Vulkanised 2024
The 6th Vulkan Developer Conference
Sunnyvale, California | February 5-7, 2024

Adding Vulkan to Pixar's Hydra Storm Renderer

Henrik Edstrom, Autodesk
Vipul Kapoor, Autodesk
Caroline Lachanski, Pixar
Sébastien Chevrel, Adobe




Agenda

- Intro to Hydra and Storm
  Henrik Edstrom, Autodesk
- Vulkan, Cross-Platform Hydra Graphics and Tooling at Autodesk
  Vipul Kapoor, Autodesk
- Hydra Graphics Interface
  Caroline Lachanski, Pixar
- Fast GPU Interoperability in Complex Hydra Applications
  Sebastien Chevrel, Adobe

Safe Harbor Statement

The presentations during this event may contain forward-looking statements about our outlook, future results and related assumptions, total addressable markets, acquisitions, products and product capabilities, and strategies. These statements reflect our best judgment based on currently known factors. Actual events or results could differ materially. Please refer to our SEC filings, including our most recent Form 10-K and Form 10-Q filings available at www.sec.gov, for important risks and other factors that may cause our actual results to differ from those in our forward-looking statements.

The forward-looking statements made in these presentations are being made as of the time and date of their live presentation. If these presentations are reviewed after the time and date of their live presentation, even if subsequently made available by us, on our website or otherwise, these presentations may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statements.

Statements regarding planned or future development efforts for our products and services are not intended to be a promise or guarantee of future availability of products, services, or features but merely reflect our current plans and based on factors currently known to us. Purchasing decisions should not be made based upon reliance on these statements.

AUTODESK

“Autodesk makes software for people who design and make things”

ARCHITECTURE, ENGINEERING & CONSTRUCTION

R RVT | C C3D | N MAN | I IWX

PRODUCT DESIGN & MANUFACTURING

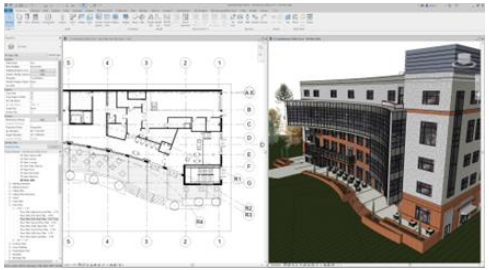
F 360 | I PRO | A CAD | V PRO

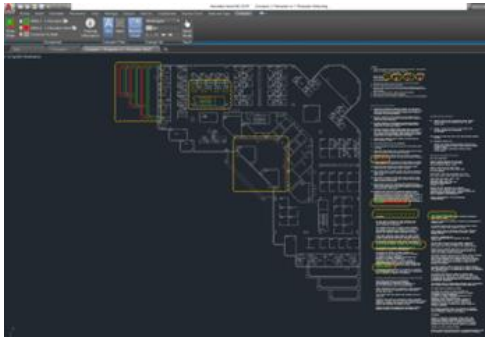
MEDIA & ENTERTAINMENT

M AYA | 3 MAX | A ARN | F FLM

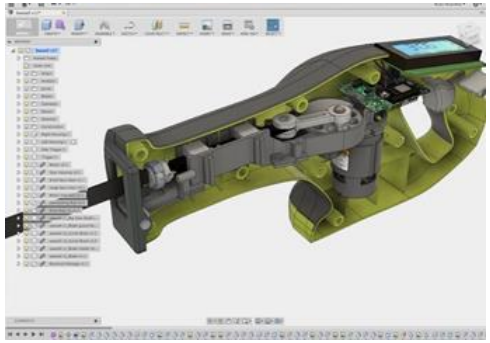
We need a wide range of graphics capabilities

