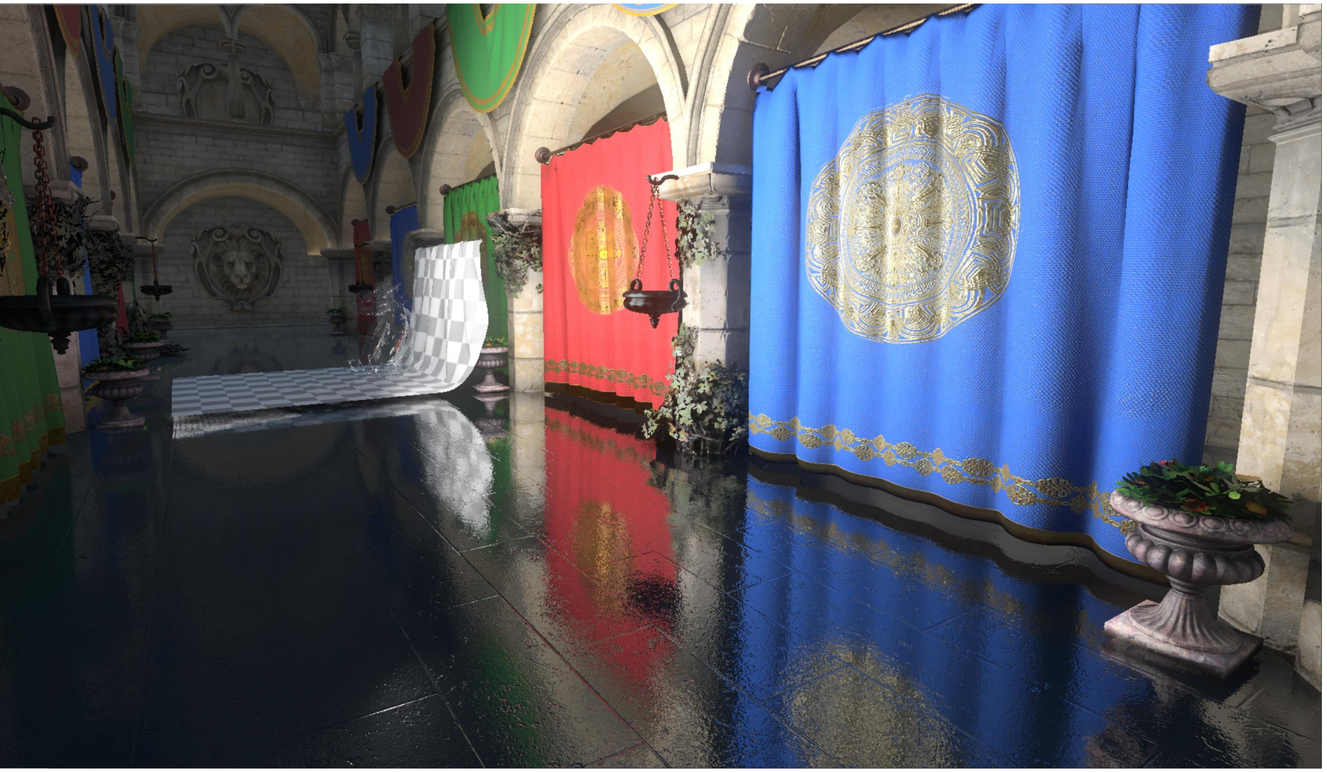
Presented at the Khronos Vulkanised 2023 Conference

NVIDIA Kickstart RT SDK Overview

Jakub Boksansky (NVIDIA) | jboksansky@nvidia.com

Kickstart RT SDK

An easy open-source ray tracing



- Goal: **Enable ray tracing features with minimal effort**
  - Raytraced reflections, GI, shadows and ambient occlusion
  - Uses simplified lighting and scene representation
- Cross-platform (Linux, ARM, Windows) and cross-API (Vulkan, Dx12 and Dx11)
- Open source: https://github.com/NVIDIAGameWorks/KickstartRT
  - Released under MIT license
- Comes with integrated denoiser (NRD)
  - NRD uses different license (NVIDIA RTX SDK License)

Kickstart RT SDK

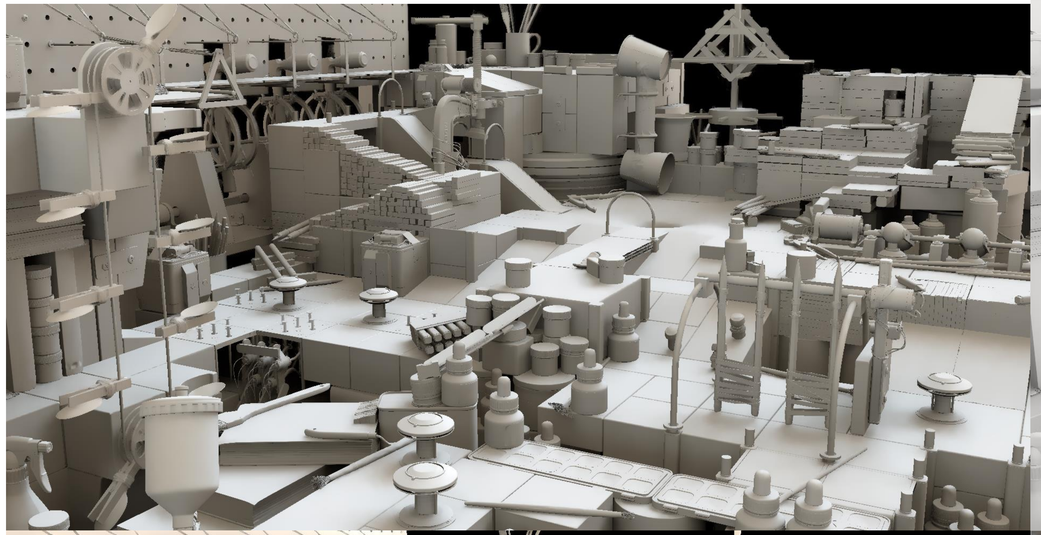
Why use this SDK?

- **Implementing ray tracing from scratch requires**
  - Building BVH of all scene geometry
    - Streaming, animation, instancing, memory allocations, …
  - Rewriting all materials and lighting to RT pipeline
    - Many shader variants
    - Complicated uber shader
    - Non-PBR materials/lighting
  - Denoising
- Kickstart RT SDK:
  - **Handles BVH builds, lighting and denoising internally**
  - With some simplifications and drawbacks
  - Supports Dx11 (via interop)

