KHRONOS GROUP

W3C

WebGL

WebGPU

glTF

WebXR

Forging the Immersive Web

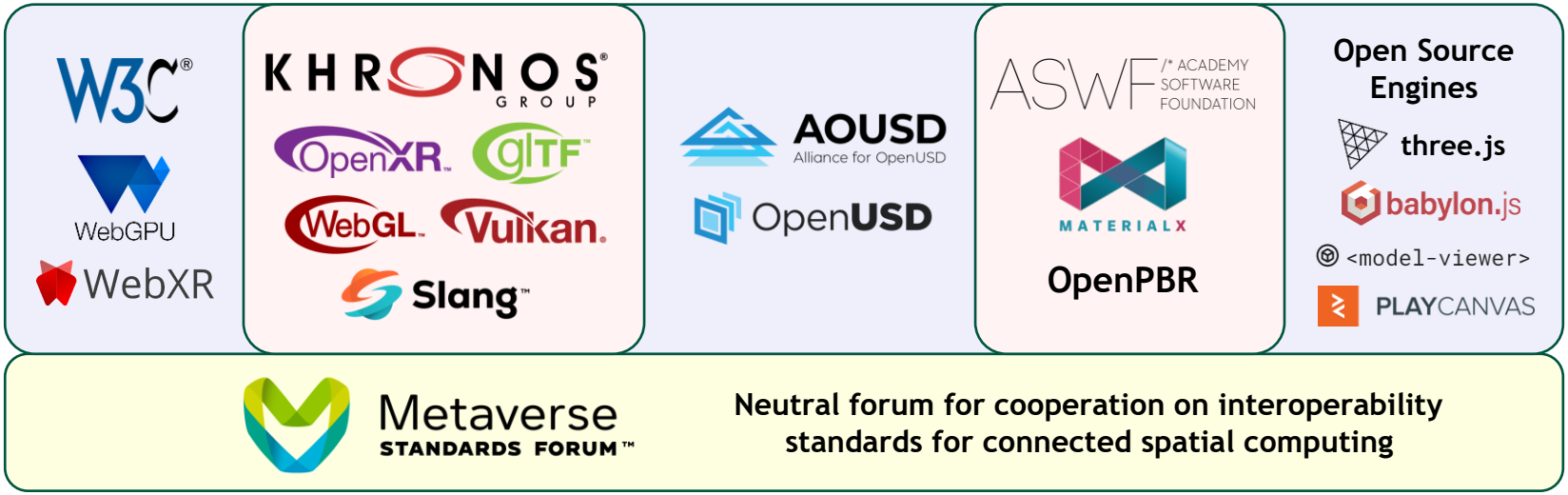
Khronos, W3C, ASWF, Three.js

Immersive Web: Standards & Open Source

Creating the Immersive Web will need and leverage the work of many standards organizations, consortia and open-source projects

Standardization is a cooperative endeavor!

Today's session will provide latest updates and how the industry is working together



Speakers

Session Title	Speaker	Length
Khronos and Web Standards	Neil Trevett, Khronos	10 minutes
W3C and the Immersive Web	Atsushi Shimono, W3C	10 minutes
WebGPU Update	Gregg Tavares, Google	10 minutes
Three.js Update	Ricardo Cabello, Google	10 minutes
OpenPBR and the Web	Julien Guertault, Adobe Frédéric Servant, Autodesk	10 minutes
Audience Q&A	All	10 minutes

KHRONOS GROUP

W3C

WebGL

WebGPU

glTF

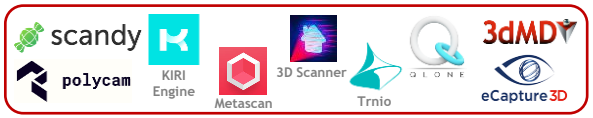
WebXR

Khronos and Web Standards

Neil Trevett, Khronos President

3D Authoring Tools

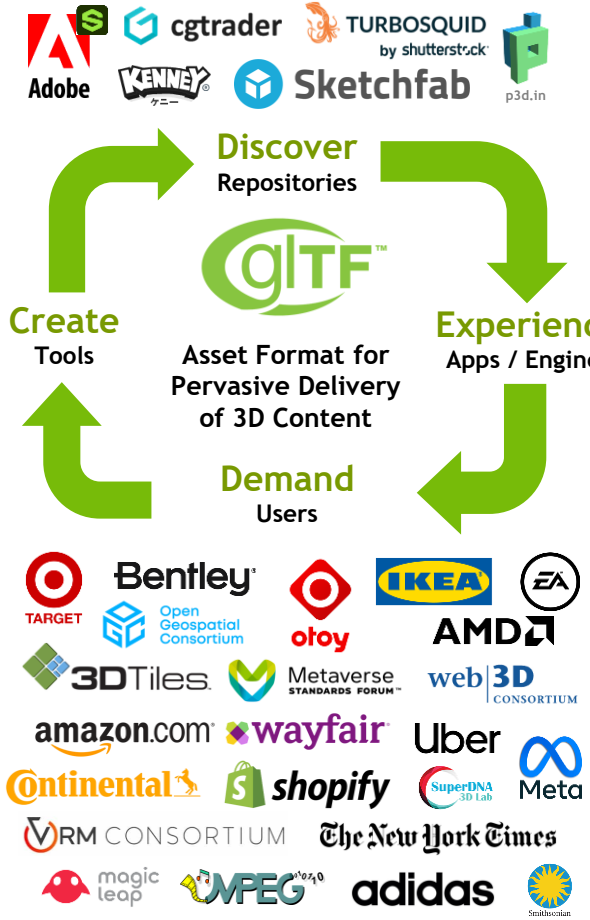
VR / AR Authoring Tools



3D Scanning Tools

Convertors, Optimizers and Loaders

Validation and Reference Tools



Game Engines

Web Engines

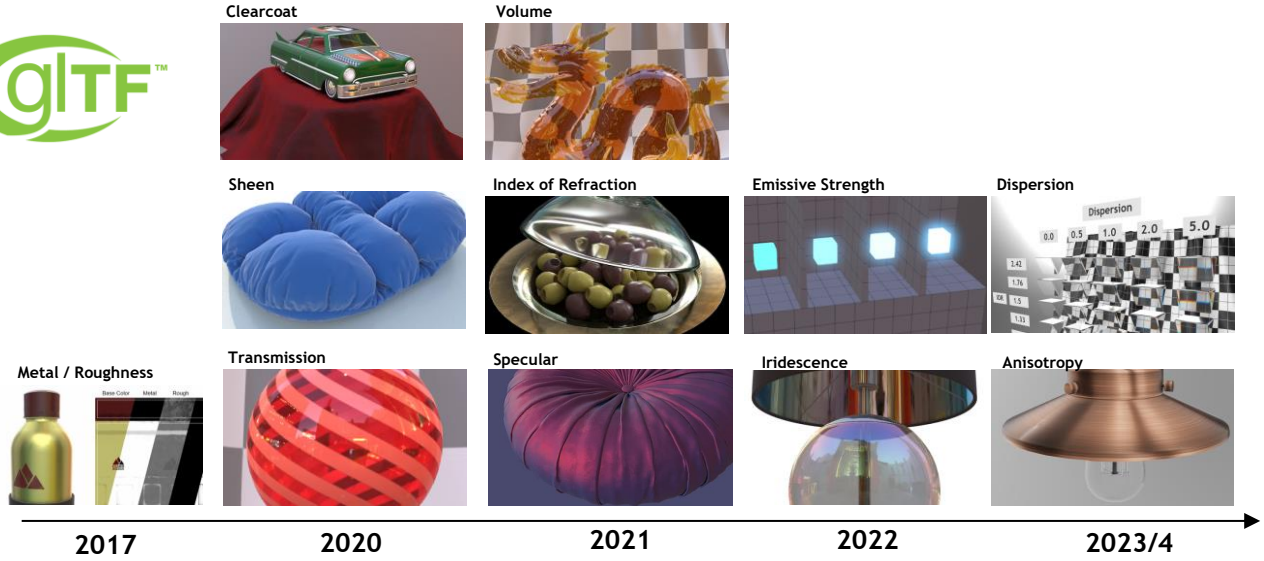
Apps and Engines

VR / AR Apps and Engines

Productivity and Social Apps

glTF PBR Materials Roadmap

Incremental consolidation and meticulous specification of *proven and accepted industry practice*



Subsurface
In development

MaterialX
1) Export glTF PBR as MaterialX node graph
2) Use MaterialX to drive procedural texture inputs into glTF PBR

OpenPBR
Working to align and resolve differences between glTF and OpenPBR

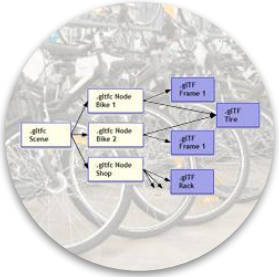
Industry Collaboration

glTF Spatial Computing Roadmap

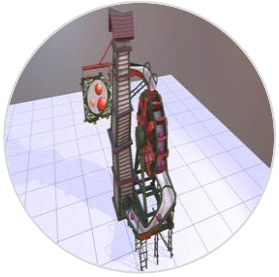
Interactivity

Node-based graph handling of user actions or events
Distillation of widespread accepted practice
Flexible computed scene state updates and animations

Compose complex scenes from referenced glTF assets
Efficiency and flexibility in transmission/delivery use cases
Placement, configuration, cache reuse, personalization, deferred loading, LODs, mesh variants etc.



