Vulkanised 2026
The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

How Autodesk VRED is Bringing Vulkan to the Automotive Industry

Oscar Sebio Cajaraville, Autodesk

What is VRED?

The Vulkan Project

- Promises
- Execution
- Raytracing
- Performance

The Vulkan Experience

- Pipelines
- Device Groups
- Material Params

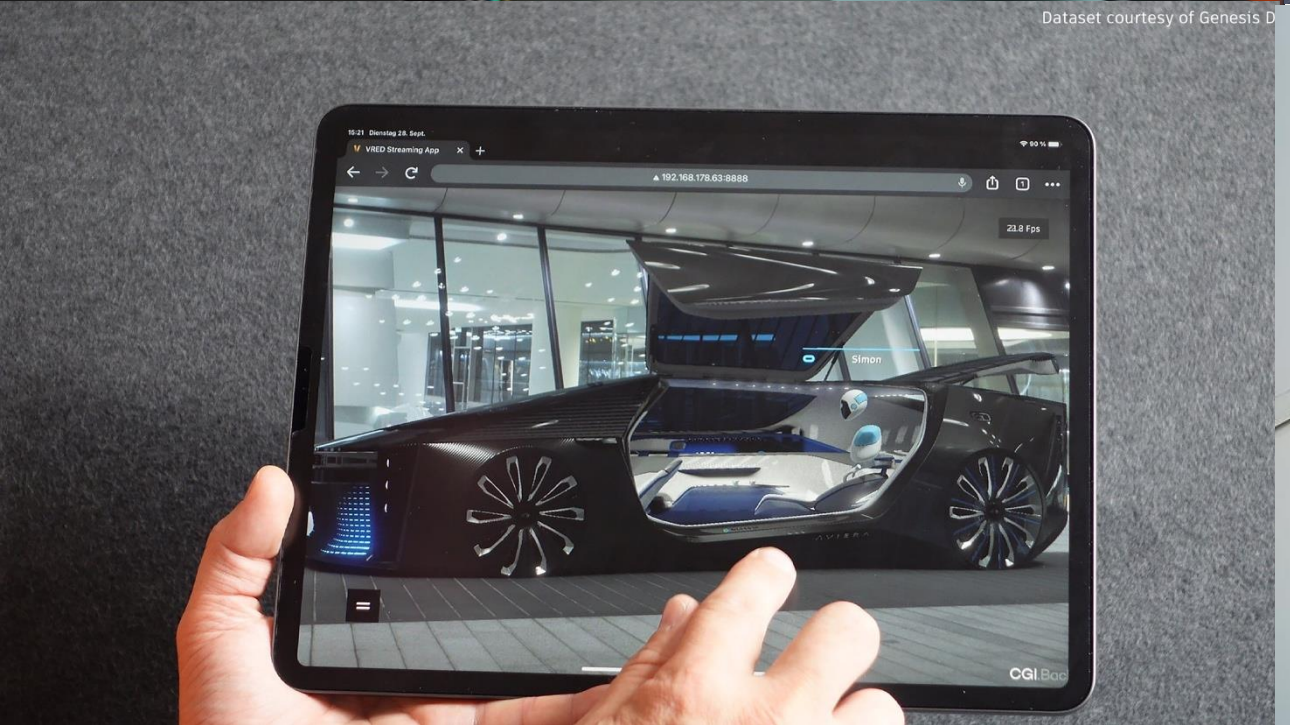
Dataset courtesy of Genesis Design

Average FPS 76.1

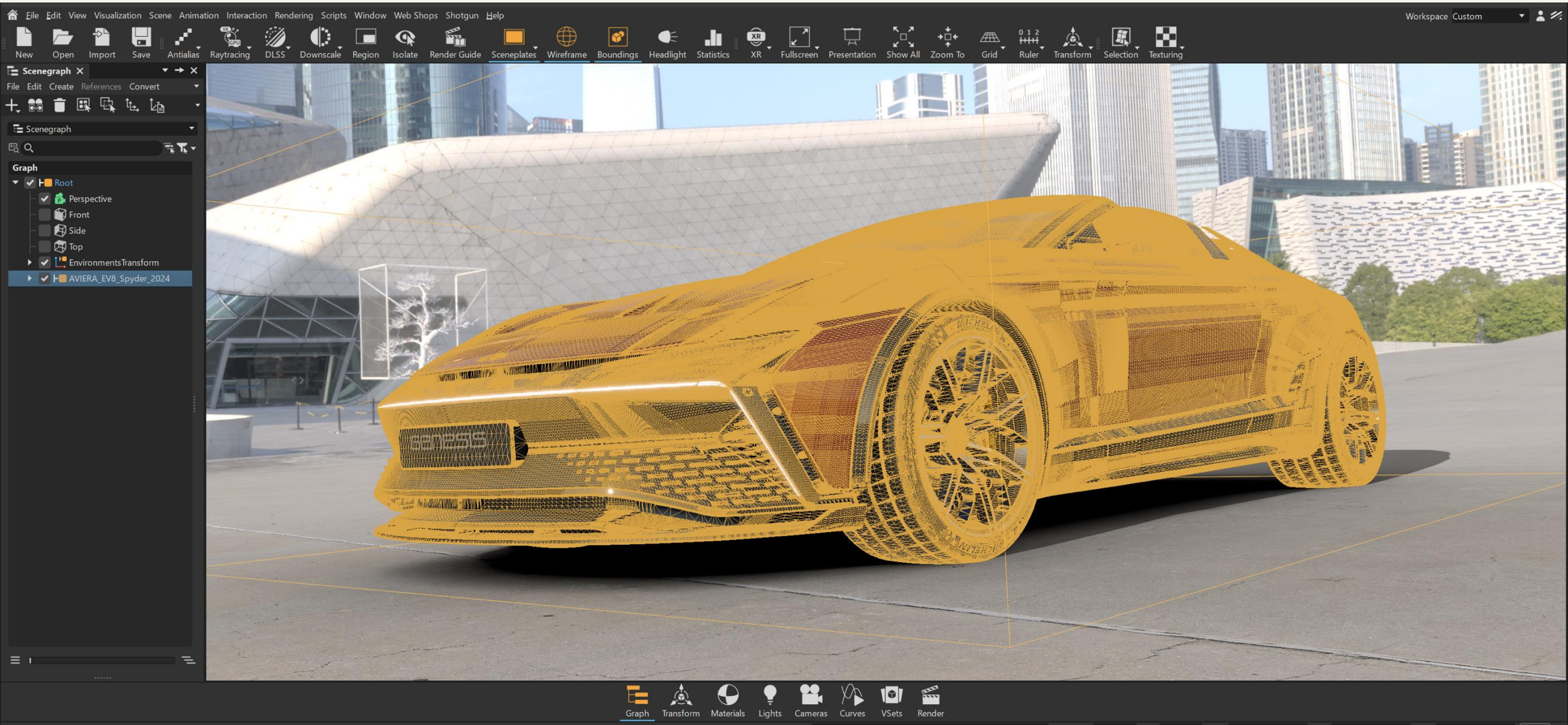
BACK

DLSS ON

Dataset courtesy of Genesis D