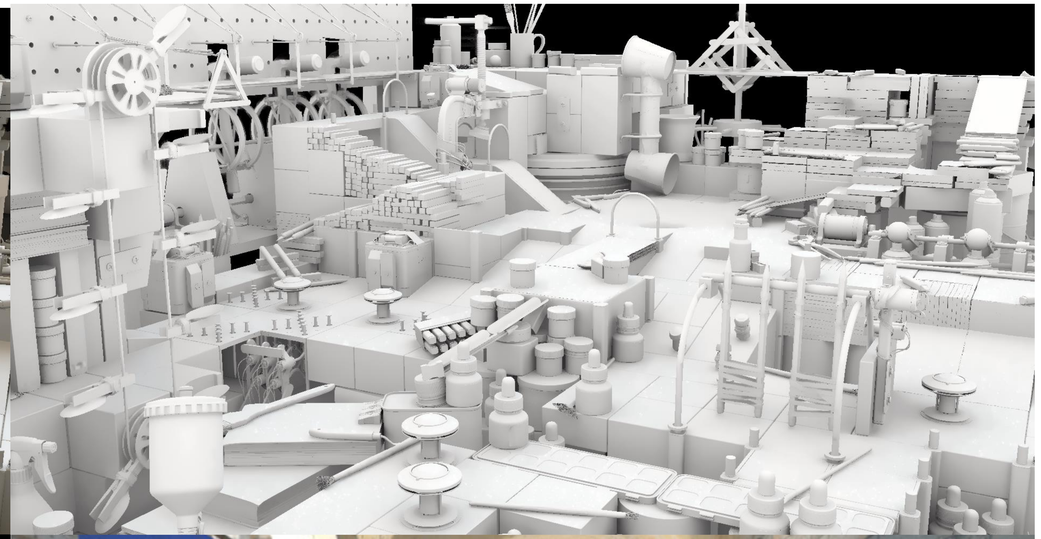
Kickstart RT SDK

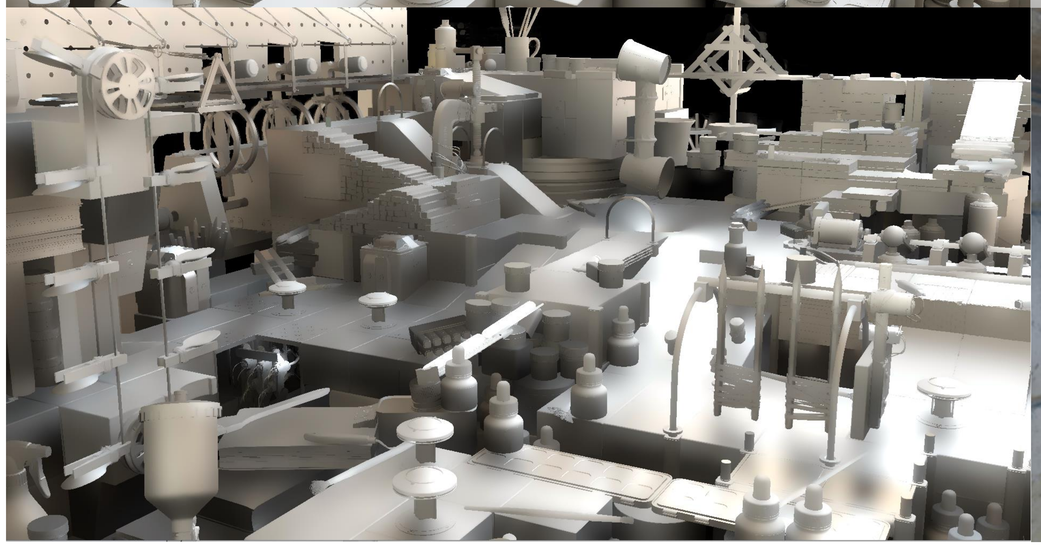
Supported effects

Diffuse GI

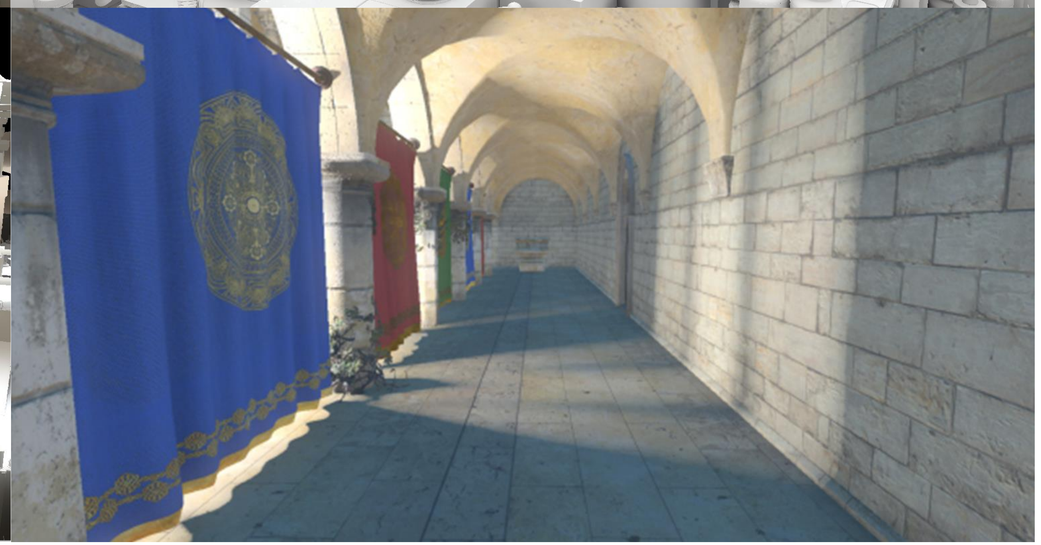


Ambient Occlusion





Reflections



Shadows


RT OFF


KICKSTART RT ON


RT OFF


KICKSTART RT ON

Kickstart RT SDK

How does it work?

- **1. Use „Lighting Cache“**
  - To replace lighting/shading in hit shaders
  - Stores shaded results from rasterization on geometry
- **2. Reuse lighting from rasterization pass**
  - Main view fills the lighting cache
  - Let‘s have auxiliary views to fill lighting cache faster
  - User-app passes shaded frame buffer to the SDK
- **3. Build BVH and tile cache inside of the SDK**
  - User-app has to submit geometry to the SDK

Kickstart RT SDK

When to use this

- **Easier to integrate than full RT**
  - No materials/lights/textures in RT shaders
  - No BVH management
  - No content changes required
  - Denoiser is built-in
- **Use cases**
  - Games
    - Especially without bindless support and/or complicated material system
  - Prototypes
    - *What would RT look like in this application?*
  - Legacy applications (Dx11)
- **It‘s open source**

Implementation Details

Implementation Details

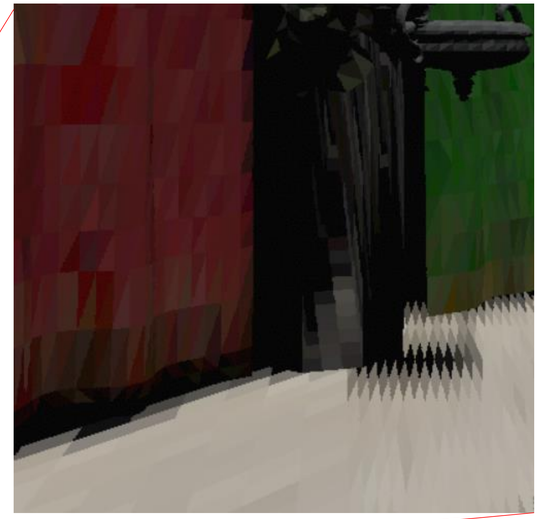
Overview

Index Buffer

Vertex Buffer

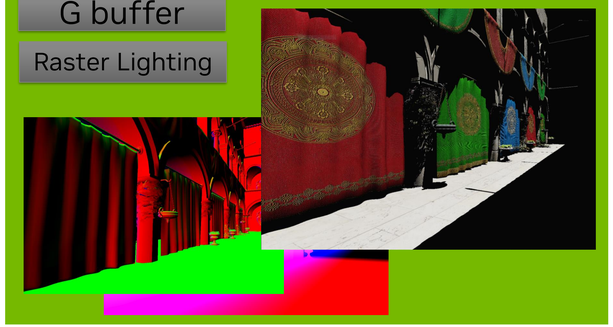
Geometry Input

BVH



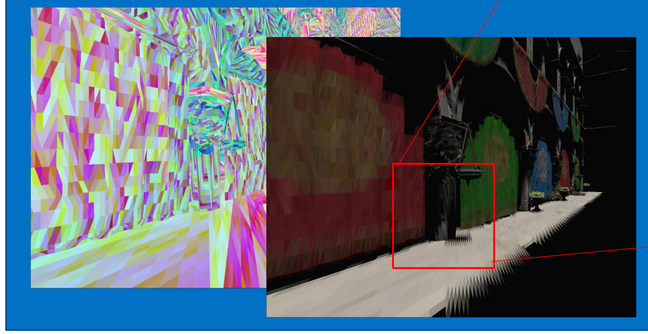
G buffer

Raster Lighting

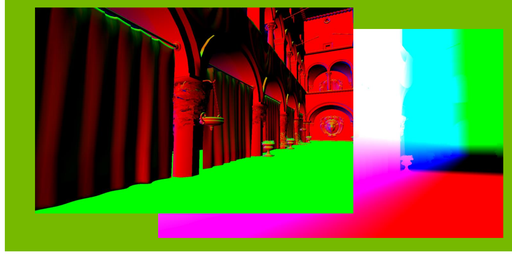


Direct Lighting Injection

World space irradiance-like cache



G buffer



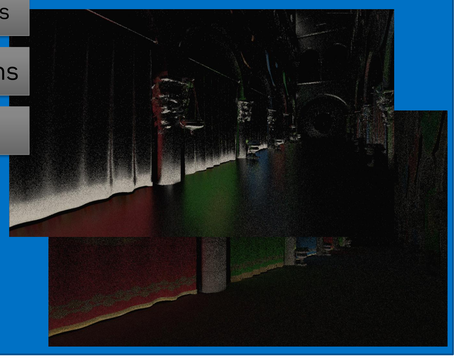
Raytracing

Result

Specular Reflections

Diffuse Reflections

RT AO

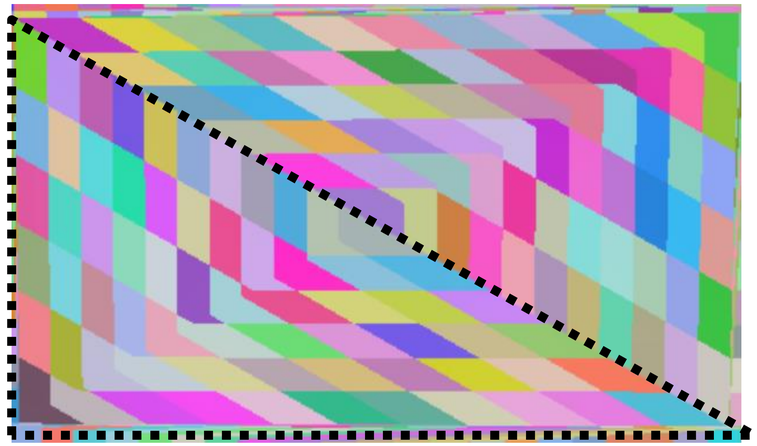


Denoiser

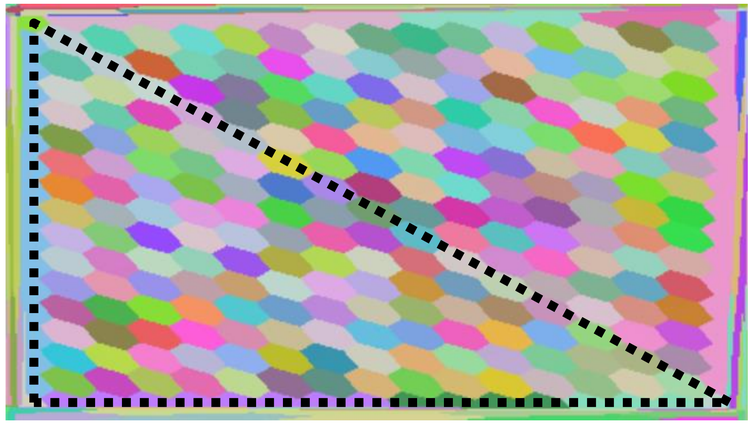
For RT results, NRD SDK is integrated to this SDK to unify interface from the user's perspective.