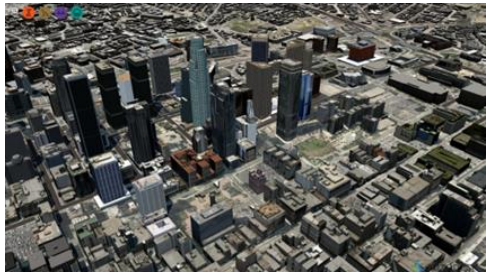
2D & Simple 3D



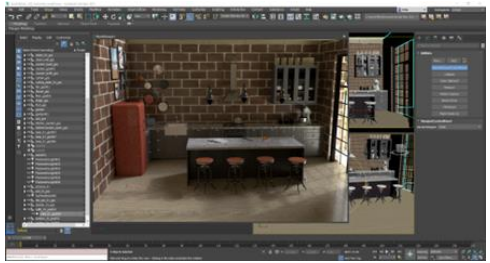


3D Modeling





Realistic Rendering



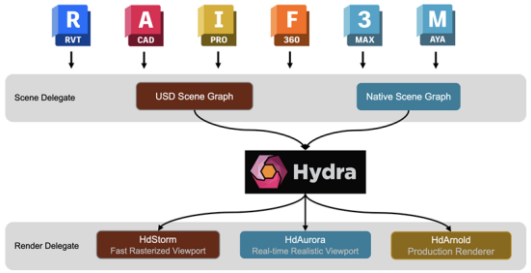
Autodesk's Graphics Objectives

Modern APIs

Open Standards

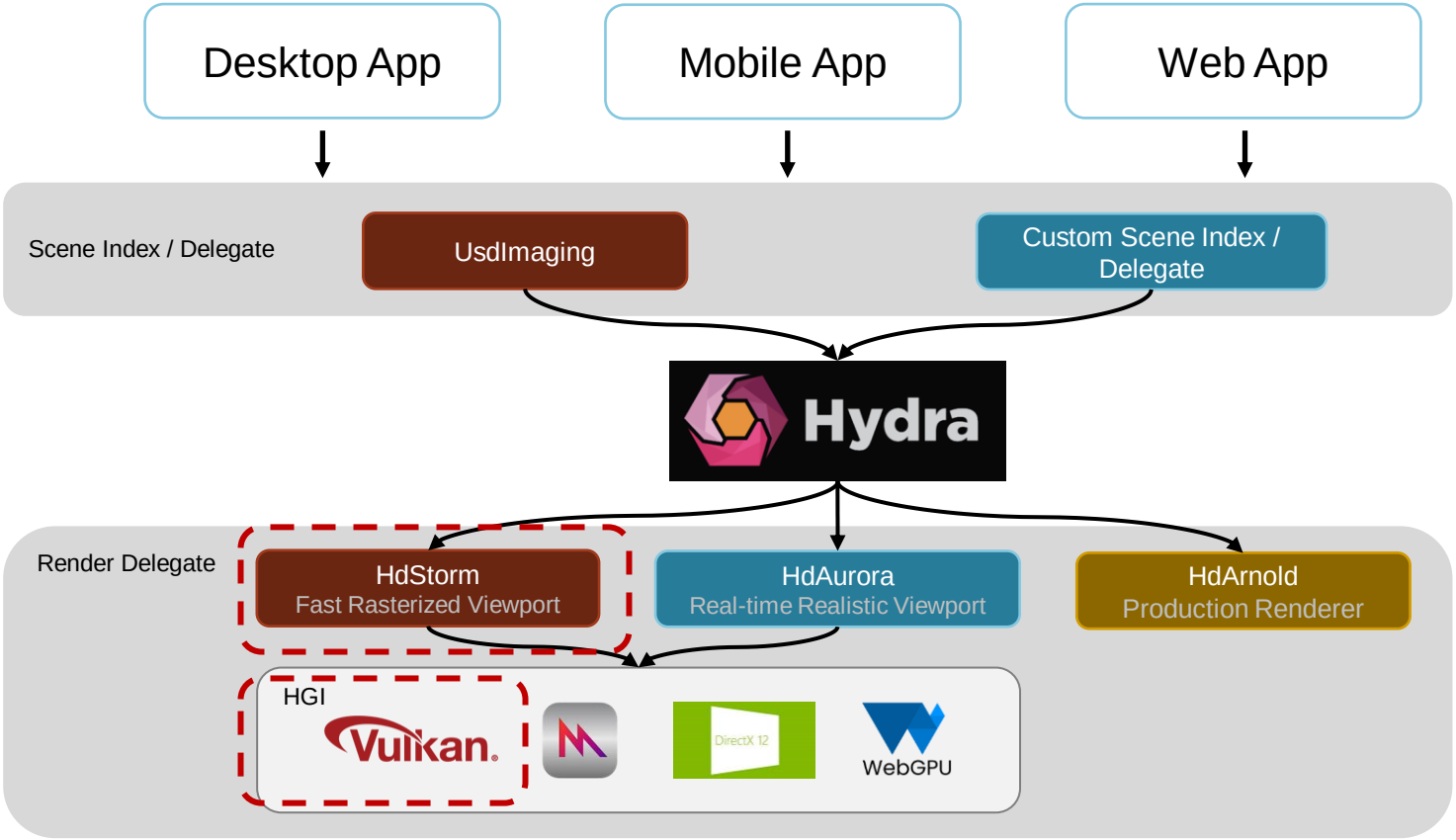
OpenPBR

Decoupled Architecture



Available on Desktop, Mobile, and Web

Hydra and Storm for Autodesk's Viewports



Demands at Autodesk

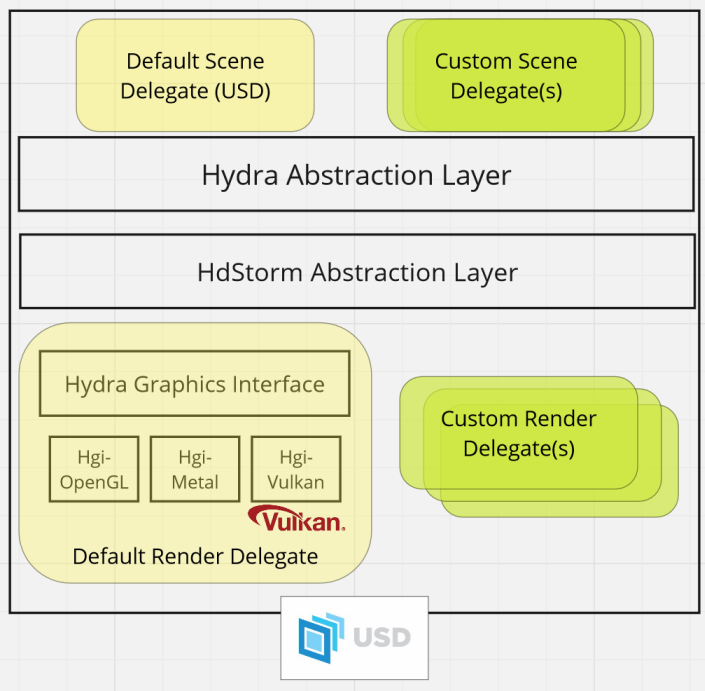
- Large Model Rendering for Industrial Applications
  - 100s GB of 3D Data
  - Billions of Triangles
  - Texture usage varying on Industry
- Cross-Platform Graphics and Visualization
  - iOS and macOS
  - Windows
  - Linux
  - Android
- Exploit Next-Gen Rendering Hardware Infrastructure

Demands at Autodesk

- Large Model Rendering for Industrial Applications
  - 100s GB of 3D Data
  - Billions of Triangles
  - Texture usage varying on Industry
- Cross-Platform Graphics and Visualization
  - iOS and macOS
  - Windows
  - Linux
  - Android
- Exploit Next-Gen Rendering Hardware Infrastructure

