runtime inputs editable and animatable (*)	

(*) Via Khronos animation extension

Representation

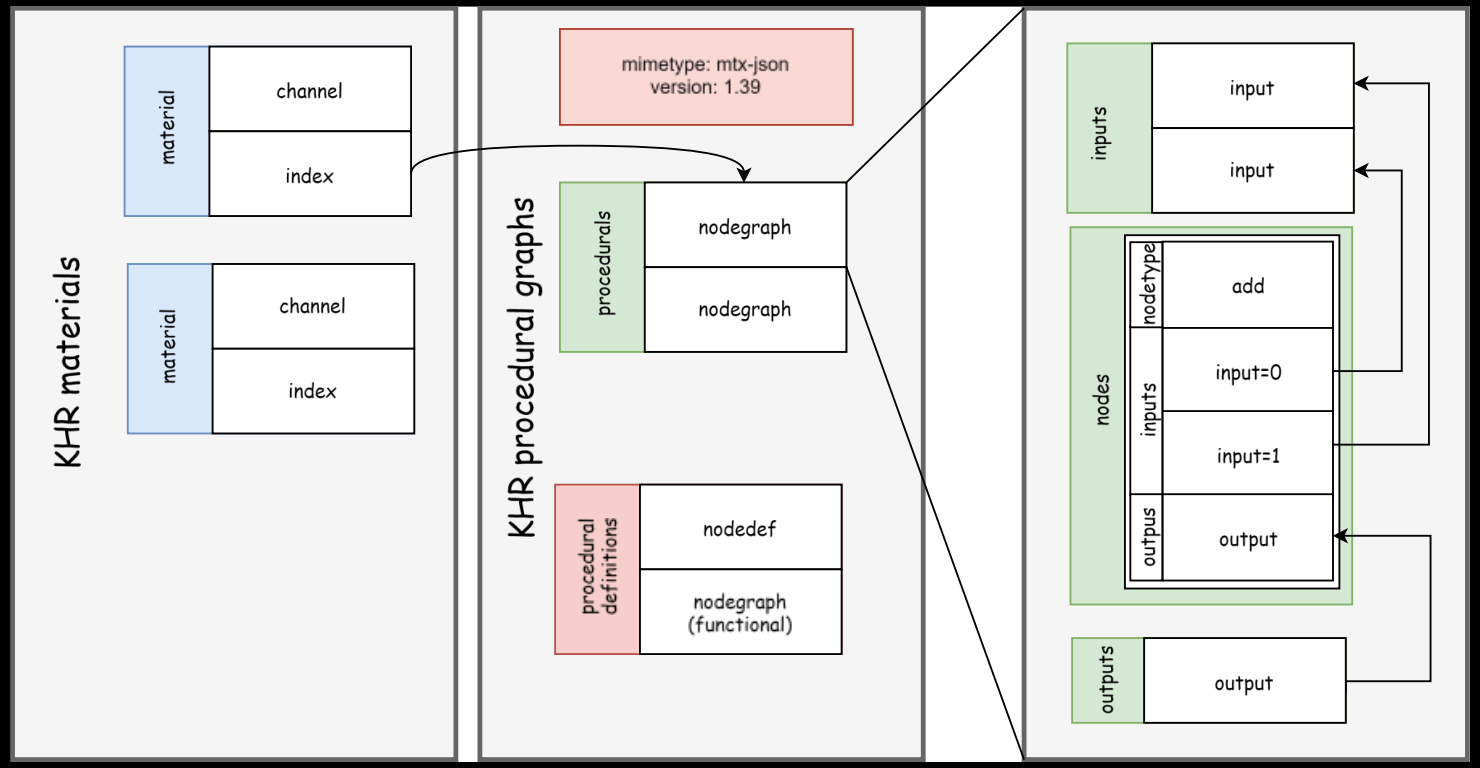
Standardized	Versioned	Runtime Format	Meta-data
Only use MaterialX node definitions **All** libraries representable (*) 	Schema fixed to MaterialX release version identifier. Separate glTF extension version id.	JSON with glTF conventions. e.g. `[x,y,z]` vector syntax Cannot be referenced  	Allow **all** meta-data to be maintained e.g. User interface meta-data

(*) Including “pbr”, “bxdf”, “nprlib”

