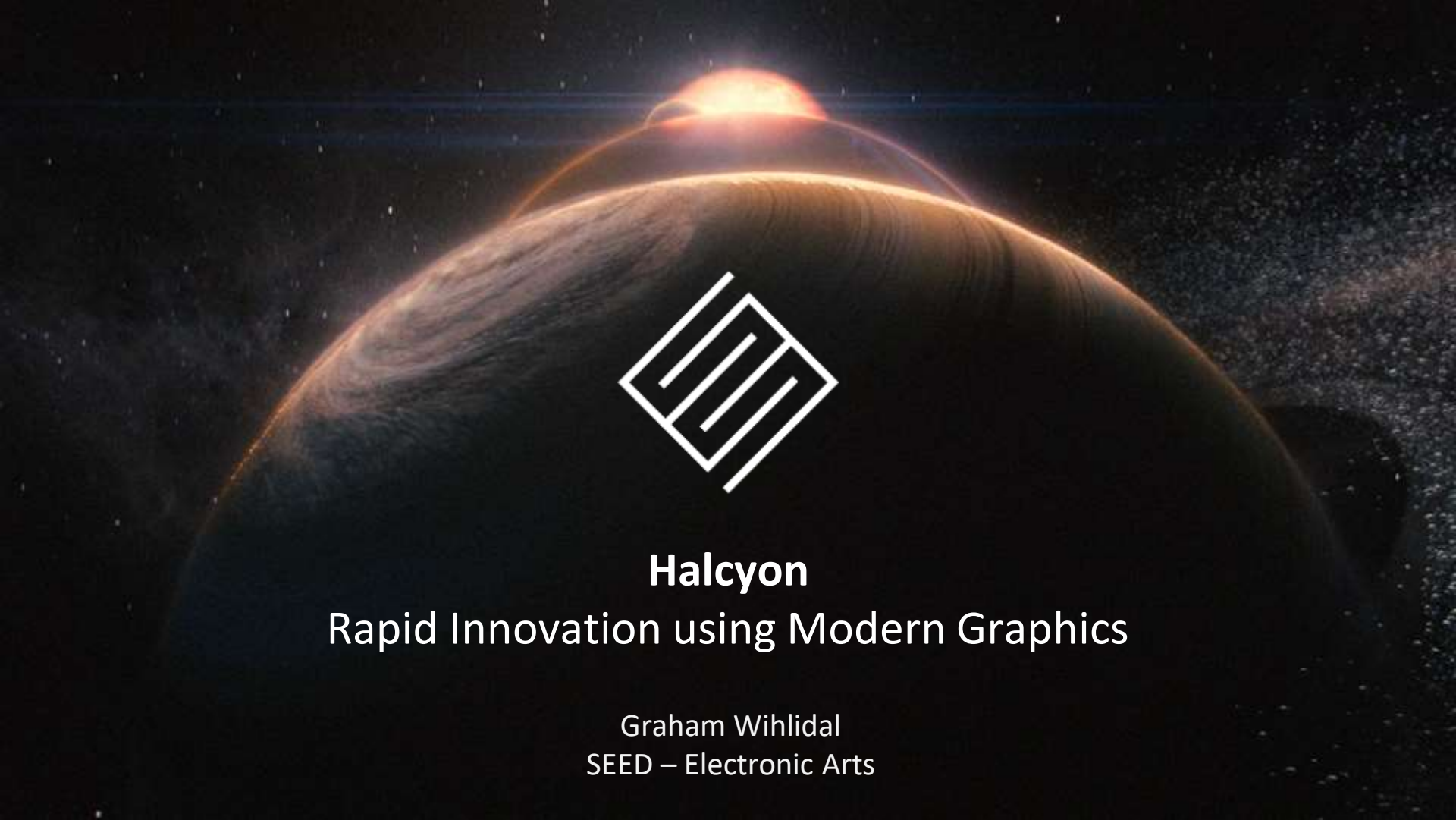


Halcyon

Rapid Innovation using Modern Graphics

Graham Wihlidal
SEED – Electronic Arts

SEED

Disruptive Technology?

- Technology that significantly alters the way your business operates.

- Often forces companies to change their business for fear of losing market share or becoming irrelevant.

Embrace Disruption

- Maintain existing business
  - Safe bets, manage risk
  - Also important!
- Avoid “horse blinders”
  - Evolving market
  - Evolving users
  - Evolving technology

Embrace Disruption

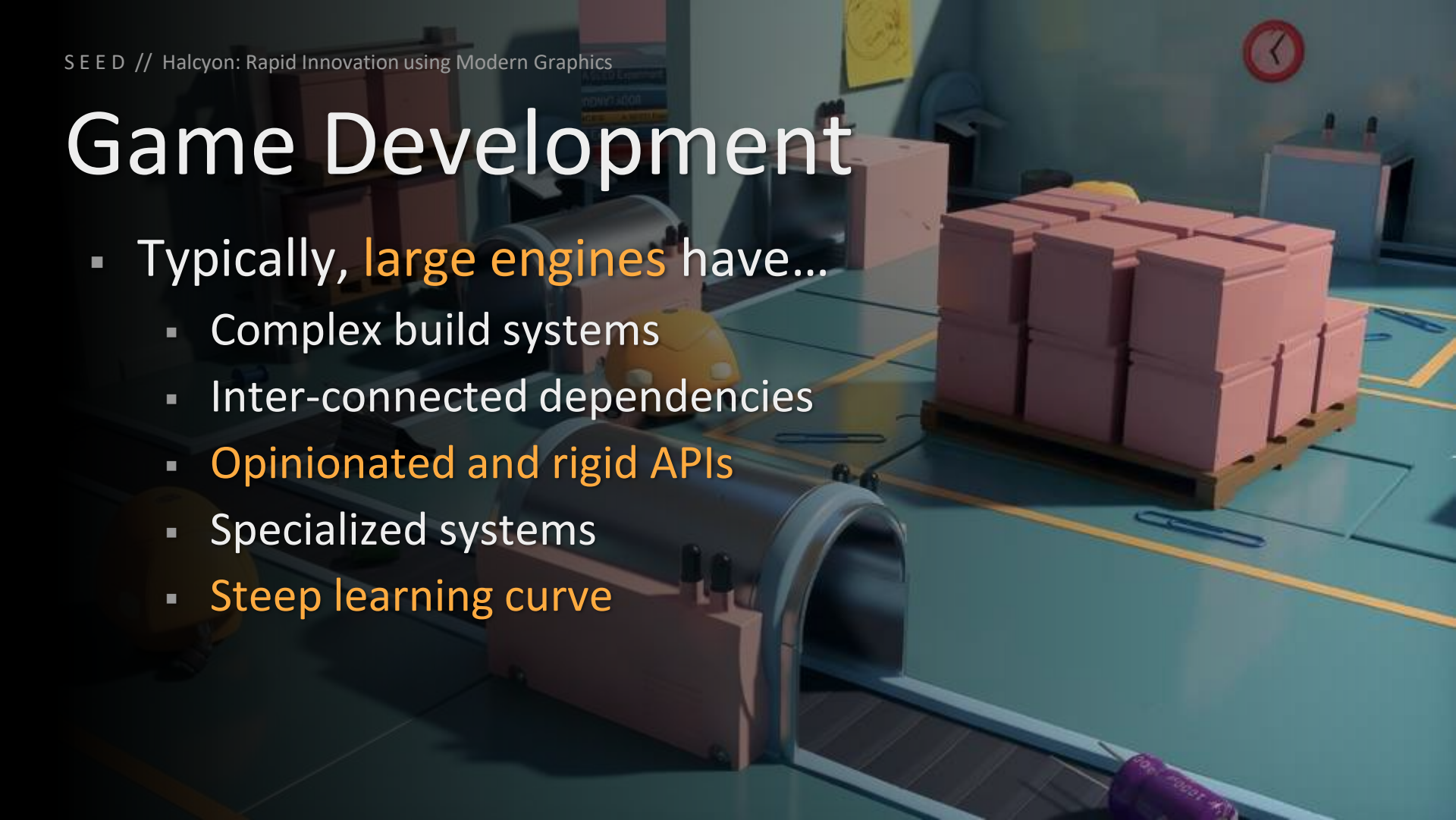
- Innovation is important!

- Don’t be Kodak!
  - Invent the digital camera
  - Fear of disruption

    “That’s cute - but don’t tell anyone about it.” **[Mui 2012]**
  - Competition wins

Game Development

- Typically, large engines have…
  - Complex build systems
  - Inter-connected dependencies
  - Opinionated and rigid APIs
  - Specialized systems
  - Steep learning curve



Why?

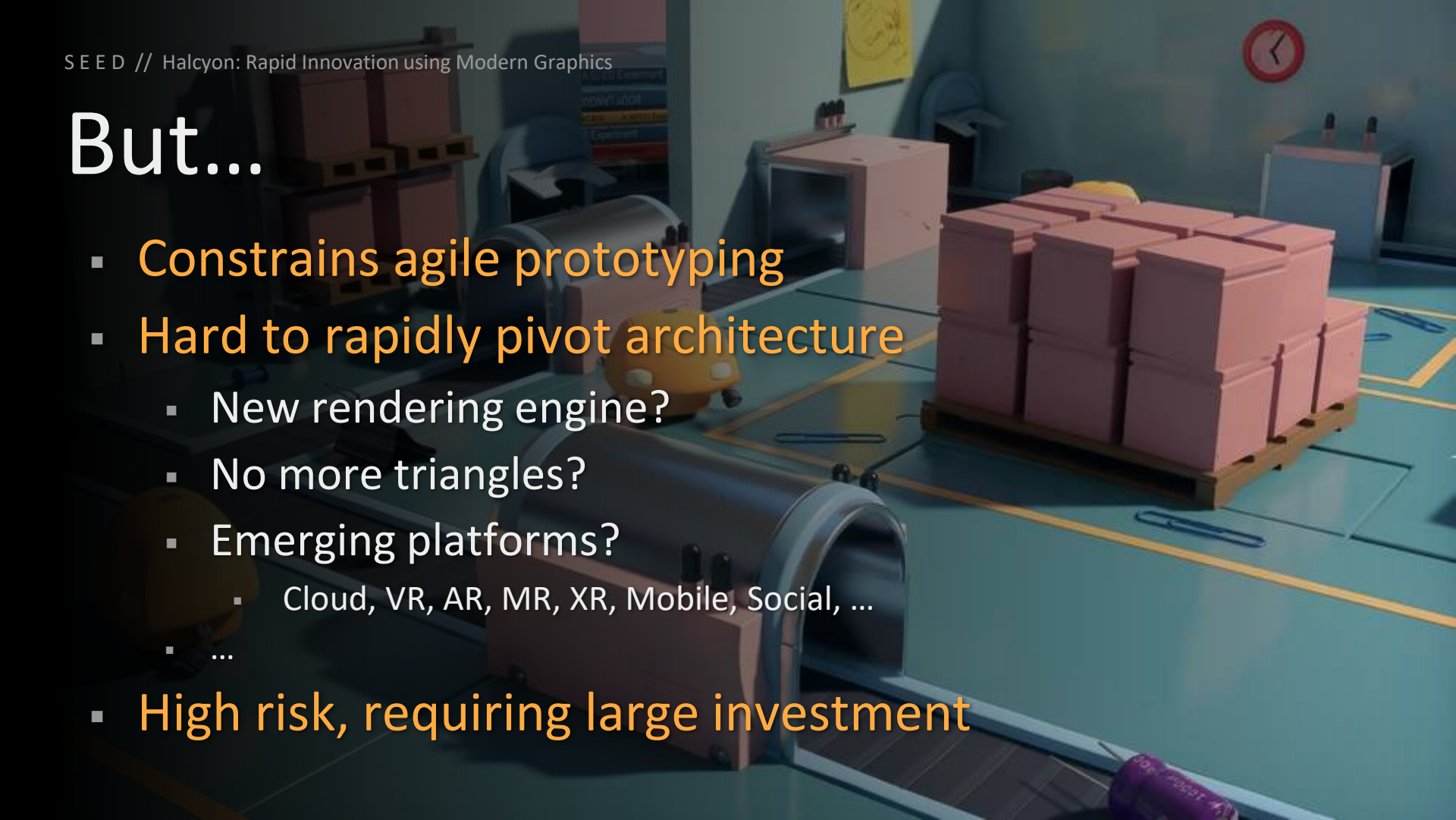
- For good reason!
  - Ship great games
- Performance > flexibility





But...

- Constrains agile prototyping
- Hard to rapidly pivot architecture
  - New rendering engine?
  - No more triangles?
  - Emerging platforms?
    - Cloud, VR, AR, MR, XR, Mobile, Social, …
  - …
- High risk, requiring large investment



Solution!

- Agile R&D engine

- Prove out technology
  - Scout ahead
  - Share findings
  - Mitigate risk
  - Guide adoption

Comparison

Game Engine

R&D Engine

Comparison

Game Engine

R&D Engine


HALCYON

Halcyon

- Rapid prototyping engine
- Windows, Linux, macOS
- Different purpose than Frostbite
  - Fast experimentation vs. AAA games

“PICA PICA”

- Exploratory mini-game & world
  [Andersson 2018] [Harmer 2018] [Opara 2018]
- Built with Halcyon
- Goals
  - Hybrid rendering
  - Clean and consistent visuals
  - Self-learning AI agents
  - Procedural worlds
  - No precomputation

PICA PICA Trailer

https://www.youtube.com/watch?v=LXo0WdlELJk

Halcyon Goals

- Minimize or eliminate busy-work
  - Artist “meta-data” meshes
    - Occlusion
    - GI / Lighting
    - Collision
    - Level-of-detail
- Live reloading of all assets
  - Insanely fast iteration times

Halcyon Goals

- Only target modern APIs
  - Vulkan 1.1
  - Direct3D 12
  - Metal 2
- Multi-GPU
  - Explicit heterogeneous mGPU
  - No AFR / No linked adapters

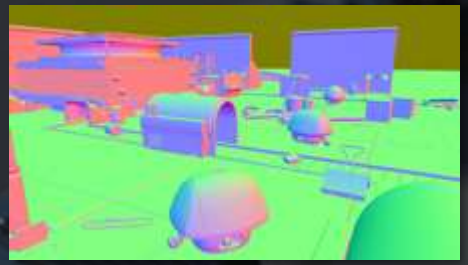
Halcyon Goals

- Scalable computation
  - All cores in a workstation
  - Multiple graphics adapters (mGPU)
  - Local cluster
  - Google Cloud Platform
- Under a single abstraction
  - Easy, right?

Halcyon Goals

- Local or remote streaming
- Minimal boilerplate code
- Variety of rendering techniques and approaches
  - Rasterization
  - Path and ray tracing
  - Hybrid

Hybrid Rendering



Deferred Shading
(raster)

Direct Shadows
(ray trace or raster)

Direct Lighting
(compute)

Reflections
(ray trace or compute)

Global Illumination
(ray trace)

Ambient Occlusion
(ray trace or compute)

Transparency & Translucency
(ray trace)

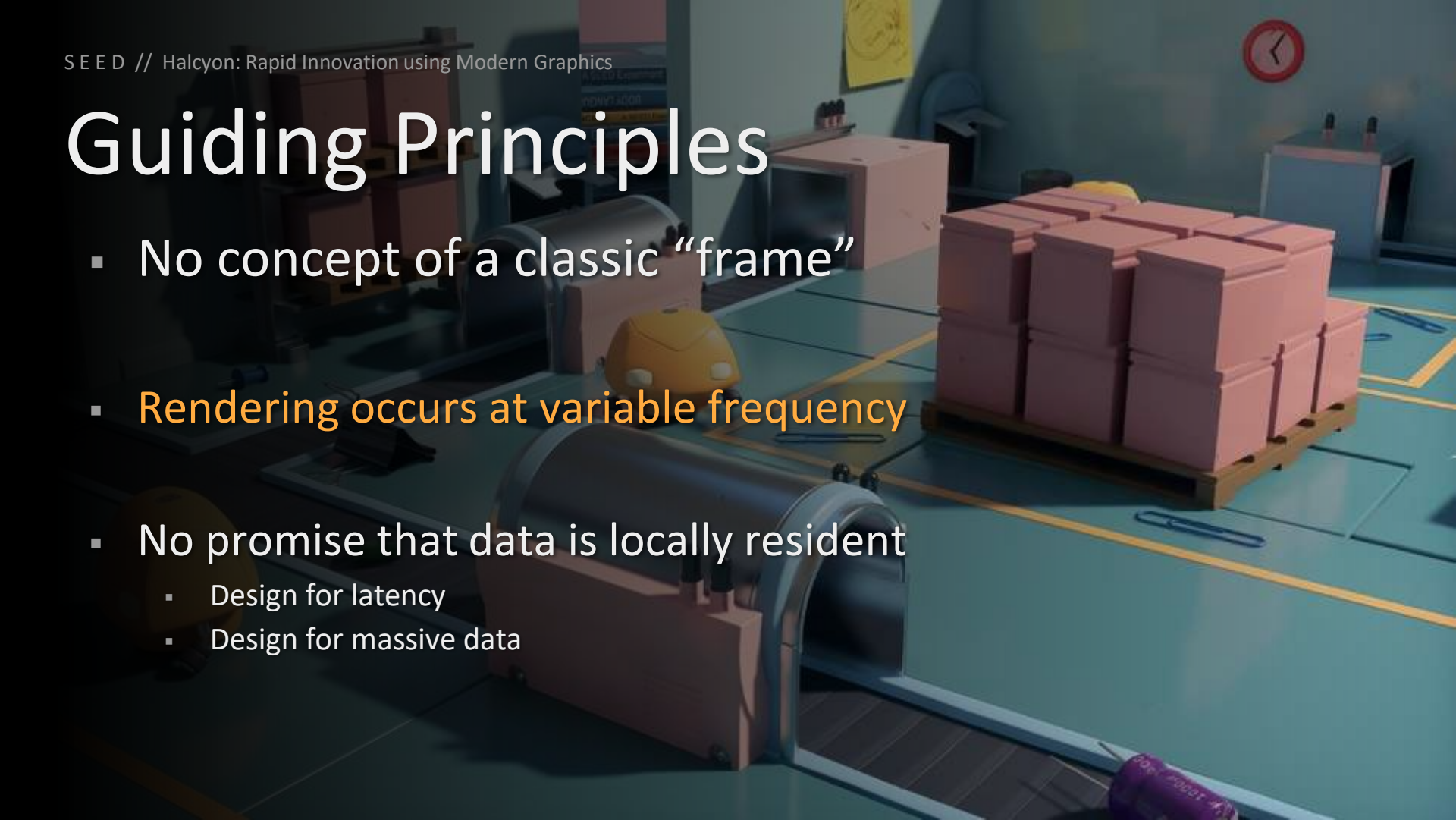
Post Processing
(compute)


Hybrid Rendering


Rasterization Only

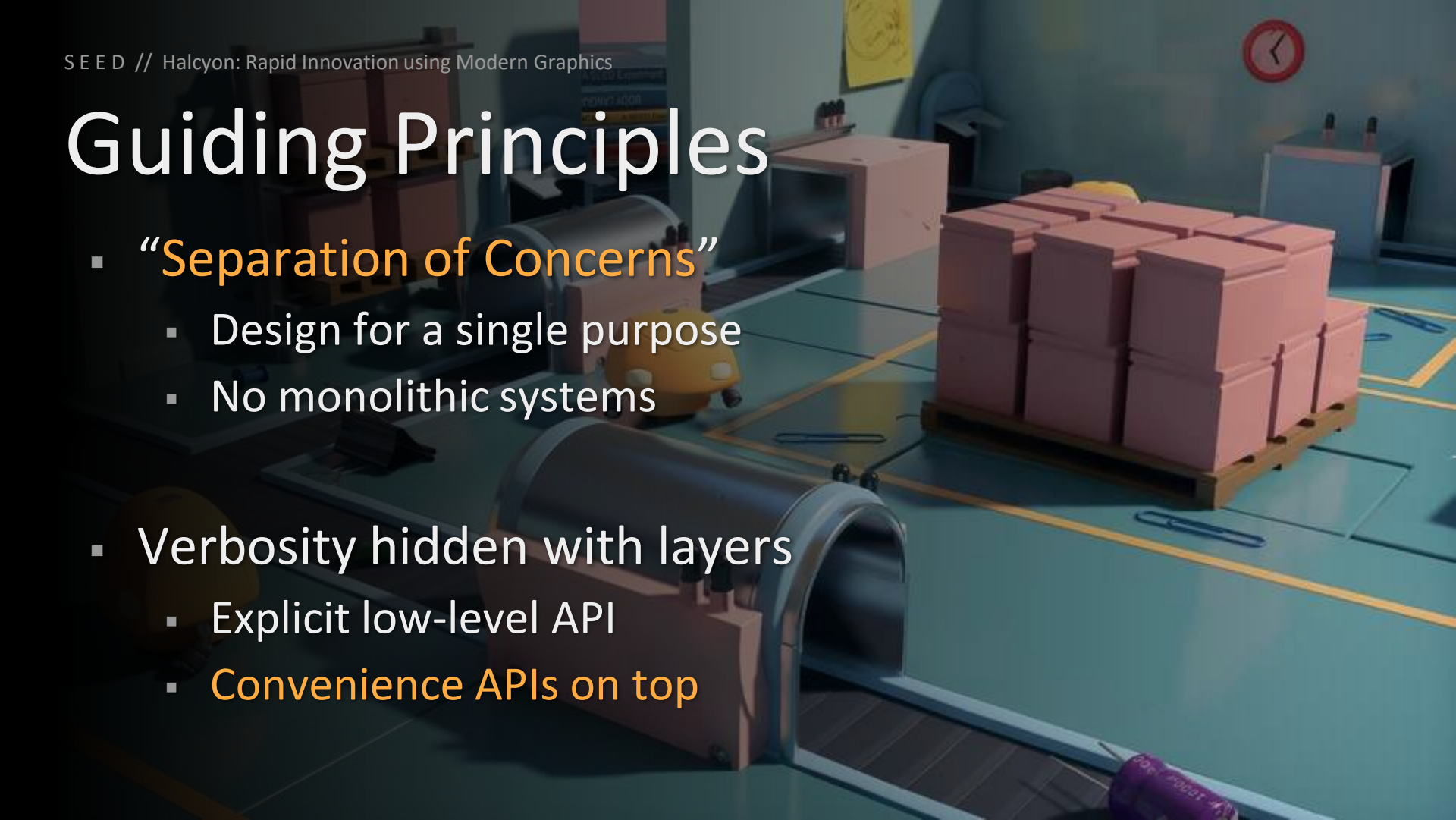
Guiding Principles

- No concept of a classic “frame”
- Rendering occurs at variable frequency
- No promise that data is locally resident
  - Design for latency
  - Design for massive data



Guiding Principles

- "Separation of Concerns"
  - Design for a single purpose
  - No monolithic systems
- Verbosity hidden with layers
  - Explicit low-level API
  - Convenience APIs on top