Implementation Details

Lighting Cache



Regular Tiles

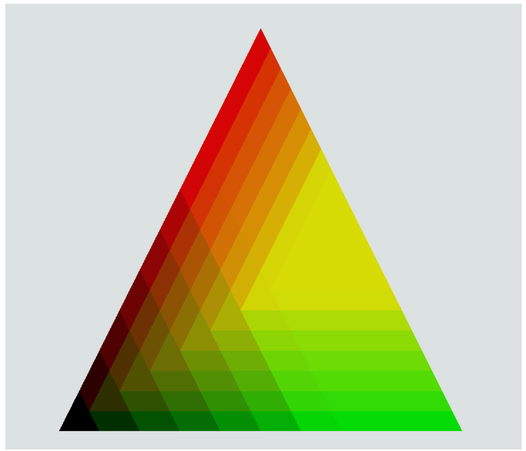


Mesh Colors

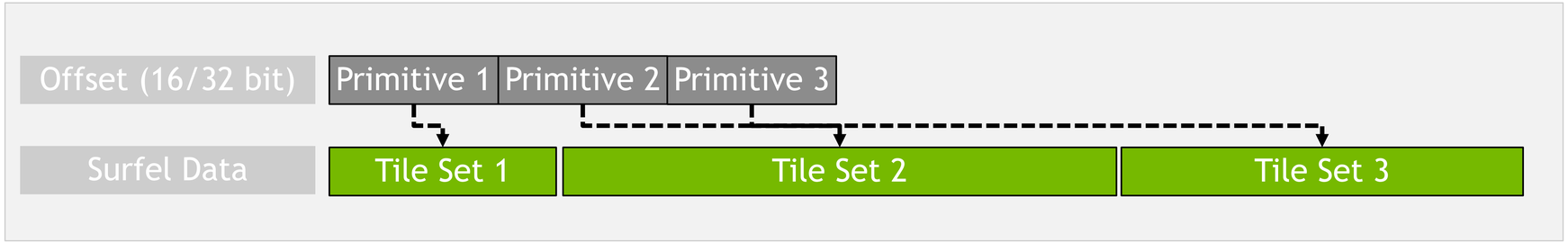
- **Direct Lighting Cache (DLC)**
  - Stores shaded results in surfels on geometry
  - 2 layouts: regular and „mesh colors“ appraoch
    - **Regular** – less memory and faster to build/query
    - **Mesh colors** – possible to interpolate between surfels
- **G-Buffer to cache projection is done by raytracing**
  - “Light Injection” pass
  - Primary (camera) ray finds entry in the tile cache
- **Temporal accumulation**
  - Exponential moving average
  - Configurable window size
  - Balances response time against temporal stability

Implementation Details

Lighting Cache

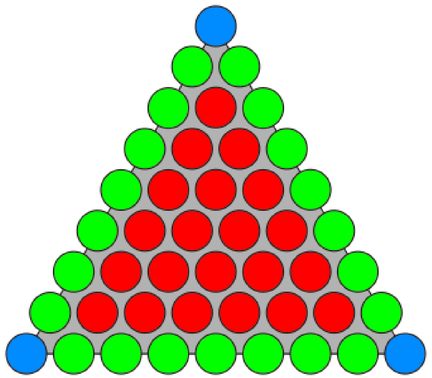


- **Regular Layout**
  - NxM patches possible for every triangle
    - But tiling is obvious in reflections!
    - Tiles are equal size
  - Simple to address cache entries
    - One level of indirection
    - Nased on barycentrics
  - Memory allocated based on triangle size before BVH is built
  - Interpolation between surfels is difficult



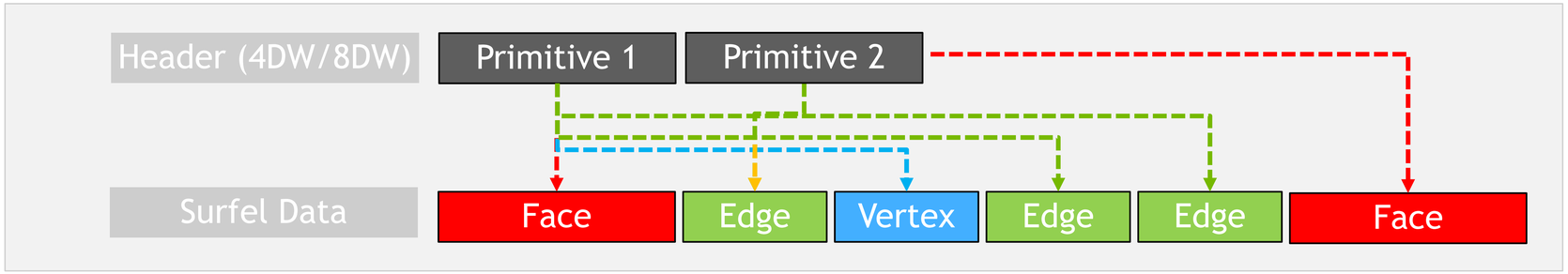
Implementation Details

Lighting Cache



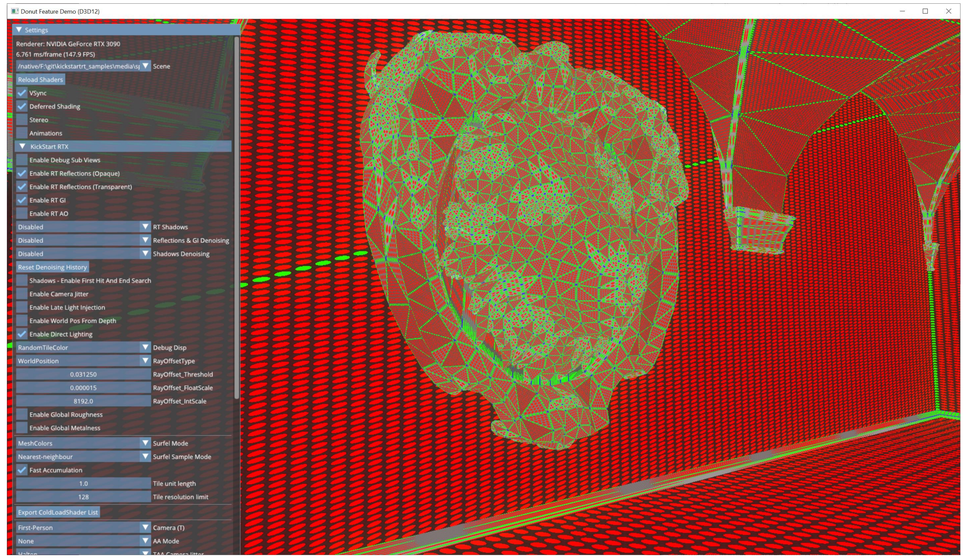
- **Mesh Colors Layout**
  - Based on Cem Yuksel‘s Mesh Colors article [1]
  - NxN patches only!
    - But we can interpolate between surfels
    - N must be a power of 2
  - More difficult to build and access
    - Needs an „edge table“, built before the BVH

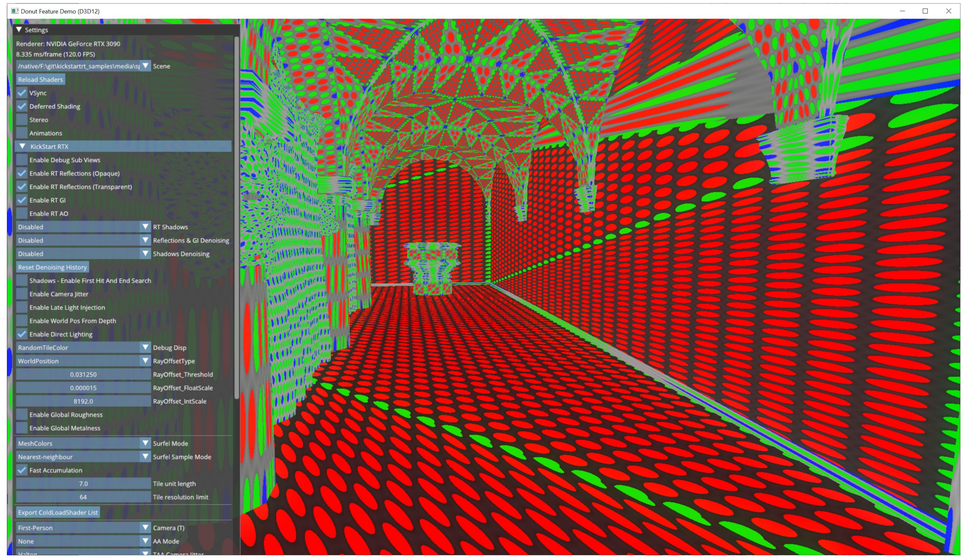
[1] http://www.cemyuksel.com/research/meshcolors/meshcolors_techreport.pdf