Complex Scenes



Physics

Describes physical properties of assets
Distillation of widespread accepted practice
Rigid Bodies: motions, collisions, materials , joints filters

Triggered and controlled from interactivity node graph
3D spatialized audio with 6DoF source/listener capabilities,
Play, stop, pause, loop, and speed controls
Splitting, merging, up/down-mixing, reverb, filtering

Audio

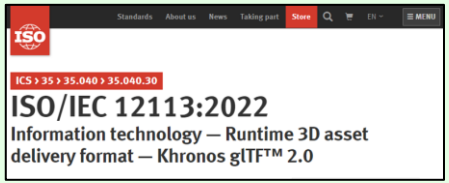
glTF as Foundational Standard

Khronos welcomes working collaboratively to leverage glTF extensibility

Market-specific extensions and use of glTF defined by partner standards organization
Accelerates development of market segment functionality
Avoid needless duplication and fragmentation

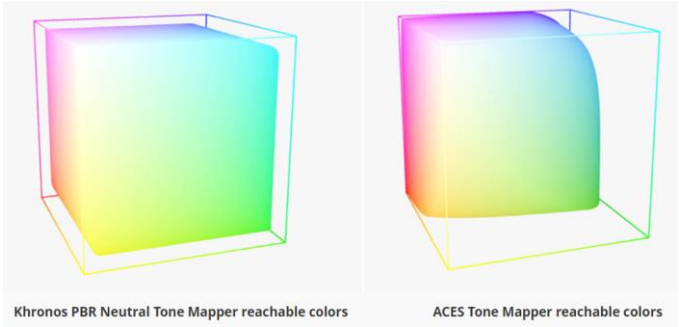
VRM	3D Tiles	MPEG	X3D	PDF	Additional Market Segments
Avatar Format *.vrm extension*	*.b3dm and .i3dm extensions*	ISO 23090-14:2023 MPEG-I *.mp4, miv, ivr*	ISO 19775-1:2023 X3D4 *.x3d extension*	ISO/TS 32007 glTF in PDF *.PDF extension*	

Foundation of
Core specification and
glTF working group extensions



Khronos PBR Neutral Tone Mapper

- **True-to-Life Color Rendering of 3D Products**
  - Released in May 2024
  - Specification and sample implementation
- **1:1 match for colors up to a certain maximum value**
  - The remainder of color space used as headroom for compressed highlights
- **Wide adoption and support by 3D tools and engines**
  - <model-viewer>, Autodesk, Babylon.js, Blender, Dassault, Filament
  - London Dynamics, Phasmatic, Three.js, and ThreeKit



Khronos 3D Commerce

Making 3D Pervasive - in the Real World

Build Once, Use Everywhere

Developing tools and techniques for 3D assets to be reliably and consistently used and displayed across diverse platforms and engines

Multiple Projects Underway

Render Showcase - evolve and expand Render Fidelity Site

Tone Mapping (PBR Neutral), exposure and lighting

Apparel: Skeletal & Facial Anchoring, Virtual Try-On, Stitching / detailing, Simulation

Content Developers test and validate assets

The equivalent of an API Conformance Test Suite (Vulkan has >3M tests)

Public comparison of multiple engines
1) Early warning of inconsistencies
2) Incentive to implement functionality

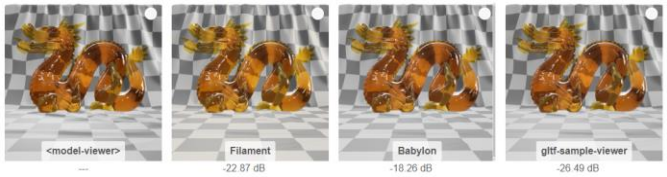
Sample Viewer
Asset Display Exemplar

Validator
Asset Correctness

Asset Library
Showcase & Test Assets

Asset Generator
Generate unit tests

Render Showcase
Web-based display comparisons



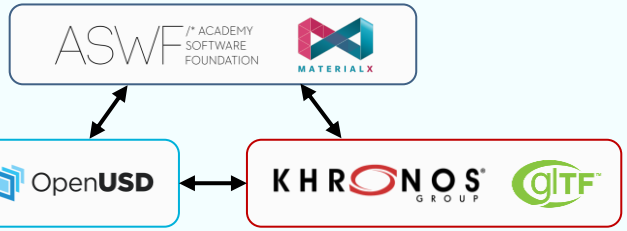
Broadening 3D Asset Cooperation

Metaverse STANDARDS FORUM™

Venue for *multiple* standards organizations and industry to cooperate *together*
Accessible membership to encourage broad participation

glTF/USD 3D Asset Interoperability Working Group

550 Members in the Forum working group includes many participants from both SDOs AND broader industry
Actively fostering interoperability understanding and roadmap cooperation between glTF and USD Communities
Wide cooperation complements 1-1 liaisons between SDOs



Participation by multiple SDOs
Each SDO evolves its own standards
Limited 1-1 interaction through liaisons

FBX Migration Project
Investigation of roadmap recommendations to Khronos and AOUSD so key toolchains can migrate from FBX dependence

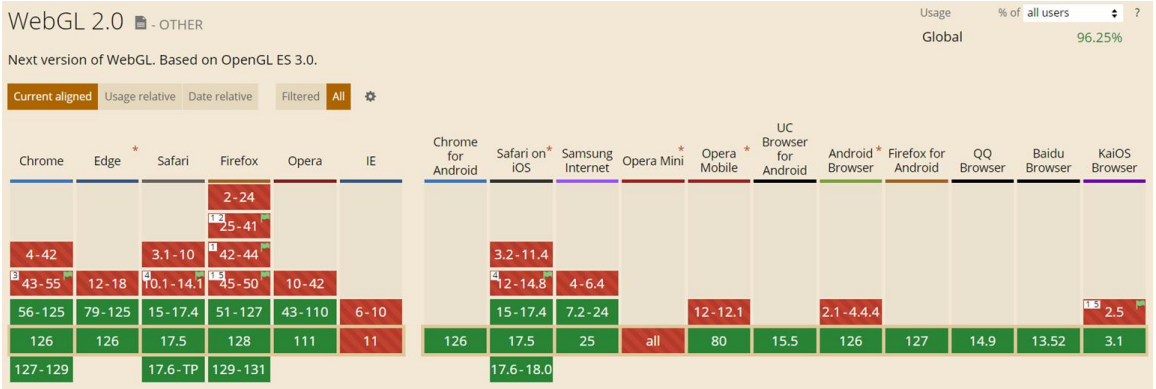
glTF ⇄ USD Conversion Project
Guidelines and recommendations for key conversion pain points with associated test assets

OpenUSD ⇅ glTF

WebGL Update

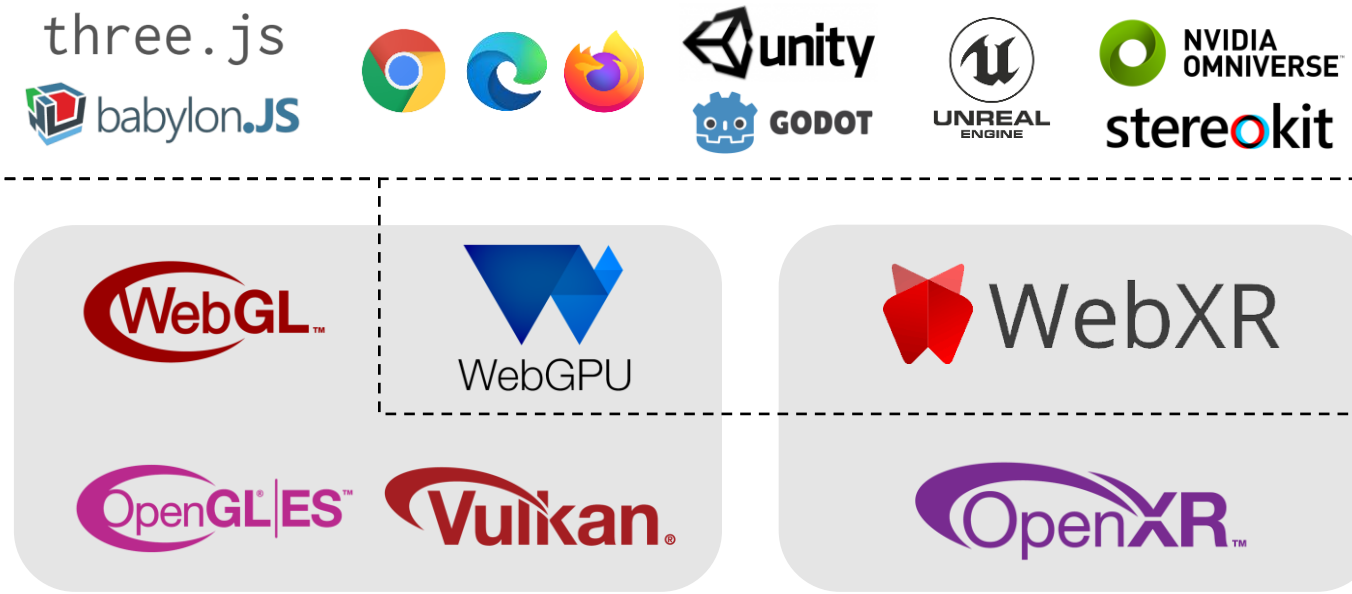
- **Khronos is fully supporting development of WebGPU at W3C**
  - Working for a smooth transition for developers between WebGL and WebGPU
  - WebGPU brings GPU Compute to the Web using Vulkan/DX12/Metal backends
- **WebGL is pervasive and will be used by many applications for many years**
  - Khronos is evolving the WebGL specification and supporting multiple implementations
  - New extensions: Pixel Local Storage and more OpenGL ES functionality