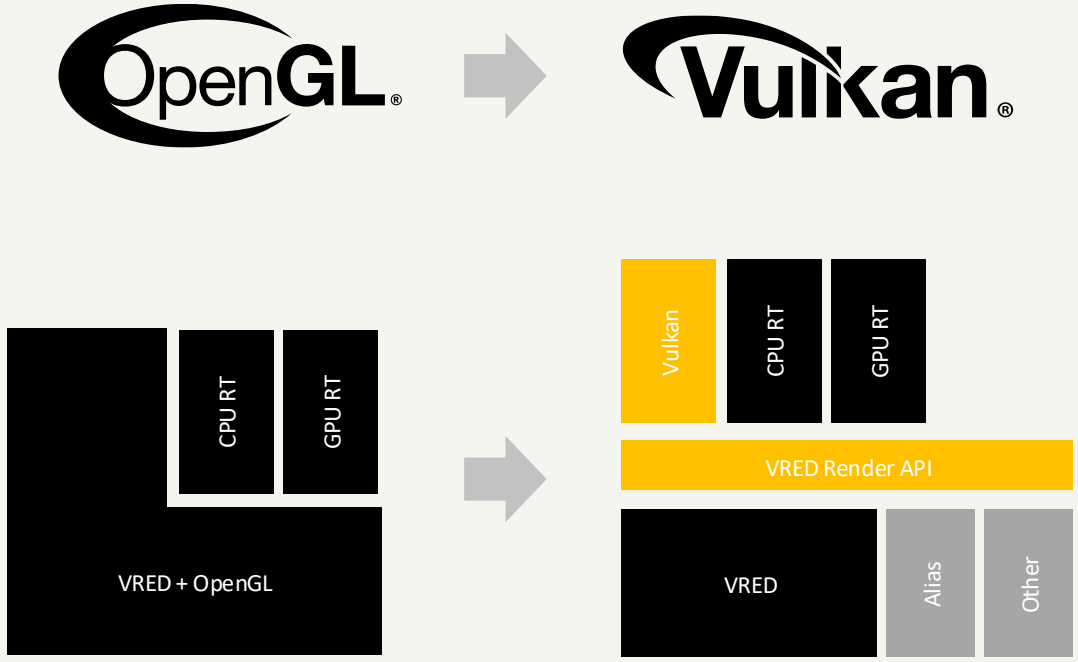


What is VRED?


File Edit View Visualization Scene Animation Interaction Rendering Scripts Window Web Shops Shotgun Help
Workspace Custom
New Open Import Save Antialias Raytracing DLSS Downscale Region Isolate Render Guide Sceneplates Wireframe Boundings Headlight Statistics XR Fullscreen Presentation Show All Zoom To Grid Ruler Transform Selection Texturing
Scenegraph
File Edit Create References Convert
Scenegraph
Graph
Root
Perspective
Front
Side
Top
EnvironmentsTransform