Implementation Details

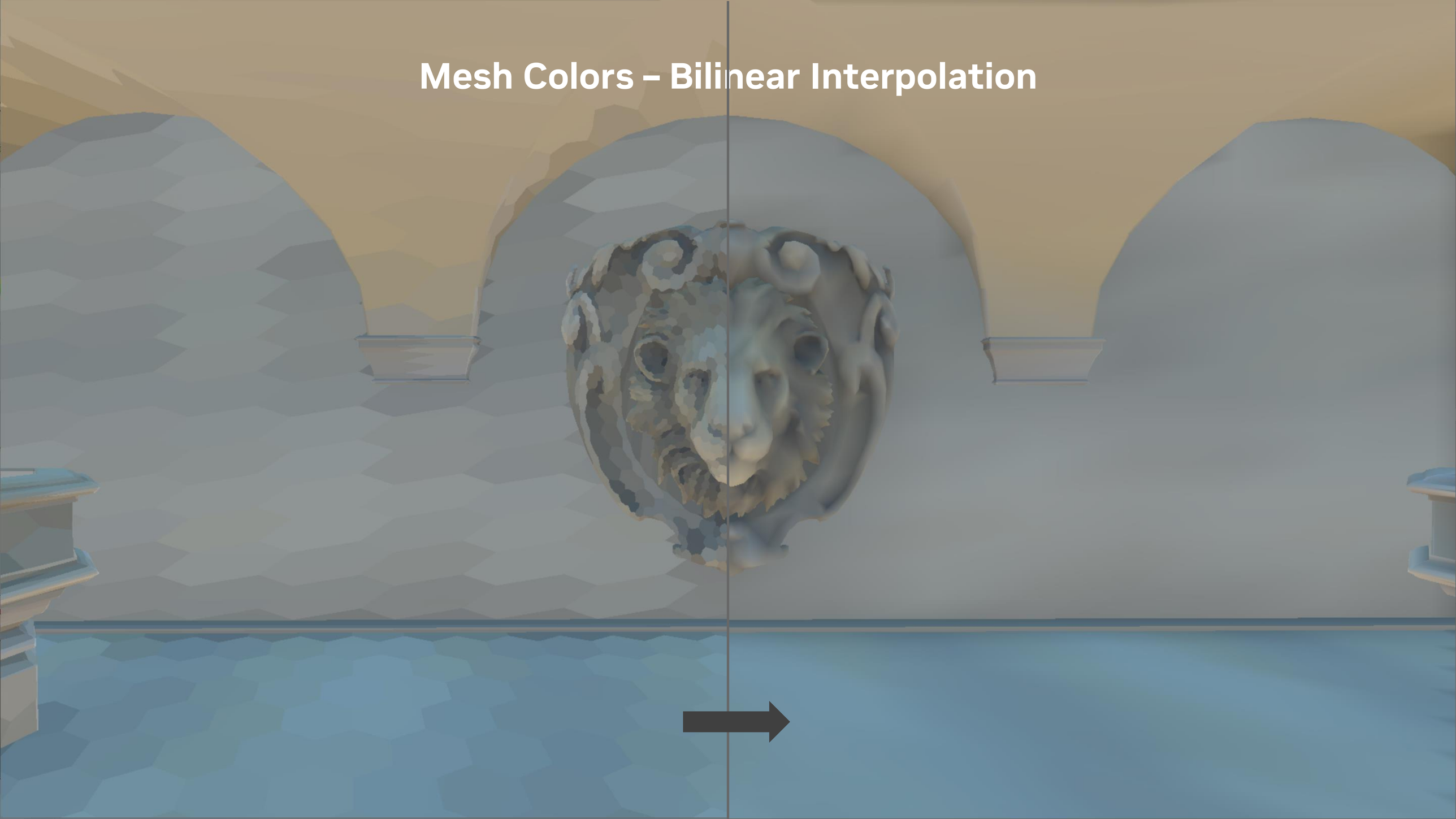
Lighting Cache – Debug Views





- **Resolution of lighting cache needs to be tuned**
  - Trade-off between speed and quality
  - Finer resolution requires more memory
  - Debug views help

Mesh Colors – Bilinear Interpolation



Implementation Details

Lighting Cache

Regular Tiles

Mesh Colors

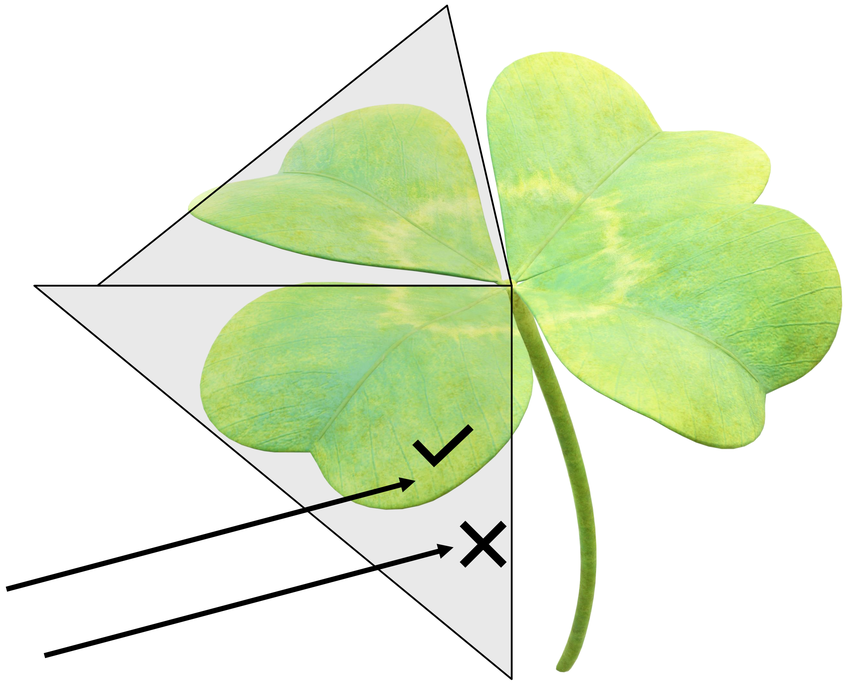
Implementation Details

Lighting Cache

What about alpha testing?!

Implementation Details

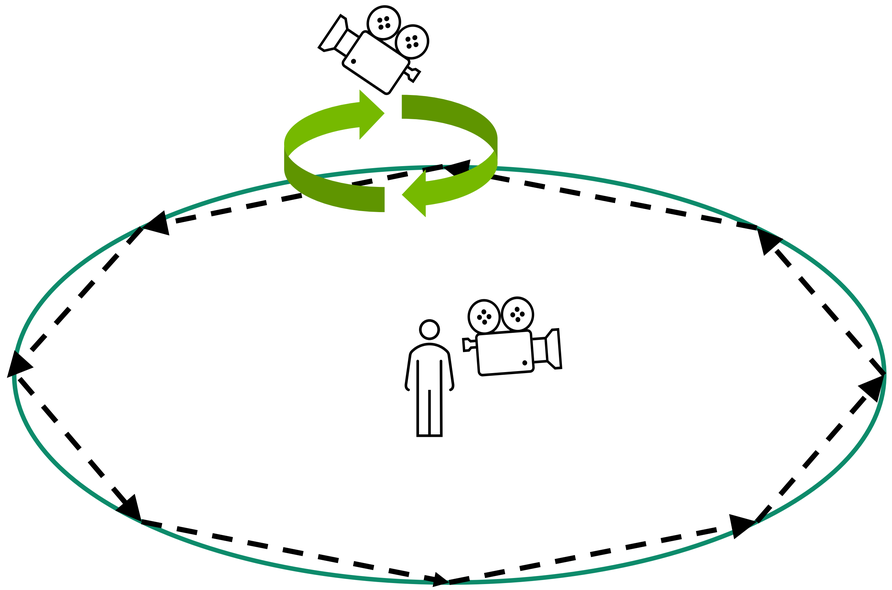
Alpha Testing



- **Currently not supported, but!**
- Should be possible to add a *visibility bit* to tile cache
  - Tiles that are never written to are considered transparent
  - Any hit shader would ignore transparent bits
- Future work

Implementation Details

Populating the Lighting Cache



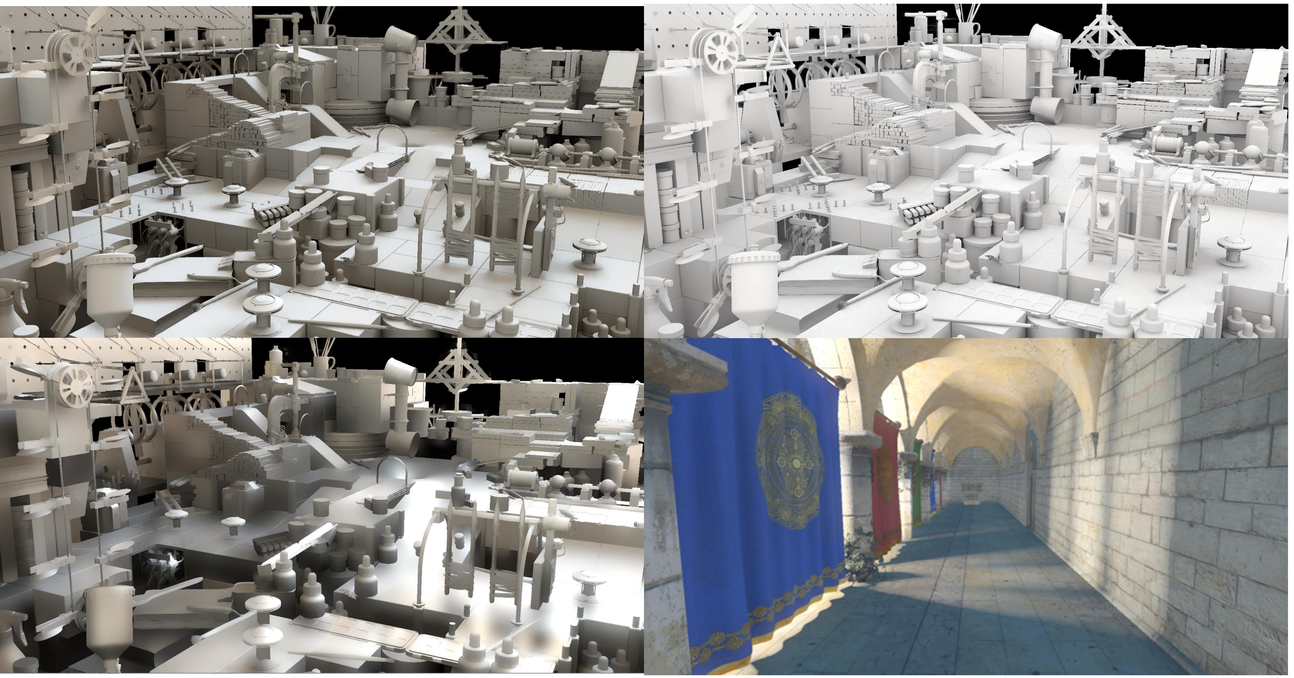
- Find lighting cache entries for pixels in the G-Buffer
  - Cast camera rays
  - Use same hit shader routine as for secondary rays
  - Write lighting data into the cache
    - Temporally accumulate (weighted average)
- SDK accepts more than 1 G-Buffer
  - Multiple views
  - Possibly low res, simpler shading
  - **Orbiting camera**
- Update every frame
  - Dynamic scenes just work