Rendering Layers


Render Backend

Render Backend

- Live-reloadable DLLs

- Enumerates adapters and capabilities
  - Swap chain support
  - Extensions (i.e. ray tracing, sub groups, …)
  - Determine adapter(s) to use

Render Backend

- Provides debugging and profiling
  - RenderDoc integration, validation layers, …
- Create and destroy render devices

Render Backend

- Direct3D 12
- Vulkan 1.1
- Metal 2
- Proxy
- Mock

Render Backend

- Direct3D 12
  - Shader Model 6.X
  - DirectX Ray Tracing
  - Bindless Resources
  - Explicit Multi-GPU
  - DirectML
  - …

Render Backend

- Vulkan 1.1
  - Sub-groups
  - Descriptor indexing
  - External memory
  - Multi-draw indirect
  - NV Ray tracing
  - …

Render Backend

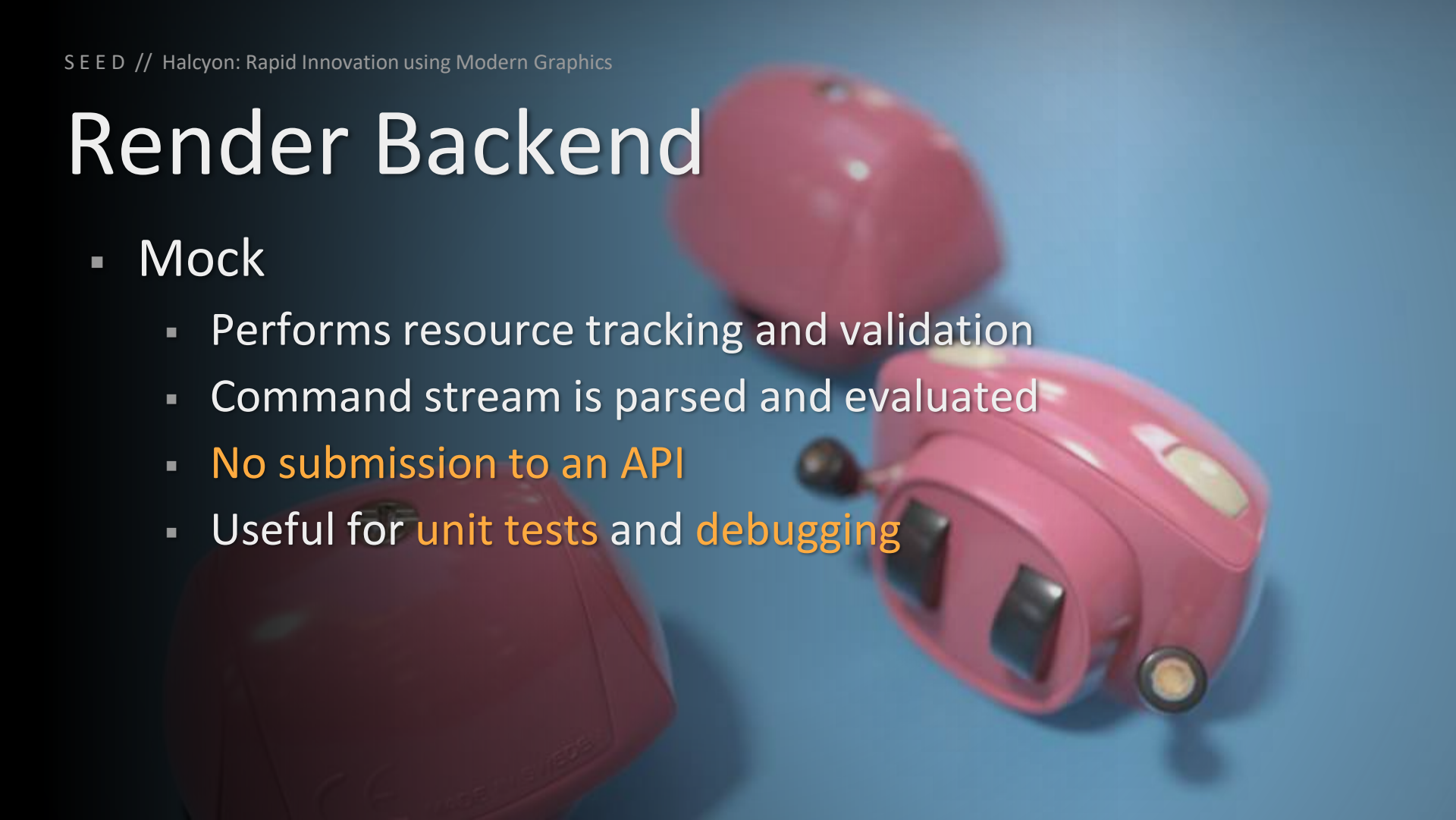
- Metal 2
  - Early development
  - Primarily desktop
  - Argument buffers
  - Machine learning
  - …

Render Backend

- Proxy
  - Discussed later in the presentation

Render Backend

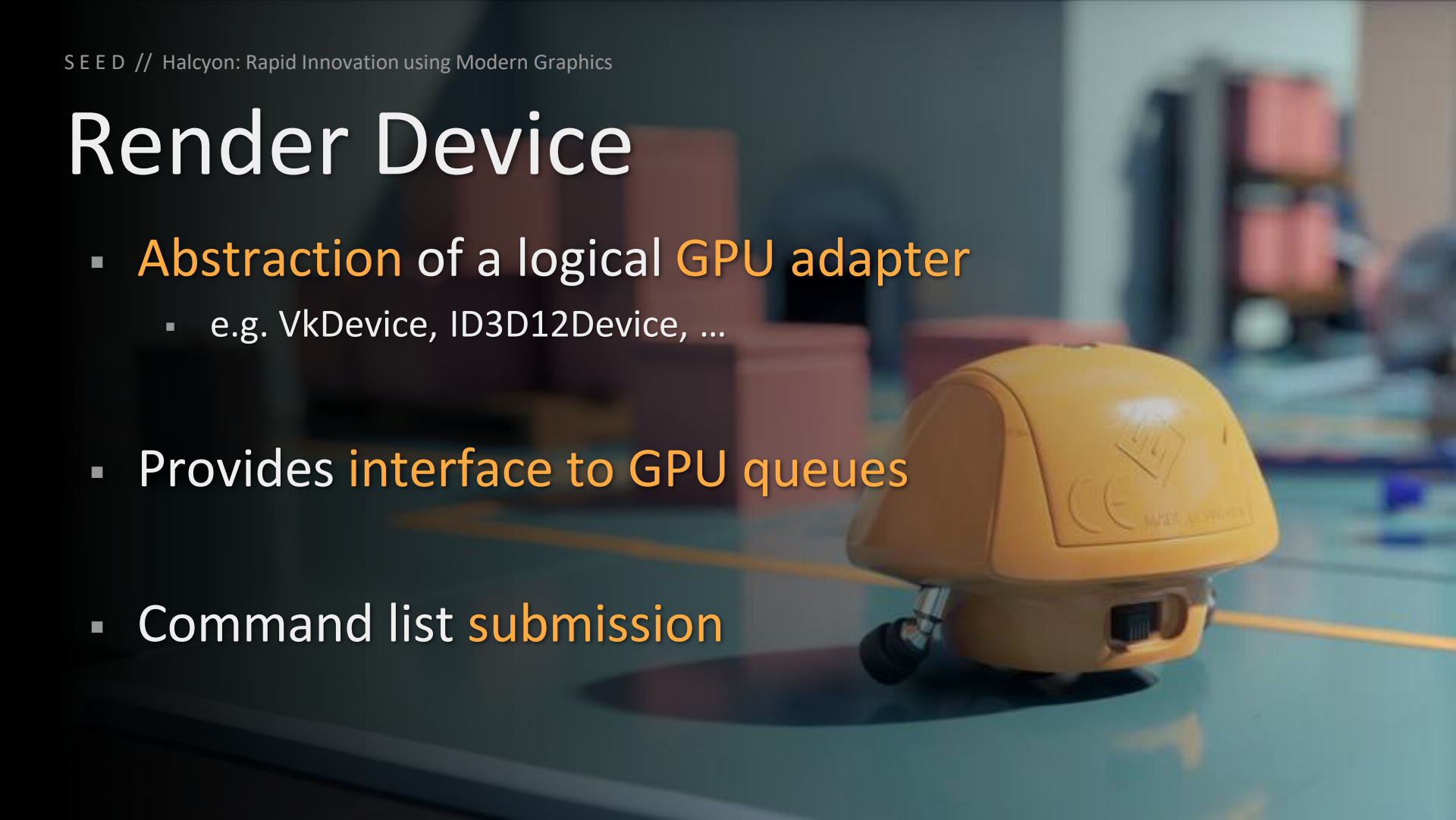
- Mock
  - Performs resource tracking and validation
  - Command stream is parsed and evaluated
  - No submission to an API
  - Useful for unit tests and debugging




Render Device

Render Device

- Abstraction of a logical GPU adapter
  - e.g. VkDevice, ID3D12Device, …
- Provides interface to GPU queues
- Command list submission



Render Device

- Ownership of GPU resources
  - Create & Destroy
- Lifetime tracking of resources
- Mapping render handles → device resources

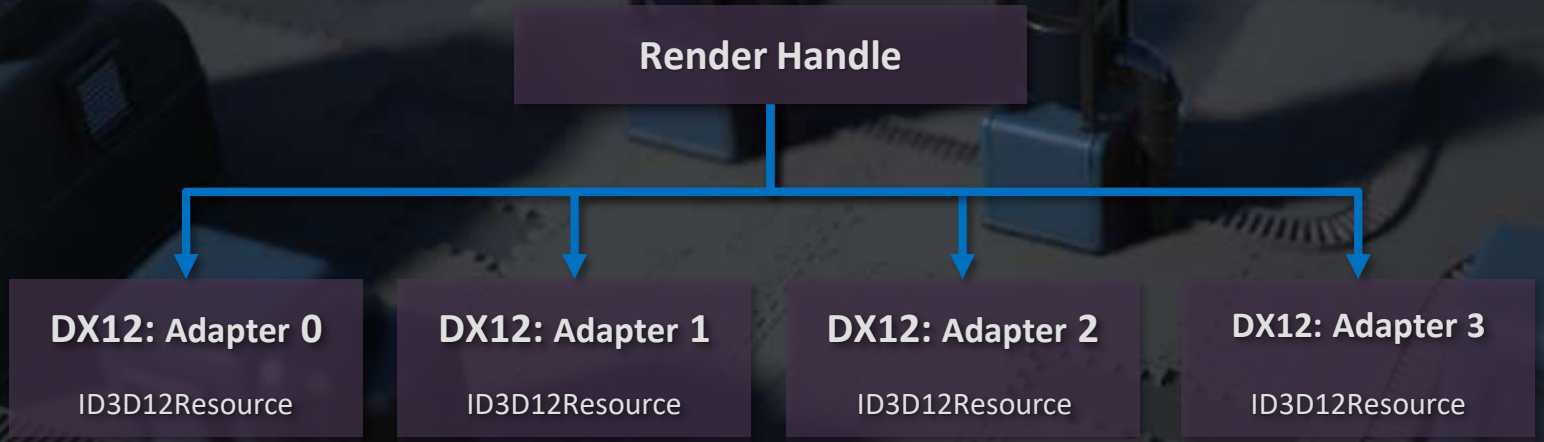

Render Handles

Render Handles

- Resources associated by handle
- Lightweight (64 bits)
- Constant-time lookup
- Type safety (i.e. buffer vs texture)
- Can be serialized or transmitted
- Generational for safety
  - e.g. double-delete, usage after delete

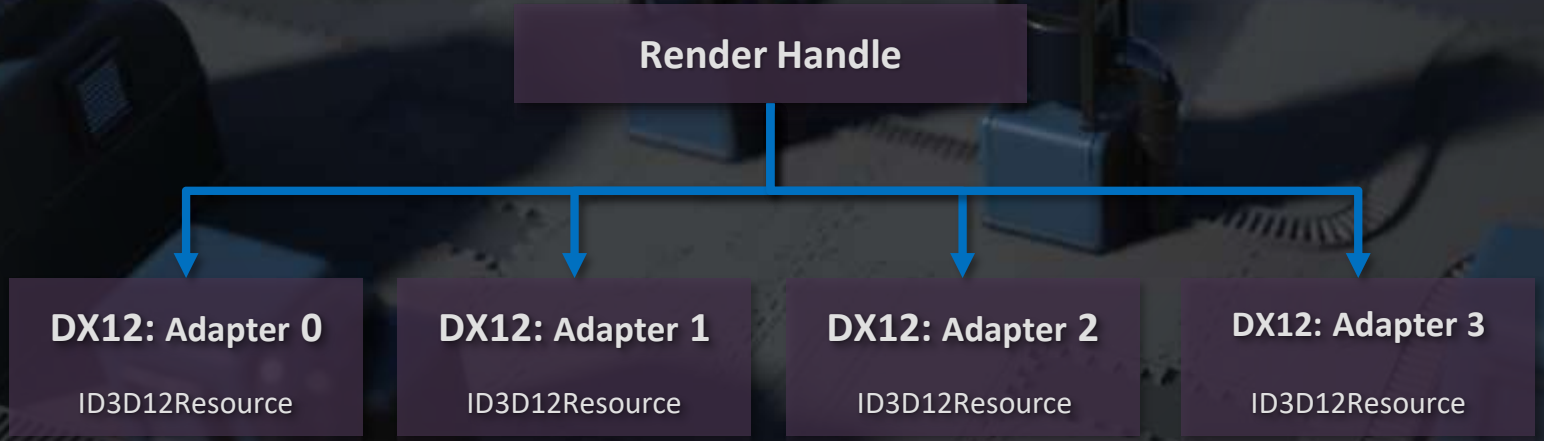
Render Handles

- Handles allow one-to-many cardinality *[handle->devices]*
- Each device can have a unique representation of the handle



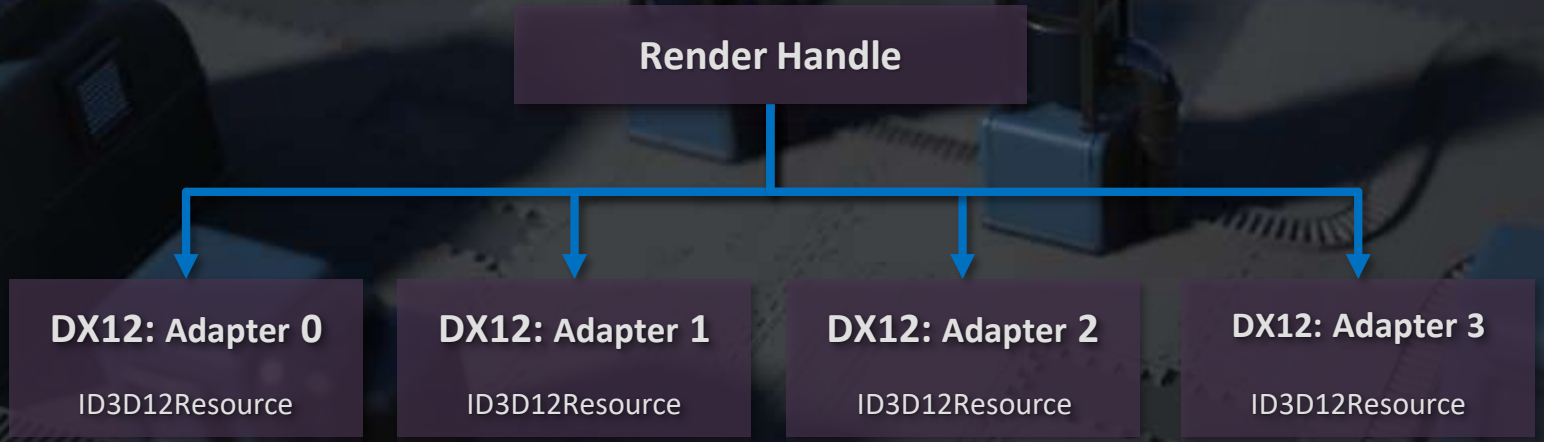
Render Handles

- Can query if a device has a handle loaded
- Safely add and remove devices
  - Handle owned by application, representation can reload on device



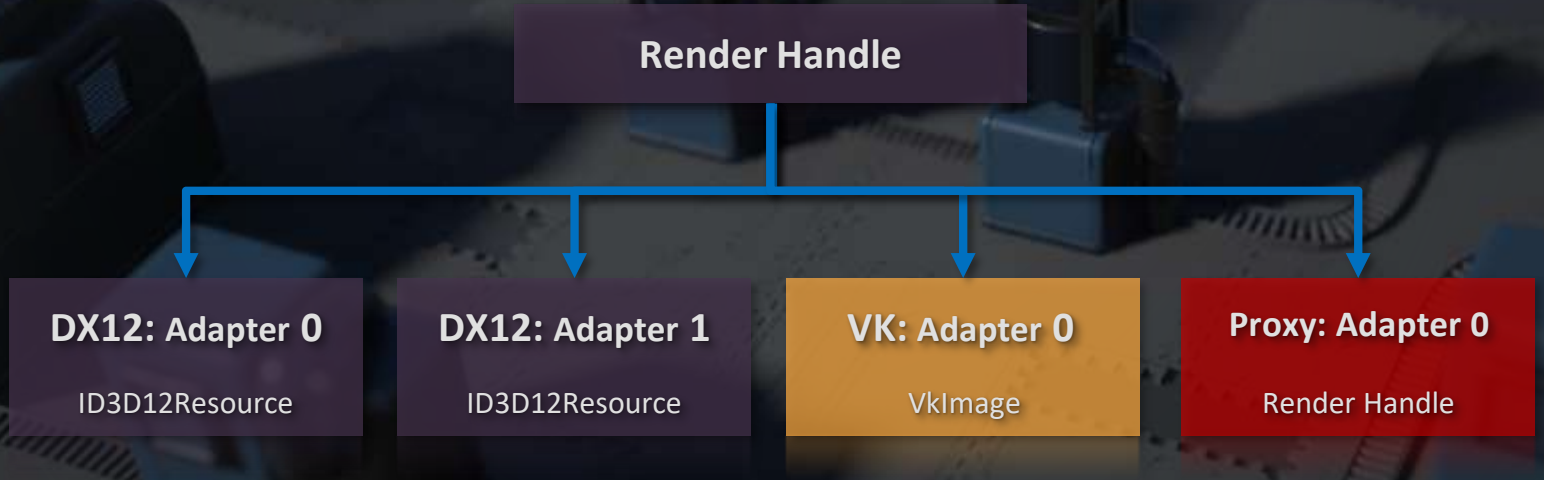
Render Handles

- Shared resources are supported
- Primary device owner, secondaries alias primary



Render Handles

- Can also mix and match backends in the same process!
  - Makes debugging new backends much easier
  - DX12 on left half of screen, VK on right half of screen




Render Commands

Render Commands

- Draw
- DrawIndirect
- Dispatch
- DispatchIndirect
- UpdateBuffer
- UpdateTexture
- CopyBuffer
- CopyTexture
- Barriers
- Transitions
- BeginTiming
- EndTiming
- ResolveTimings
- BeginEvent
- EndEvent
- BeginRenderPass
- EndRenderPass
- RayTrace
- UpdateTopLevel
- UpdateBottomLevel
- UpdateShaderTable

Render Commands

- Queue type specified
- Spec validation
  - Allowed to run?
  - ***e.g. draws on compute***
- Automatic scheduling
  - Where can it run?
  - Async compute

```
enum class RenderCommandQueueType : uint8
{
    None = 0x00,
    Copy = 0x01,
    Compute = 0x02,
    Graphics = 0x04,
    All = Copy | Compute | Graphics,
};

struct RenderCommand
{
    RenderCommandType type = RenderCommandType::Count;
};

template<RenderCommandType TYPE, RenderCommandQueueType QUEUETYPE>
struct RenderCommandTyped : RenderCommand
{
    static const RenderCommandType Type = TYPE;
    static const RenderCommandQueueType QueueType = QUEUETYPE;

    RenderCommandTyped() { type = Type; }
};
```

Render Commands

```
struct RenderCommandDispatch : RenderCommandTyped<RenderCommandType::Dispatch, RenderCommandQueueType::Compute>
{
    RenderResourceHandle pipelineState;
    ShaderArgument shaderArguments[MaxShaderParameters];
    uint32 shaderArgumentsCount = 0;

    uint32 dispatchX = 0;
    uint32 dispatchY = 0;
    uint32 dispatchZ = 0;
};
```

Render Command List

- Encodes high level commands
- Tracks queue types encountered
  - Queue mask indicating scheduling rules
- Commands are stateless - parallel recording

Render Compilation

- Render command lists are “compiled”
  - Translation to low level API
  - Can compile once, submit multiple times