AVIERA_EV8_Spyder_2024
Graph Transform Materials Lights Cameras Curves VSets Render

AUTODESK

The Vulkan Project



Better, Faster, Prettier

Support multiple applications

More FPS for the same quality

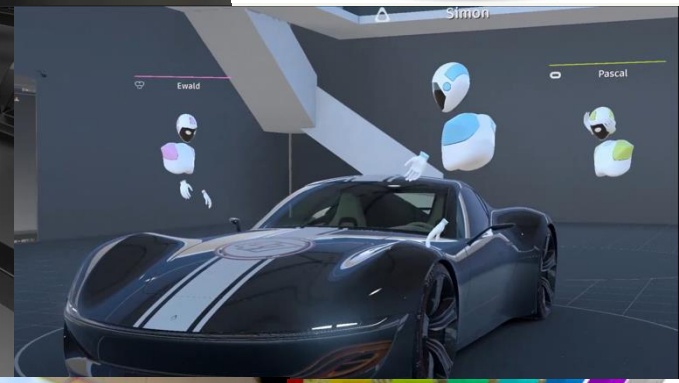
Raytracing!

TODO list

- 25+ materials
- Light sources
- Order Independent Transparency
- Volume rendering
- Post-process
- Gizmos + other auxiliary geometry
- XR support
- Multi-GPU support
- Display Cluster (wall displays, CAVE setups)
- Raytraced reflections and environment shadows