Implementation Details

Effects

Diffuse GI

Ambient Occlusion



Reflections

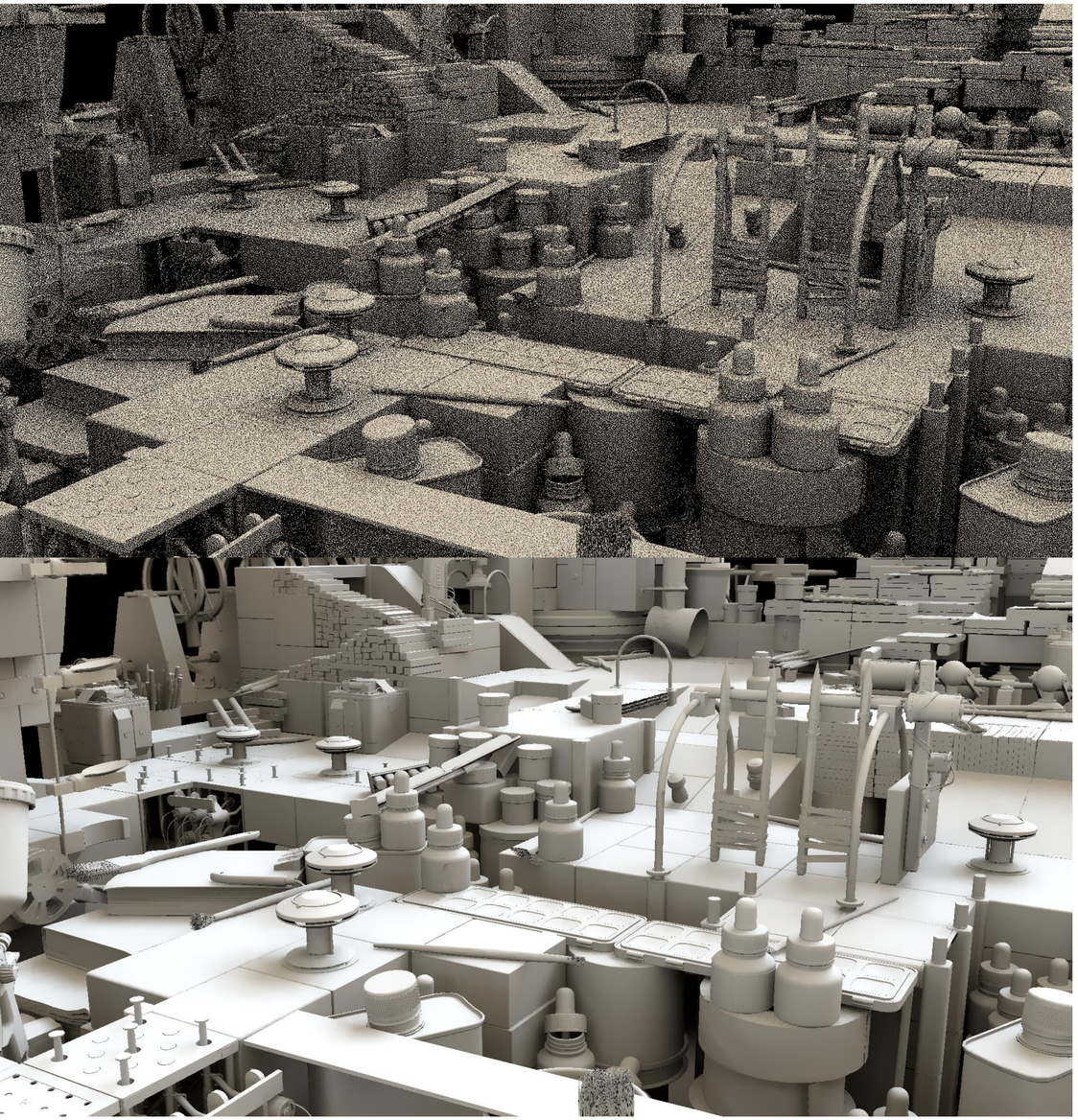
Shadows

- **Diffuse GI**
  - Lambertian or Disney diffuse
  - 1 bounce
- **Specular**
  - GG-X model
  - Respects roughness (mirrors possible)
    - Optional roughness remapping (clamp into plausible range)
- **Ambient occlusion**
  - Adjustable range
- **Shadows**
  - Directional, Point and Spot lights

Implementation Details

Denoiser

Diffuse GI

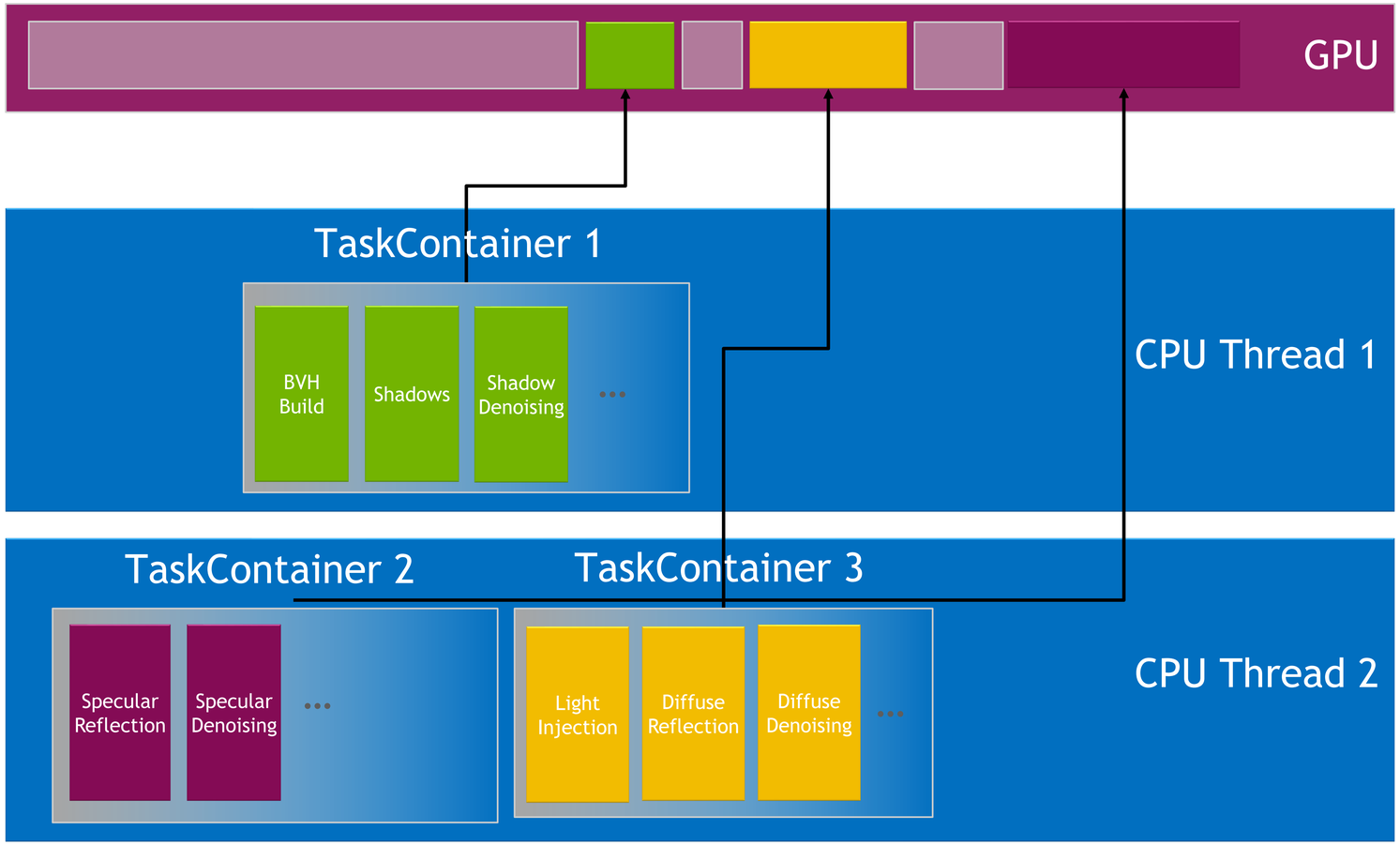


- **NRD (NVIDIA Real-Time Denoisers)**
  - Denoises all provided effects including shadows
  - Is optional
  - GI denoiser for AO
  - Albedo Demodulation
  - Different license than Kickstart RT