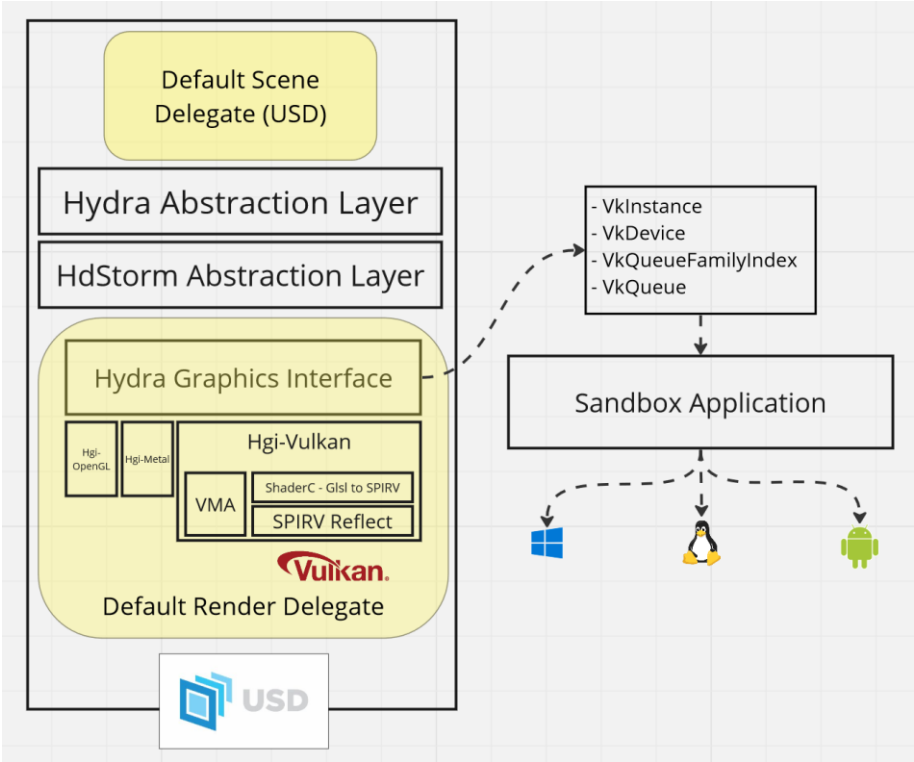
OpenUSD for Autodesk Platform

- Adopt on **OpenUSD,** leverage on **Open-Source** Solutions and Community
- Stabilize and Mature **HgiVulkan** backend for Windows, Linux and Android Platforms
- **Solve** Large Model Viewing challenges for Industrial Applications and add to **OpenUSD** ecosystem



Current State

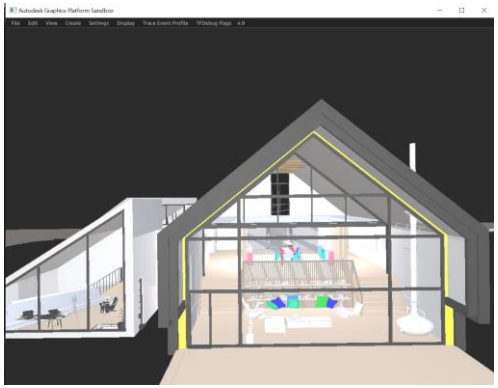
- Composition and Presentation Framework is part of Sandbox Application
- Necessary Vulkan handles are acquired from OpenUSD::HgiVulkan Implementation
- Sandbox Application is capable of rendering on Windows, Linux and Android
- Stable and capable of rendering mid-large models

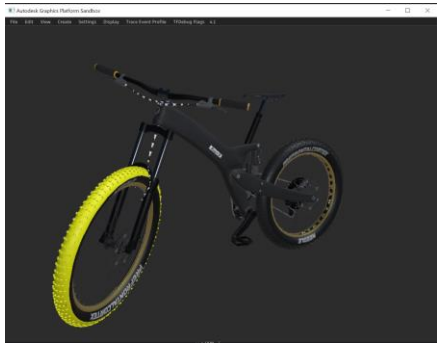


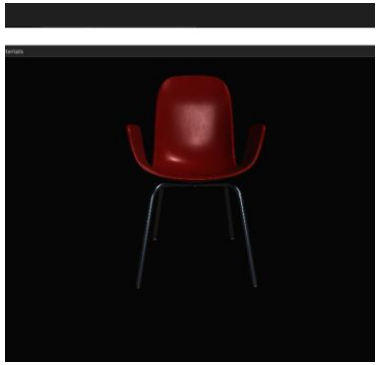
Current state

- Fixed stability issues related to HgiVulkan
- Established build environment for OpenUSD on Android
- Presented progress at Siggraph 2023
- CPU based HLOD implementation on propriety asset formats
- Technical collaboration with Pixar and Adobe
  - Bring HgiVulkan to feature parity with OpenGL and Metal backends
  - Expand and complete unit testing support

Current State







Debugging and Profiling Infrastructure

	Windows	Linux	Android
Correct API usage and stability	Vulkan Debug and Validation Layers		
GPU Frame Debugger	RenderDoc		
	Nvidia Nsight		Snapdragon Profiler
GPU Performance Profiling	Nvidia Nsight AMD RGP	Nvidia Nsight	Snapdragon Profiler
GPU Memory Profiling	VMA Dump VK_EXT_device_memory_report AMD Memory Visualizer	VMA Dump VK_EXT_device_memory_report	VMA Dump VK_EXT_device_memory_report
Cross Platform Capture and Replay	GfxReconstruct		

- Tooling Stability Support from Qualcomm, Nvidia, AMD, Intel and LunarG

Next Steps

- Ensure Vulkan spec compatibility
- Provide Render Graph support
  - Provide context independent pipeline states
  - Efficient layout and synchronization heuristic for Render Graphs
  - Provide async command buffer recording support
- GPU accelerated HLOD and Occlusion Culling
- SwiftShader support for Software Rasterization