WebGL 2.0 is available on >96% of browsers



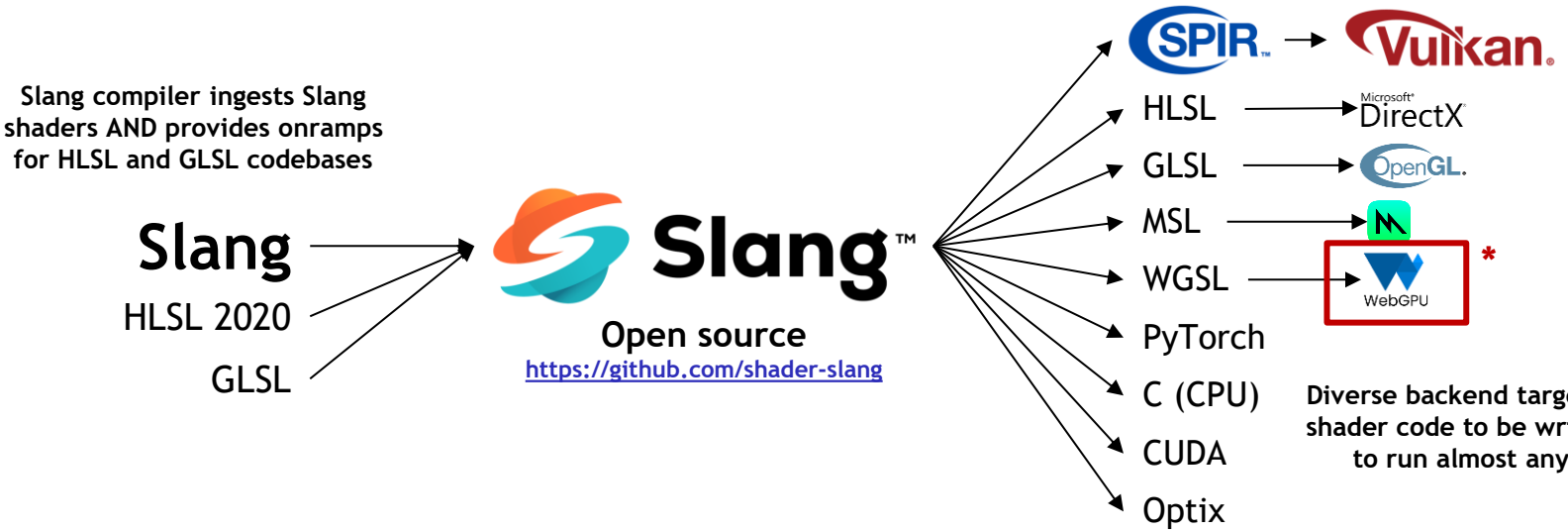
Khronos and W3C - XR Cooperation

XR Applications and Engines have access to native and JavaScript APIs with aligned functionality



Open-Source, Cross-Platform Compiler

Khronos now hosts the Slang open-source shading language and compiler
Builds on 15 years development at NVIDIA
Modular code, portable deployment, and neural computation



*** How can Slang add value to the WebGPU Community?**

KHRONOS GROUP

W3C

WebGL

WebGPU

glTF

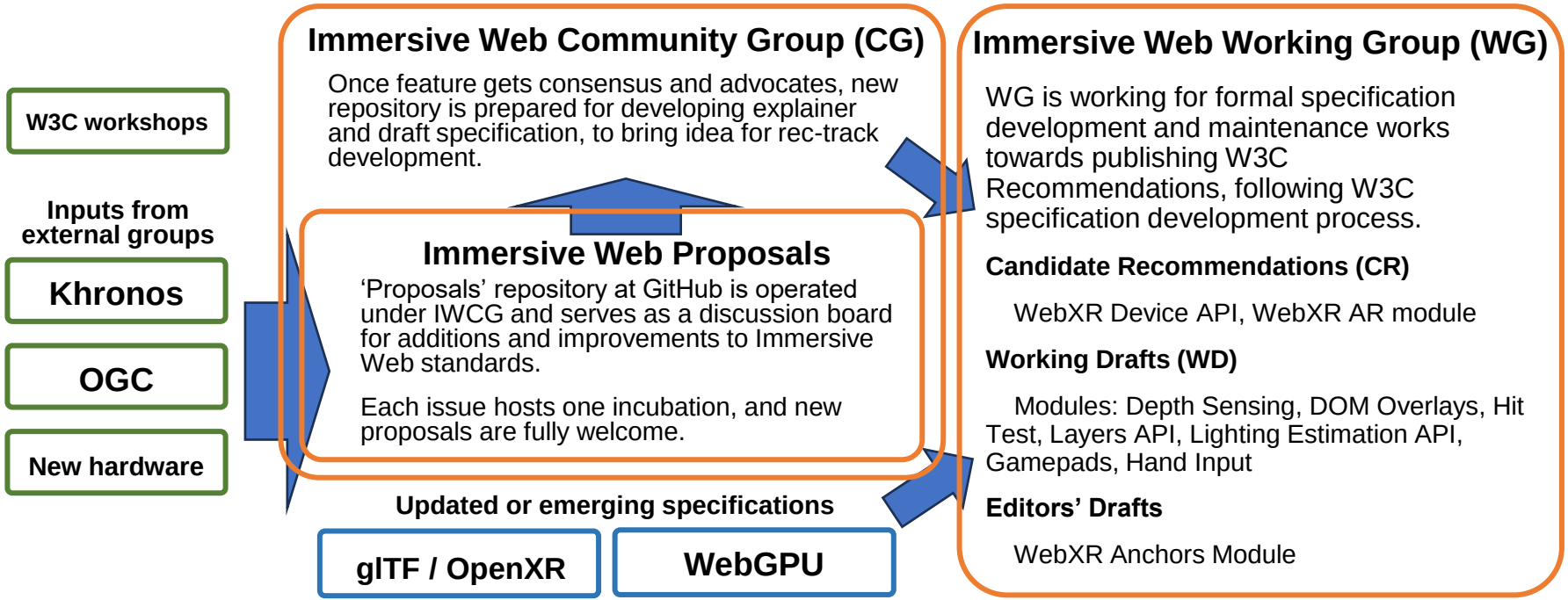
WebXR

W3C and the Immersive Web

Atsushi Shimono, W3C

Immersive Web at W3C

Immersive Web groups (IWWG/IWCG) at W3C help bring high-performance Virtual Reality (VR) and Augmented Reality (AR) (collectively known as XR) to the open Web via APIs to interact with XR devices and sensors in browsers



https://www.w3.org/immersive-web/list_spec.html

WebXR and modules

Real world geometry module

Raw camera access module

Body tracking module

Marker tracking module

Geo alignment module

Anchors module

Lighting Estimation API

Support for exposing estimates of environment lighting conditions, to be used for surface of virtual objects.

Hand Input module

Extends XRInputSource with the functionality to track articulated hand poses. (hand and joint)

Layers API

Enables to initialize XR session with layers to be shown on XR screen. Several types of screen are provided.

Hit Test module

Add interfaces to cast rays into real world environment, to get a point of a ray intersected with physical object.

Gamepads module

Extends XRInputSource for gamepad, based on integration with Gamepad API.

DOM Overlays module

Enables to initialize XR session with specific DOM element, to be shown on XR screen.

Depth sensing module

Provides XRDepthInformation interface for 2D depth information.

WebXR Device API (Core)

WebXR Device API defines core of Immersive Web feature from initialization, session handling, Layers, Input, and Events.

This core specification provides Navigator.xr, XRSystem, XRSession, XRSpace, XRView, XRPose, XRInputSource, XRLayer (XRWebGLLayer), XRSessionEvent,

WebXR AR module

Defines behavior of ‘immersive-ar’ session mode of navigator.xr

WebXR-WebGPU binding

Immersive Web Recent Updates

Even (most of) WebXR Device API (Core) is already stable and just waiting for meeting with W3C REC criteria (mostly waiting second implementation), updates are under discussion on various areas:

- to incorporate new and emerging specifications with updating Core
- to enable new hardware capabilities with adding new modules
- to enable emerging software driven requirements

During 2024 rechartering of the Immersive Web WG, we decided to turn several specifications into living standard based model, with keeping other modules within standard W3C rec-track development model – finishing with Recommendation publication, for enabling updates to core specification and some key modules in timely manner – especially supporting new key technologies (like WebGPU) and hardware capabilities.

Topics or key interests of group participants change time by time, like dropping interest on geolocation (or metaverse/smartcity?) based interactions among XR sessions. Recent topics are:

- Adding WebGPU support into WebXR Device API
- <module> element
- Transformation among data, on framerate, matrix, spatial, etc.
- New hardware feature enablement
  - Body tracking

Adding WebGPU along with WebGL interfaces

Originally, WebGL was solely supported within WebXR, as XRWebGLLayer extends XRLayer, which provides a WebGL framebuffer to render into 3D graphics on the XR device. (XRLayer was defined as the base class only for XRWebGLLayer at the initial moment, and gained modules to extend like WebXR Layers API Level 1).

Similar to other data format related issue, following WebGPU specification gets matured, integration of WebGPU into WebXR became major topic.

Massive discussion was made recently within WG calls and during TPAC 2024 (Sep 2024), several baseline agreement made, but still some agreements and decisions are required to move into actual spec text update.

New <model> element into HTML

After several conversations and breakout sessions at a W3C workshop on the Games on the Web (2019), discussion started to add new HTML element named as <model>.

Following recent releases/announcement of new hardware, attentions increased on this area especially to have capability of this feature for head mount typed XR device. Recent discussions includes:

- Supported data format – minimum supported set?
- How element will exist within XR, stereoscopic entity lives in 3D space
  - Background, transparency, view, etc...
- Interaction with HTML based feature, including CSS, into 3D model data

Still baseline (required feature) is under discussion, Immersive Web CG decided to increase frequency of group meetings to have discussion on this area.