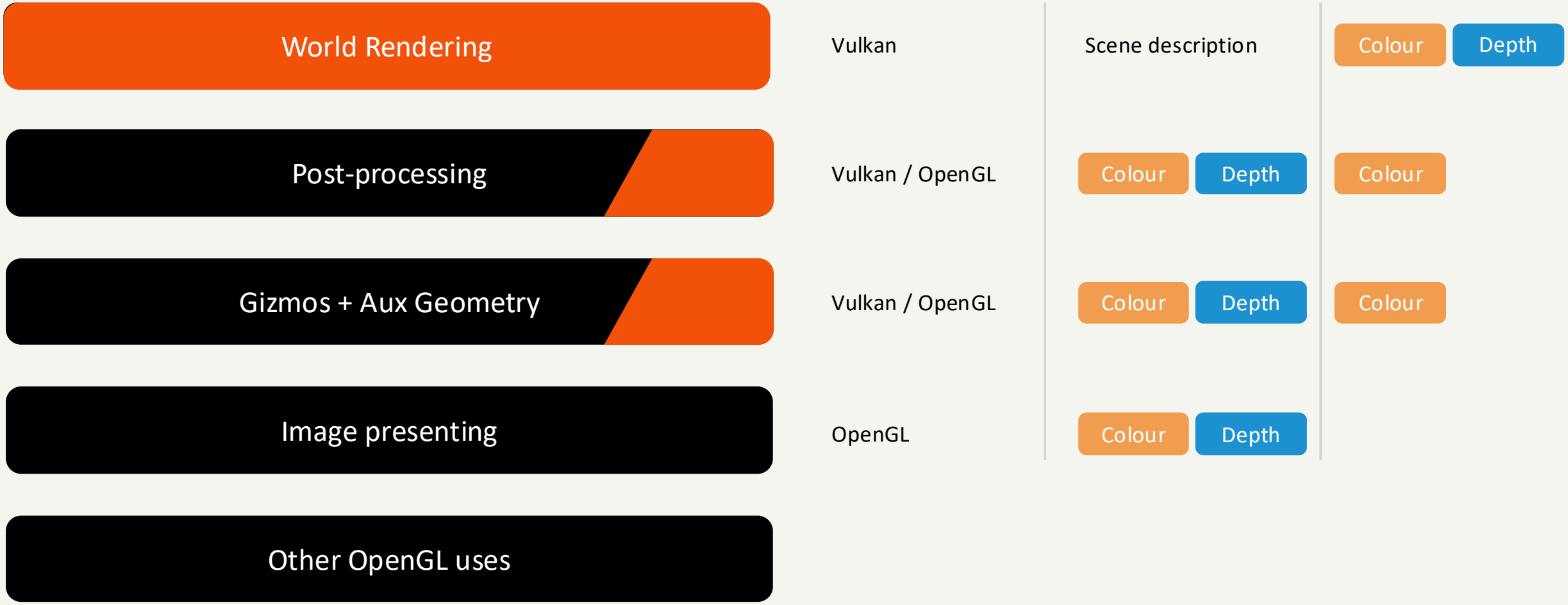





Vulkan-OpenGL interop

VK_KHR_external_memory, VK_KHR_external_semaphore



Raytracing with Vulkan

GPU Raytracer: Global Illumination

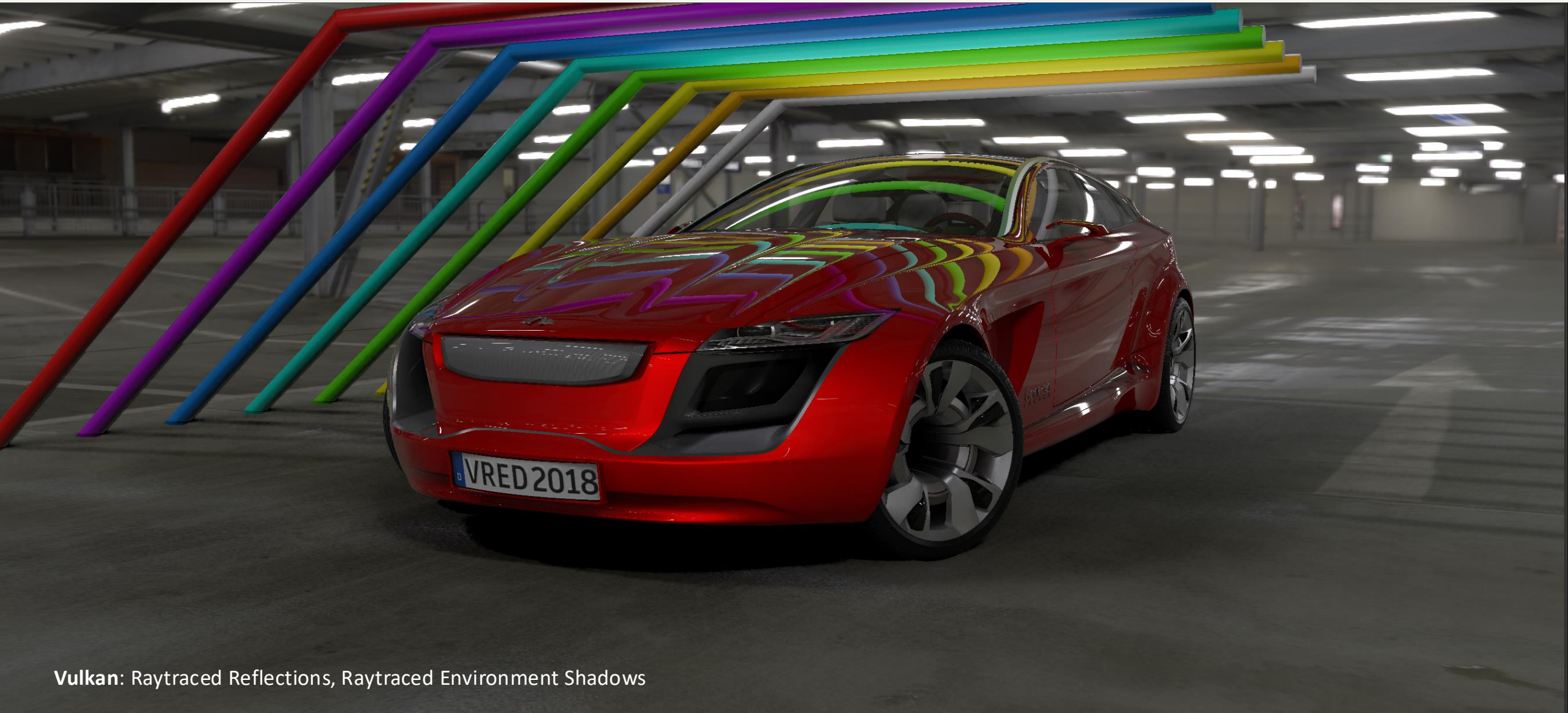
Raytracing with Vulkan

Vulkan: No Raytracing, Baked Shadows

Raytracing with Vulkan

Vulkan: No Raytracing, No baking

Raytracing with Vulkan



Vulkan: Raytraced Reflections, Raytraced Environment Shadows

Raytracing with Vulkan

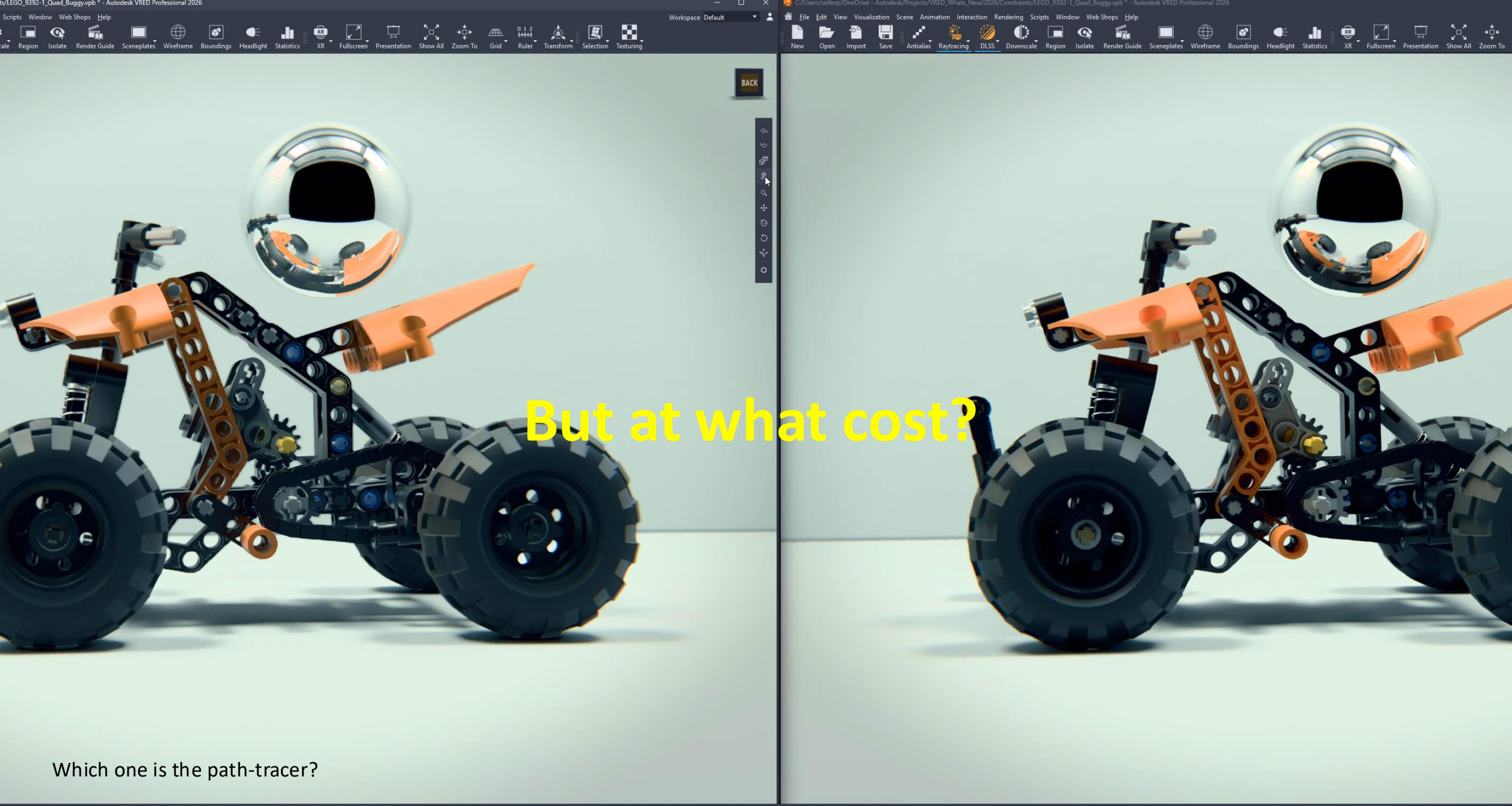
Vulkan: No Raytracing, Baked Shadows

Raytracing with Vulkan

Vulkan: Raytraced Reflections, Raytraced Environment Shadows

Raytracing with Vulkan

GPU Raytracer: Global Illumination

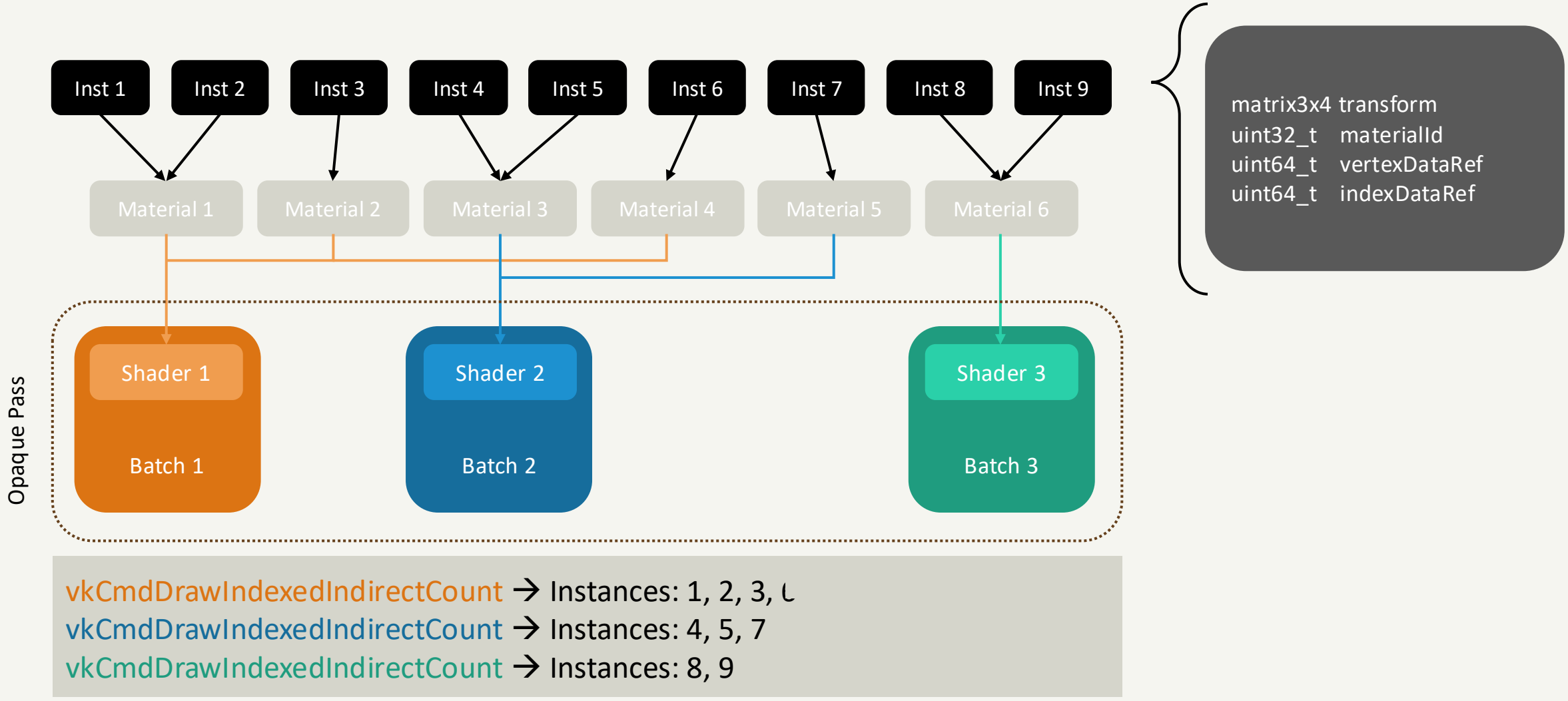


Which one is the path-tracer?

Video courtesy of Pascal Seifert


Average FPS 10.6
Nodes culltested 234,668
Nodes culled 35,459
Occlusion culling On
Occlusion tests 198,985
Occlusion culled 163,125
Nodes drawn 73,709
Transparent Nodes drawn 1,960
Triangles drawn 395,648,049
Lines drawn 0
Points drawn 0
