Comparison of Ray Tracing GPU Implementations

2021-2022

Universitat Politècnica de València
Dept. of Computer Systems and Computation

Master's Thesis
Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Author: Luis Carlos Catalá Martínez
Director: Francisco José Abad Cerdá

Contents

Problem and Objectives

- Ray Tracing is widely used in industry (film, videogames, simulation...)
- GPU acceleration **required**
- Multiple libraries and APIs for GPU acceleration
- Benchmarks heavily outdated (15 years since last publication)

Problem and Objectives

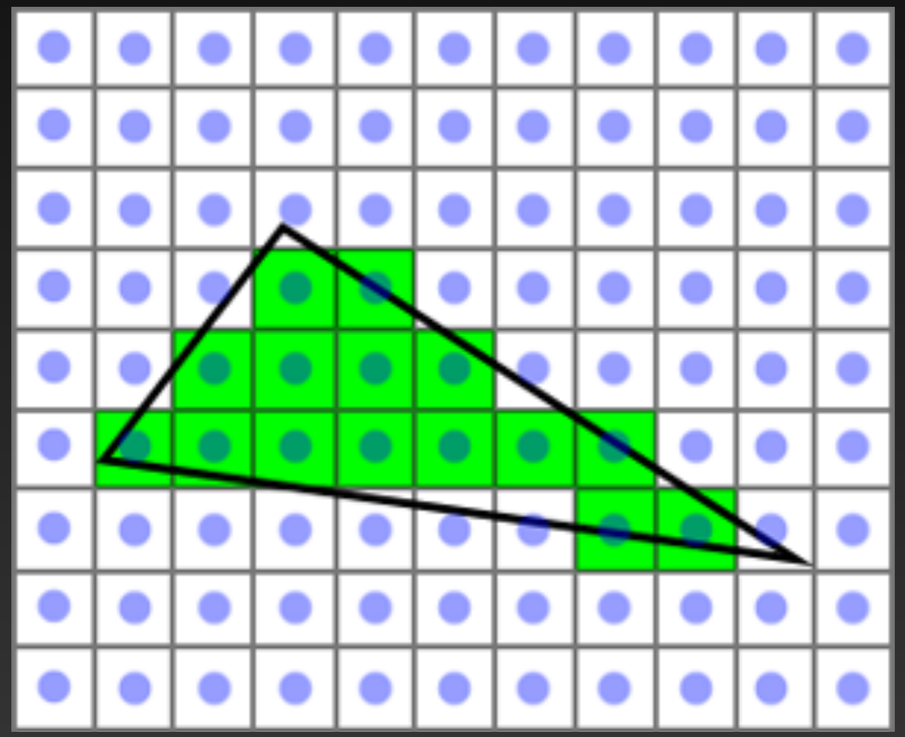
- Most prevalent libraries: Vulkan and Nvidia OptiX
- **Our goal:** build an updated benchmark comparing these
  - Different scenes and rendering settings
  - Frame Render Time, Acceleration Structure Build Time and Memory Usage
  - Comparison in the same hardware and across different systems

State of the Art
Techniques

- Computer Graphics: computer-aided image generation
- Several techniques: Rasterization, Ray Tracing

State of the Art
Technique: Rasterization

- Vector Graphics → pixels
  - Displayed on a monitor, stored, etc.
- Only maps geometry, not color
  - Requires Pixel/Fragment Shaders
- Extremely fast
  - Fixed function hardware
  - Most used for videogames