- Serial operation (memcpy speed)
  - Perfect redundant state filtering

Vulkan Compilation

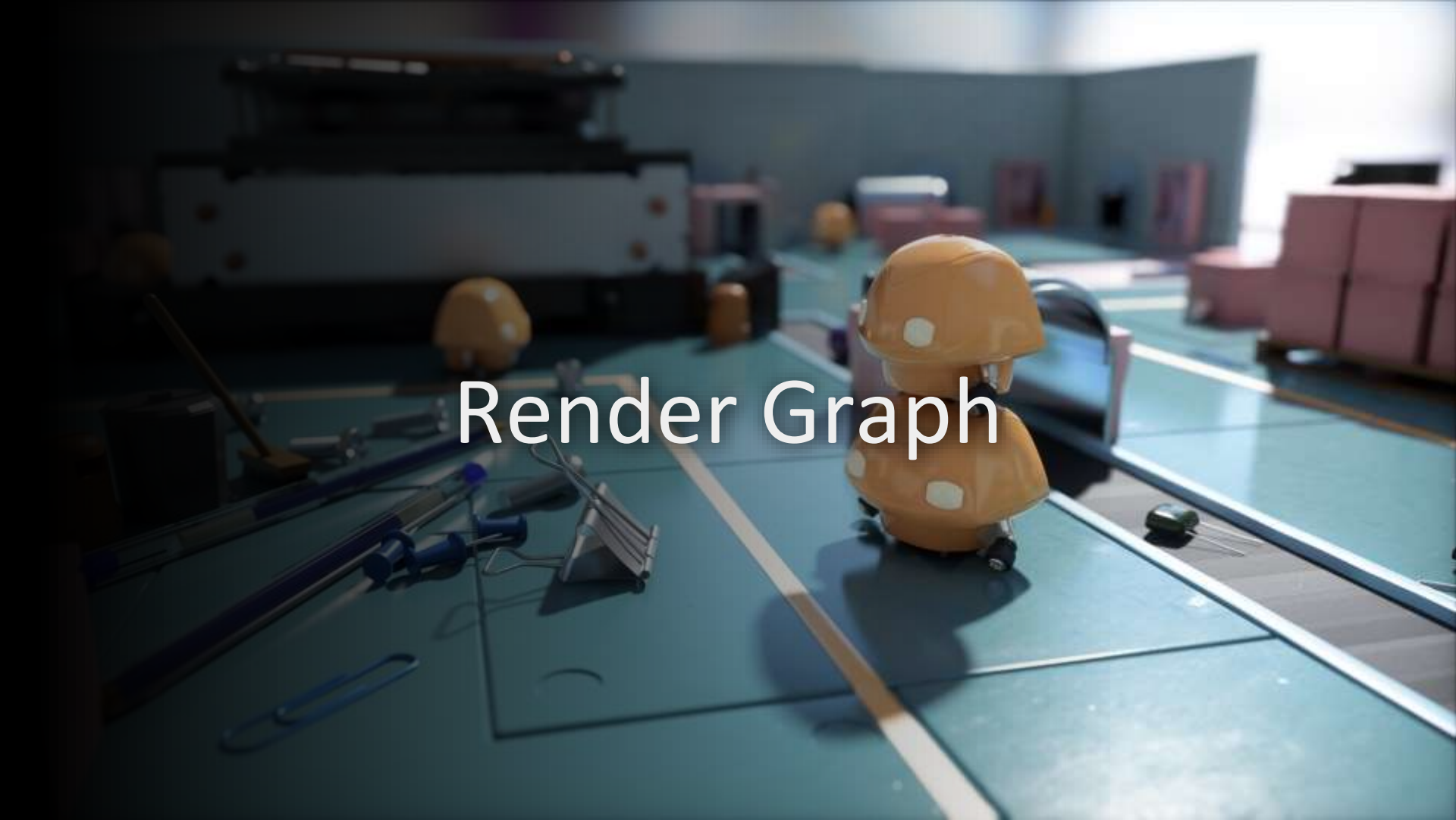
- Translate commands
  - Read command list
  - Write Vulkan API

```
#define COMPILE_PACKET(TYPE_STRUCT)                                          \
    case TYPE_STRUCT::Type:                                                  \
        if (!compileCommand(*static_cast<const TYPE_STRUCT*>(command)))      \
            return false;                                                    \
        break

    for (const auto* command : recorded)
    {
        switch (command->type)
        {
            COMPILE_PACKET(RenderCommandDraw);
            COMPILE_PACKET(RenderCommandDrawIndirect);
            COMPILE_PACKET(RenderCommandDispatch);
            COMPILE_PACKET(RenderCommandDispatchIndirect);
            COMPILE_PACKET(RenderCommandUpdateBuffer);
            COMPILE_PACKET(RenderCommandUpdateTexture);
            COMPILE_PACKET(RenderCommandCopyBuffer);
            COMPILE_PACKET(RenderCommandCopyTexture);
            COMPILE_PACKET(RenderCommandBarriers);
            COMPILE_PACKET(RenderCommandTransitions);
            COMPILE_PACKET(RenderCommandBeginTiming);
            COMPILE_PACKET(RenderCommandEndTiming);
            COMPILE_PACKET(RenderCommandResolveTimings);
            COMPILE_PACKET(RenderCommandBeginEvent);
            COMPILE_PACKET(RenderCommandEndEvent);
            COMPILE_PACKET(RenderCommandBeginRenderPass);
            COMPILE_PACKET(RenderCommandEndRenderPass);
            break;
        default:
            HcyFail();
            return false;
        }
    }

#undef COMPILE_PACKET
```

```
bool RenderCompileContextVulkan::compileCommand(const RenderCommandDispatch& command)
{
    const auto pipelineState = device.getComputePipelineState(command.pipelineState);
    if (!applyPipelineState(pipelineState))
    {
        return false;
    }

    applyShaderArguments(command.pipelineState, command.shaderArguments, command.shaderArgumentsCount);
    applyTransitions();

    vkCmdDispatch(commandBuffer, command.dispatchX, command.dispatchY, command.dispatchZ);
    return true;
}
```

```
bool RenderCompileContextVulkan::compileCommand(const RenderCommandBeginTiming& command)
{
    auto timingHeap = const_cast<RenderTimingHeapVulkan*>(device.getTimingHeap(command.timingHeap));
    HcyAssert(timingHeap);
    const uint32 slot = 2 * command.region + 0;
    HcyAssert(slot >= 0 && slot < (timingHeap->desc.regionCount * 2));
    auto& buffer = timingHeap->buffers[timingHeap->currentBuffer];
    vkCmdWriteTimestamp(
        commandBuffer,
        VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT,
        buffer.queryPool,
        slot);
    buffer.regions[command.region].begin = slot;
    return true;
}
```

```cpp
bool RenderCompileContextVulkan::compileCommand(const RenderCommandResolveTimings& command)
{
    auto timingHeap = const_cast<RenderTimingHeapVulkan*>(device.getTimingHeap(command.timingHeap));
    HcyAssert(timingHeap);
    HcyAssert(command.regionCount > 0);
    HcyAssert(command.regionCount <= timingHeap->desc.regionCount);
    auto& buffer = timingHeap->buffers[timingHeap->currentBuffer];
    buffer.resolveStart = buffer.regions[command.regionStart].begin;
    buffer.resolveCount = (command.regionCount * 2);

    vkCmdCopyQueryPoolResults(
        commandBuffer,
        buffer.queryPool,
        buffer.resolveStart,
        buffer.resolveCount,
        buffer.readBack->buffer(),
        buffer.resolveStart * sizeof(uint64),
        sizeof(uint64),
        VK_QUERY_RESULT_64_BIT | VK_QUERY_RESULT_WAIT_BIT);

    buffer.writeFence->signalGpu(queue.commandQueue());

    vkCmdResetQueryPool(commandBuffer, buffer.queryPool, buffer.resolveStart, buffer.resolveCount);

    timingHeap->previousBuffer = timingHeap->currentBuffer;
    timingHeap->currentBuffer = (device.m_frameIndex % timingHeap->buffers.size());

    return true;
}
```


Render Graph

Render Graph

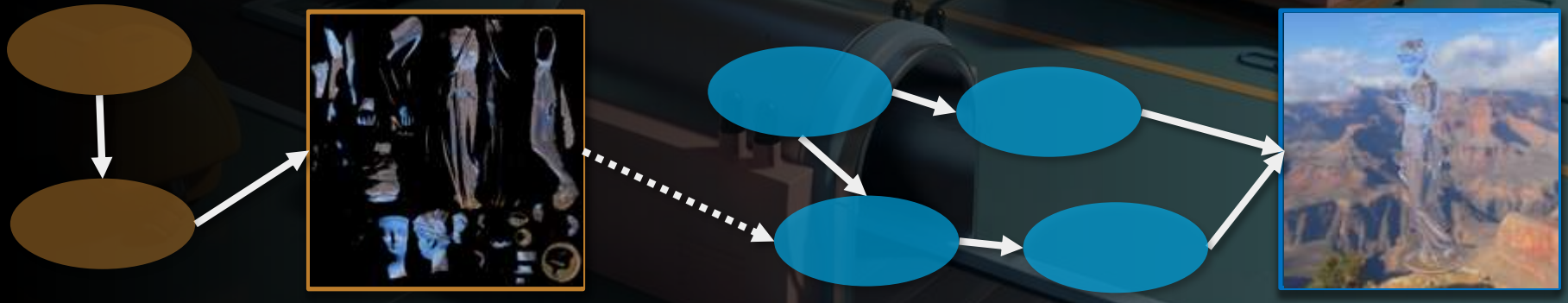
- Inspired by FrameGraph **[O'Donnell 2017]**
- Automatically handle transient resources
- Import explicitly managed resources
- Automatic resource transitions
  - Render target batching
  - DiscardResource
  - Memory aliasing barriers
  - …

Render Graph

- **Frame** Graph → **Render** Graph: No concept of a “frame”
- Fully automatic transitions and split barriers
- Single implementation, regardless of backend
  - Translation from high level render command stream
  - API differences hidden from render graph
- Support for mGPU
  - Mostly implicit and automatic
  - Can specify a scheduling policy

Render Graph

- Composition of multiple graphs at varying frequencies
  - Same GPU: async compute
  - mGPU: graphs per GPU
  - Out-of-core: server cluster, remote streaming



Render Graph

- Composition of multiple graphs at varying frequencies
  - e.g. translucency, refraction, global illumination

Render Graph

- Two phases

- Graph construction
  - Specify inputs and outputs
  - Serial operation (by design)

- Graph evaluation
  - Highly parallelized
  - Record high level render commands
  - Automatic barriers and transitions

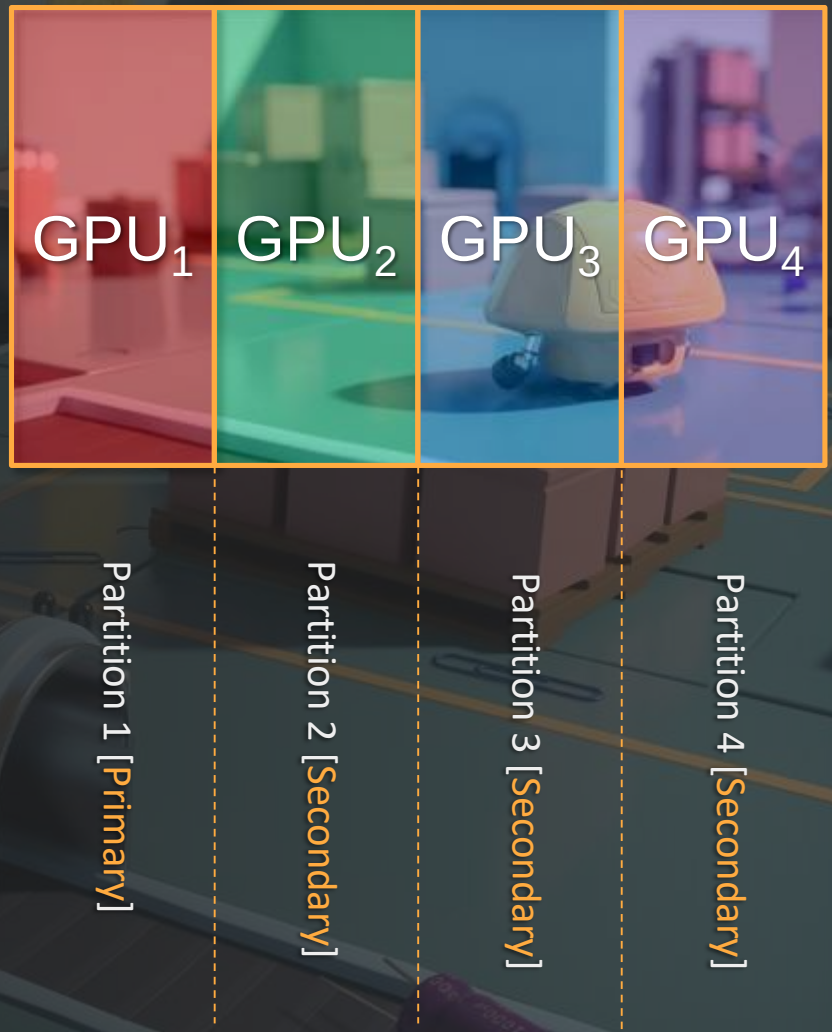
```
renderGraph.addPassCallback("Present Pass", [&](RenderGraphBuild& build)
{
    auto& outputTexData = scope.get<RenderGraphOutputTexture>();
    const auto& viewData = scope.get<RenderGraphViewData>();
    const auto& custom = scope.getOptional<RenderGraphCustomFinalTexture>();

    auto finalTexture = build.read(
        custom ? custom->finalTexture
        : scope.get<RenderGraphFinalTexture>().finalTexture, RenderBindFlags::ShaderResource);

    auto outputTexture = outputTexData.outputTexture = build.write(outputTexData.outputTexture, RenderBindFlags::UnorderedAccess);

    return [=](RenderGraphRegistry& registry, RenderCommandList& commandList)
    {
        RenderPoint point = {};
        RenderBox box = {};
        box.w = viewData.viewWidth;
        box.h = viewData.viewHeight;

        commandList.copyTexture(
            registry.getTexture(outputTexture),
            0,
            point,
            registry.getTexture(finalTexture),
            0,
            box);
    };
});
```

← Construction phase

← Evaluation phase

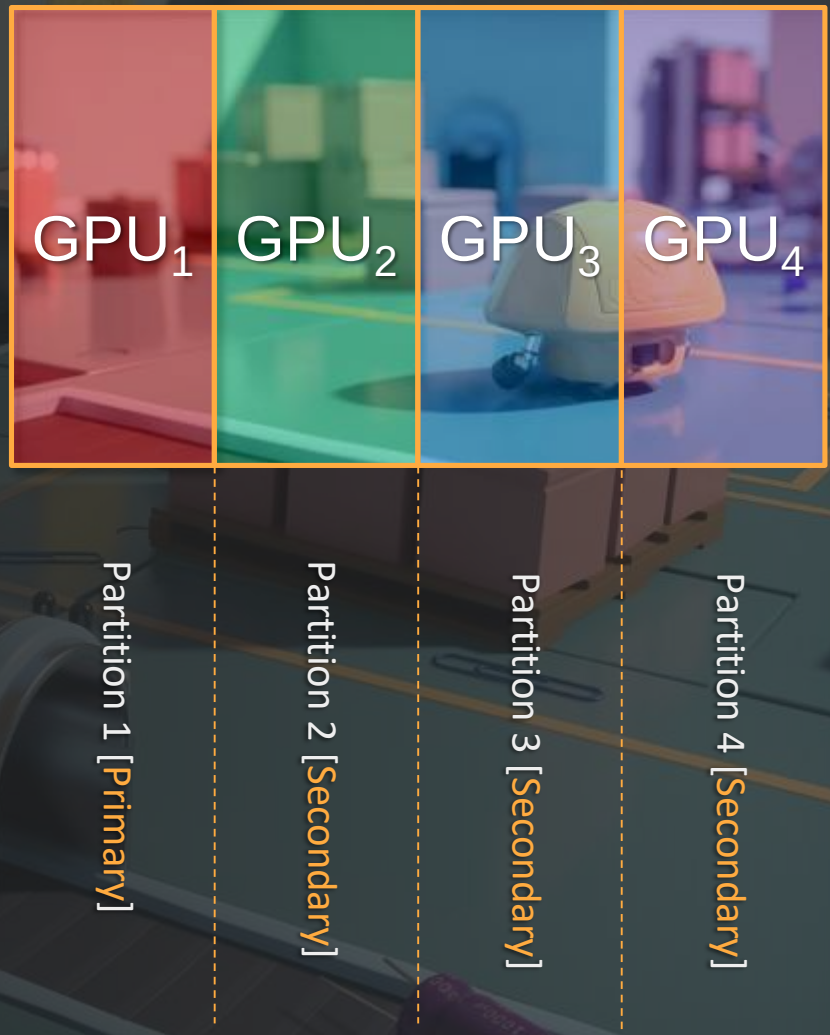
Render Graph

- Explicit heterogeneous mGPU
- Parallel fork-join approach
- Resources copied through system memory using copy queue
  - ~1ms for every 15mb transferred
- Minimize PCI-E transfers
  - Immutable data replicated
  - Tightly pack data



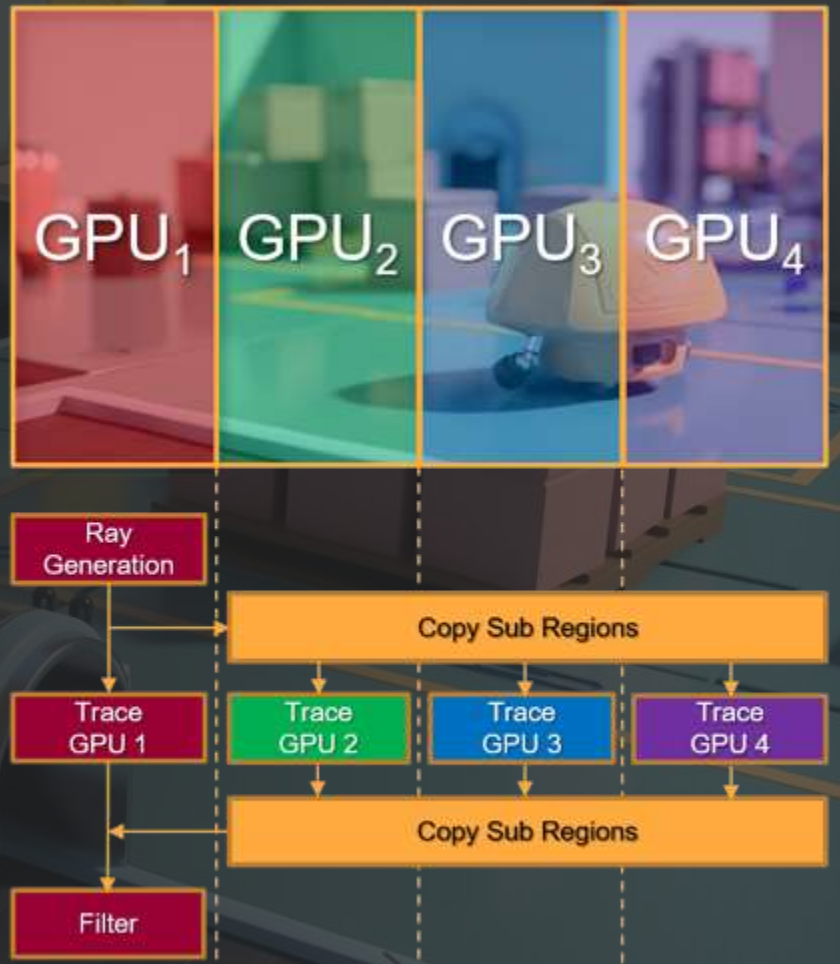
Render Graph

- Workloads are divided into **partitions**
  - Based on GPU device count
- Single primary device
- Other devices are secondaries
- Variety of scheduling and transfer patterns are necessary
- Simple rules engine



Render Graph

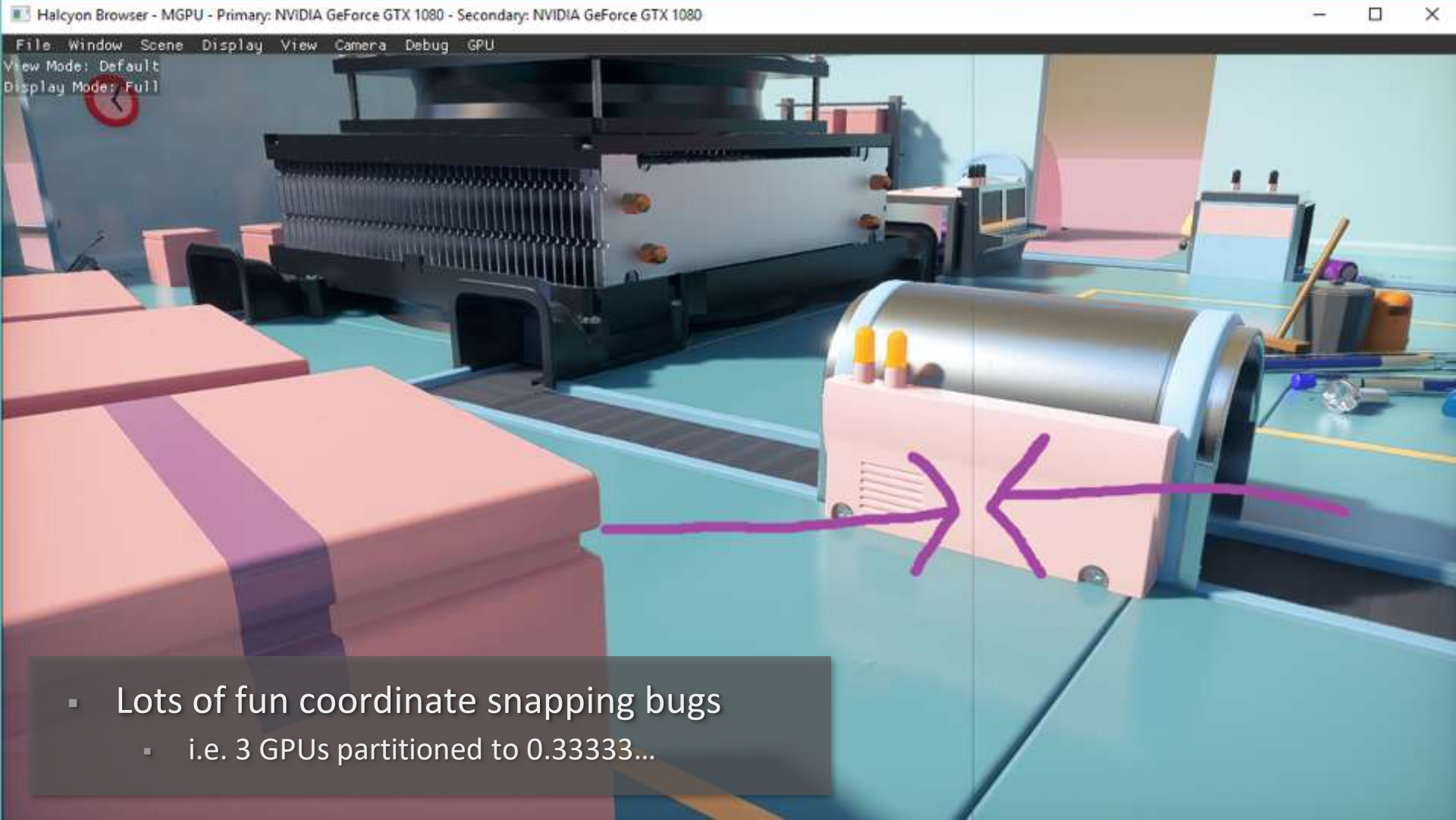
- Run ray generation on primary GPU
- Copy results in sub-regions to other GPUs
- Run tracing phases on separate GPUs
- Copy tracing results back to primary GPU
- Run filtering on primary GPU