Make Anything

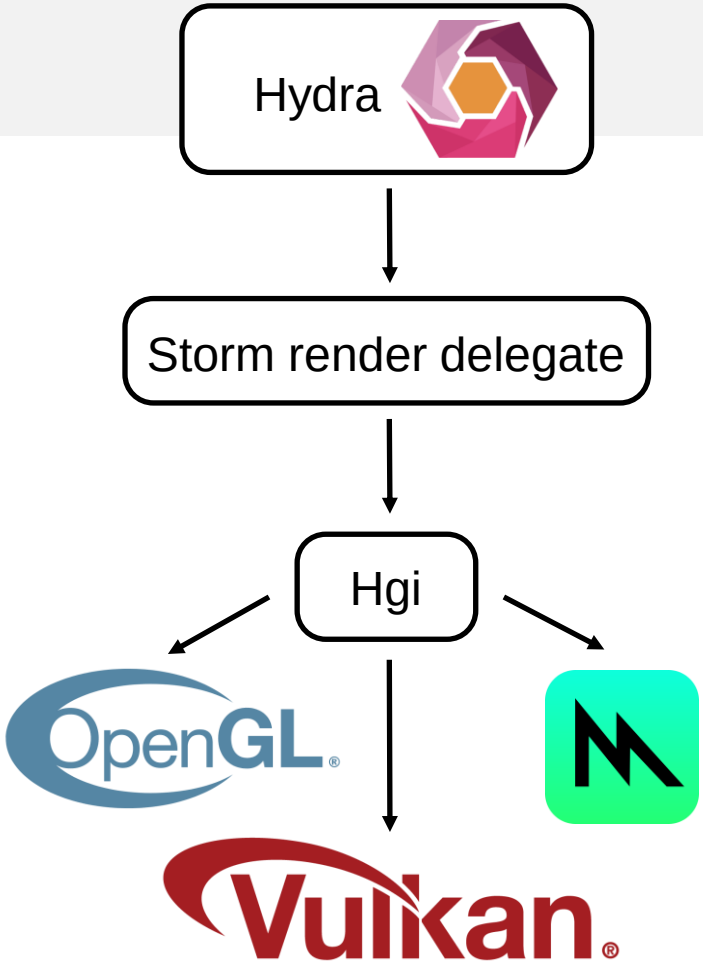
PIXAR
SOFTWARE R&D

Hydra Graphics Interface

Caroline Lachanski
Software Engineer
clachanski@pixar.com

HYDRA GRAPHICS INTERFACE (HGI)

- **Hydra** originally an OpenGL-based renderer
  - Meant as ground truth visualization for USD
- OpenGL render delegate component became **Storm**
  - Used in apps like usdview and Presto
- **Hgi** is graphics API abstraction layer
  - HgiGL currently used internally
  - HgiMetal result of collaboration with Apple
  - HgiVulkan now the focus
- Pixar goal to shift from OpenGL to Vulkan internally
- How to write renderer independent of graphics API without disrupting users?