Expectations for the Immersive Web from WebXR Japanese community

- There is an active community about XR and events are often held.
  - For example, this month we will host “XR Kaigi,” the largest XR-specific conference in Japan. Also, on December 21, there will be two VRChat-related events, and I will be hosting the “LODGE XR Talk.
- Companies and organizations associated with XR are equally excited.
  - As you know, VRM, a file format for 3D avatars from Japan, is based on glTF.
    - “pixiv/three-vrm” allows us to handle VRM with WebXR :)
  - Palan Inc. is another Japanese company that specializes in WebXR. Their WebXR solutions are used throughout Japan.
- And although not numerous, there are individual developers who are particularly strong in WebXR.
  - In particular, @ninisan_drumath san continues to disseminate Babylon.js and WebXR Device API, and @wakufactory san is the developer of “SpatialShell”, a playground for WebXR.
  - WebXR Advent Calendar 2024, an advent calendar dedicated to WebXR, is also underway.
    - However, we have not gathered this year and would like to make it more exciting :(
- I expect WebXR to be more exciting than ever now that Apple Vision Pro supports the WebXR Device API.
  - In 2025, we plan to hold the WebXR Developer Conference Tokyo in Japan, a conference for developers specializing in XR among other XRs.
  - I would like to see more interaction between developers who use game engines such as Unity and web front-end engineers who are strong in JavaScript.

KHRONOS GROUP

W3C

WebGL

WebGPU

glTF

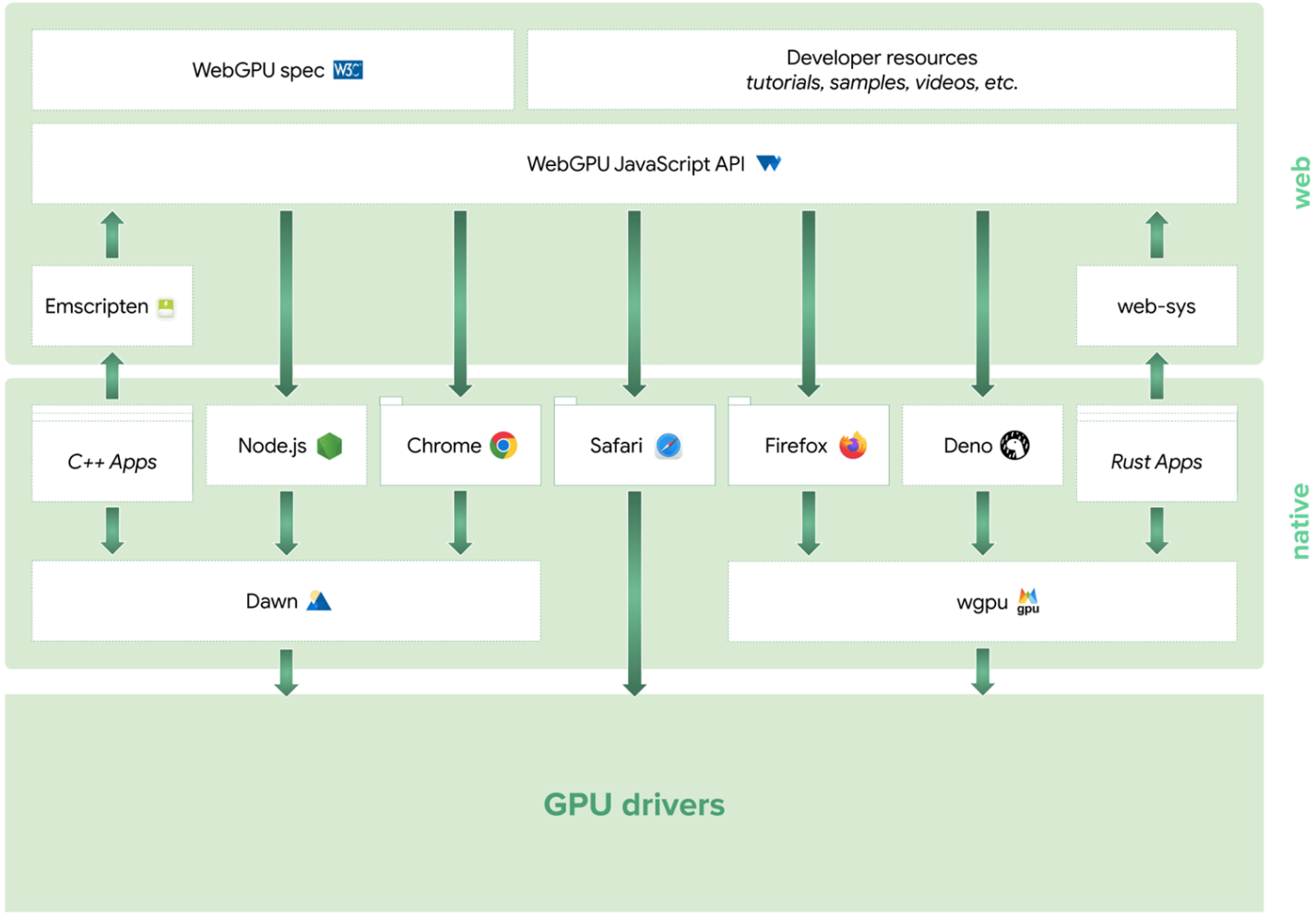
WebXR

WebGPU Update

Gregg Tavares, Google

WebGPU?

- Cross Platform
  GPU Graphics and Compute
- DirectX, Vulkan, Metal, OpenGL (*)
- Windows, Mac, Linux, ChromeOS, Android, iOS
- Chrome, Edge, Firefox, Safari, ….
- WGSL (WebGPU Shading Language)


WebGPU spec
Developer resources
tutorials, samples, videos, etc.
WebGPU JavaScript API
web
Emscripten
web-sys
C++ Apps
Node.js
Chrome
Safari
Firefox
Deno
Rust Apps
native
Dawn
wgpu
GPU drivers

WebGPU - **Hello Triangle in 60 seconds**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});

const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - Create A Device

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});

const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - **Setup the Canvas**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});

const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - **Compile Shaders**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});

const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - **Setup a Pipeline**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});
```

```
const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - **Encode a Command Buffer**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});

const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - **Start a Render Pass**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});

const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - **Execute the Pipeline**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});

const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - **Submit the Commands**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, format: 'rgba8unorm' });

const module = device.createShaderModule({ code: `
    @vertex fn vs(@builtin(vertex_index) vNdx : u32)
        -> @builtin(position) vec4f {
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx], 0.0, 1.0);
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```

```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
      targets: [{format: 'rgba8unorm'}],
    },
});

const encoder = device.createCommandEncoder();
const target = context.getCurrentTexture();
const pass = encoder.beginRenderPass({
    colorAttachments: [{
        view: target.createView(),
        loadOp: 'clear',
        storeOp: 'store',
    }],
});
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

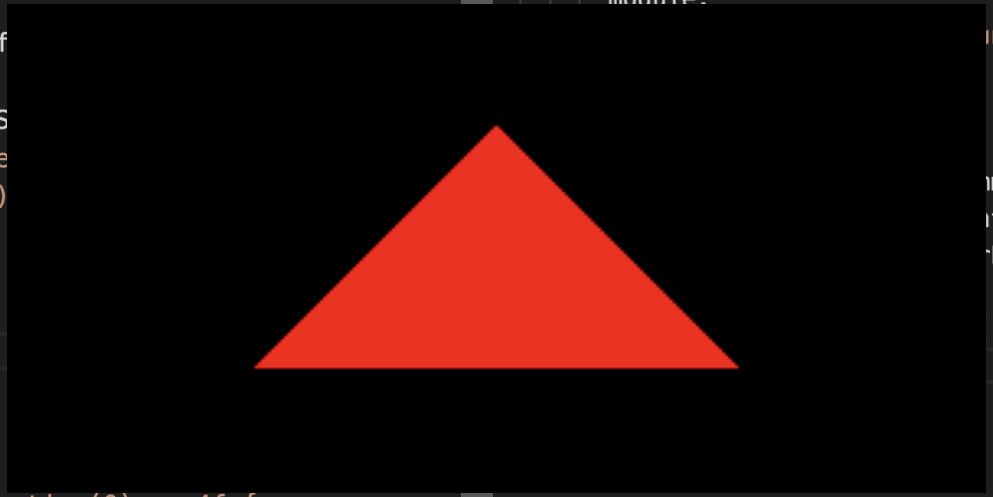
WebGPU - **Result**

```
const adapter = await navigator.gpu.requestAdapter();
const device = await adapter.requestDevice();

const context = document.querySelector("canvas")
  .getContext('webgpu');
context.configure({ device, f

const module = device.createS
    @vertex fn vs(@builtin(ve
        -> @builtin(position)
      let pos = array(
        vec2f( 0.0,  0.5),
        vec2f(-0.5, -0.5),
        vec2f( 0.5, -0.5));
      return vec4(pos[vNdx],
    }

    @fragment fn fs() -> @location(0) vec4f {
      return vec4f(1.0, 0.0, 0.0, 1.0);
    }`,
});
```