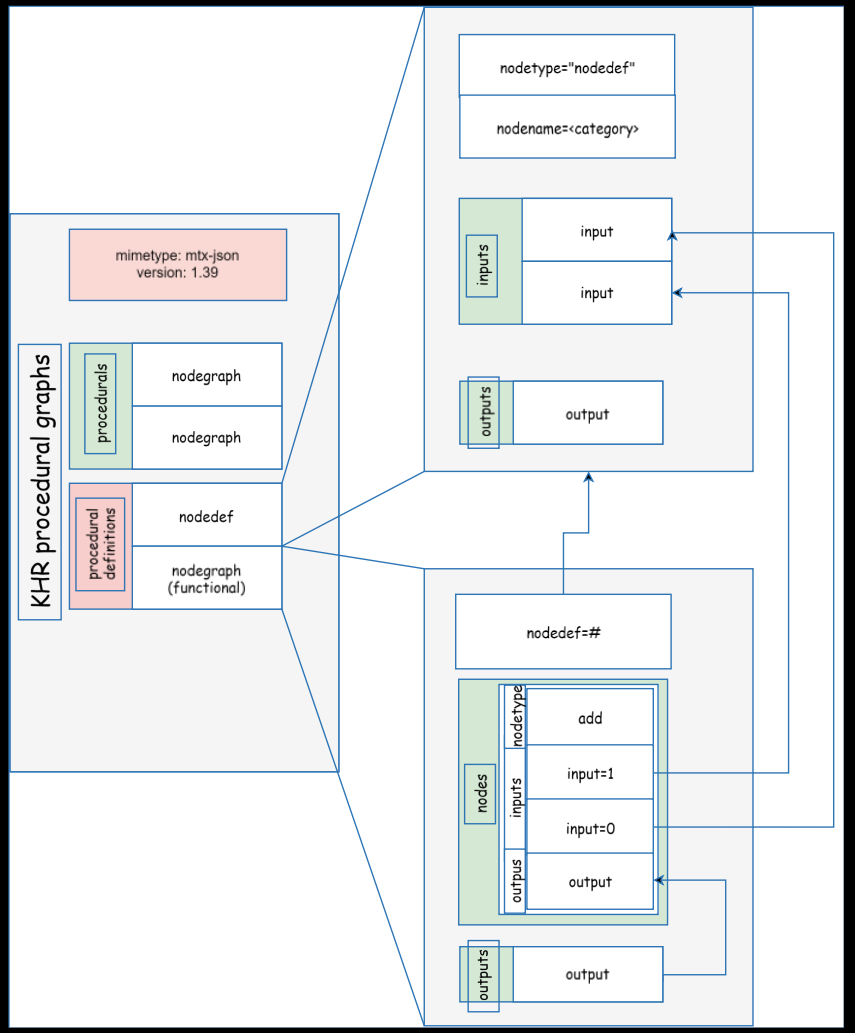
Representation : Components

Types	Identifiers	Nodes + Graphs	Definitions
Boolean, numeric and URIs No strings, arrays, enumerations, structures	Name/ path identifiers not required Reference by storage location (glTF convention)	Graphs required “procedural” encapsulation Interface “publishing” required	Proposed: Follow MaterialX schema Convertible to / from MaterialX and OpenUSD.

Representation : Component Diagram



Material Node Graphs



Node Definitions

Representation: Format Comparison

Feature	MaterialX	OpenUSD	glTF
Component String Identifiers	Yes	Yes	Optional
Numeric Tuples	string	list ()	array []
Connection Syntax	string (name in context)	Absolute Path	numeric index
Explicit Node Outputs	No	Yes	Yes
Node Type Grouping	No	No	Yes
Reference to nodedef on node instance	Yes	Yes	Optional
NodeGraph Nesting	"Yes" (not implemented)	Yes	No
Nodegraph Inputs settable and animatable	Yes (keyframes not part of data model)	Yes	Yes
Referencing	Yes	Yes	No
Definition Versioning	Yes	Yes	Yes
Meta-Data	Yes	Yes	Yes
Node Definition	Yes	Yes	In Progress

Fidelity

Texture Procedurals

Current specification: Only pattern graphs

Non-destructive / additive: Both rasterized and procedural version of pattern can be specified.

Placement

Bypass matrix “baking” issues:

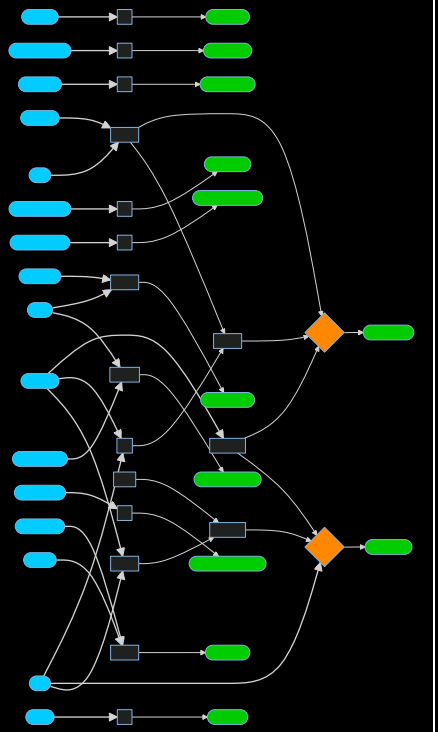
- Rotation order
- TRS order
- Pivot
- Deg vs Rad (*)