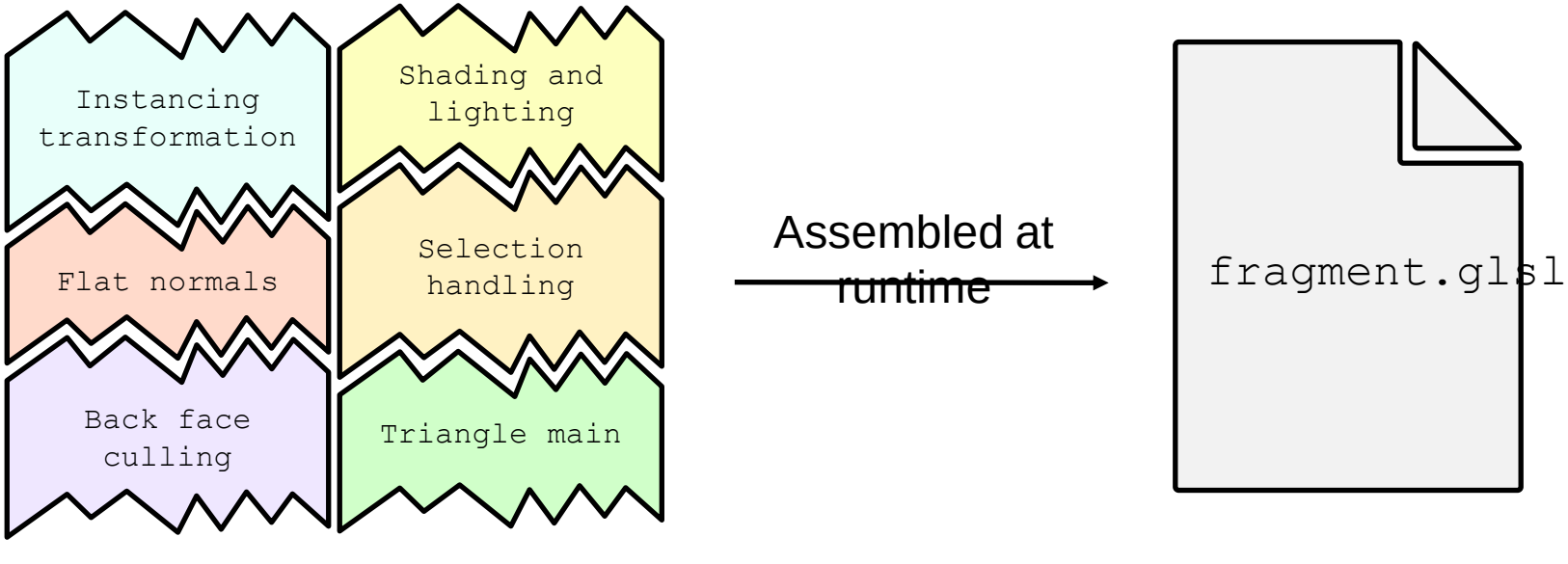
TRANSITIONING TO HGI

- Storm written with OpenGL in mind, Hgi written with modern APIs in mind
- OpenGL state machine to explicit pipeline
  - HgiVulkan: commands are recorded in command buffer → command buffer is submitted
  - HgiGL: functions are accumulated in stack → GL state captured → functions (GL calls) called → GL state restored
- Lingering GL code and GL concepts
- Vulkan validation layers

```
HgiGLOpsFn
HgiGLOps::SetViewport(GfVec4i const& vp)
{
    return [vp] {
        glViewport(vp[0], vp[1], vp[2], vp[3]);
    };
}
```

GLSLFX

- GLSLFX is domain language for defining shaders
  - Defines imports, configurations, and shading code snippets



Multiple shader snippets

Completed shader to compile

HOW TO WRITE SHADER RESOURCES?

- GLSL is original shading language of choice

- Shader resources originally hardcoded in shader snippets
  - Shader stage inputs and outputs
  - Texture and data buffer declarations
  - Interpolation modifiers
  - Location and binding indices
  - Other layout qualifiers (e.g. “early_fragment_tests” for the FS)

- Wanted shader language-independent way of declaring shader’s resources and resource layout

SHADER RESOURCE LAYOUTS

- Extended GLSLFX to include “layout” section
  - Corresponds to “glsl” section of same name
  - Processed at runtime to fill descriptors, which are processed by shadergen to produce shading code

```
-------------------------------------------------------------------
-- glsl Mesh.Vertex

out VertexData
{
    vec4 Peye;
    vec3 Neye;
} outData;

void main(void)
{
            . . .
    outData.Peye = vec4(GetWorldToViewMatrix() * transform * point);
    outData.Neye = GetNormal();
    gl_Position = vec4(GetProjectionMatrix() * outData.Peye);
}
```

Before resource layouts

```
-------------------------------------------------------------------
-- layout Mesh.Vertex

[
    ["out block", "VertexData", "outData",
        ["vec4", "Peye"],
        ["vec3", "Neye"]
    ]
]

-------------------------------------------------------------------
-- glsl Mesh.Vertex

void main(void)
{
    . . .
    outData.Peye = vec4(GetWorldToViewMatrix() * transform * point);
    outData.Neye = GetNormal();
    gl_Position = vec4(GetProjectionMatrix() * outData.Peye);
}
```

With resource layouts

SHADER GENERATION

- API-specific shader creation is handled with Hgi shadergen system
- Set of classes that generate API-specific shading code
- Fed by descriptors
  - `HgiShaderFunctionTextureDesc`, `HgiShaderFunctionBufferDesc`, `HgiShaderFunctionFragmentDesc`, etc.
- Behind abstraction layer, we can deal with resource declaration, builtin function and keyword name differences, extension names, etc.

```
struct HgiShaderFunctionTextureDesc
{
    std::string nameInShader;
    uint32_t dimensions;
    uint32_t bindIndex;
    size_t arraySize;
    bool writable;
        . . .
};
```