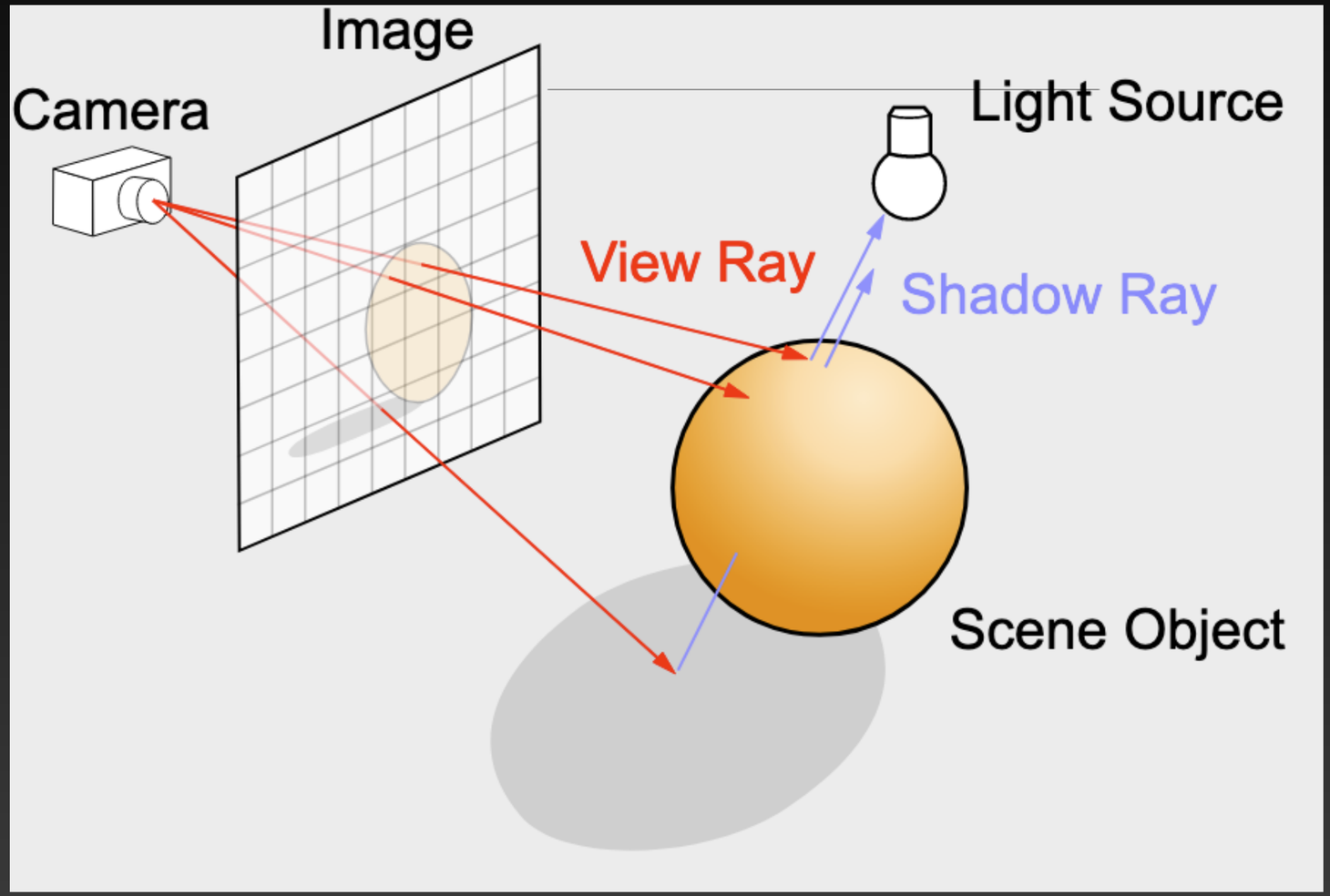


State of the Art
Technique: Ray Tracing

- Main focus of this work
- Slower but higher fidelity than Rasterization
- Generate images modeling light transport
  - Trace path from “eye” through each pixel in a virtual screen
  - Calculate color of object visible through it
  - Rays are tested for intersection with scene geometry
- Sending rays **from** the camera is **faster** than the other way (less rays)

State of the Art

Technique: Ray Tracing



State of the Art
Technologies

- 2 use cases: **photorealism** and **real-time graphics**
- Both require libraries to communicate with GPU for acceleration
- Nowadays main options are **OptiX** and **Vulkan** respectively

State of the Art
Technology: Vulkan

- Announced by Khronos Group in 2015 (next generation OpenGL)
- High performance, real time graphics
- Lower level control than other APIs at the time
  - Lower overhead, higher performance
- Based on AMD Mantle, copied by DirectX12 and Metal

State of the Art
Technology: Vulkan

- Unified API for desktop and mobile, unlike OpenGL
- Not locked to any specific platform, unlike Metal or DirectX
- Lower CPU usage with optimizations like instruction batching
- Multi-threading friendly
- Pre-compiled shaders (SPIR-V)
- Extension system for extra functionality
  - Ray Tracing support
- Widely used in videogame industry

State of the Art
Technology: OptiX

- Released in 2009 as part of Nvidia GameWorks
- Based on CUDA (required Nvidia GPU)
- Relatively low level since version 7 (higher than Vulkan)
- Also used in areas where line-of-sight is important
  (optical/acoustical design, radiation/electromagnetic research, AI, etc.)