Fidelity

Shading Models

Bake not required to map models

Shader translation graphs supported e.g. OpenPBR -> glTF



Colorspace

Meta-data supported

Working colorspace fixed

Input colorspace glTF default: sRGB

Units

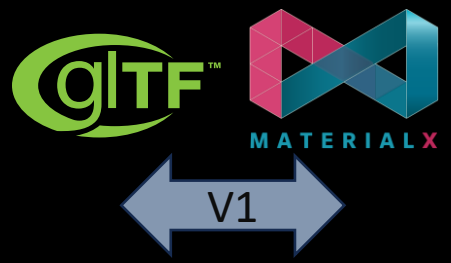
Arbitrary meta-data supported

Includes units (e.g. distance, angles)

Support

Tooling

V1 of MaterialX / glTF conversion available



Resource Binding

Stream bind detached from texture bind

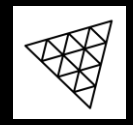
Name remapping required for

Validation

Tooling has test suite examples

round-trip unit test validation

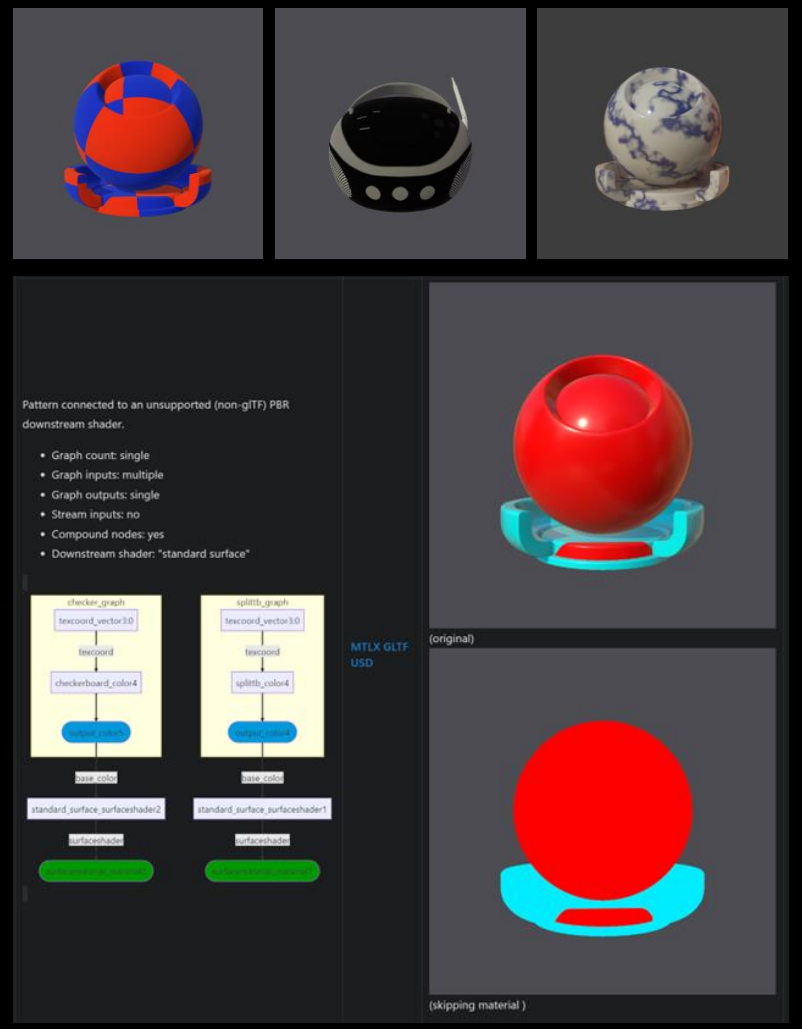
Runtime

ThreeJS  support being investigated

NVIDIA MDL support being investigated

glTF MaterialX Convertor

- Python bi-directional converter for MaterialX documents
- Github: glTF_MaterialX_Converter
- Dependents: 1.39.1 MaterialX minimum
- Phase 1: Completed
  - Patterns mapped to color on glTF PBR and unlit
  - Sample graphs: MTLX, glTF, USD
  - Round-trip Validation: “functional equivalence” test
  - Reference rendering: MaterialXView (MaterialX)



Summary / Next Steps

Specification

Core support logic complete

JSON Schema forthcoming

Completion target: 2025

Adoption

Unity / ThreeJS workflow

MDL workflow

Gather more adopters for prototyping

Procedurals

Finish spec for definitions and variants

Examine shading graphs using PBR nodes

Interop

Tooling Continuation

Validate workflow