Integration

Integration

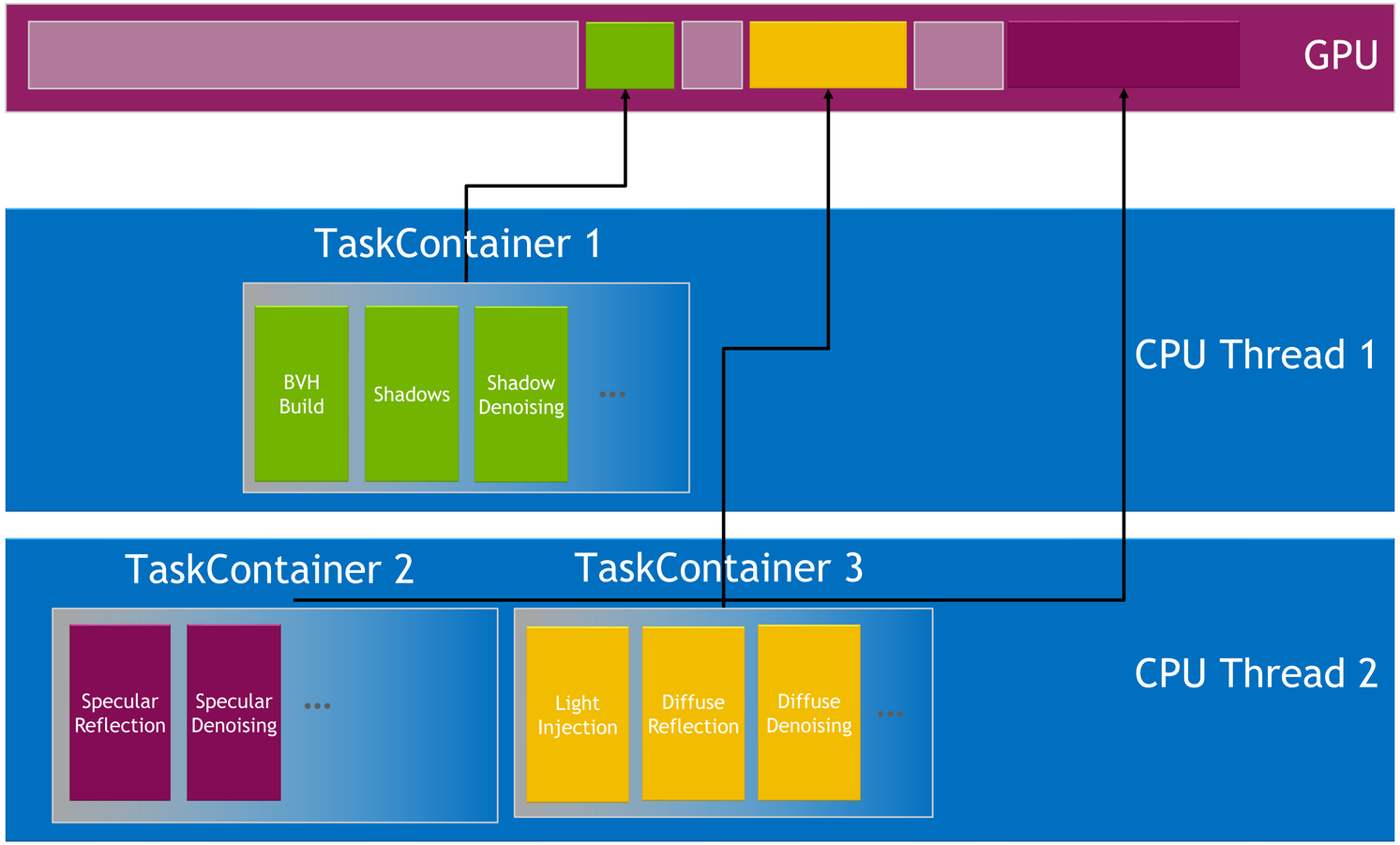
Main Concepts



- **1. Execution Context**
  - Created on initialization
  - Maps to VkDevice / ID3D12Device
  - Allocates resources, creates task containers
  - manages the lifetime of objects (BLAS/TLAS) and the internal state
- **2. Task Container**
  - Maps to ID3D12CommandList / D3D12 Command List
  - Used to schedule Render Tasks on the GPU
  - Managed by application
- **3. Render Task**
  - A high-level task or a „render pass“ of the SDK
  - E.g., BVH build, Diffuse GI Pass, Denoising task, ...
  - Scheduled and executed using task containers

Integration

Main Concepts



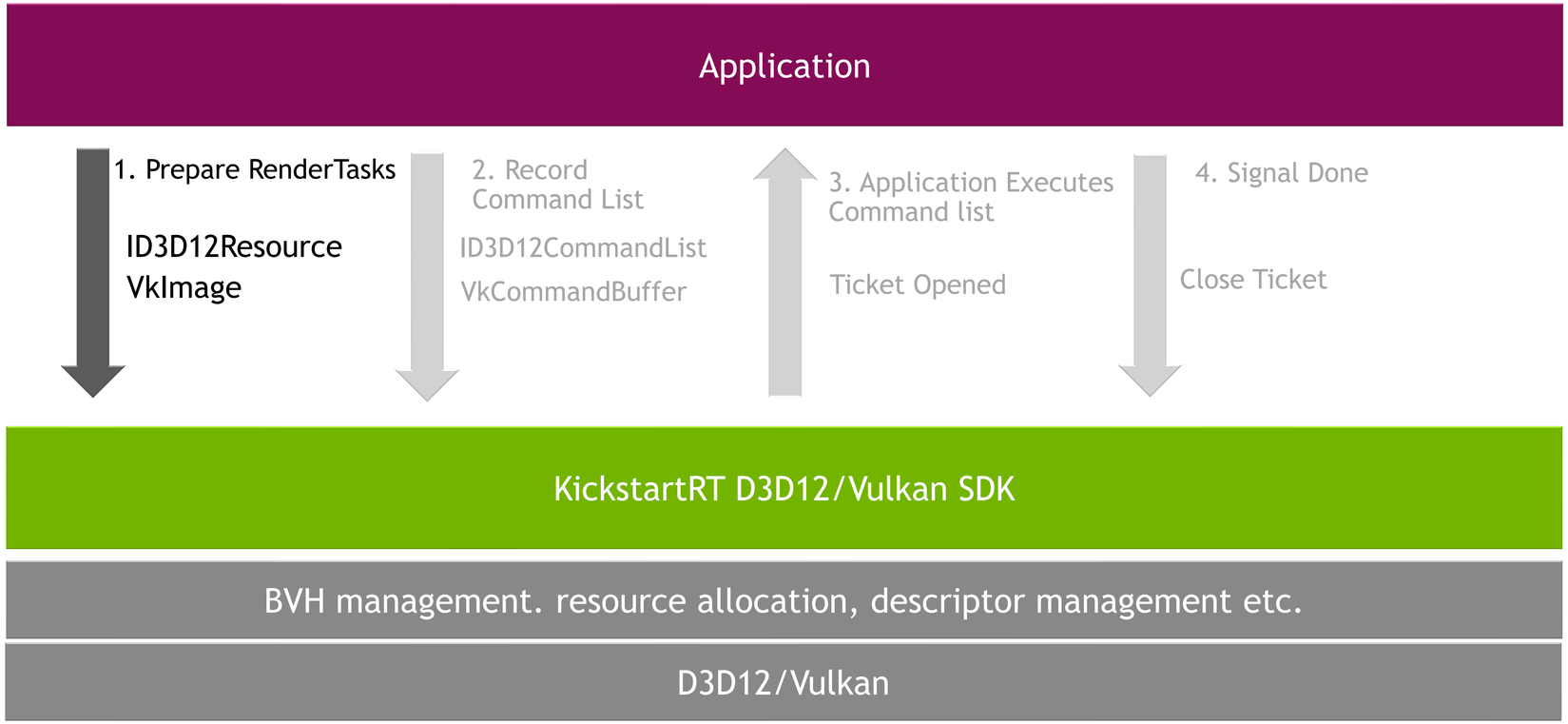
- **1. Execution Context**
- **2. Task Container**
- **3. Render Task**

- These form an abstraction over selected graphics API (Vulkan/D3D)
- SDK doesn‘t own or create any threads
- SDK doesn‘t own or create and command lists
- Possible parallelism

Integration

Application Flow

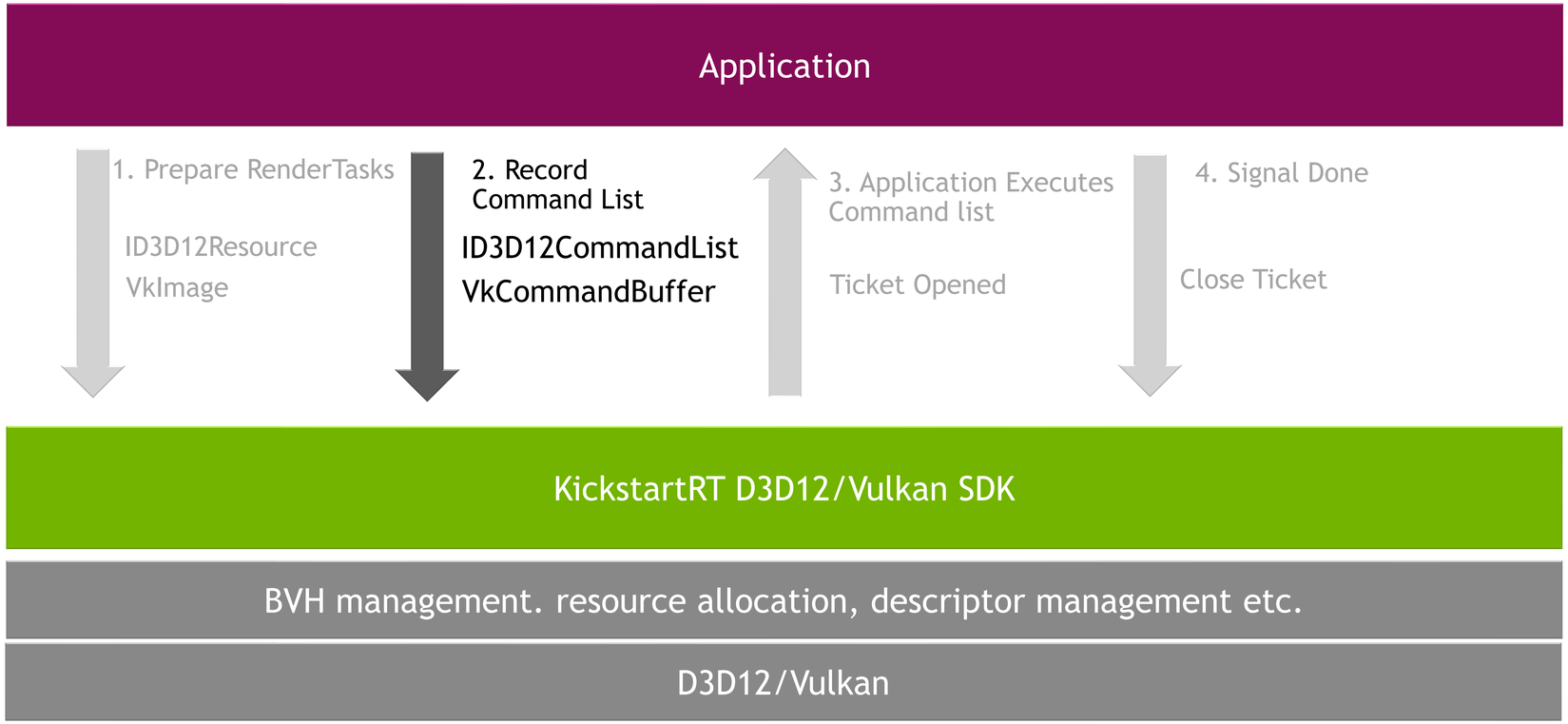
1. Prepare a render task (e.g. BVH build for new geometry, rendering of reflections or denoising)