```
const pipeline = device.createRenderPipeline({
    layout: 'auto',
    vertex: { module },
    fragment: {
      module,
                                    norm'}],
                                    mandEncoder();
                                    tTexture();
                                    Pass({
                                    ),
pass.setPipeline(pipeline);
pass.draw(3);
pass.end();
device.queue.submit([encoder.finish()]);
```

WebGPU - It’s not just Web

- JavaScript/TypeScript
  - Chromium/Firefox/Safari
  - Node/Deno
  - React Native
- C/C++ (Dawn) - and Emscripten
- Rust (wgpu) - web
- Python
- Go
- Java/Kotlin

WebGPU - Recent Features

- Colorspaces (srgb, display-p3)
- HDR (brighter colors)
- Clip Distances
- Dual Source Blending
- More Vertex Formats
- F16
- DP4A (Dot Product of 4 Elements and Accumulate)

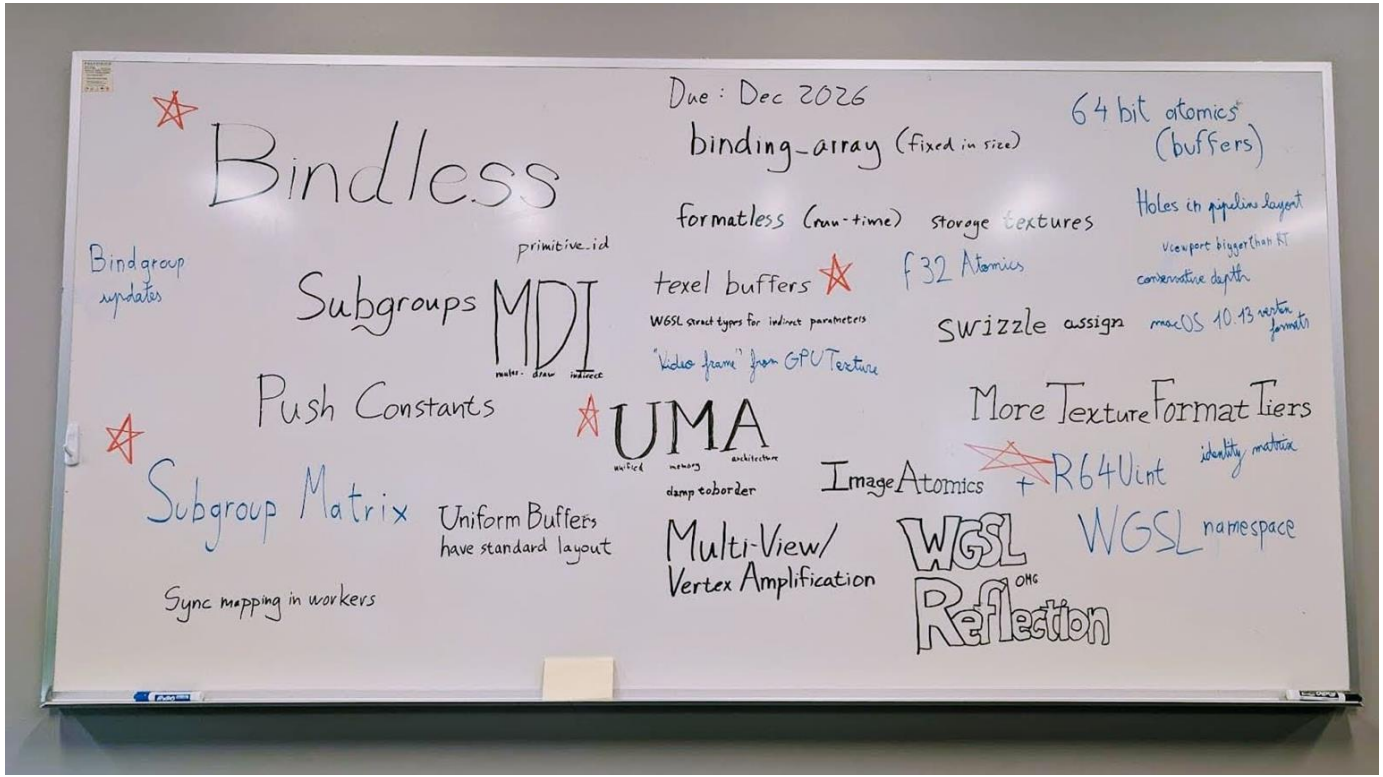
WebGPU - RSN (Real Soon Now)

- Safari
- Firefox
- Multi-draw Indirect
- Subgroups
- Compat (WebGPU light on OpenGL ES)

WebGPU - XR

- Collaborating with Apple on the API design.
- Available for testing in the coming weeks on:
  - Android (Cardboard and ARCore)
  - Windows (OpenXR)
- Enabled with the “WebXR/WebGPU Binding” flag in about:flags.
- Looking for developer feedback on ease of use, missing features, etc.
- Currently some known performance bumps, but optimizations are in-progress.
- Explainer: https://github.com/immersive-web/WebXR-WebGPU-Binding
- Minimal Samples: https://immersive-web.github.io/webxr-samples/webgpu/

WebGPU: Next Steps



WebGPU Links

- The Spec: https://www.w3.org/TR/webgpu/
- WGSL: https://www.w3.org/TR/WGSL/
- Spec Discussion: https://github.com/gpuweb/gpuweb
- Tests: https://github.com/gpuweb/cts
- Dawn: https://dawn.googlesource.com/dawn
- WGPU: https://github.com/gfx-rs/wgpu
- React Native: https://github.com/wcandillon/react-native-webgpu
- Samples: https://webgpu.github.io/webgpu-samples/
- Tutorials: https://webgpufundamentals.org/

Three.js Update

Ricardo Cabello, Google

https://mrdoob.github.io/talks/202412-siggraph/

KHRONOS GROUP

W3C WebGL WebGPU glTF WebXR

OpenPBR Update