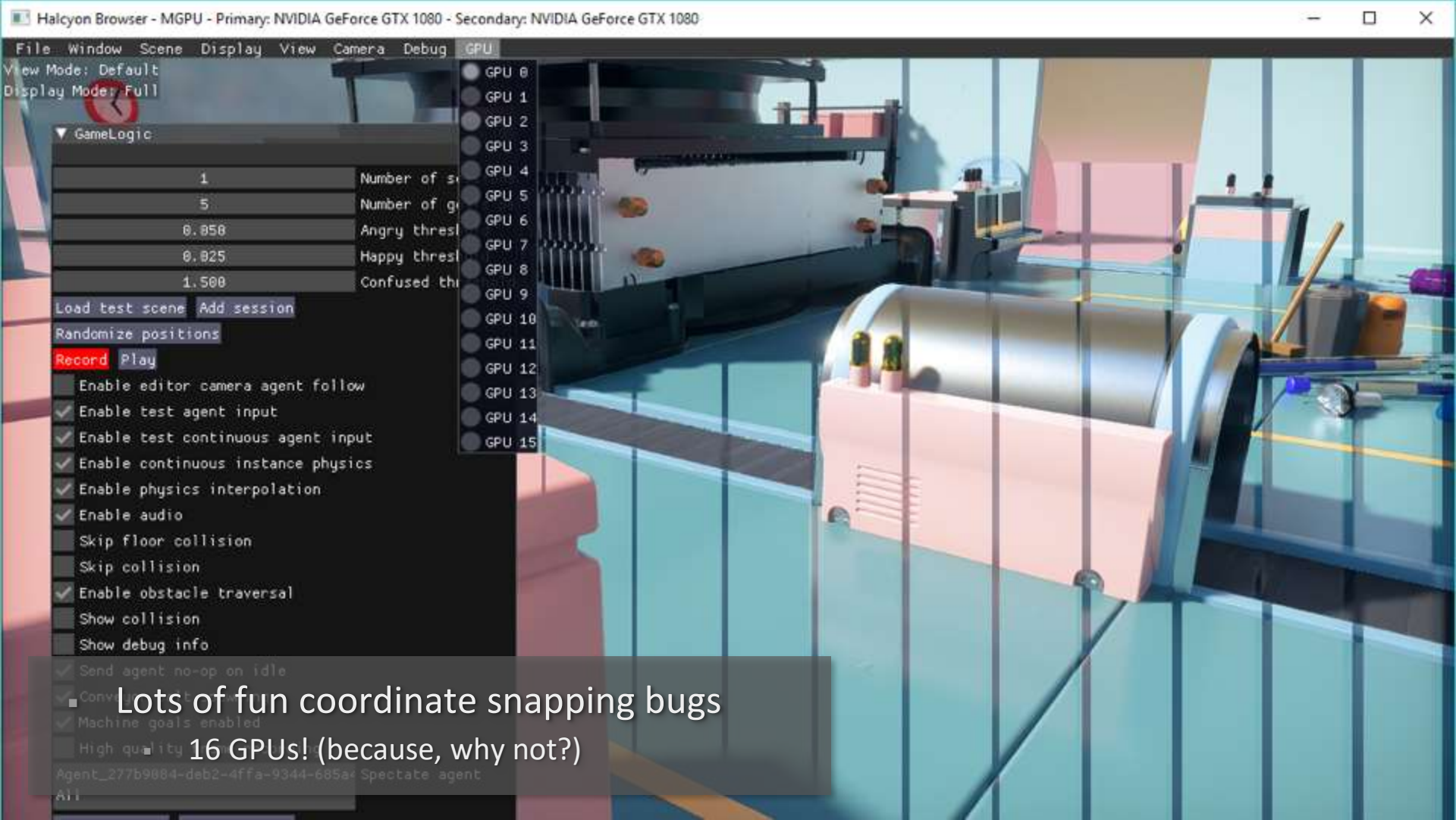


Render Graph

- Only width is divided
- Simplifies textures vs. buffers
- Passes are unaware of GPU count

```
glm::vec4 partitionWindow = scheduleData.mgpuShadows ? registry.partitionIsolated() : registry.partitionAll();
const uint32 dispatchOffset = uint32(desc.width * partitionWindow.x);
const uint32 dispatchWidth = uint32(desc.width * partitionWindow.z);

// ...

commandList.dispatch2d(
    pipelineState,
    { ShaderArgument(dynamicConstants.buffer, constantsOffset, registry.createShaderViews(srvs, uavs)) },
    dispatchWidth, desc.height
);
```


Halcyon Browser - MGPU - Primary: NVIDIA GeForce GTX 1080 - Secondary: NVIDIA GeForce GTX 1080
File Window Scene Display View Camera Debug GPU
View Mode: Default
Display Mode: Full
Lots of fun coordinate snapping bugs
i.e. 3 GPUs partitioned to 0.33333…


Halcyon Browser - MGPU - Primary: NVIDIA GeForce GTX 1080 - Secondary: NVIDIA GeForce GTX 1080
File Window Scene Display View Camera Debug GPU
View Mode: Default
Display Mode: Full
GameLogic
GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
GPU 5
GPU 6
GPU 7
GPU 8
GPU 9
GPU 10
GPU 11
GPU 12
GPU 13
GPU 14
GPU 15
1
5
0.050
0.025
1.500
Load test scene
Add session
Randomize positions
Record
Play
Enable editor camera agent follow
Enable test agent input
Enable test continuous agent input
Enable continuous instance physics
Enable physics interpolation
Enable audio
Skip floor collision
Skip collision
Enable obstacle traversal
Show collision
Show debug info
Lots of fun coordinate snapping bugs
16 GPUs! (because, why not?)

Render Graph

- RenderGraphSchedule
  - **NoDevices** → Pass is disabled
  - **AllDevices** → Pass runs on all devices
  - **PrimaryDevice** → Pass only runs on primary device
  - **SecondaryDevices** → Pass runs on secondaries if count > 1, otherwise primary
  - **OnlySecondaryDevices** → Pass only runs on secondary devices, disabled unless mGPU

Requested Per Pass →

```
void setSchedule(RenderGraphSchedule passSchedule);

void setTransfer(
    RenderGraphResource resource,
    RenderTransferPartition src,
    RenderTransferPartition dst,
    RenderTransferFilter dstFilter);
```

Render Graph

- RenderTransferPartition
  - **PartitionAll** → Select all partitions from device
  - **PartitionIsolated** → Select isolated region from device
- RenderTransferFilter
  - **AllDevices** → Transfer completes on all devices
  - **PrimaryDevice** → Transfer completes on the primary device
  - **SecondaryDevices** → Transfer completes on all secondary devices

Requested Per Pass →

```
void setSchedule(RenderGraphSchedule passSchedule);

void setTransfer(
    RenderGraphResource resource,
    RenderTransferPartition src,
    RenderTransferPartition dst,
    RenderTransferFilter dstFilter);
```

Render Graph

- **PartitionAll → PartitionAll**
  - Copies full resource on one GPU to full resource on all specified GPUs
- **PartitionAll → PartitionIsolated**
  - Copies full resource on one GPU to isolated regions on all specified GPUs (partial copies)
- **PartitionIsolated → PartitionAll**
  - **(Invalid configuration)**
- **PartitionIsolated → PartitionIsolated**
  - Copies isolated region on one GPU to isolated regions on all specified GPUs (partial copies)

```
renderGraph.addPassCallback("Shadow Mask Pack", [&](RenderGraphBuild& build)
{
    build.setSchedule(RenderGraphSchedule::AllDevices);

    // ...

    build.setTransfer(
        outputTexture,
        RenderTransferPartition::PartitionIsolated,
        RenderTransferPartition::PartitionIsolated,
        RenderTransferFilter::PrimaryDevice);

    auto pipelineState = pipelines.pipelineState(ShaderPipelineId::ShadowMaskPack);

    return [=](RenderGraphRegistry& registry, RenderCommandList& commandList)
    {
        glm::vec4 partitionWindow = scheduleData.mgpuShadows ? registry.partitionIsolated() : registry.partitionAll();
        const uint32 dispatchOffset = uint32(desc.width * partitionWindow.x);
        const uint32 dispatchWidth = uint32(desc.width * partitionWindow.z);

        // ...

        commandList.dispatch2d(
            pipelineState,
            { ShaderArgument(dynamicConstants.buffer, constantsOffset, registry.createShaderViews(srvs, uavs)) },
            dispatchWidth, desc.height
        );
    };
});
```

Devices this pass will run on

Schedule transfers in or out

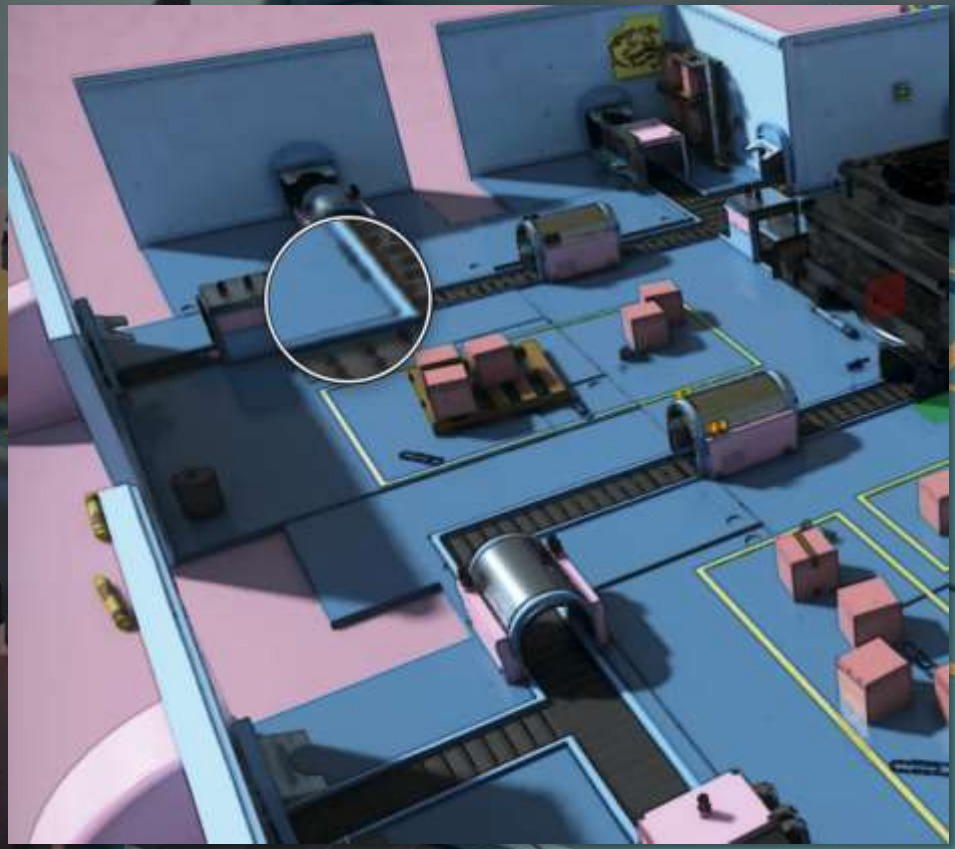
Scaling work dimensions for each GPU


Halcyon Browser - MGPU - Primary: NVIDIA GeForce GTX 1080 - Secondary: NVIDIA GeForce GTX 1080
File Window Scene Display View Camera Debug GPU
View Mode: Default
Some bugs were obvious

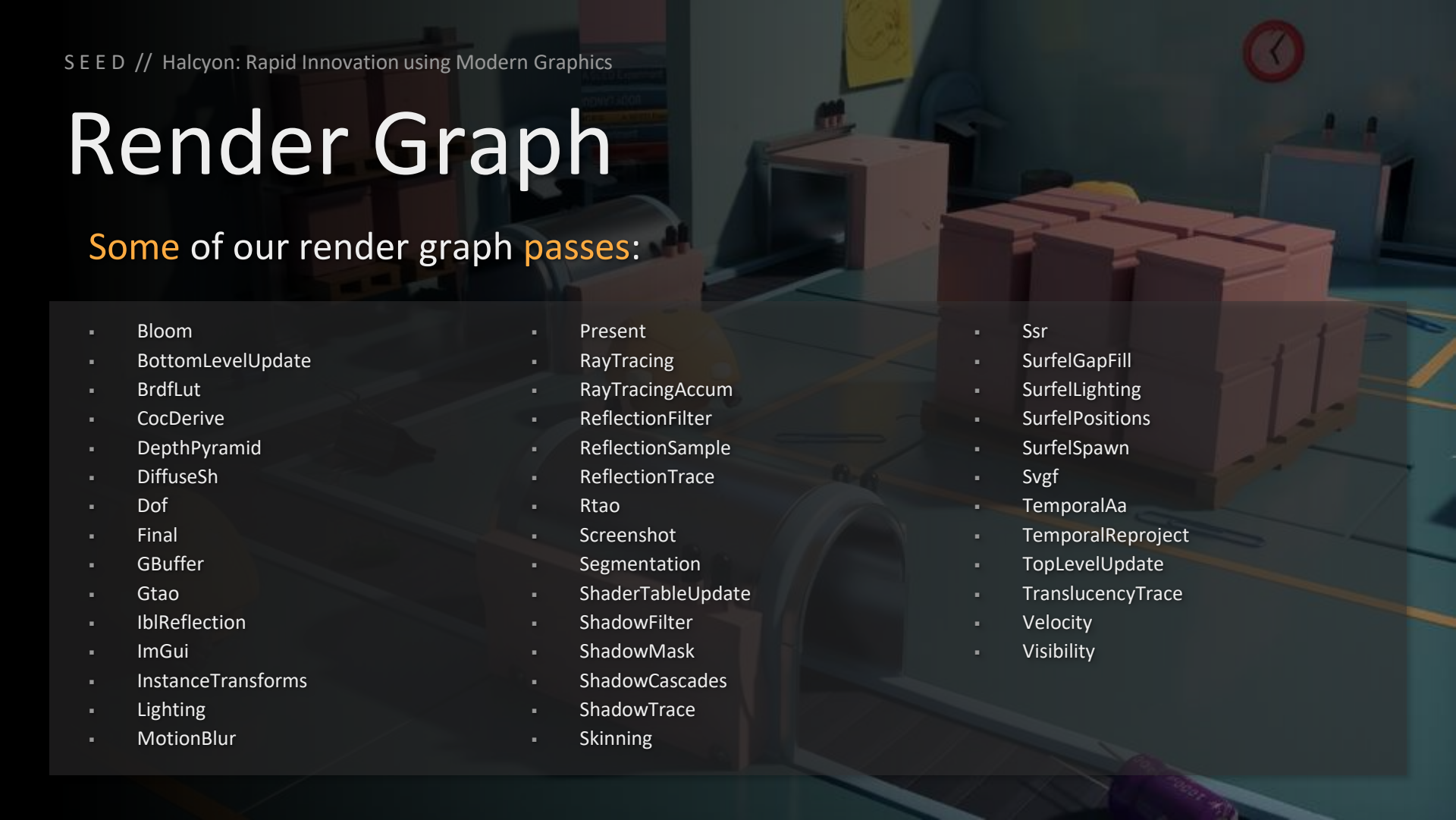

Some bugs were obvious

Render Graph

- Some bugs were subtle
  - Weird cell shading? ☺
- Incorrect transfers?
  - Transfers in (input data)
  - Transfers out (result data)
- Incorrect scheduling?
  - Pass not running
  - Pass running when it shouldn't
  - Partition window



Render Graph



Some of our render graph passes:

- Bloom
- BottomLevelUpdate
- BrdfLut
- CocDerive
- DepthPyramid
- DiffuseSh
- Dof
- Final
- GBuffer
- Gtao
- IblReflection
- ImGui
- InstanceTransforms
- Lighting
- MotionBlur
- Present
- RayTracing
- RayTracingAccum
- ReflectionFilter
- ReflectionSample
- ReflectionTrace
- Rtao
- Screenshot
- Segmentation
- ShaderTableUpdate
- ShadowFilter
- ShadowMask
- ShadowCascades
- ShadowTrace
- Skinning
- Ssr
- SurfelGapFill
- SurfelLighting
- SurfelPositions
- SurfelSpawn
- Svgf
- TemporalAa
- TemporalReproject
- TopLevelUpdate
- TranslucencyTrace
- Velocity
- Visibility

Render Graph

- Implicit data flow via explicit scopes
  - *"Long-distance"* extensible parameter passing
- Scope given to each render pass
  - Can create nested scope for sub-graph
  - Results stored into scope
- Hygiene via nesting and shadowing

```
{
    gbuffer <- render_opaque()
    gbuffer <- render_decals(gbuffer)

    {
        gbuffer <- render_opaque()
        render_lighting(gbuffer)
    } -> envmap

    apply_envmap(gbuffer, envmap)
}
```

```
struct RenderGraphAreaLight
{
    RenderGraphResource triangleLightList;
    uint32 triangleCount;
};
```

Render Graph

- Lookup by type
  - `scope.get<T>() -> &T`
- Parameters in “plain old data” structs
  - RenderGraphResource, RenderHandle
  - float, int, mat4, etc.

```
{
    gbuffer <- render_opaque()
    gbuffer <- render_decals(gbuffer)

    {
        gbuffer <- render_opaque()
        render_lighting(gbuffer)
    } -> envmap

    apply_envmap(gbuffer, envmap)
}
```

```
struct RenderGraphAreaLight
{
    RenderGraphResource triangleLightList;
    uint32 triangleCount;
};
```

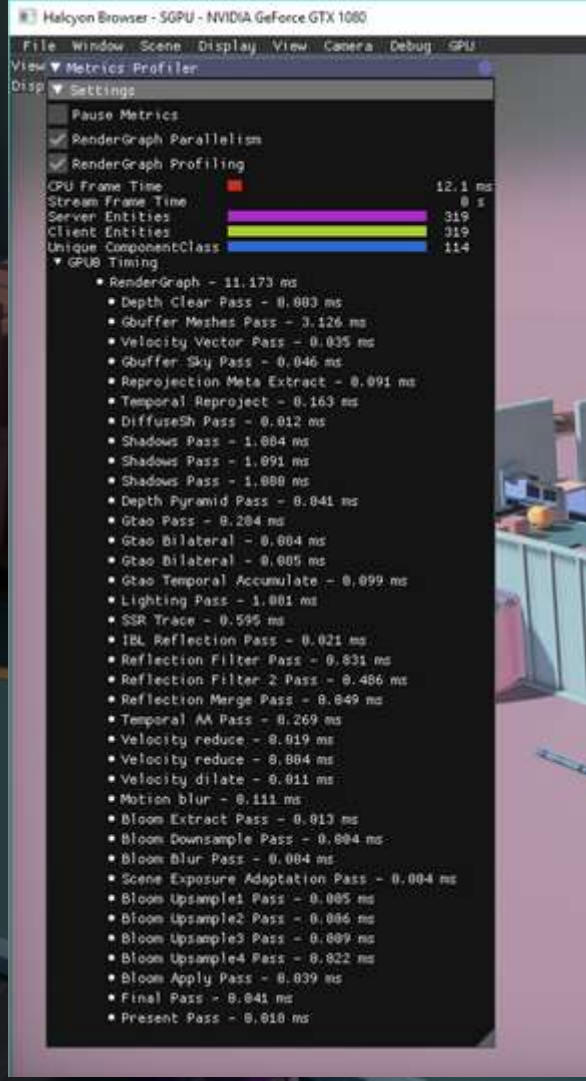
Render Graph DSL

- Experimental
- Macro Magic

```
_def_pass(Present) {
    _use_pipeline(PresentCs) {
        auto& output = _scope(RenderGraphOutputTexture);
        const auto viewData = _scope(RgViewData);

        auto inputTexture = rg.read(_scope(RgMainTexture).texture, RenderBindFlags::ShaderResource);
        output.texture = rg.write(output.texture, RenderBindFlags::UnorderedAccess);

        _when_drawn {
            _def_srvs(
                _res_tex2d(inputTexture),
            );

            _def_uavs(
                _res_tex2d(output.texture),
            );

            commandList.dispatch2d(
                _shader_args,
                viewData.viewWidth,
                viewData.viewHeight
            );
        };
    };
}
```