Integration

Application Flow

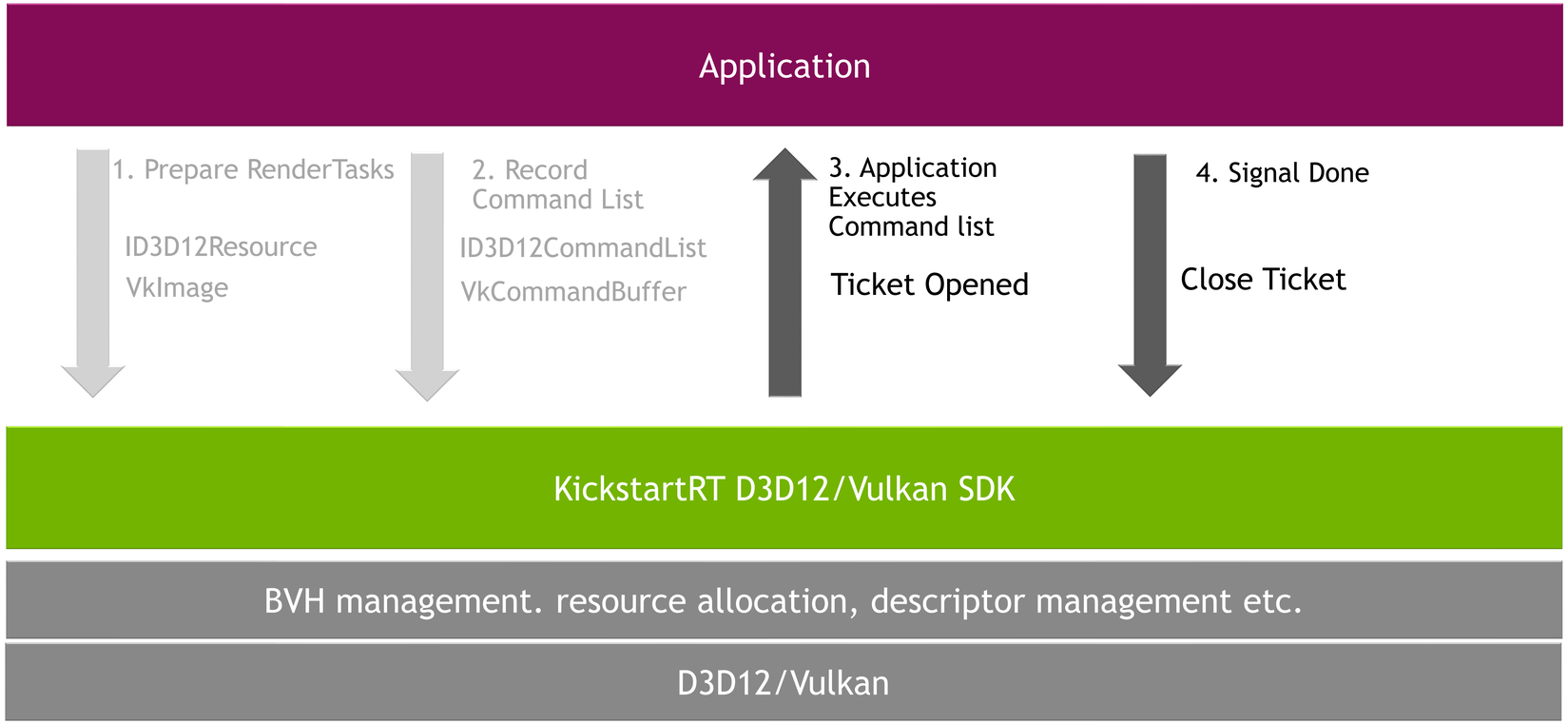
2. Schedule the task for execution within a Task Container (cmd. list). Blocking call with possible memory allocations



Integration

Application Flow

2. Application executes the cmd. list of SDKs task container. Here is room for async compute. Once done, singal to the SDK that tasks are completed



Integration

Graphics API Abstraction

- **Application side:**
  - Include Kickstart header file with selected API
    - Use native pointers to resources (VKImage, ID3D12Resource*)
    - Use VK namespace: `KickstartRT::VK`
  - **Wrappings of device and cmd. buffer**: Execution Context, Task Container, Render Tasks
- **SDK Side:**
  - Internally, we use a custom abstraction layer
    - GraphicsAPI.h, GraphicsAPI.cpp
  - **#if defined(GRAPHICS_API_D3D12)**
  - **#if defined(GRAPHICS_API_VK)**

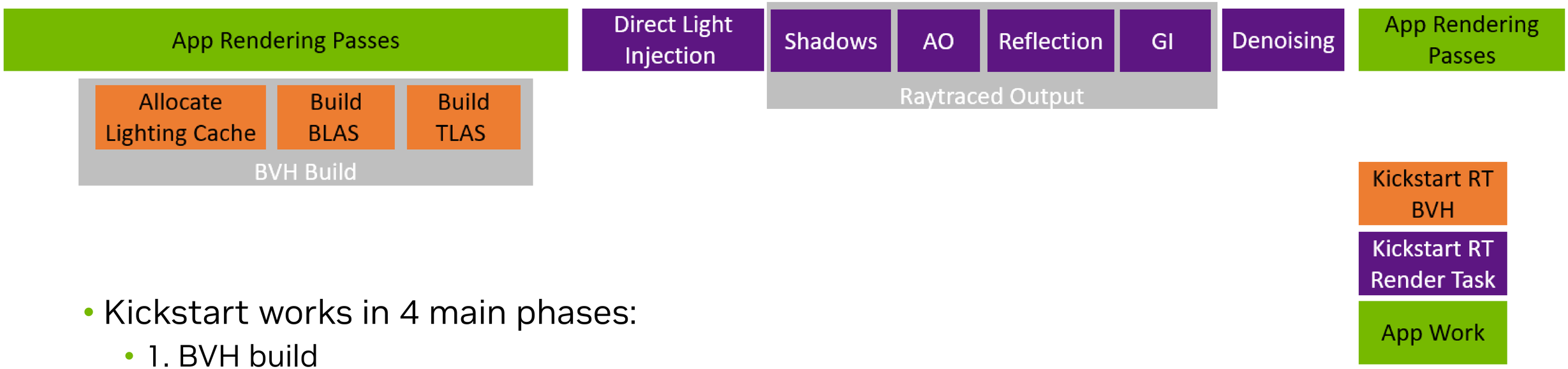
```
#define KickstartRT_Graphics_API_Vulkan
#include "KickstartRT.h"

KickstartRT::Status sts =
KickstartRT::VK::ExecuteContext::Init(&settings,
&m_SDKContext.m_vk->m_executeContext);


KickstartRT::VK::RenderTask::DenoisingTaskCommon
dTaskCommon;
```

Integration

Overview



- Kickstart works in 4 main phases:
  - 1. BVH build
  - 2. Direct Light Injection
  - 3. Raytracing
  - 4. Denoising