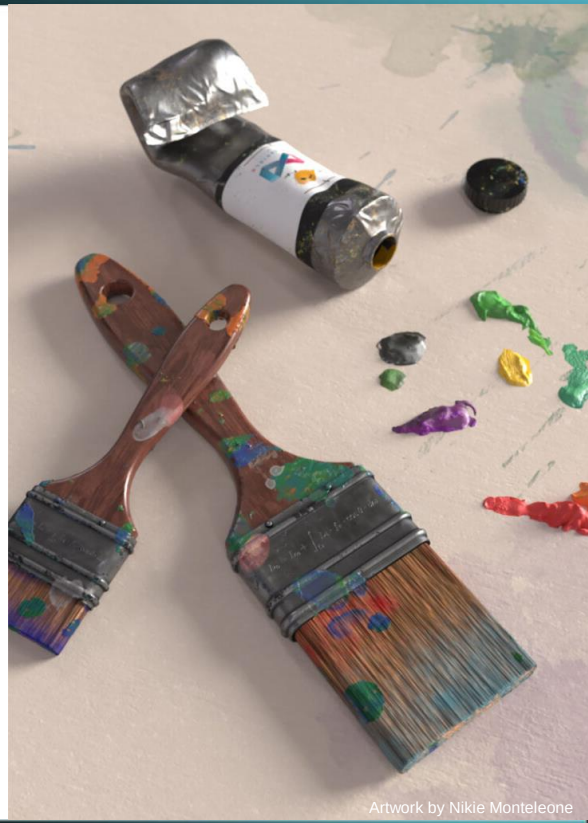
Julien Guertault, Adobe
Frédéric Servant, Autodesk

OpenPBR update agenda

- State of OpenPBR
- New features overview
- Integrations
- Future work

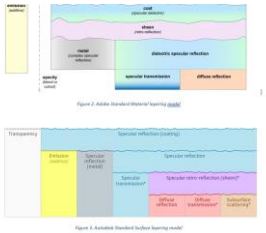
A new standard

- Merge Adobe & Autodesk material models
- Physically based
- Artist friendly
- Open governance
- Reference implementation



OpenPBR Timeline

Nov 2022 — **Common ground**



Apr 2023 — **New specification**

Jun 2023 — **ASWF**

MATERIALX

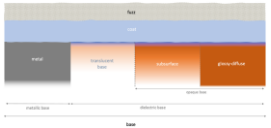
ASWF /* ACADEMY SOFTWARE FOUNDATION

Aug 2023 — **Announce and private reviews**

NVIDIA · PIXAR ANIMATION STUDIOS · imageworks · Industrial Light & Magic · EPIC GAMES · chaos · blender · SideFX

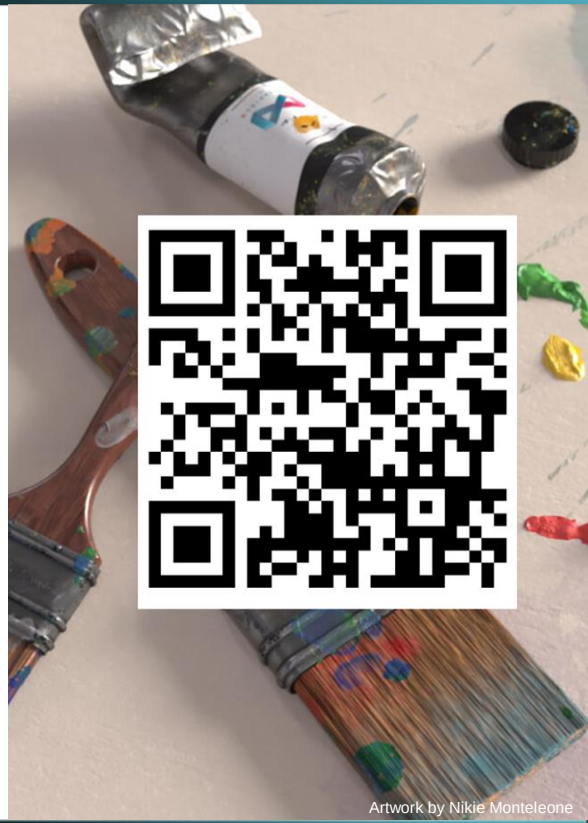
Oct 2023 — **Public preview**

June 2024 — **1.0 release**

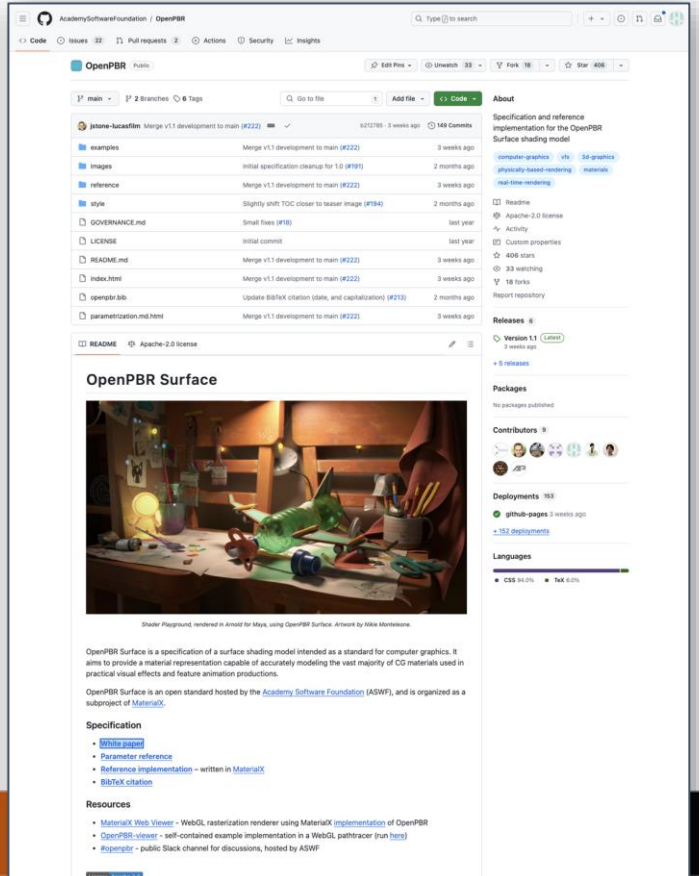


OpenPBR Status

- Finalized 1.1 specification
- Enhancements to previous models
- MaterialX reference implementation
- ASWF governance model
- Major interest from end-users and vendors



Open source repo and specification



AcademySoftwareFoundation / OpenPBR

Code · Issues 22 · Pull requests 2 · Actions · Security · Insights

OpenPBR Public

main · 2 Branches · 6 Tags · Go to file · Add file · Code

jstone-lucasfilm Merge v1.1 development to main (#222) · b212785 · 3 weeks ago · 149 Commits

examples	Merge v1.1 development to main (#222)	3 weeks ago
images	Initial specification cleanup for 1.0 (#191)	2 months ago
reference	Merge v1.1 development to main (#222)	3 weeks ago
style	Slightly shift TOC closer to teaser image (#194)	2 months ago
GOVERNANCE.md	Small fixes (#18)	last year
LICENSE	Initial commit	last year
README.md	Merge v1.1 development to main (#222)	3 weeks ago
index.html	Merge v1.1 development to main (#222)	3 weeks ago
openpbr.bib	Update BibTeX citation (date, and capitalization) (#213)	2 months ago
parametrization.md.html	Merge v1.1 development to main (#222)	3 weeks ago

README · Apache-2.0 license

About

Specification and reference implementation for the OpenPBR Surface shading model