SHADER GENERATION EXAMPLE

- OpenGL GLSL builtin vertex stage input variables `gl_VertexID` and `gl_InstanceID`
- Vulkan GLSL extension replaces* those with `gl_VertexIndex` and `gl_InstanceIndex`
- We want shader writers to be able to use these variables without having to think about the backend differences
- Map variables "`hd_VertexID`" and "`hd_InstanceID`" to a non-backend-specific role
- Each backend's shadergen emits code defining `hd_VertexID and hd_InstanceID` to correct thing

OpenGL GLSL:

```
uint hd_VertexID = gl_VertexId;
uint hd_InstanceID = gl_InstanceId;
```

Vulkan GLSL:

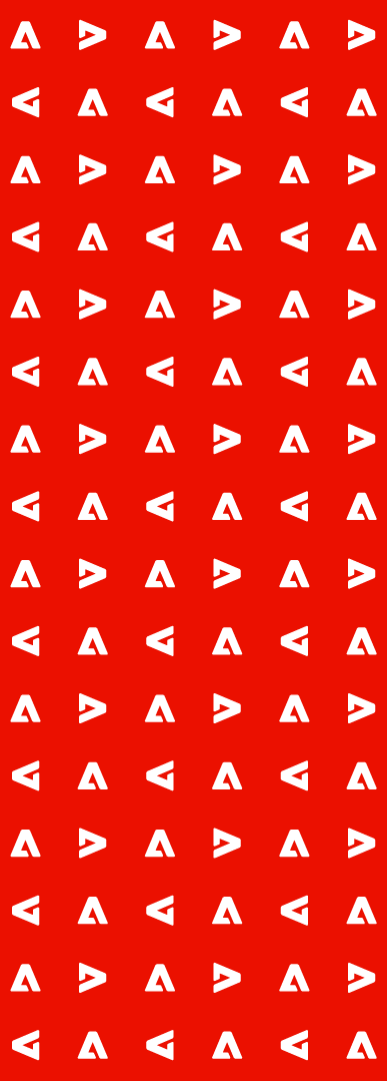
```
uint hd_VertexID = gl_VertexIndex;
uint hd_InstanceID = gl_InstanceIndex;
```

Metal shading language:

```
uint hd_VertexID[[vertex_id]],
uint hd_InstanceID[[instance_id]],
```

Fast GPU interoperability in complex Hydra applications

Sébastien Chevrel – Senior 3D Graphics Engineer



Problem statement

Adobe is building a framework to develop next generation 3D applications based on USD.

Performant applications require interoperability of GPU resources between components:

- **Renderer**: custom photorealistic renderer using low level graphics API that does not depend on Hgi
- **Hydra delegate:** for the renderer that interfaces with USD.
- **3D viewport UI** : selection, gizmos, manipulators, color management, etc.
- **2D traditional UI:** GPU accelerated compositing and traditional UI rendering (QtQuick, imGui)

These components need to share framebuffers / AOVs on the GPU in different Vulkan instances, avoiding CPU copies and coordinate lifetime without necessarily knowing about each others.

Sharing buffers between Vulkan instances

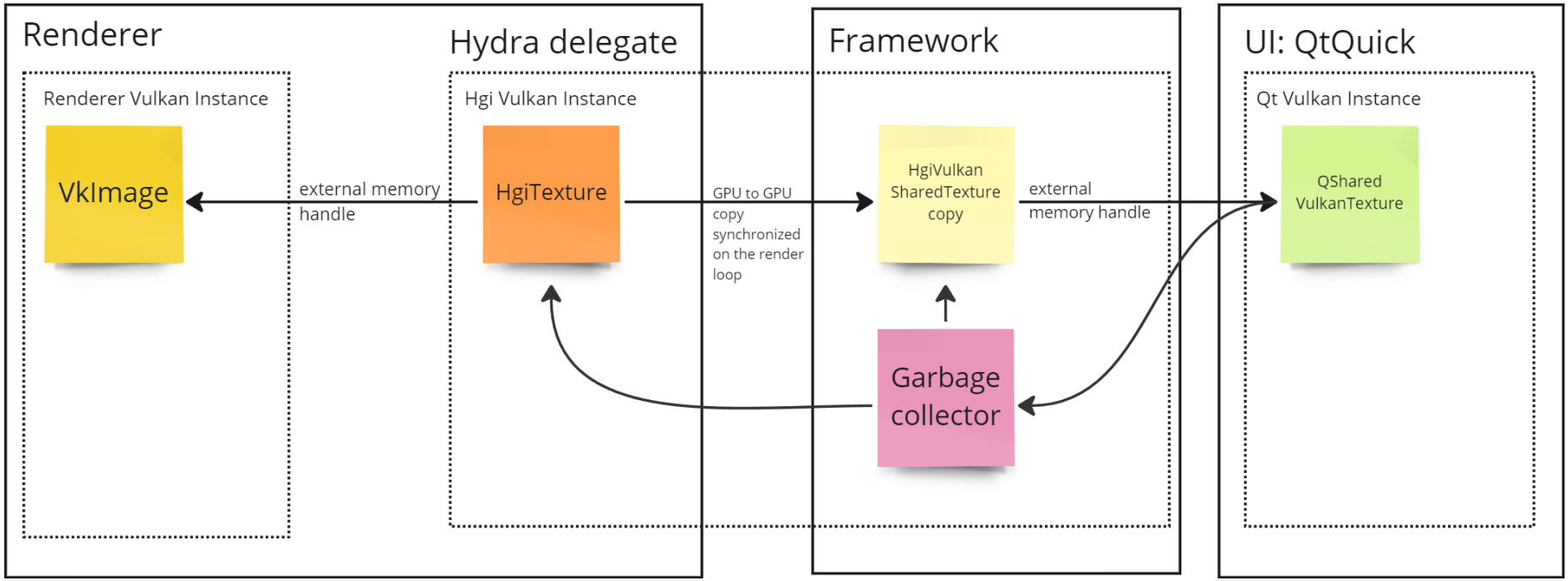
External memory extension part of Vulkan core since 1.1 : VK_KHR_external_memory