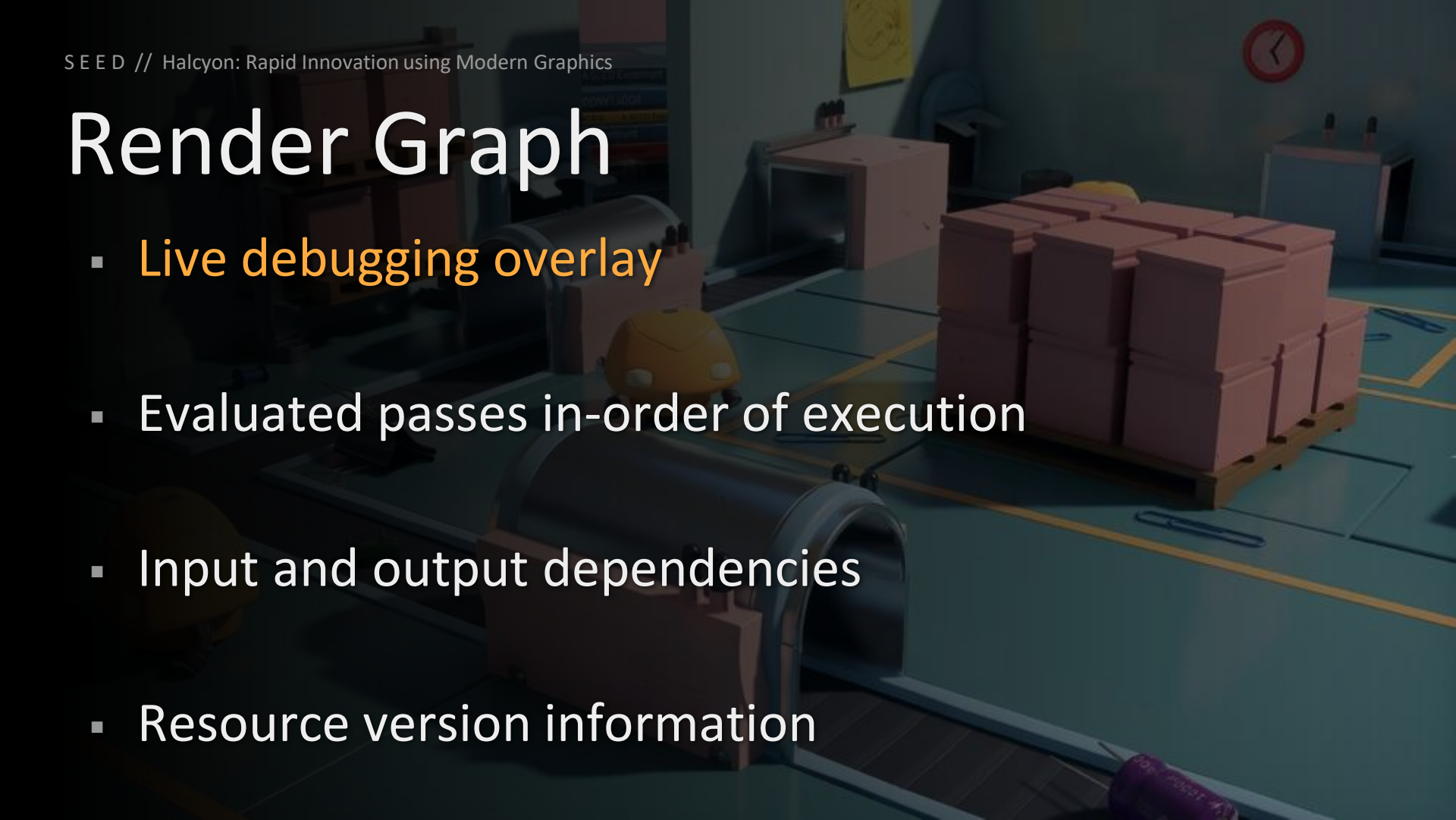
Render Graph

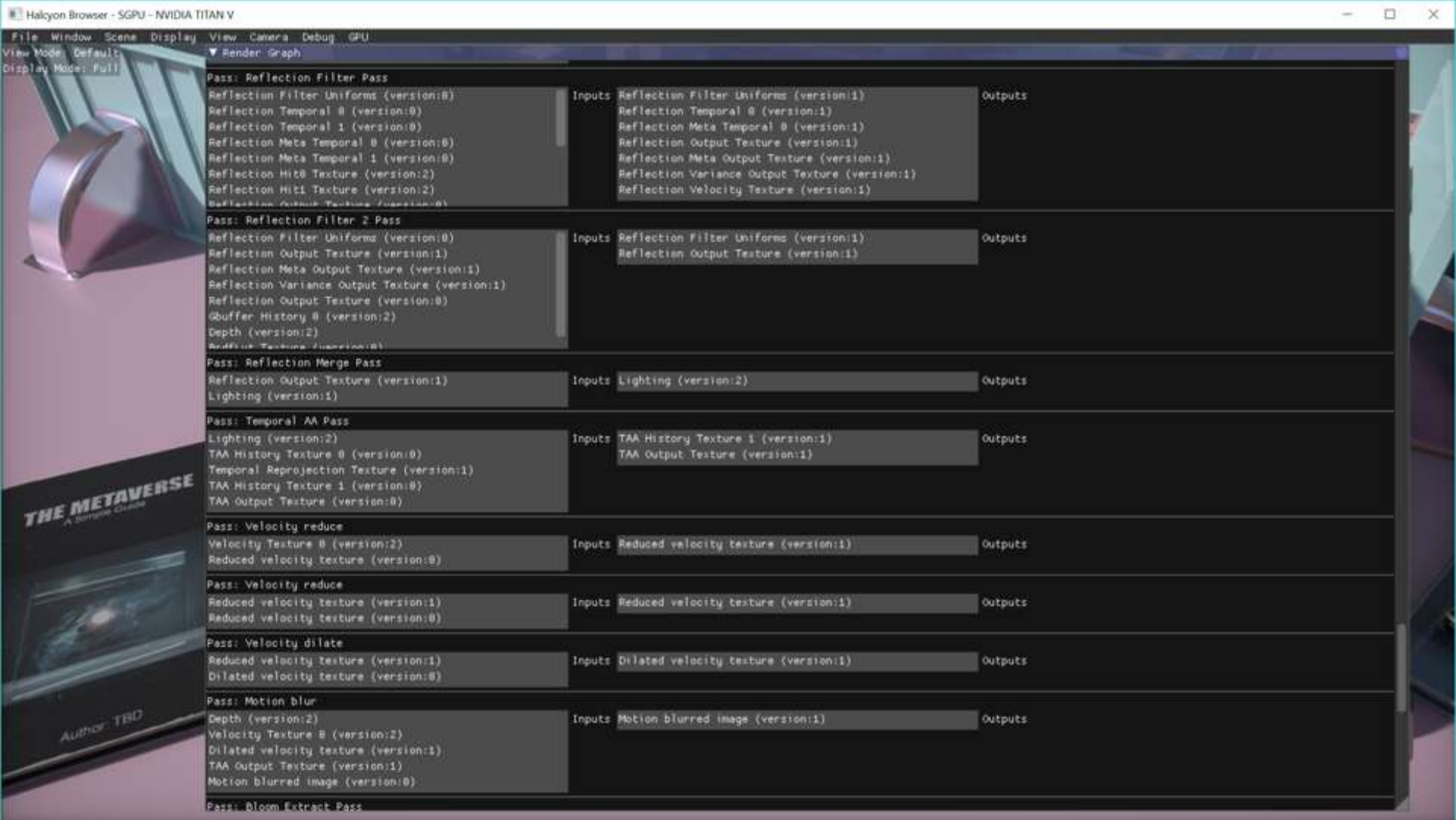
- Automatic profiling data
- GPU and CPU counters per-pass
- Works with mGPU
  - Each GPU is profiled

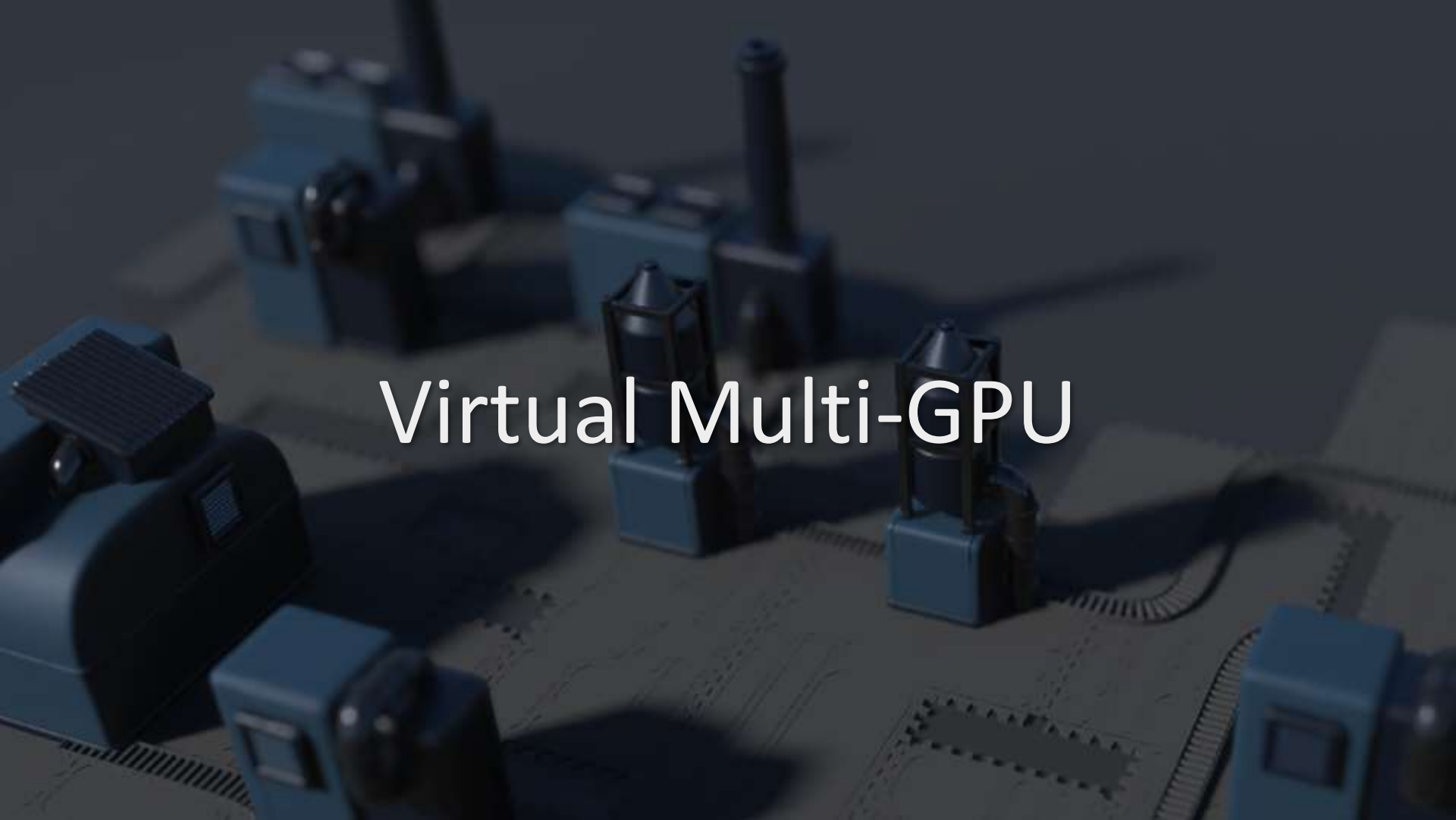


Render Graph

- Live debugging overlay
- Evaluated passes in-order of execution
- Input and output dependencies
- Resource version information




Halcyon Browser - SGPU - NVIDIA TITAN V
File Window Scene Display View Camera Debug GPU
View Mode: Default
Display Mode: Full
▼ Render Graph
Pass: Reflection Filter Pass
Reflection Filter Uniforms (version:0)
Reflection Temporal 0 (version:0)
Reflection Temporal 1 (version:0)
Reflection Meta Temporal 0 (version:0)
Reflection Meta Temporal 1 (version:0)
Reflection Hit0 Texture (version:2)
Reflection Hit1 Texture (version:2)
Inputs
Reflection Filter Uniforms (version:1)
Reflection Temporal 0 (version:1)
Reflection Meta Temporal 0 (version:1)
Reflection Output Texture (version:1)
Reflection Meta Output Texture (version:1)
Reflection Variance Output Texture (version:1)
Reflection Velocity Texture (version:1)
Outputs
Pass: Reflection Filter 2 Pass
Reflection Filter Uniforms (version:0)
Reflection Output Texture (version:1)
Reflection Meta Output Texture (version:1)
Reflection Variance Output Texture (version:1)
Reflection Output Texture (version:0)
Gbuffer History 0 (version:2)
Depth (version:2)
Inputs
Reflection Filter Uniforms (version:1)
Reflection Output Texture (version:1)
Outputs
Pass: Reflection Merge Pass
Reflection Output Texture (version:1)
Lighting (version:1)
Inputs
Lighting (version:2)
Outputs
Pass: Temporal AA Pass
Lighting (version:2)
TAA History Texture 0 (version:0)
Temporal Reprojection Texture (version:1)
TAA History Texture 1 (version:0)
TAA Output Texture (version:0)
Inputs
TAA History Texture 1 (version:1)
TAA Output Texture (version:1)
Outputs
Pass: Velocity reduce
Velocity Texture 0 (version:2)
Reduced velocity texture (version:0)
Inputs
Reduced velocity texture (version:1)
Outputs
Pass: Velocity reduce
Reduced velocity texture (version:1)
Reduced velocity texture (version:0)
Inputs
Reduced velocity texture (version:1)
Outputs
Pass: Velocity dilate
Reduced velocity texture (version:1)
Dilated velocity texture (version:0)
Inputs
Dilated velocity texture (version:1)
Outputs
Pass: Motion blur
Depth (version:2)
Velocity Texture 0 (version:2)
Dilated velocity texture (version:1)
TAA Output Texture (version:1)
Motion blurred image (version:0)
Inputs
Motion blurred image (version:1)
Outputs
Pass: Bloom Extract Pass
THE METAVERSE
A Simple Guide
Author: TBD

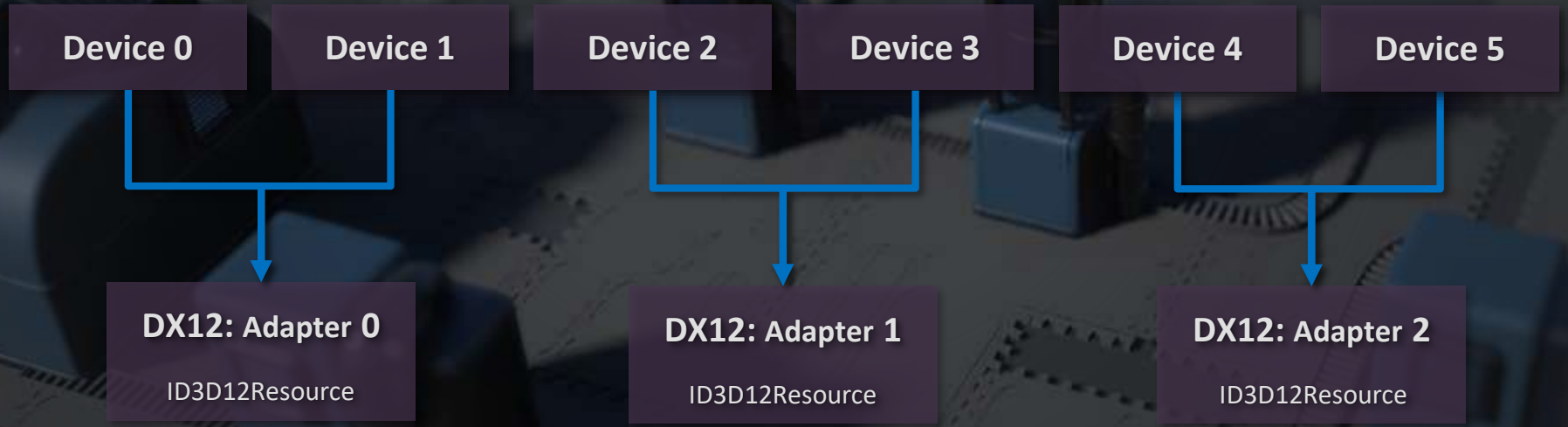

Virtual Multi-GPU

Virtual Multi-GPU

- Most developers have single GPU
- Uncommon for 2 GPU machines
- Rare for 3+ GPU
  - Practical for show floor and cranking up to 11
  - Impractical for regular development ☺

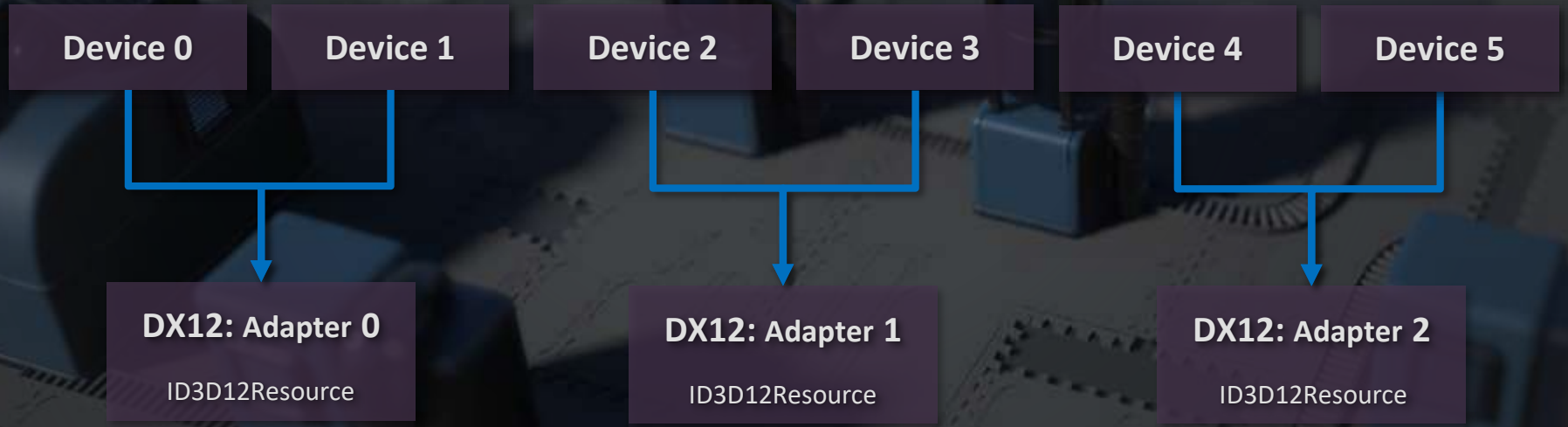
Virtual Multi-GPU

- Build device indirection table
- Virtual device index → adapter index



Virtual Multi-GPU

- Create multiple instances of a device
- Virtual GPUs execute sequentially (WDDM)



Virtual Multi-GPU

- Increases overall wall time (don't use for profiling)
- Amazing for development and testing!

Virtual Multi-GPU

- PICA PICA developers all had 1 GPU
- Limited testing with 2 GPUs
- Show floor at GDC 2018 was 4 GPUs
  - Virtual-only testing…
  - Crossed fingers
  - Worked flawlessly!

Virtual Multi-GPU

- Develop and debug multi-GPU with only a single GPU
- Virtual mGPU reliably reproduces most bugs!
- Entire features developed without physical mGPU
  - i.e. Surfel GI (the night before GDC.. ☺)


Render Proxy

Render Proxy

- Remote render backend
  - Any API / Any OS

Render Proxy

- Render API calls are routed remotely
- Uses gRPC (high performance RPC framework)
- Use an API on an incompatible OS
  - e.g. Direct3D 12 on macOS or Linux

Render Proxy

- Scale large workloads with a GPU cluster
  - Some API as render graph mGPU
- Only rendering is routed, scene state is local
- Work from the couch!
  - i.e. NV ray tracing on a MacBook ☺

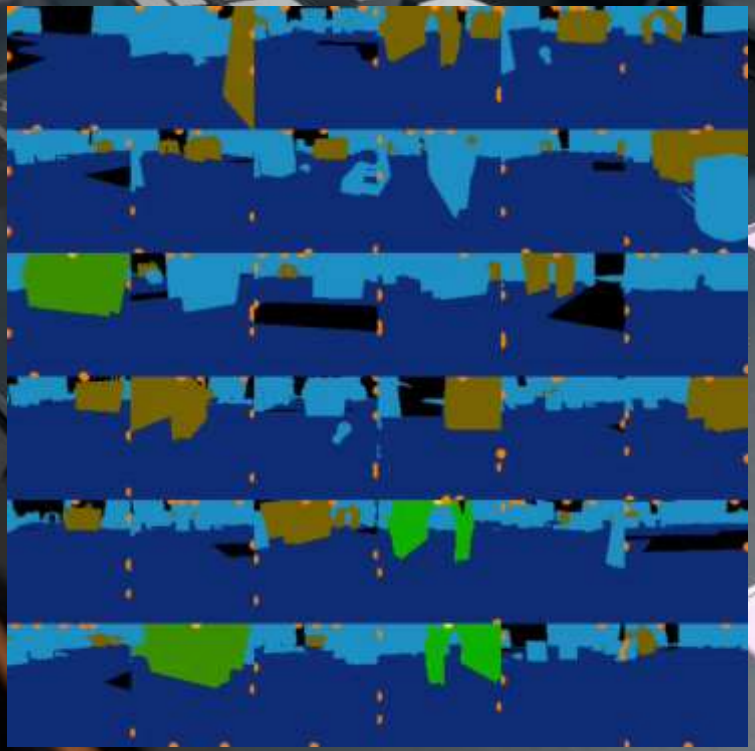
Render Proxy

- The possibilities are endless!


Machine Learning

Machine Learning

- Deep reinforcement learning
- Rendering 36 semantic views
- Training with TensorFlow
  - On-premise GPU cluster
- Inference with TensorFlow
  - CPU AVX2
  - In-process



Machine Learning

- Adding inferencing with DirectML
  - Hardware accelerated inferencing operators
  - Resource management
  - Schedule ML work explicitly
  - Interleave ML work with other GPU workloads
- Fall back for other APIs

Machine Learning

- Treat trained ML models like any other 3D asset
- Render Graph abstractions
  - Reference the same render resources
  - Similar to chaining compute passes
- Record “meta” render commands
  - Backends can fuse or transform, if desired

Asset Pipelines

Asset Pipelines

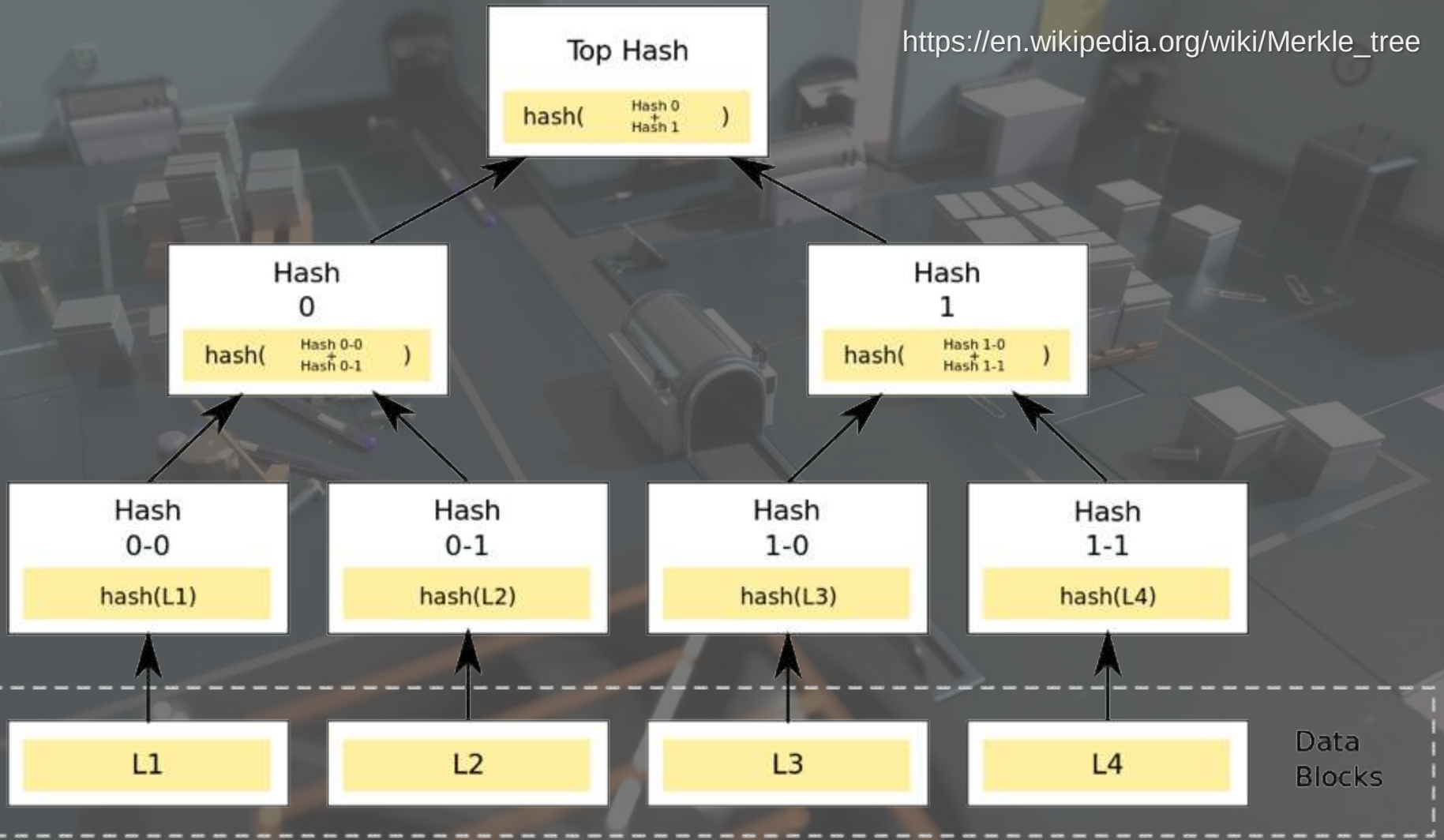
- Geometry
- Animations
- Shaders
- Sounds
- Music
- Textures
- Scenes
- etc.

100% Rust!