Performance

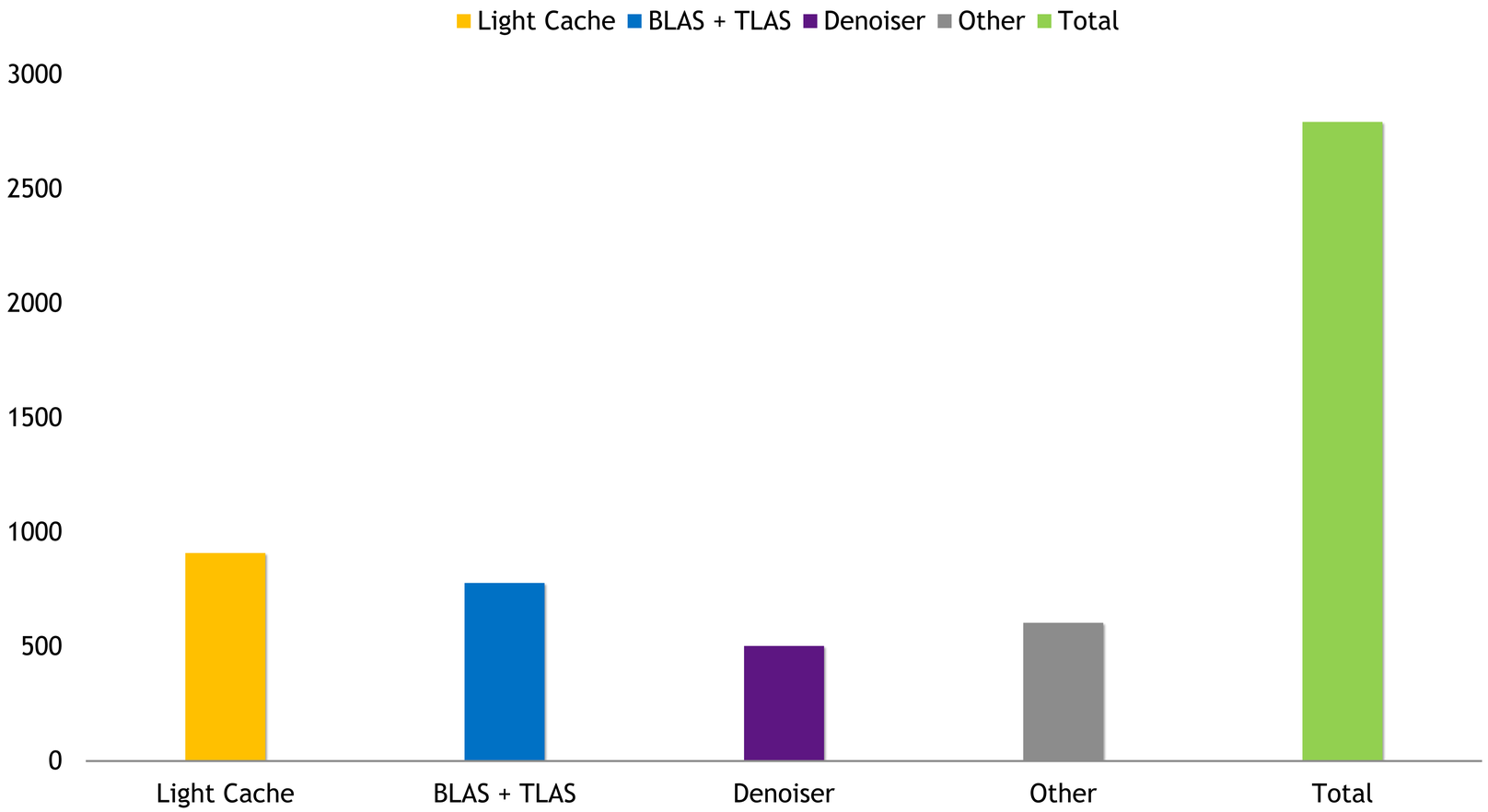
Speed

RTX 3070 @1440p	Kickstart RT
“Extra G-Buffer” @ 640x360	1.0 ms
Light Injection	0.1 ms
Reflections	1.2 ms
GI	1.0 ms

Test Integration into game - resolution @1440p, 5k geometry instances

Performance

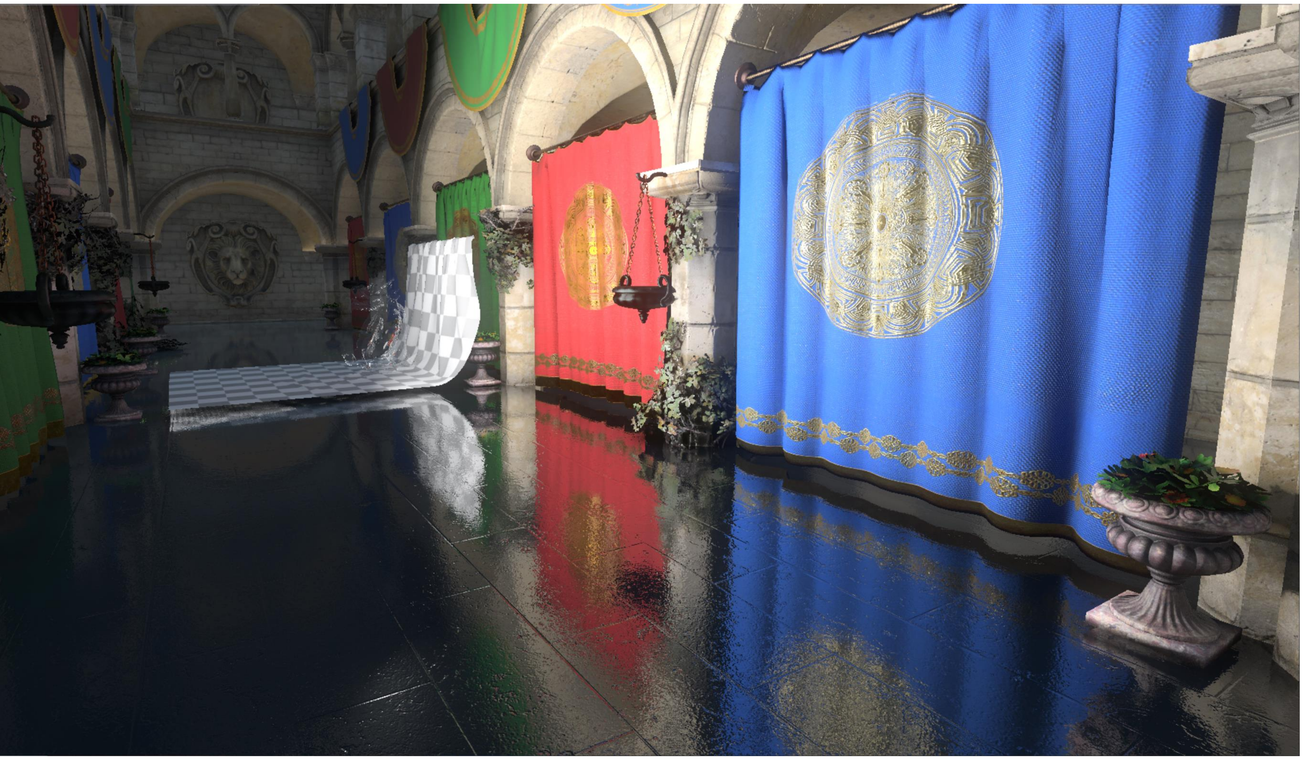
Memory Requirements



Test Integration into game - resolution @1440p, 10k geometry instances

Performance

Finetuning Tips

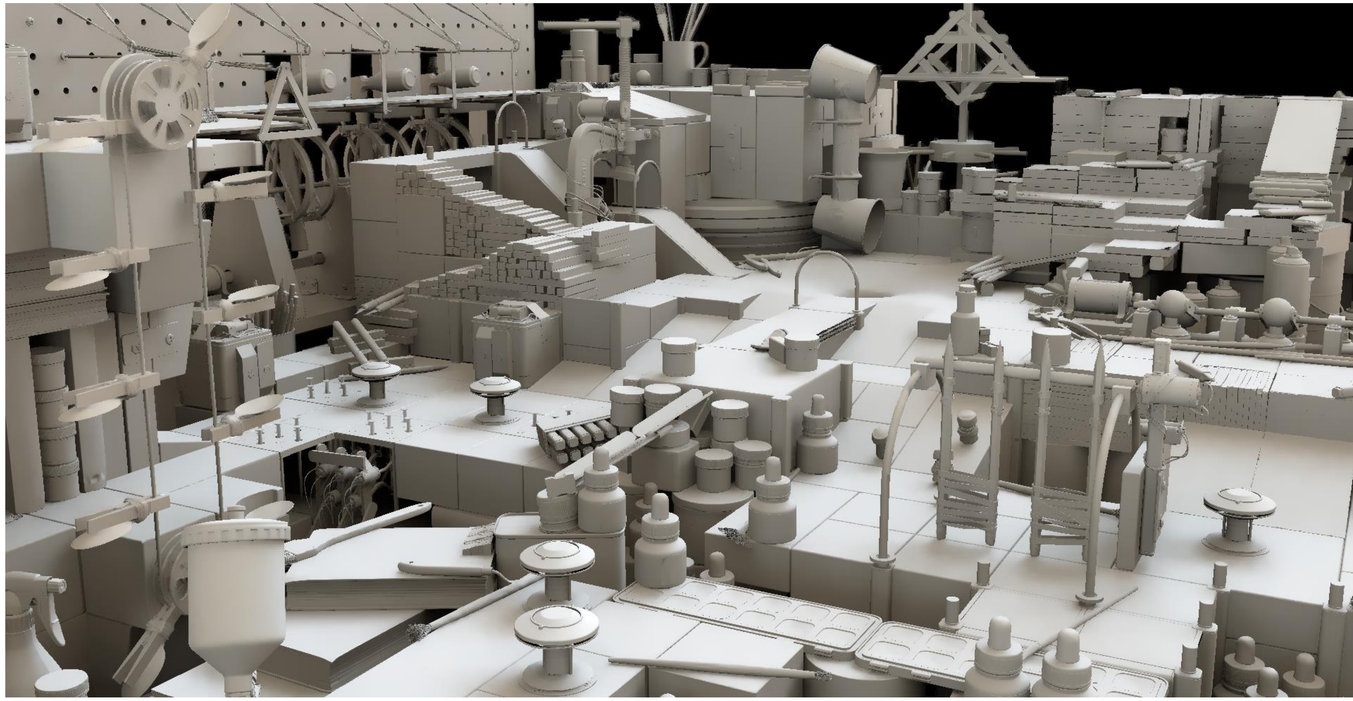


- **Throttle BVH builds**
  - Limit number of BVHs built per frame
- **Tile Cache Size**
  - Coarser cache uses less memory and is faster
- **Checkerboard Rendering**
  - Halves number of rays traced, SDK upscales automatically
- **Choose between *TraceRayEXT()* and *RayQueryEXT()***
  - Depending on application, one might outperform other

Kickstart RT SDK

Summary

- **An open-source SDK for easy to integrate ray tracing**
  - Windows & Linux
  - Vulkan & Dx11, Dx12
- No content changes required
- Handles all RT tasks internally
- Features common effects: GI, reflections, shadows



Kickstart RT SDK

Q&A

https://github.com/NVIDIAGameWorks/KickstartRT

https://github.com/NVIDIAGameWorks/KickstartRT_demo


NVIDIA