- Resources are created with tagging for external memory sharing: dedicated block or dedicated pool if using VMA.
- An opaque handle can be exported (Windows Handle, or Posix FD).
- Along with some information about the memory pool size, allocation offset and memory type, other instances can use this handle to recreate resources pointing to the same shared memory.
- Some synchronization may be necessary, or at least making copies of the framebuffers on the GPU to avoid tearing or contention.
- This mechanism is normally used to share across graphics APIs (OpenGL, DirectX, Vulkan). The Vulkan-Vulkan case is not widely used or documented.

Lifetime management

- Different modules need to use resources asynchronously in different threads at different times.
- Because of the shared memory, modules that own resources cannot release them directly when they are done using them.
- There is the need for a global garbage collection type system where the last user of a resource (typically the UI) can release it when it is done using it.
- Only then the various modules are notified and can release the actual resources.
- Making copies can help shorten the chain of lifetime dependency.

Example application structure



USD support for interoperability

- *Hgi* does not currently support external memory. *Hgi* interop is slow via CPU copy.
- Our current solution uses a modified version of USD/*Hgi* that enables us to wrap existing GPU resources inside a *HgiTexture*.
- This allows us to use the same mechanism as *HdStorm* to pass GPU buffers to *UsdImagingGl*.
- Adobe uses a graphics API abstraction layer similar to *Hgi* internally.
- We’re moving away from using *UsdImagingGl* and are now using our own low-level implementation to interface with the Hydra delegate using our internal abstraction layer and bypassing *Hgi*.
- Regardless, it may be valuable for USD to support external memory: create resources that can be shared, or import existing resources created in a different instance/API into *Hgi*.

Compatibility and issues

- We've heard in the graphics dev community that interoperability of graphics API is fragile and prone to breaking on some platforms, hardware, or driver versions.
- In our experience this may be an outdated belief, we have yet to encounter any issues, it has worked reliably for us on Windows and Linux with high-end GPUs typical in 3D/VFX.
- But this new platform has only been tested internally and issues may arise when this is widely deployed with the public.


Adobe

Links and references

- OpenUSD homepage: https://openusd.org/
- OpenUSD GitHub: https://github.com/PixarAnimationStudios/OpenUSD
- Hydra intro: https://openusd.org/files/Siggraph2019_Hydra.pdf
- Alliance for OpenUSD: https://aousd.org/
- AOUSD Forum: https://forum.aousd.org/
- Vulkan SIGGRAPH BOF 2023 (recording on YouTube):
  - *"Vulkan and Open Source Graphics at Autodesk"*
  - *"Vulkan for Cross-Platform Viewing of Large AEC Models"*
  - *"Vulkan Ray Tracing in Aurora: An Open Source Real-Time Path Tracer"*
- Autodesk Graphics Platform team contact: agp@autodesk.com
- Aurora source code: https://github.com/Autodesk/Aurora
- Aurora contact: aurora@autodesk.com
- Autodesk Open Source landing page: https://opensource.autodesk.com/