Asset Pipelines

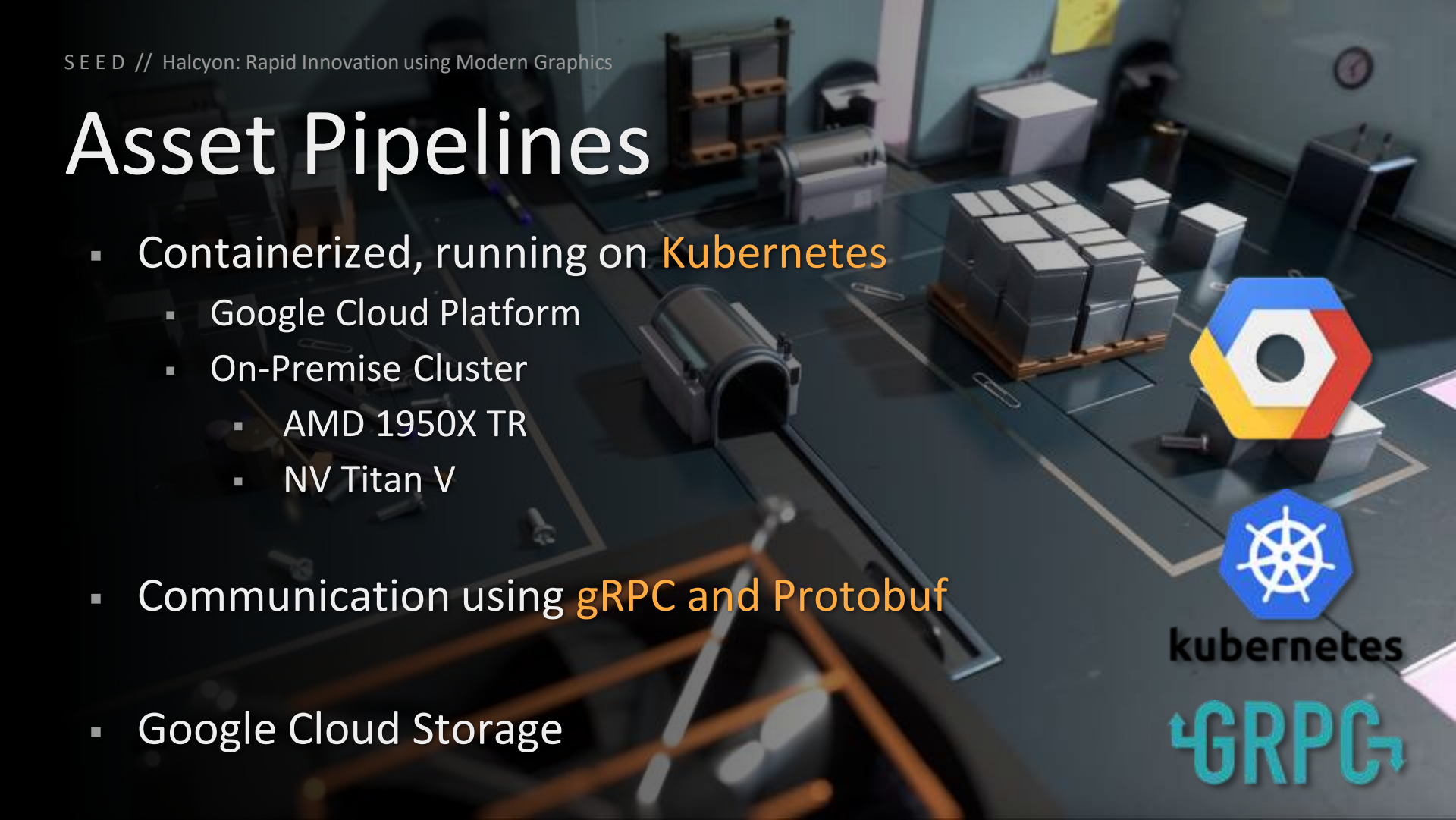
- Everything is content addressable
  - Hash of data is the identity
  - Sha256
- Merkle trees!
  - Dependency evaluation

https://en.wikipedia.org/wiki/Merkle_tree



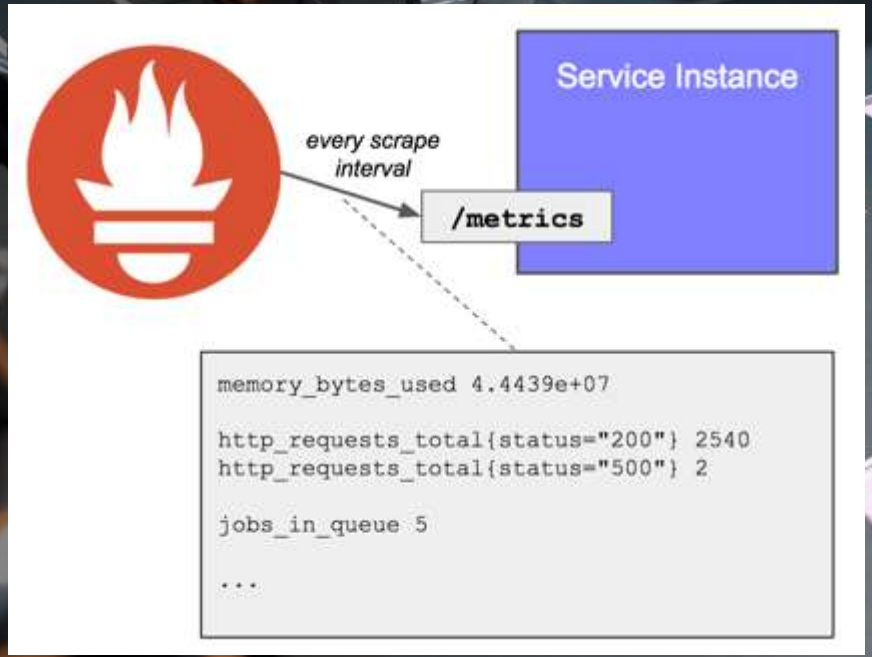
Asset Pipelines

- Containerized, running on Kubernetes
  - Google Cloud Platform
  - On-Premise Cluster
    - AMD 1950X TR
    - NV Titan V
- Communication using gRPC and Protobuf
- Google Cloud Storage



Asset Pipelines

- Analytics with Prometheus and Grafana
  - Publish custom metrics
  - Scraped into rich UI
  - Collecting data is important!




GPUs
29
GPU Utilization
1.0%
Memory Utilization
0%
GPU Utilization / Card
Memory Utilization / Card
GPU Temperature
Power Usage
Clock (Shader)
Clock (Streaming)
Clock (Pe/Encoder)
Nvidia driver version


Shaders

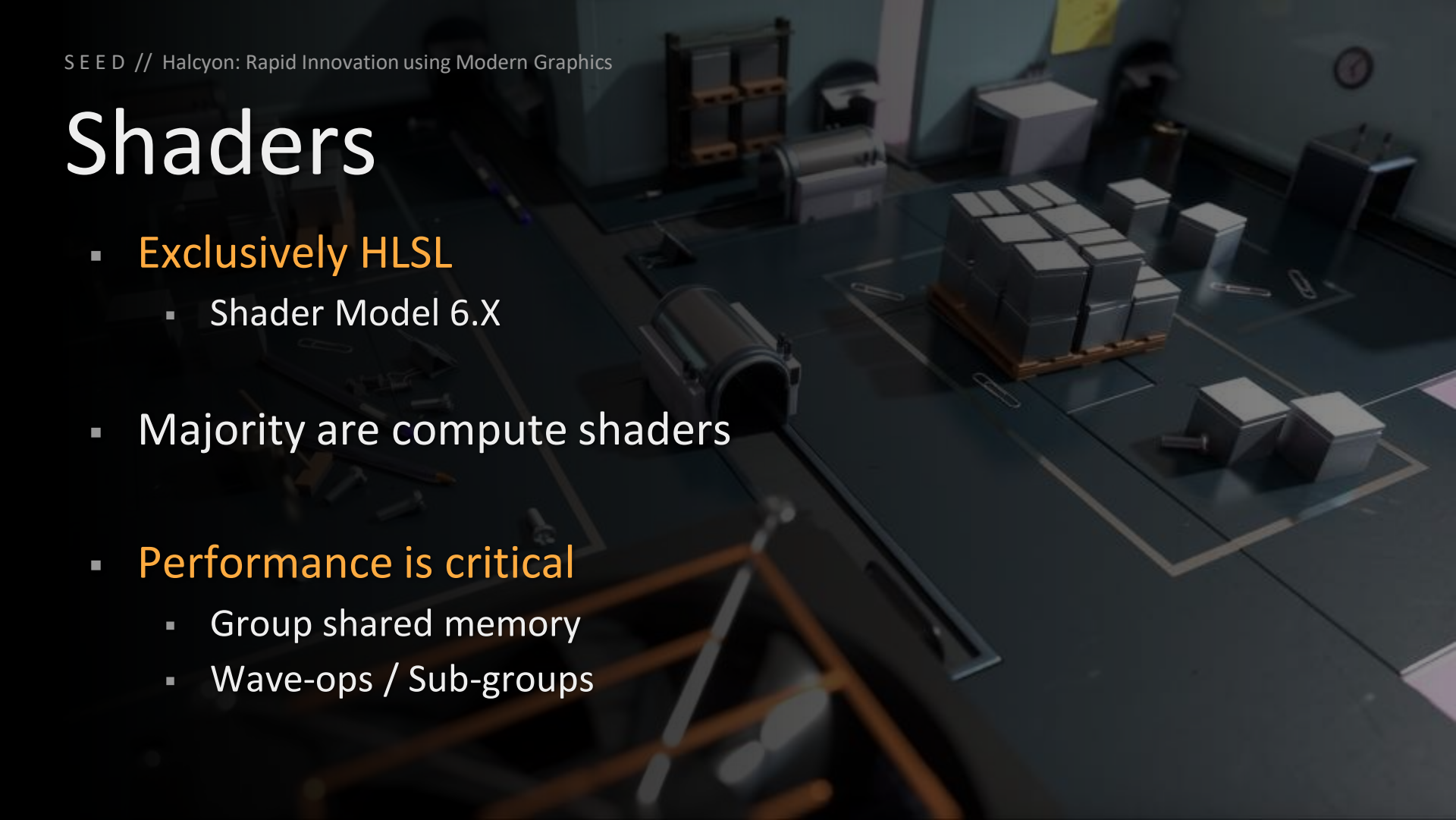
Shaders

- Complex materials
  - Multiple microfacet layers
  - **[Stachowiak 2018]**
- Energy conserving
  - Automatic Fresnel between layers
- All lighting & rendering modes
  - Raster, path-traced reference, hybrid
- Iterate with different looks
  - Bake down permutations for production

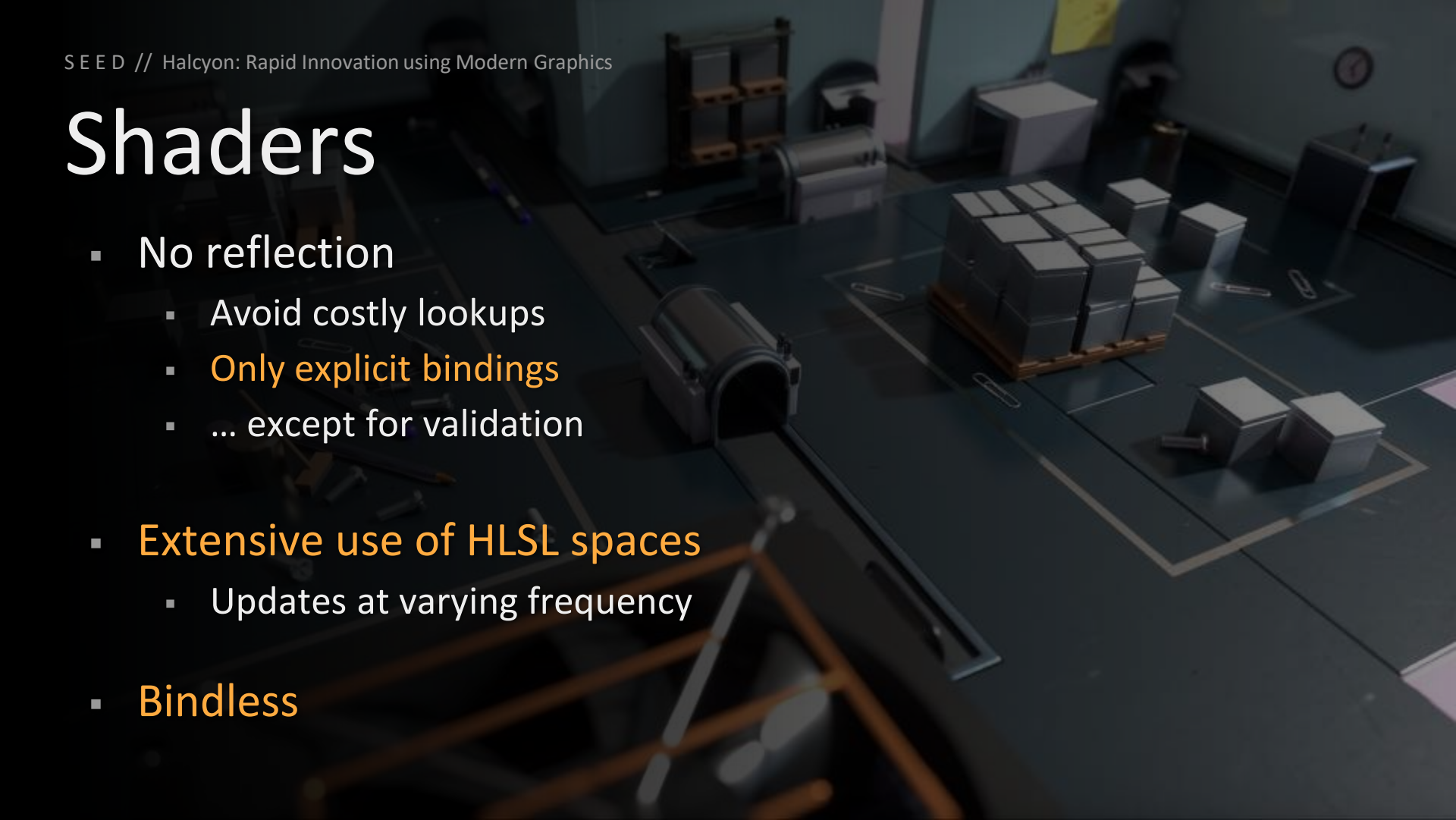
Objects with Multi-Layered Materials

Shaders

- Exclusively HLSL
  - Shader Model 6.X
- Majority are compute shaders
- Performance is critical
  - Group shared memory
  - Wave-ops / Sub-groups

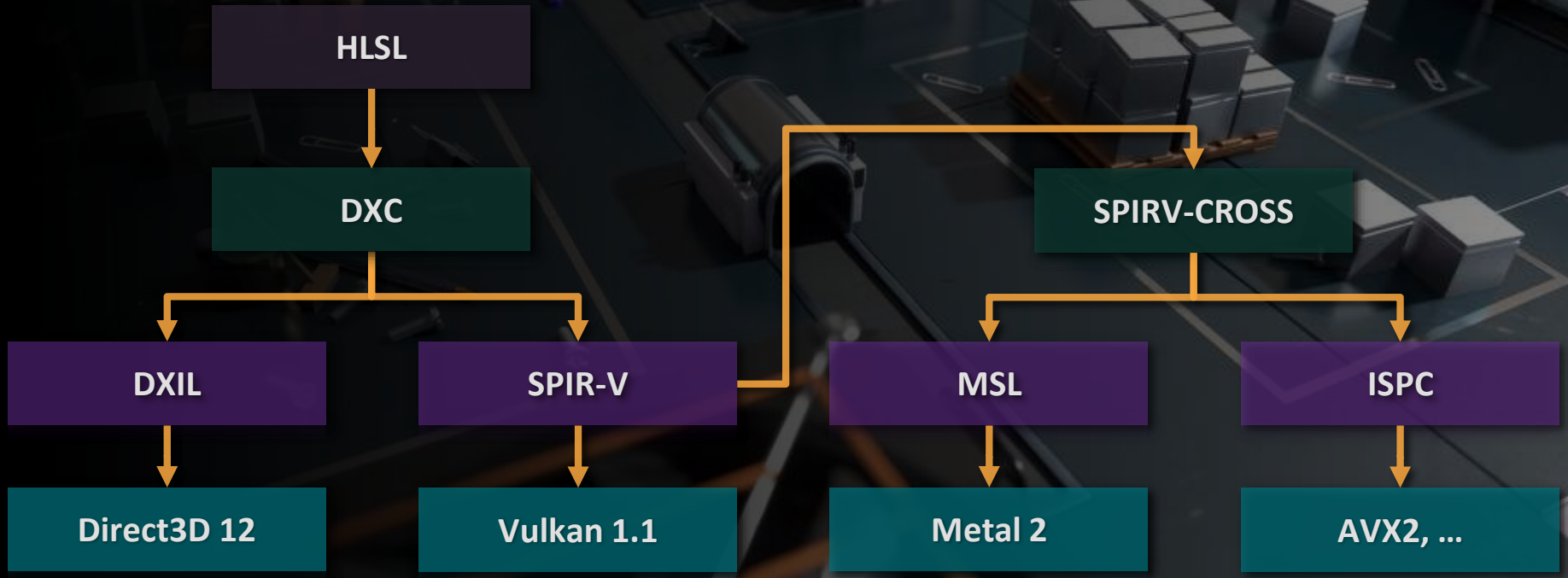


Shaders



- No reflection
  - Avoid costly lookups
  - Only explicit bindings
  - … except for validation
- Extensive use of HLSL spaces
  - Updates at varying frequency
- Bindless

Shaders


HLSL
DXC
SPIRV-CROSS
DXIL
SPIR-V
MSL
ISPC
Direct3D 12
Vulkan 1.1
Metal 2
AVX2, …

Shader Arguments

- Commands refer to resources with “Shader Arguments”
  - Each argument represents an HLSL space
  - MaxShaderParameters → 4 [Configurable]
  - # of spaces, not # of resources

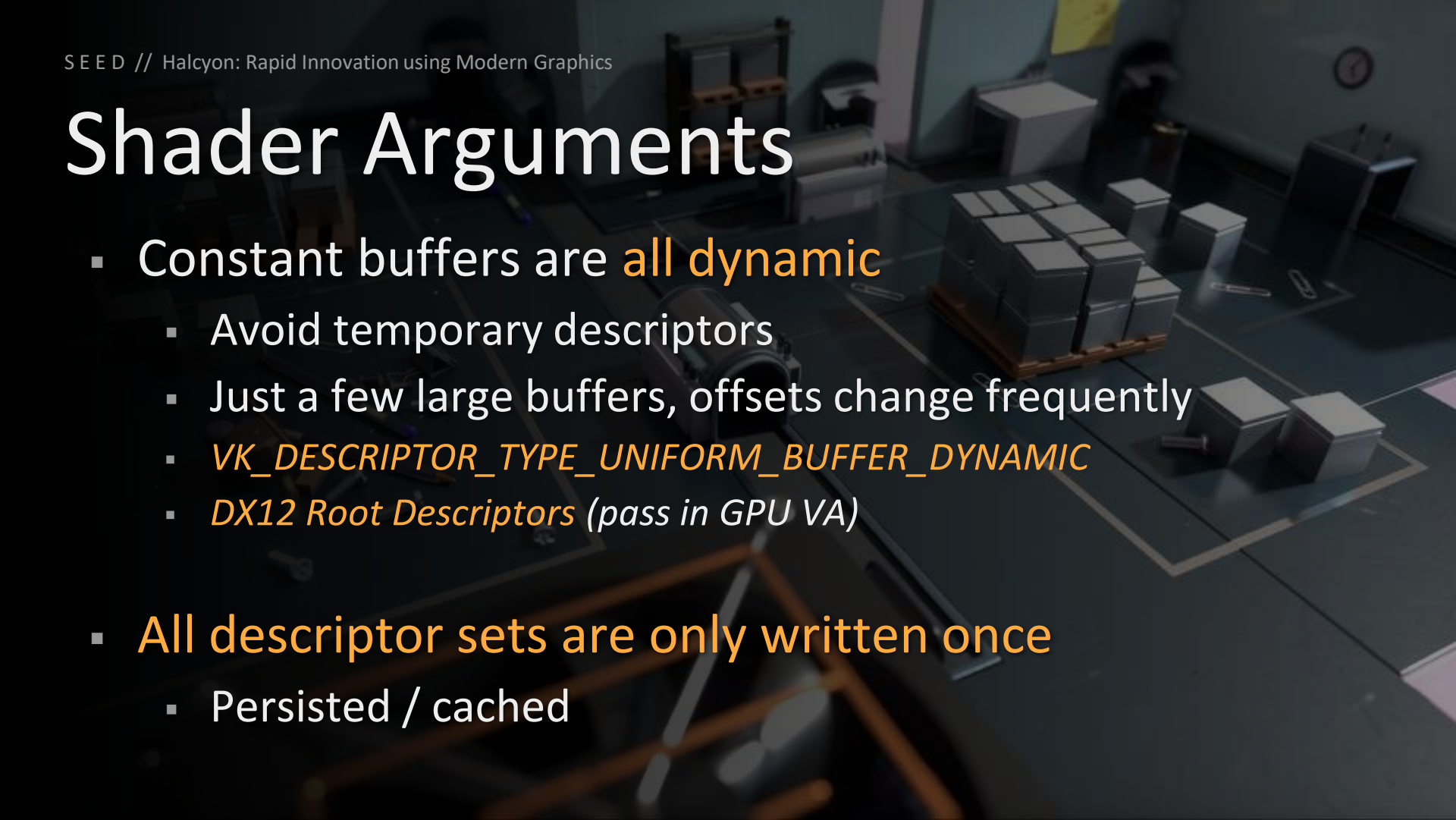
```
struct RenderCommandDispatch : RenderCommandTyped<RenderCommandType::Dispatch, RenderCommandQueueType::Compute>
{
    RenderResourceHandle pipelineState;
    ShaderArgument shaderArguments[MaxShaderParameters];
    uint32 shaderArgumentsCount = 0;

    uint32 dispatchX = 0;
    uint32 dispatchY = 0;
    uint32 dispatchZ = 0;
};
```