State of the Art
Technology: OptiX

- Based in **kernels**
  - Instructions defined by the user written in CUDA-C or PTX
  - Define how rays behave in specific situations (hitting different surfaces, missing, etc.)
  - Communicate CPU and GPU with CUDA buffers
- Scene geometry hierarchy
- Transparent scaling
- Prominent in film industry

Analysis
Experiment Design

- Tracking Frame Time, Acceleration Structure (AS) Build Time and Memory Usage
- Varying screen resolution

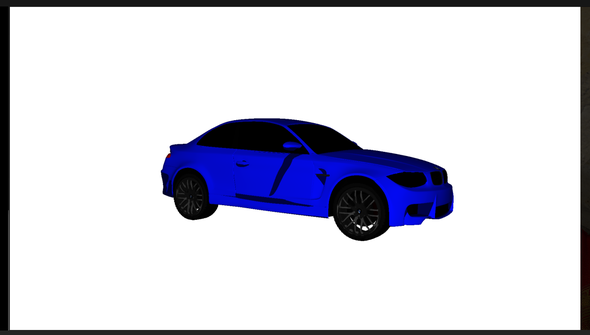
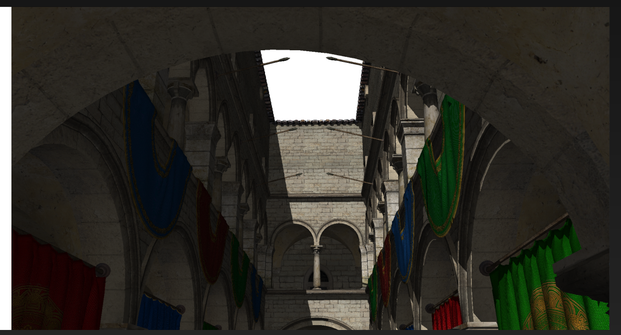
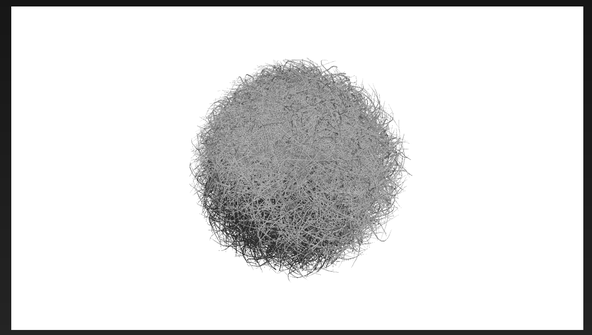
Analysis
Experiment Design

- Different scene complexities

Model Name	Triangles	Vertices
Viking Room	2000	2600
BMW	385079	249772
Sponza	262267	184330
Hairball	2880000	1441098