Lights active 0
Vertices transformed 1,186,944,147
Material changes 487
Matrix changes 72,309
Shadow map changes 41
OpenGL
230,000 instances
400,000,000 triangles drawn
70,000 glDrawRangeElements()
500 shader changes
~10 fps
Dataset courtesy of Mastenbroek

Draw-call batching



vkCmdDrawIndexedIndirectCount → Instances: 1, 2, 3, ι
vkCmdDrawIndexedIndirectCount → Instances: 4, 5, 7
vkCmdDrawIndexedIndirectCount → Instances: 8, 9


Average FPS 18.3
Vulkan
~18 fps


Average FPS 18.3


Average FPS 25.1
RIGHT
FRONT
Our Vulkan Odyssey
Dataset courtesy of Mastenbroek

Expectation vs Reality: Pipelines

Expectation

Static render pipelines and render-passes provides a performance gain.

Reality

Render-passes are only really useful in tiled GPUs, which we do not support.

Not using dynamic rendering complicates pipeline creation and we do not have any data to support any potential performance gains from it.

Message to past self

Take advantage of the dynamic rendering!

Expectation vs Reality: Device Groups

Expectation

Work with devices groups should be easy for rendering two eyes at the same time with different cameras…

Reality

No vendor information on what setups will produce device groups.

No obvious way to transfer memory from one device to another, no examples either.

Beware of uncommonly used extensions, as we encountered driver bugs.

No support from debugging and profiling tools.

```
typedef struct VkCommandBufferSubmitInfo {
    VkStructureType sType;
    const void* pNext;
    VkCommandBuffer commandBuffer;
    uint32_t deviceMask;
} VkCommandBufferSubmitInfo;
```

Message to past self

Sometimes is ok to blame the driver

deviceMask is a bitmask indicating which devices in a device group execute the command buffer. A deviceMask of 0 is equivalent to setting all bits corresponding to valid devices in the group to 1.

Expectation vs Reality: Material Parameters

Expectation

We can access material parameters given the material id/index.

All material params needs to be the exact same size so we can index them or split them by material type (which complicates indexing).

Reality

You have Buffer Device Address!

We could access the material data directly by its address, instead of indexing it from a single buffer.

As some “components” of the material parameters are shared for multiple materials, we could reduce shader complexity by passing the address of the component.

Instance

```
matrix3x4 transform;
uint32_t  materialId;
uint64_t  vertexDataRef;
uint64_t  indexDataRef;
```

PlasticMaterialParams

```
struct DiffuseComponent diff;
struct TransparencyComponent trans;
struct PlasticMaterialParams plastic;
```

MetalMaterialParams

```
struct DiffuseComponent diff;
struct TransparencyComponent trans;
struct MetalMaterialParams metal;
int _pad[1234];
```

Message to past self

Research extensions.

Figure out how they can help you.

Verify that you are reading the most up t

documentation.

Questions?

Special thanks to the VRED Team

Particularly the Rendering Workgroup

DLSS ON

Make Anything