Shader Arguments

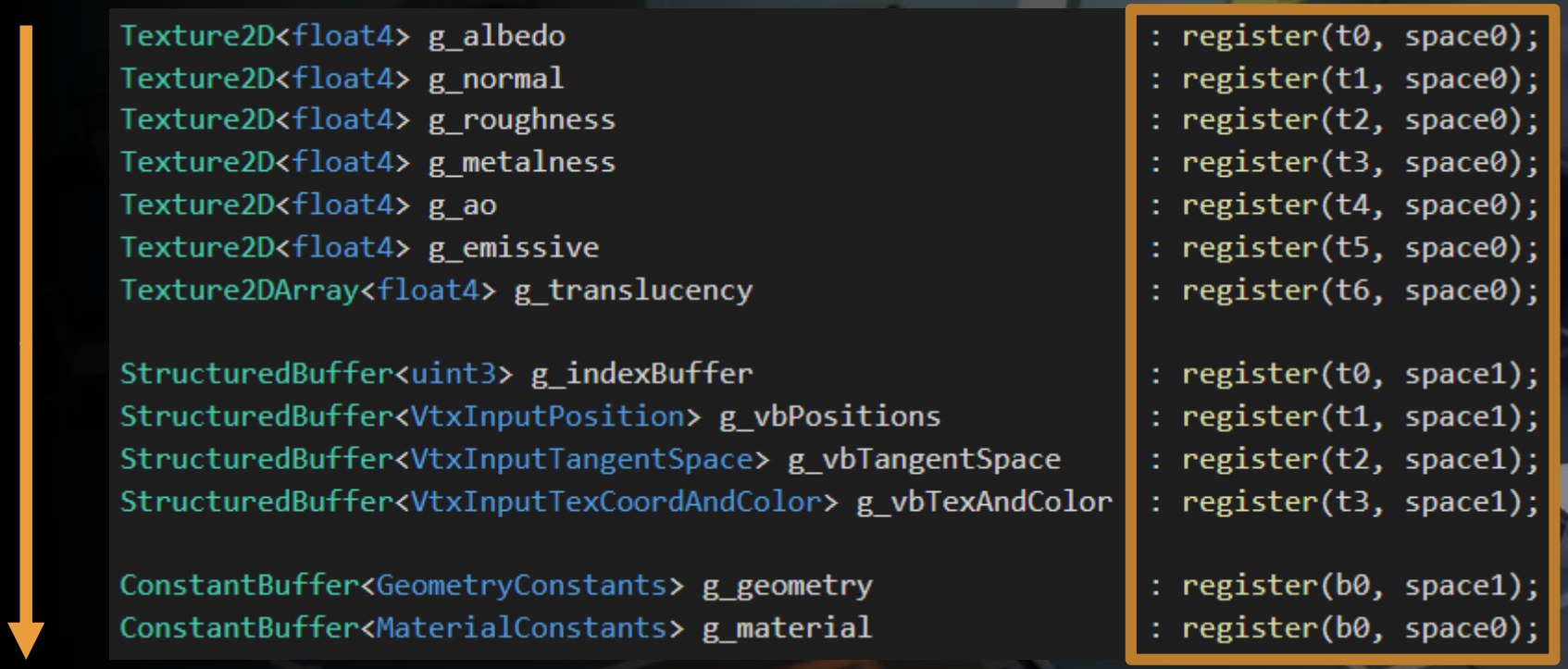
- Each argument contains:
  - “ShaderViews” handle
  - Constant buffer handle and offset

- “ShaderViews”
  - Collection of SRV and UAV handles

Shader Arguments



- Constant buffers are all dynamic
  - Avoid temporary descriptors
  - Just a few large buffers, offsets change frequently
  - *VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC*
  - *DX12 Root Descriptors (pass in GPU VA)*
- All descriptor sets are only written once
  - Persisted / cached



```
Texture2D<float4> g_albedo                          : register(t0, space0);
Texture2D<float4> g_normal                          : register(t1, space0);
Texture2D<float4> g_roughness                       : register(t2, space0);
Texture2D<float4> g_metalness                       : register(t3, space0);
Texture2D<float4> g_ao                              : register(t4, space0);
Texture2D<float4> g_emissive                        : register(t5, space0);
Texture2DArray<float4> g_translucency               : register(t6, space0);

StructuredBuffer<uint3> g_indexBuffer               : register(t0, space1);
StructuredBuffer<VtxInputPosition> g_vbPositions    : register(t1, space1);
StructuredBuffer<VtxInputTangentSpace> g_vbTangentSpace : register(t2, space1);
StructuredBuffer<VtxInputTexCoordAndColor> g_vbTexAndColor : register(t3, space1);

ConstantBuffer<GeometryConstants> g_geometry        : register(b0, space1);
ConstantBuffer<MaterialConstants> g_material        : register(b0, space0);
```

```
{
    // space0
    ShaderArgument(materialConstants.buffer, materialConstantsOffset, materialShaderViews),

    // space1
    ShaderArgument(geometryConstants.buffer, geometryConstantsOffset, geometryShaderViews)
},
```

SPIR-V Patching

- Shader compilation (HLSL → SPIR-V)
  - Patch SPIR-V to match DX12
  - Using **spirv-reflect** from Hai and Cort

- spvReflectCreateShaderModule
- **spvReflectEnumerateDescriptorSets**
- **spvReflectChangeDescriptorBindingNumbers**
- spvReflectGetCodeSize / spvReflectGetCode
- spvReflectDestroyShaderModule

SPIR-V Patching

- **SPV_REFLECT_RESOURCE_FLAG_SRV**
  - Offset += 1000
- **SPV_REFLECT_RESOURCE_FLAG_SAMPLER**
  - Offset += 2000
- **SPV_REFLECT_RESOURCE_FLAG_UAV**
  - Offset += 3000

SPIR-V Patching

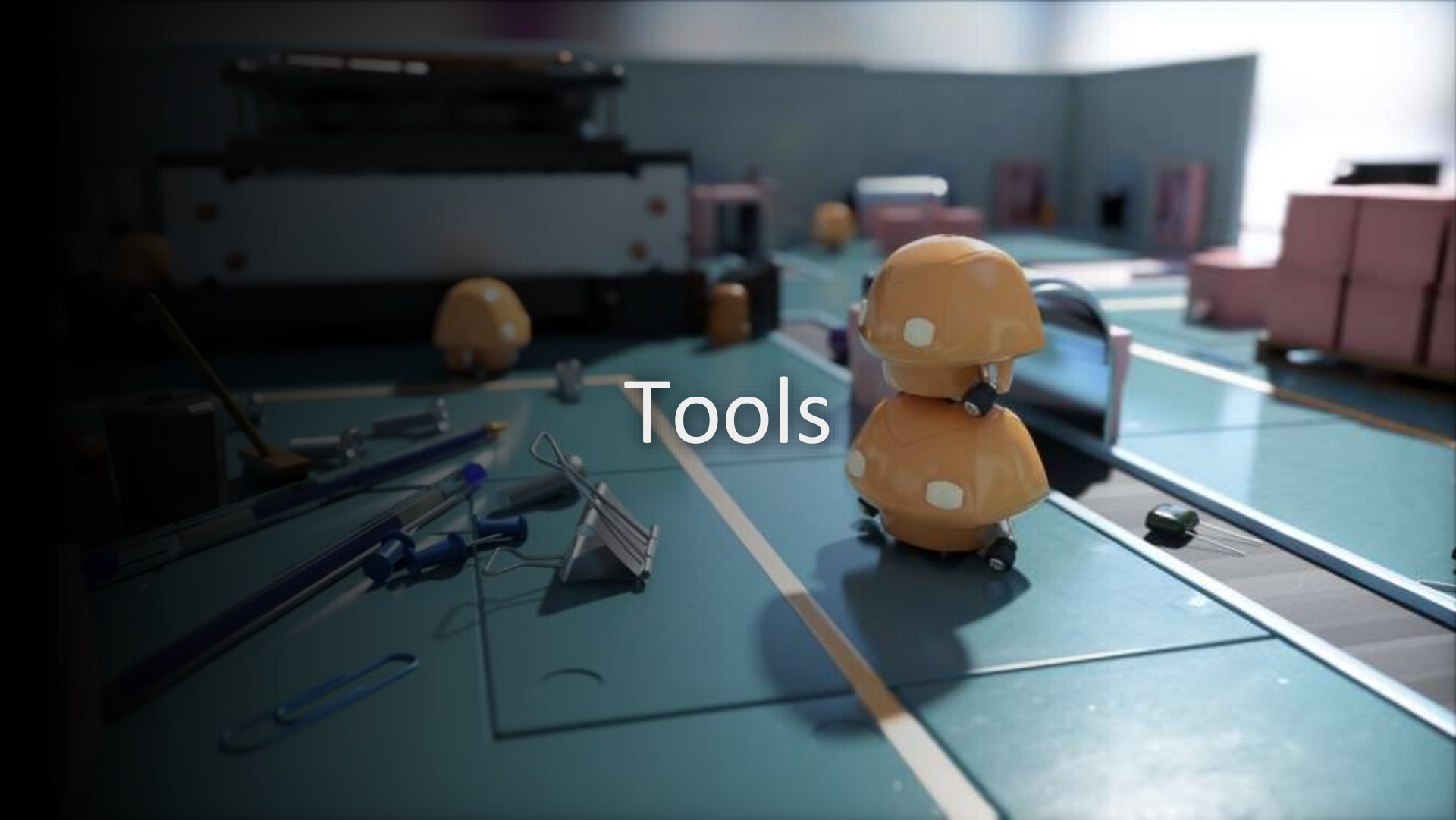
- **SPV_REFLECT_RESOURCE_FLAG_CBV**
  - Offset Unchanged: 0
  - Descriptor Set += **MAX_SHADER_ARGUMENTS**

- CBVs move to their own descriptor sets
  - ShaderViews become persistent and immutable

SPIR-V Patching

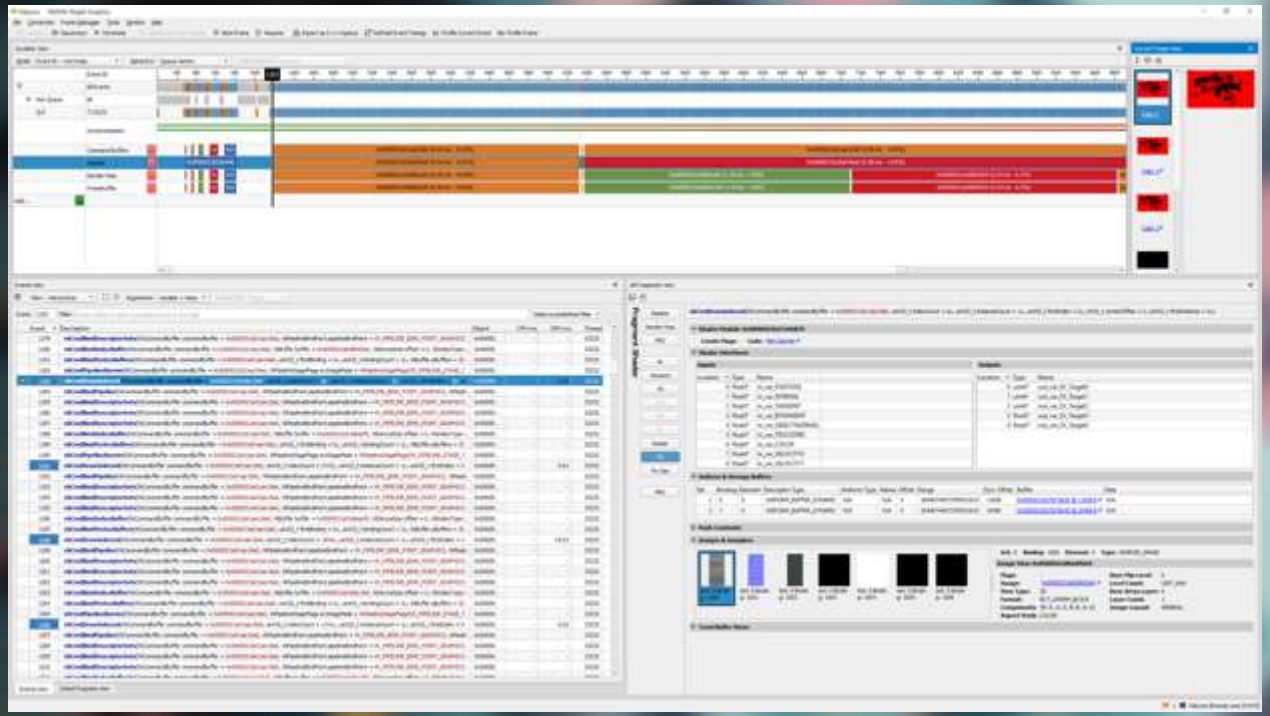
- If 2 of 4 HLSL spaces in use:

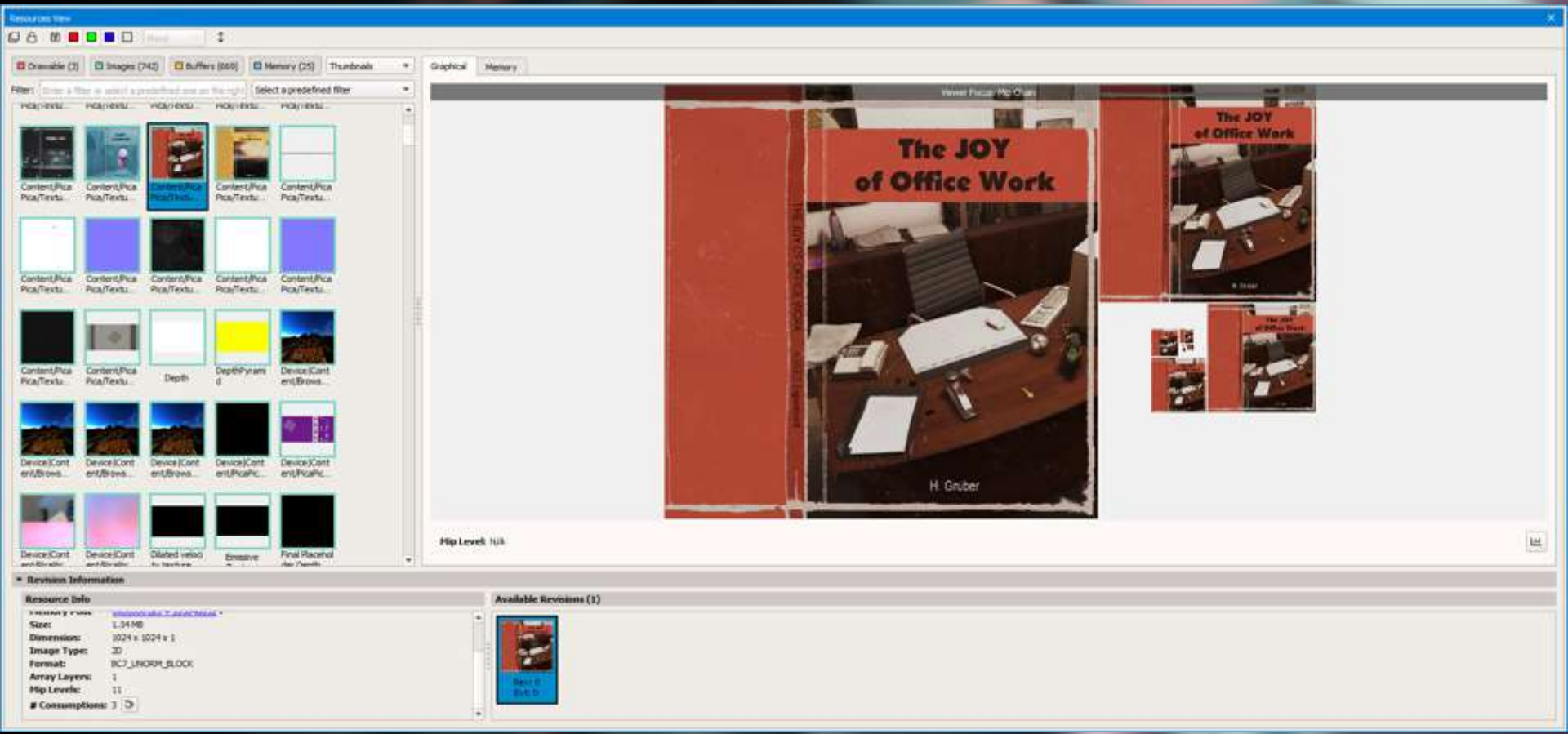
Set 0	SRVs (>=1000)	Samplers (>=2000)	UAVs (>=3000)
Set 1	SRVs (>=1000)	Samplers (>=2000)	UAVs (>=3000)
Set 2	Unbound		
Set 3	Unbound		
Set 4	Dynamic Constant Buffer (Offset: 0)		
Set 5	Dynamic Constant Buffer (Offset: 0)		


Tools

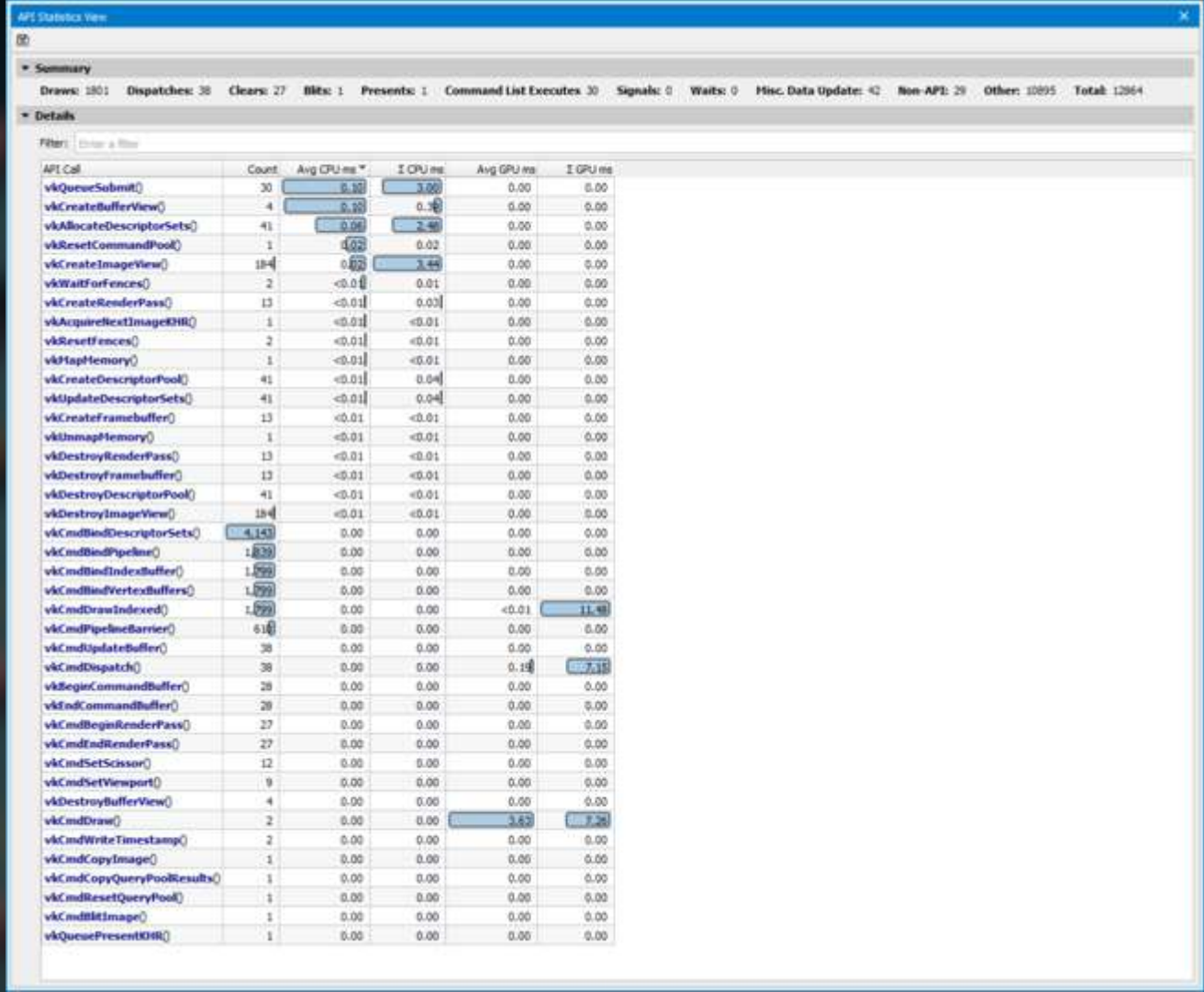
Tools

- RenderDoc
- NV Nsight
- AMD RGP



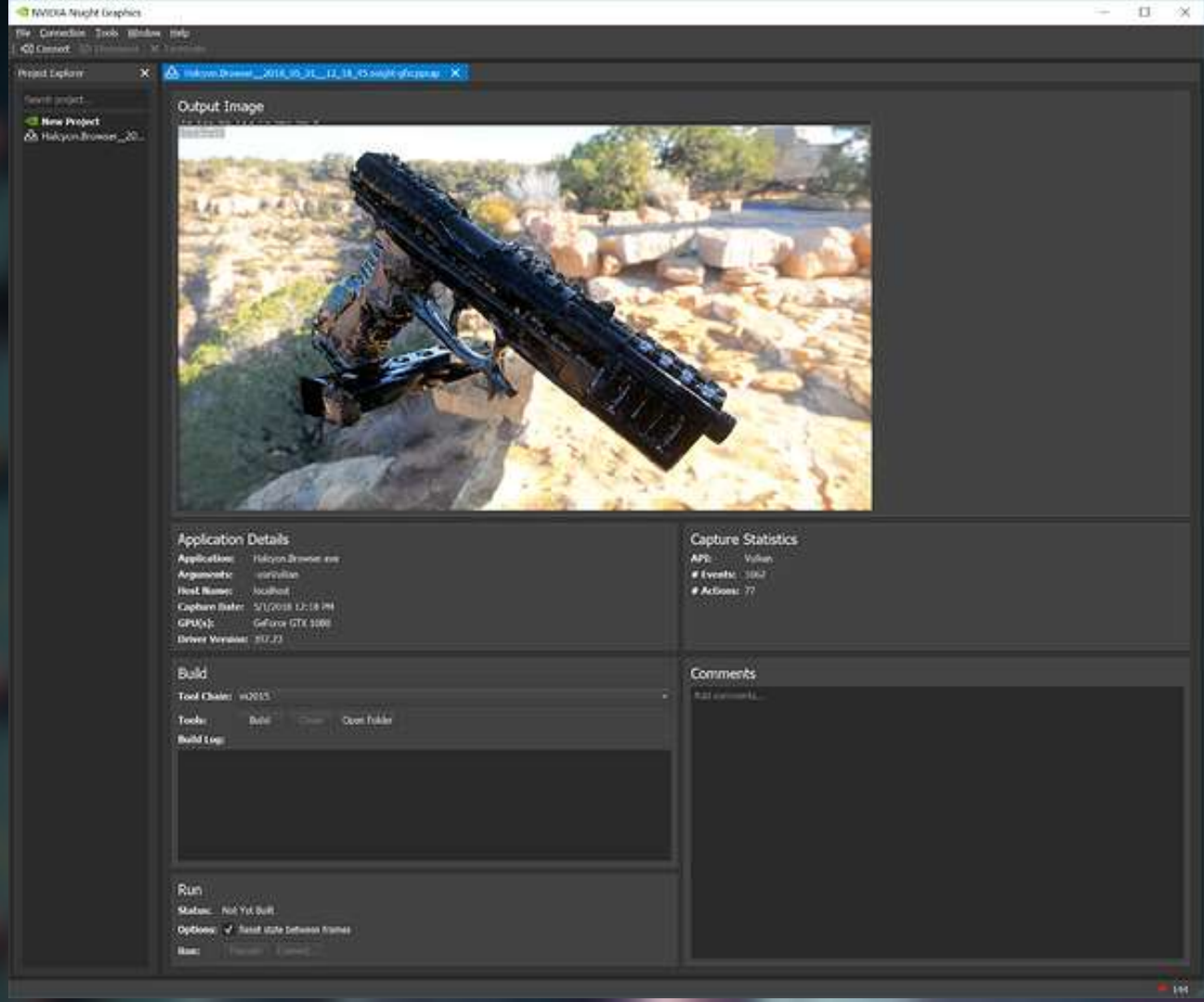

Resources View
Graphical
Memory
Thumbnails
Select a predefined filter
The JOY of Office Work
H. Gruber
Mip Level: N/A
Revision Information
Resource Info
Available Revisions (1)
Size: 1.34MB
Dimension: 1024 x 1024 x 1
Image Type: 2D
Format: BC7_UNORM_BLOCK
Array Layers: 1
Mip Levels: 11
Consumptions: 3