computer-graphics · vfx · 3d-graphics · physically-based-rendering · materials · real-time-rendering

Readme · Apache-2.0 license · Activity · Custom properties · 406 stars · 33 watching · 18 forks · Report repository

Releases 6

Version 1.1 Latest · 3 weeks ago

+ 5 releases

Packages

No packages published

Contributors 9

Deployments 153

github-pages 3 weeks ago

+ 152 deployments

Languages

CSS 94.0% · TeX 6.0%

OpenPBR Surface

Shader Playground, rendered in Arnold for Maya, using OpenPBR Surface. Artwork by Nikie Monteleone.

OpenPBR Surface is a specification of a surface shading model intended as a standard for computer graphics. It aims to provide a material representation capable of accurately modeling the vast majority of CG materials used in practical visual effects and feature animation productions.

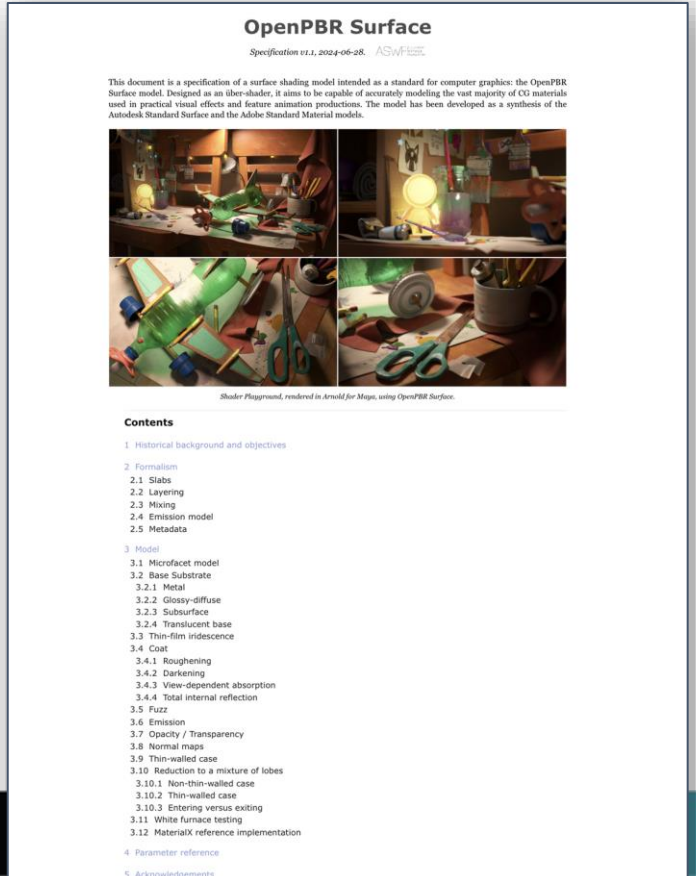
OpenPBR Surface is an open standard hosted by the Academy Software Foundation (ASWF), and is organized as a subproject of MaterialX.

Specification

- White paper
- Parameter reference
- Reference implementation – written in MaterialX
- BibTeX citation

Resources

- MaterialX Web Viewer - WebGL rasterization renderer using MaterialX implementation of OpenPBR
- OpenPBR-viewer - self-contained example implementation in a WebGL pathtracer (run here)
- #openpbr - public Slack channel for discussions, hosted by ASWF



OpenPBR Surface

Specification v1.1, 2024-06-28. ASWF

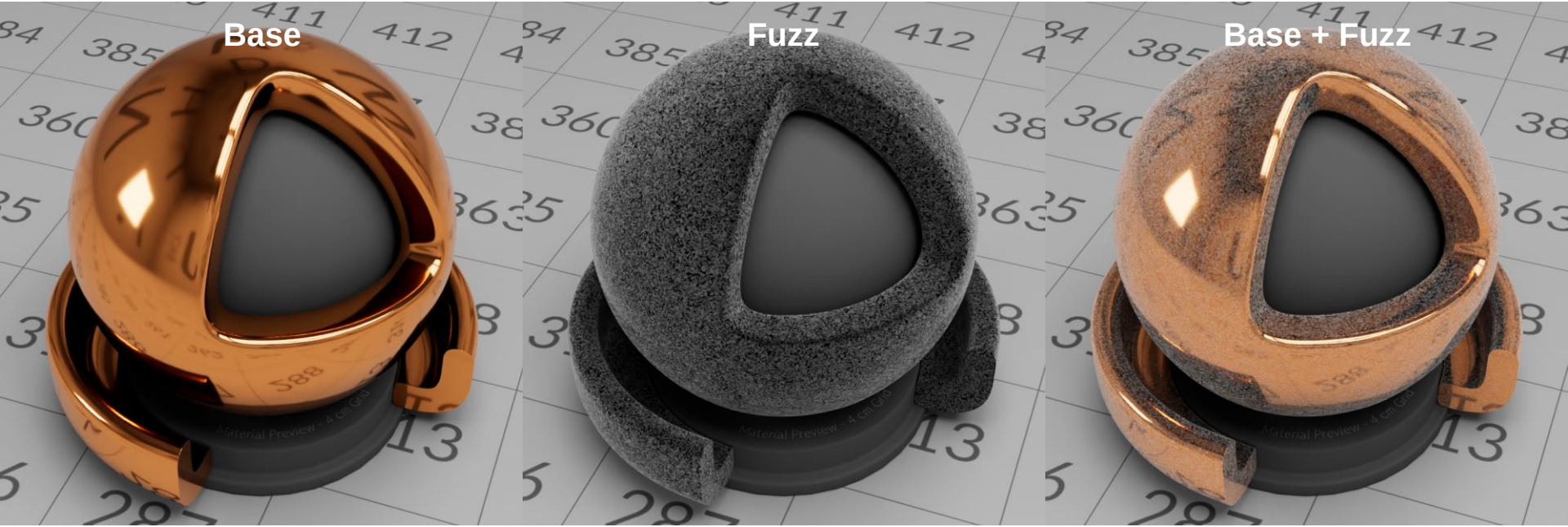
This document is a specification of a surface shading model intended as a standard for computer graphics: the OpenPBR Surface model. Designed as an über-shader, it aims to be capable of accurately modeling the vast majority of CG materials used in practical visual effects and feature animation productions. The model has been developed as a synthesis of the Autodesk Standard Surface and the Adobe Standard Material models.

Shader Playground, rendered in Arnold for Maya, using OpenPBR Surface.

Contents

Features

More expressive layer ordering



Energy-preserving Oren-Nayar

https://arxiv.org/abs/2410.18026

Art-directable metal model