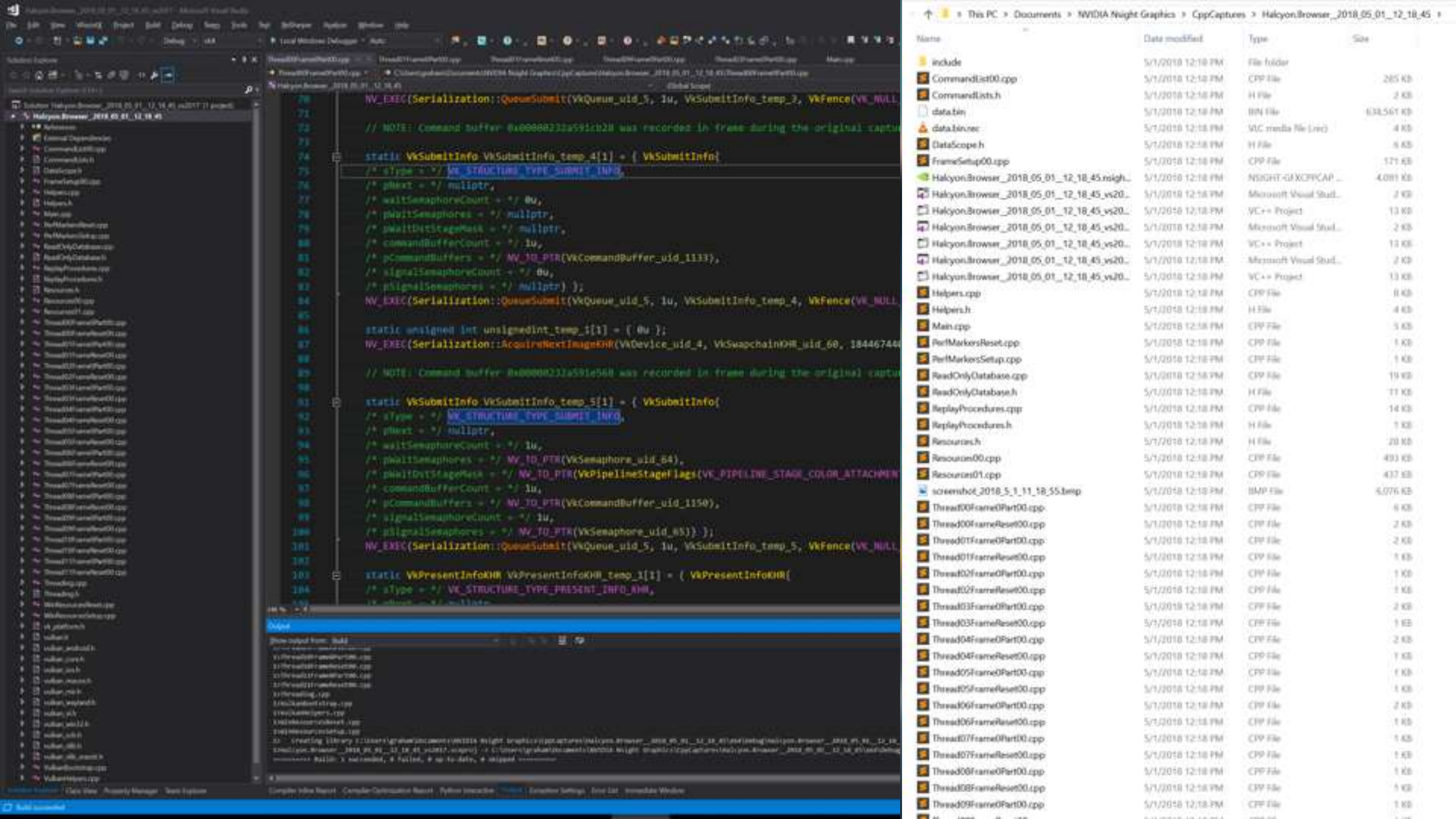
Tools


API Statistics View
Summary
Draws: 1801 Dispatches: 38 Clears: 27 Blits: 1 Presents: 1 Command List Executes: 30 Signals: 0 Waits: 0 Misc. Data Update: 42 Non-API: 29 Other: 10895 Total: 12864
Details
Filter:
API Call | Count | Avg CPU ms | Σ CPU ms | Avg GPU ms | Σ GPU ms
vkQueueSubmit() 30 0.10 3.00 0.00 0.00
vkCreateBufferView() 4 0.10 0.39 0.00 0.00
vkAllocateDescriptorSets() 41 0.06 2.40 0.00 0.00
vkResetCommandPool() 1 0.02 0.02 0.00 0.00
vkCreateImageView() 184 0.02 3.44 0.00 0.00
vkWaitForFences() 2 <0.01 0.01 0.00 0.00
vkCreateRenderPass() 13 <0.01 0.03 0.00 0.00
vkAcquireNextImageKHR() 1 <0.01 <0.01 0.00 0.00
vkResetFences() 2 <0.01 <0.01 0.00 0.00
vkMapMemory() 1 <0.01 <0.01 0.00 0.00
vkCreateDescriptorPool() 41 <0.01 0.04 0.00 0.00
vkUpdateDescriptorSets() 41 <0.01 0.04 0.00 0.00
vkCreateFramebuffer() 13 <0.01 <0.01 0.00 0.00
vkUnmapMemory() 1 <0.01 <0.01 0.00 0.00
vkDestroyRenderPass() 13 <0.01 <0.01 0.00 0.00
vkDestroyFramebuffer() 13 <0.01 <0.01 0.00 0.00
vkDestroyDescriptorPool() 41 <0.01 <0.01 0.00 0.00
vkDestroyImageView() 184 <0.01 <0.01 0.00 0.00
vkCmdBindDescriptorSets() 4,143 0.00 0.00 0.00 0.00
vkCmdBindPipeline() 1,839 0.00 0.00 0.00 0.00
vkCmdBindIndexBuffer() 1,799 0.00 0.00 0.00 0.00
vkCmdBindVertexBuffers() 1,799 0.00 0.00 0.00 0.00
vkCmdDrawIndexed() 1,799 0.00 0.00 <0.01 11.48
vkCmdPipelineBarrier() 618 0.00 0.00 0.00 0.00
vkCmdUpdateBuffer() 38 0.00 0.00 0.00 0.00
vkCmdDispatch() 38 0.00 0.00 0.19 7.15
vkBeginCommandBuffer() 28 0.00 0.00 0.00 0.00
vkEndCommandBuffer() 28 0.00 0.00 0.00 0.00
vkCmdBeginRenderPass() 27 0.00 0.00 0.00 0.00
vkCmdEndRenderPass() 27 0.00 0.00 0.00 0.00
vkCmdSetScissor() 12 0.00 0.00 0.00 0.00
vkCmdSetViewport() 9 0.00 0.00 0.00 0.00
vkDestroyBufferView() 4 0.00 0.00 0.00 0.00
vkCmdDraw() 2 0.00 0.00 3.62 7.24
vkCmdWriteTimestamp() 2 0.00 0.00 0.00 0.00
vkCmdCopyImage() 1 0.00 0.00 0.00 0.00
vkCmdCopyQueryPoolResults() 1 0.00 0.00 0.00 0.00
vkCmdResetQueryPool() 1 0.00 0.00 0.00 0.00
vkCmdBlitImage() 1 0.00 0.00 0.00 0.00
vkQueuePresentKHR() 1 0.00 0.00 0.00 0.00

Tools

- C++ Export!
- Standalone

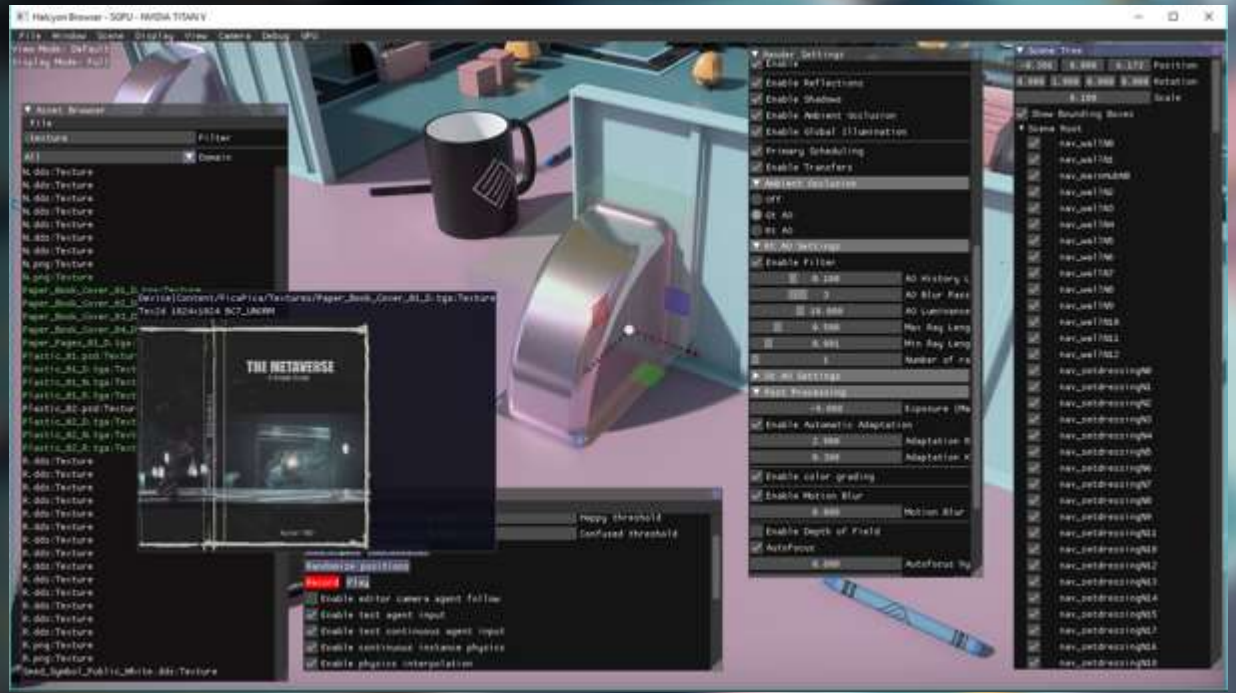


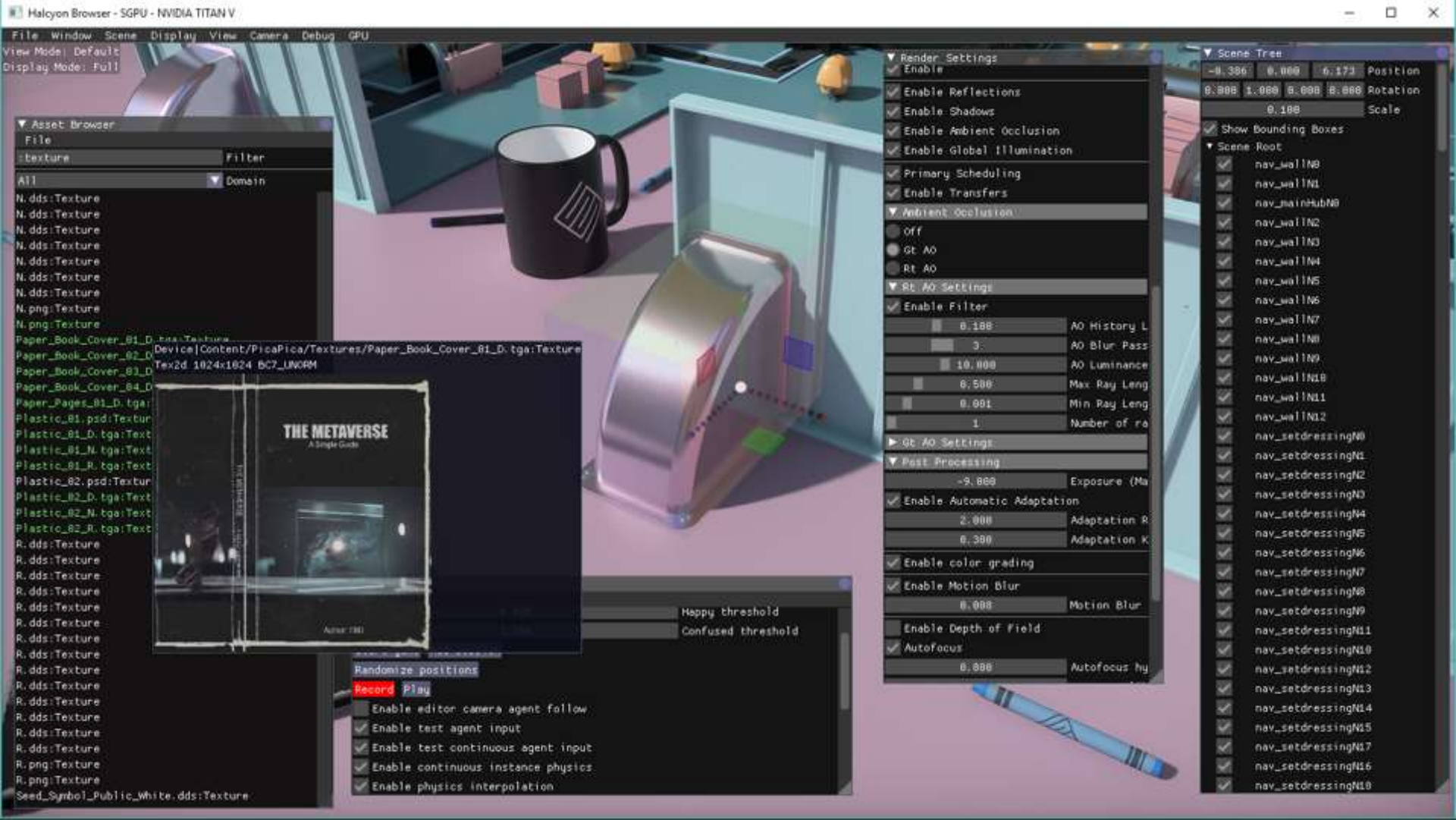

This PC > Documents > NVIDIA Nsight Graphics > CppCaptures > Halcyon.Browser_2018_05_01__12_18_45
Name
Date modified
Type
Size
include
CommandList00.cpp
CommandLists.h
data.bin
data.bin.rec
DataScope.h
FrameSetup00.cpp
Helpers.cpp
Helpers.h
Main.cpp
PerfMarkersReset.cpp
PerfMarkersSetup.cpp
ReadOnlyDatabase.cpp
ReadOnlyDatabase.h
ReplayProcedures.cpp
ReplayProcedures.h
Resources.h
Resources00.cpp
Resources01.cpp
Thread00Frame0Part00.cpp
Thread00FrameReset00.cpp
Thread01Frame0Part00.cpp
Thread01FrameReset00.cpp
Thread02Frame0Part00.cpp
Thread02FrameReset00.cpp
Thread03Frame0Part00.cpp
Thread03FrameReset00.cpp
Thread04Frame0Part00.cpp
Thread04FrameReset00.cpp
Thread05Frame0Part00.cpp
Thread05FrameReset00.cpp
Thread06Frame0Part00.cpp
Thread06FrameReset00.cpp
Thread07Frame0Part00.cpp
Thread07FrameReset00.cpp
Thread08Frame0Part00.cpp
Thread08FrameReset00.cpp
Thread09Frame0Part00.cpp
static VkSubmitInfo VkSubmitInfo_temp_4[1] = { VkSubmitInfo{
/* sType = */ VK_STRUCTURE_TYPE_SUBMIT_INFO,
/* pNext = */ nullptr,
/* waitSemaphoreCount = */ 0u,
/* pWaitSemaphores = */ nullptr,
/* pWaitDstStageMask = */ nullptr,
/* commandBufferCount = */ 1u,
/* pCommandBuffers = */ NV_TO_PTR(VkCommandBuffer_uid_1133),
/* signalSemaphoreCount = */ 0u,
/* pSignalSemaphores = */ nullptr) };
NV_EXEC(Serialization::QueueSubmit(VkQueue_uid_5, 1u, VkSubmitInfo_temp_4, VkFence(VK_NULL_
static unsigned int unsignedint_temp_1[1] = { 0u };
static VkSubmitInfo VkSubmitInfo_temp_5[1] = { VkSubmitInfo{
/* sType = */ VK_STRUCTURE_TYPE_SUBMIT_INFO,
/* pNext = */ nullptr,
/* waitSemaphoreCount = */ 1u,
/* pWaitSemaphores = */ NV_TO_PTR(VkSemaphore_uid_64),
/* commandBufferCount = */ 1u,
/* pCommandBuffers = */ NV_TO_PTR(VkCommandBuffer_uid_1150),
/* signalSemaphoreCount = */ 1u,
/* pSignalSemaphores = */ NV_TO_PTR(VkSemaphore_uid_65)) };
NV_EXEC(Serialization::QueueSubmit(VkQueue_uid_5, 1u, VkSubmitInfo_temp_5, VkFence(VK_NULL_
static VkPresentInfoKHR VkPresentInfoKHR_temp_1[1] = { VkPresentInfoKHR{
/* sType = */ VK_STRUCTURE_TYPE_PRESENT_INFO_KHR,


Nvda.Replayer
File Window Scene Display View Camera Debug GPU

Dear ImGui + ImGuizmo

- Live tweaking
- Very useful!




Halcyon Browser - SGPU - NVIDIA TITAN V
File Window Scene Display View Camera Debug GPU
View Mode: Default
Display Mode: Full
Asset Browser
File
:texture
Filter
All
Domain
N.dds:Texture
N.png:Texture
Paper_Book_Cover_01_D.tga:Texture
Paper_Book_Cover_02_D
Paper_Book_Cover_03_D
Paper_Book_Cover_04_D
Paper_Pages_01_D.tga:
Plastic_01.psd:Textur
Plastic_01_D.tga:Text
Plastic_01_N.tga:Text
Plastic_01_R.tga:Text
Plastic_02.psd:Textur
Plastic_02_D.tga:Text
Plastic_02_N.tga:Text
Plastic_02_R.tga:Text
R.dds:Texture
R.png:Texture
Seed_Symbol_Public_White.dds:Texture
Device|Content/PicaPica/Textures/Paper_Book_Cover_01_D.tga:Texture
Tex2d 1024x1024 BC7_UNORM
THE METAVERSE
A Simple Guide
Happy threshold
Confused threshold
Randomize positions
Record
Play
Enable editor camera agent follow
Enable test agent input
Enable test continuous agent input
Enable continuous instance physics
Enable physics interpolation
Render Settings
Enable
Enable Reflections
Enable Shadows
Enable Ambient Occlusion
Enable Global Illumination
Primary Scheduling
Enable Transfers
Ambient Occlusion
Off
Gt AO
Rt AO
Rt AO Settings
Enable Filter
0.100
AO History L
3
AO Blur Pass
10.000
AO Luminance
0.500
Max Ray Leng
0.001
Min Ray Leng
1
Number of ra
Gt AO Settings
Post Processing
-9.000
Exposure (Ma
Enable Automatic Adaptation
2.000
Adaptation R
0.300
Adaptation K
Enable color grading
Enable Motion Blur
0.000
Motion Blur
Enable Depth of Field
Autofocus
0.000
Autofocus hy
Scene Tree
-0.386
0.000
6.173
Position
0.000
1.000
0.000
0.000
Rotation
0.100
Scale
Show Bounding Boxes
Scene Root
nav_wallN0
nav_wallN1
nav_mainHubN0
nav_wallN2
nav_wallN3
nav_wallN4
nav_wallN5
nav_wallN6
nav_wallN7
nav_wallN8
nav_wallN9
nav_wallN10
nav_wallN11
nav_wallN12
nav_setdressingN0
nav_setdressingN1
nav_setdressingN2
nav_setdressingN3
nav_setdressingN4
nav_setdressingN5
nav_setdressingN6
nav_setdressingN7
nav_setdressingN8
nav_setdressingN9
nav_setdressingN11
nav_setdressingN10
nav_setdressingN12
nav_setdressingN13
nav_setdressingN14
nav_setdressingN15
nav_setdressingN17
nav_setdressingN16
nav_setdressingN18

References

- **[Wihlidal 2018]** Graham Wihlidal, Colin Barré-Brisebois. “Modern Graphics Abstractions & Real-Time Ray Tracing” .
  available online
- **[Wihlidal 2018]** Graham Wihlidal. “Halcyon + Vulkan”.
  available online
- **[Stachowiak 2018]** Tomasz Stachowiak. “Towards Effortless Photorealism Through Real-Time Raytracing”.
  available online
- **[Andersson 2018]** Johan Andersson, Colin Barré-Brisebois.“*DirectX: Evolving Microsoft's Graphics Platform”.*
  available online
- **[Harmer 2018]** Jack Harmer, Linus Gisslén, Henrik Holst, Joakim Bergdahl, Tom Olsson, Kristoffer Sjöö and Magnus Nordin. “Imitation Learning with Concurrent Actions in 3D Games”.
  available online
- **[Opara 2018]** Anastasia Opara. “Creativity of Rules and Patterns”.
  available online
- **[O’Donnell 2017]** Yuriy O’Donnell. “Frame Graph: Extensible Rendering Architecture in Frostbite”.
  available online
- **[Mui 2012]** Chunka Mui**,** Forbes. “How Kodak Failed”.
  available online


SEED // SEARCH FOR EXTRAORDINARY EXPERIENCES DIVISION
STOCKHOLM – LOS ANGELES – MONTRÉAL – REMOTE
SEED.EA.COM
WE'RE HIRING!

Thank You

- Reboot Develop Blue & Khronos

- Halcyon Team @ SEED
  - Andrew Lauritzen
  - Colin Barré-Brisebois
  - Jasper Bekkers
  - Henrik Halén
  - Carlos Gonzalez-Ochoa
  - Graham Wihlidal

Graham Wihlidal
graham@ea.com
@gwihlidal

Questions?

www.khr.io/slack