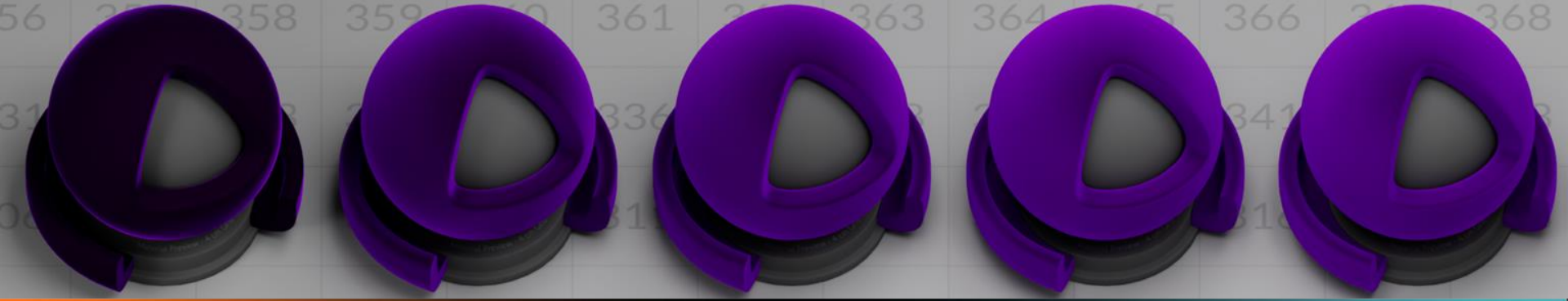
Gulbrandsen

F82-Tint

New fuzz model

New: Zeltner '22 sheen

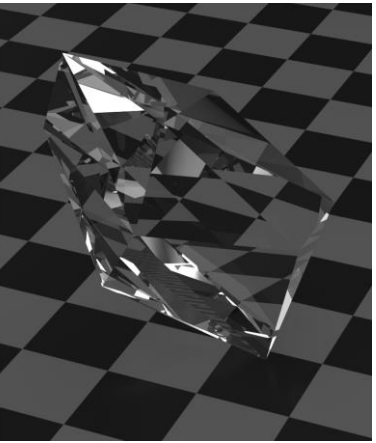
Old: SPI sheen

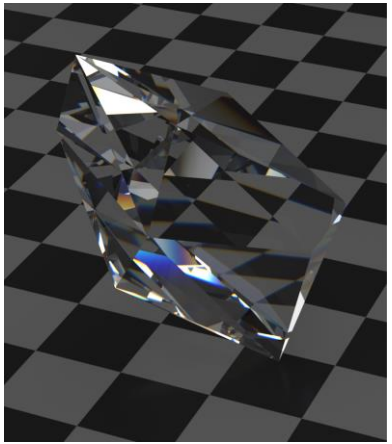


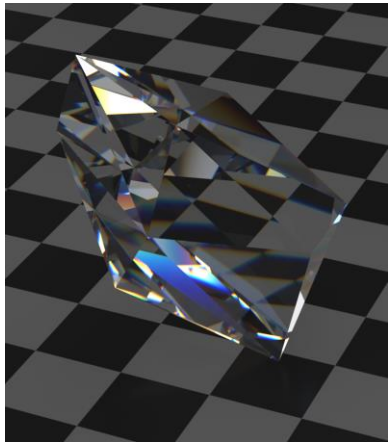
Coat darkening

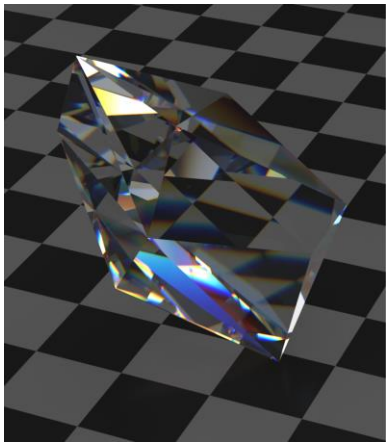
0.00 0.25 0.50 0.75 1.00

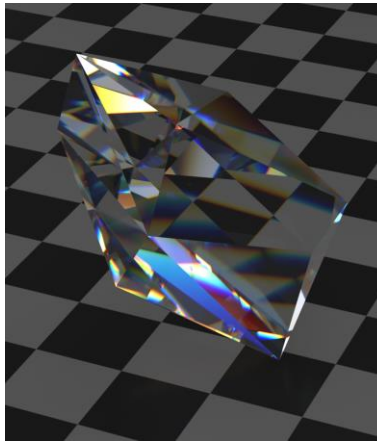
Dispersion scale


0.0


0.5

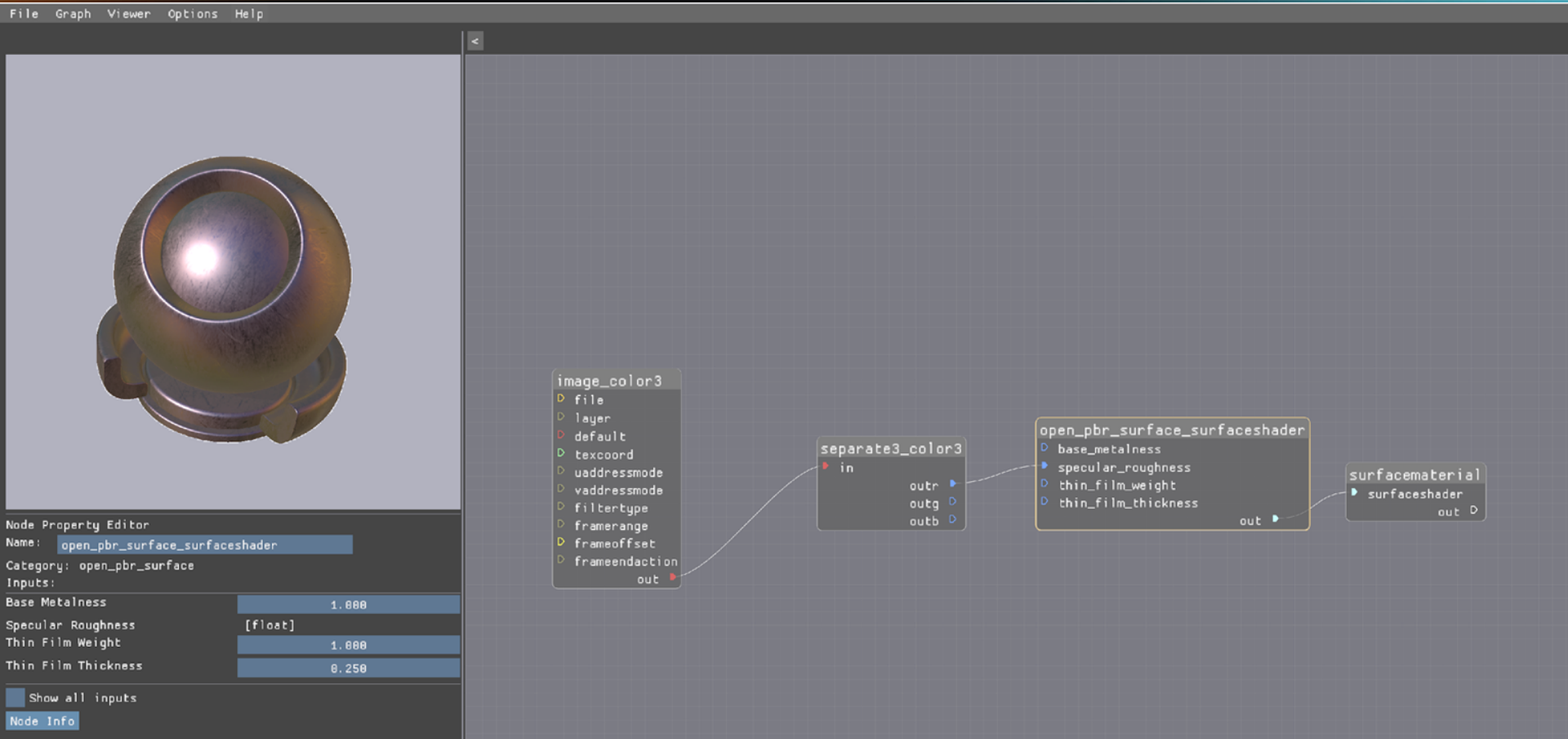

1.0


1.5


2.0

Integrating OpenPBR

Easiest integration: MaterialX



OpenPBR integrations

- Today: MaterialX 1.38.10-openpbr & OpenUSD 24.11
- Future: MaterialX 1.39 & OpenUSD 25.x
- Shader translation graphs from / to Standard Surface

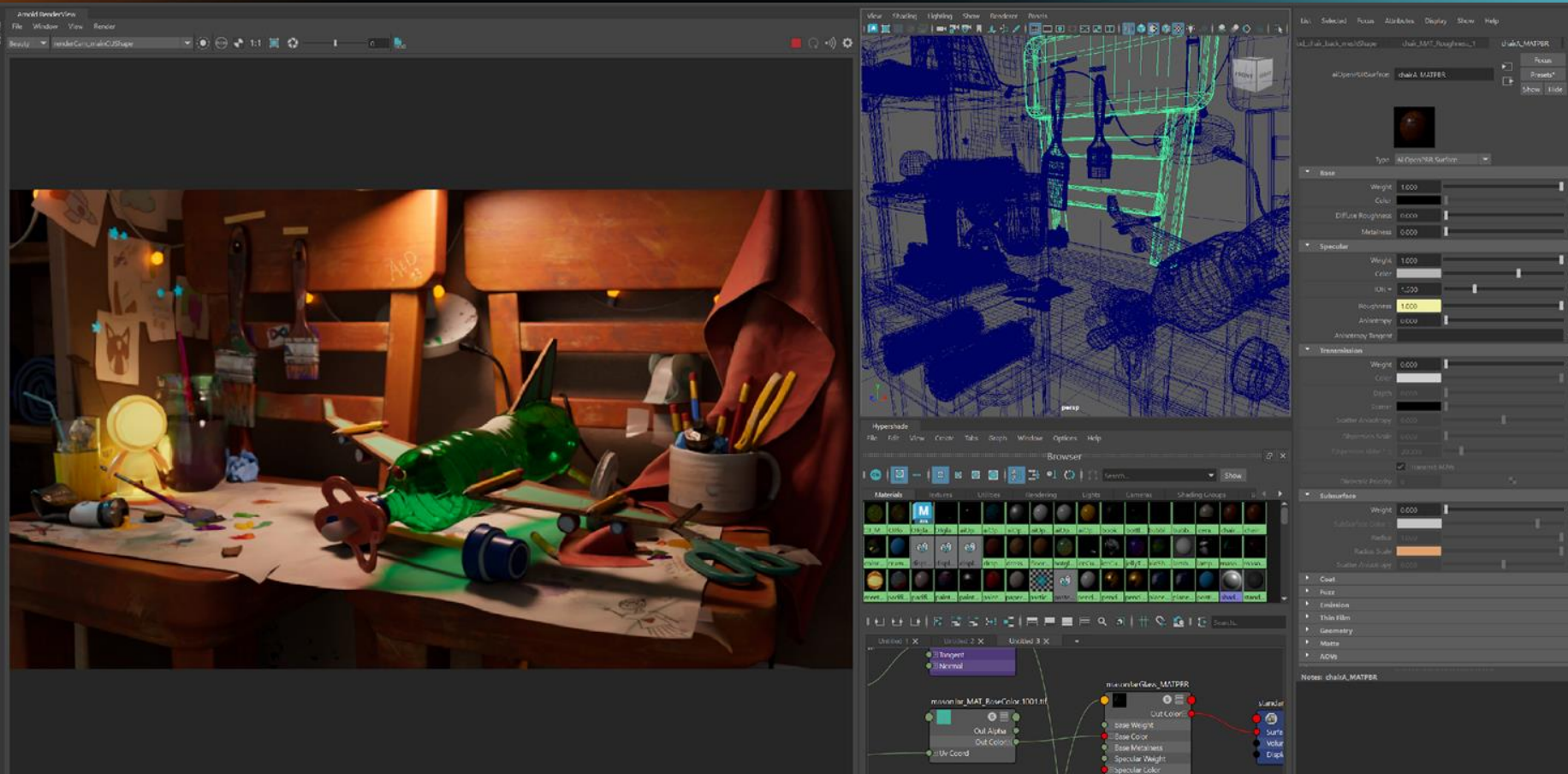
WIP Integrations Showcase

OpenPBR integration: Adobe Substance

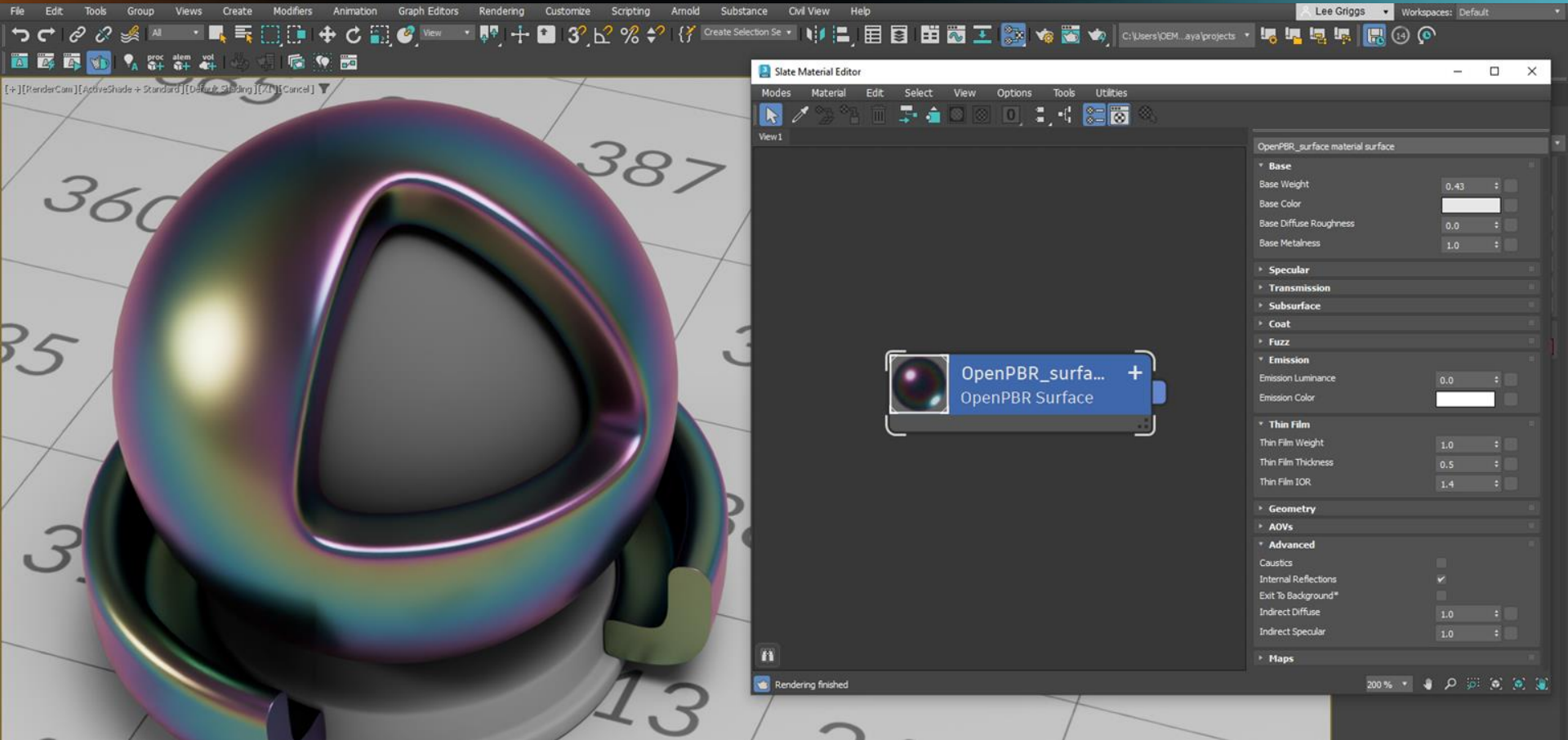
OpenPBR integration: Autodesk Arnold

Artwork by Nikie Monteleone

OpenPBR integration: Autodesk Maya



OpenPBR integration: Autodesk 3ds Max



OpenPBR integration: NVIDIA Omniverse

OpenPBR integration: SideFX Karma

OpenPBR: Current and Future Topics

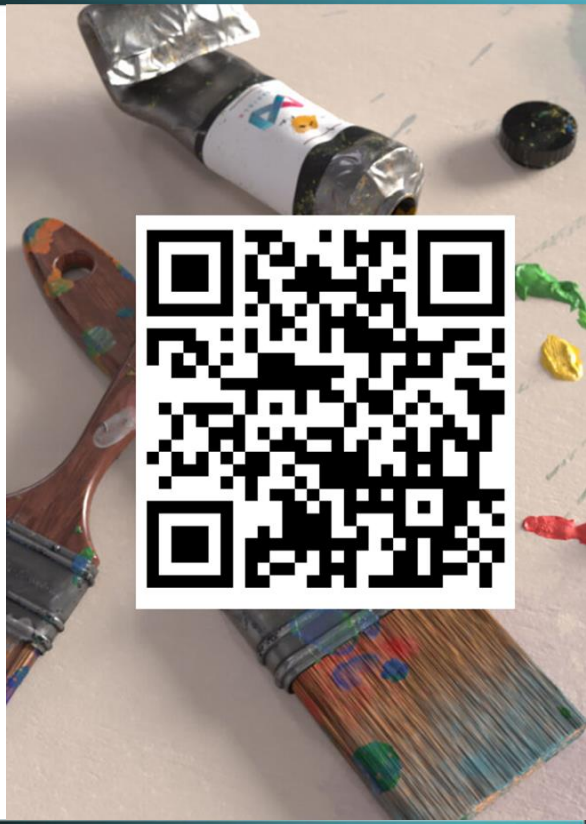
- Retro-reflectivity
- Flakes & glints

- Realtime approximations & authoring
- Implementation compliance & visual fidelity

- OpenPBR beyond surfaces (Volumes, Hair)

Contributions welcome!

github.com/AcademySoftwareFoundation/OpenPBR



Khronos BOFs at SIGGRAPH Asia

Day	Time / Room	Session Title	Standards and Projects
Tuesday 3rd	1:00-2:00PM, G408	Khronos Fast Forward	Vulkan, OpenXR, Slang, ANARI, glTF
Wednesday 4th	1:00-2:00PM, G407	Slang Shading Language	Slang
Wednesday 4th	3:30-4:30PM, G407	Immersive Web with Khronos and W3C	WebGL, WebXR, WebGPU, three.js
Thursday 5th	2:15-3:15PM, G407	OpenXR Update and Roadmap	OpenXR
Thursday 5th	3:30-5:30PM, G407	Vulkan Update and Ecosystem	Vulkan, Vulkan SC, Slang
Friday 6th	1:00-2:00PM, G408	glTF 3D Transmission Format	glTF, VRM Avatar Format

All BOF slides and videos will be uploaded to the
Khronos SIGGRAPH event page

SIGGRAPH ASIA 東京
2024 TOKYO 3–6 DEC

Khronos BOFs

KHRONOS GROUP

Khronos Information
www.khronos.org
memberservices@khronosgroup.org