Analysis
Experiment Design

- Different scene complexities

Analysis
Experiment Design

- Hardware configurations

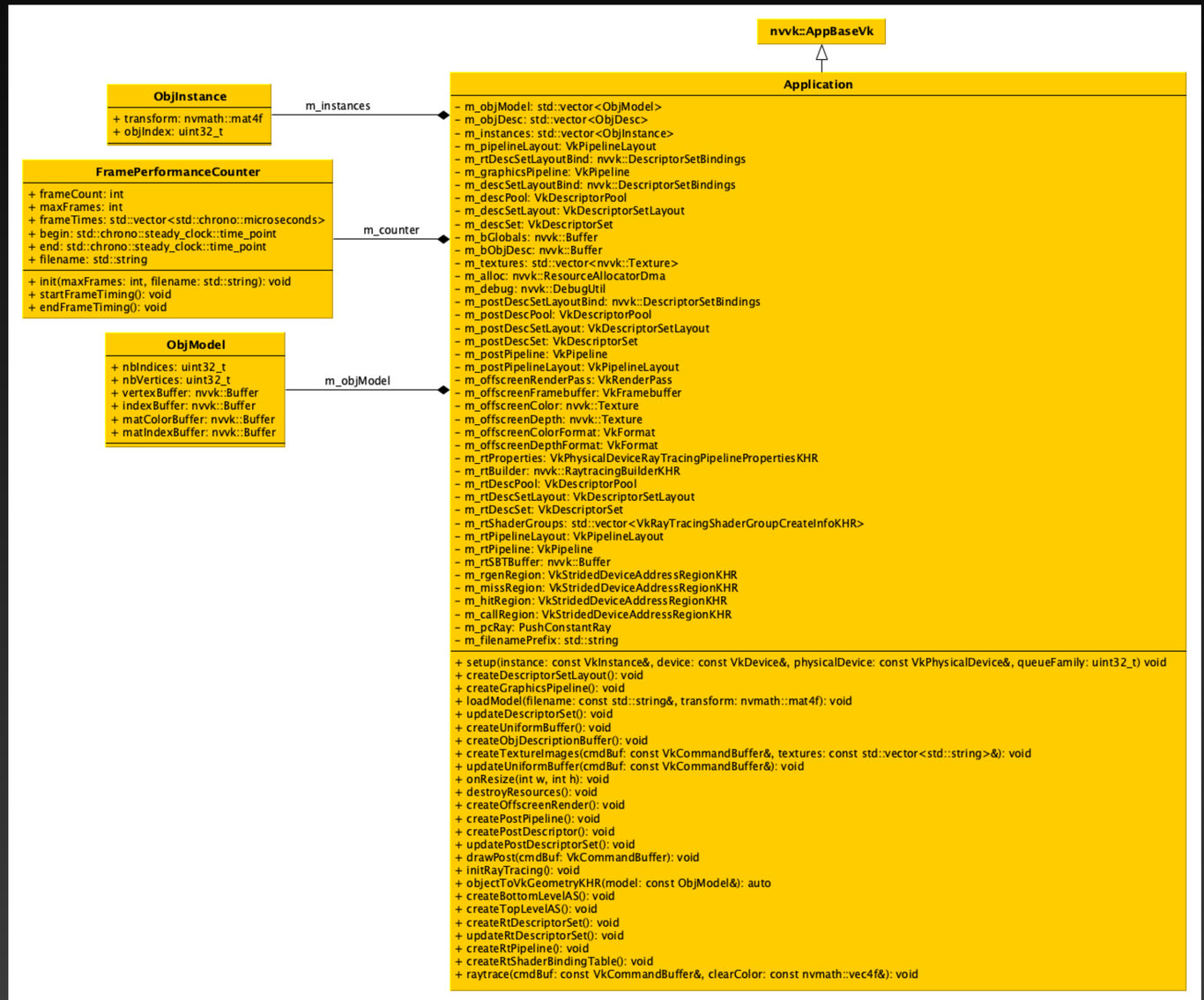
CPU	GPU	RAM	OS
Intel Core i7-12700K, 3.60GHz	Nvidia GeForce RTX 3070	16 GB	Windows 11, 64 bit
AMD Ryzen 5 3600 3.60Hz BOX	Nvidia GeForce RTX 2070 Super	16 GB	Windows 10, 64 bit
Intel Core i5-9600K, 3.70GHz	Nvidia GeForce RTX 3060 Ti	16 GB	Windows 10, 64 bit
Intel Core i9-9900K 3.6GHz	Nvidia GeForce RTX 3080	16 GB	Virtual Machine with Linux host, Windows 10, 64 bit guest
Intel Core i7 10th gen	Nvidia GeForce RTX 3080	32 GB	Windows 10, 64 bit
AMD Ryzen 7 3800x, 3.9GHz	Nvidia GeForce RTX 2070 Super	64 GB	Windows 11, 64 bit
Intel Core i9-12900F	Nvidia GeForce RTX 2060	32 GB	Windows 10, 64 bit

Design
Vulkan Renderer

- **Rasterization** renderer for **baseline**
  - Monolithic **Application** class
  - Auxiliary data structures for grouping information
  - Almost no auxiliary libraries, handling low level operations ourselves
  - Largest code size of all
- Auxiliary class for measuring performance: **FramePerformanceCounter**

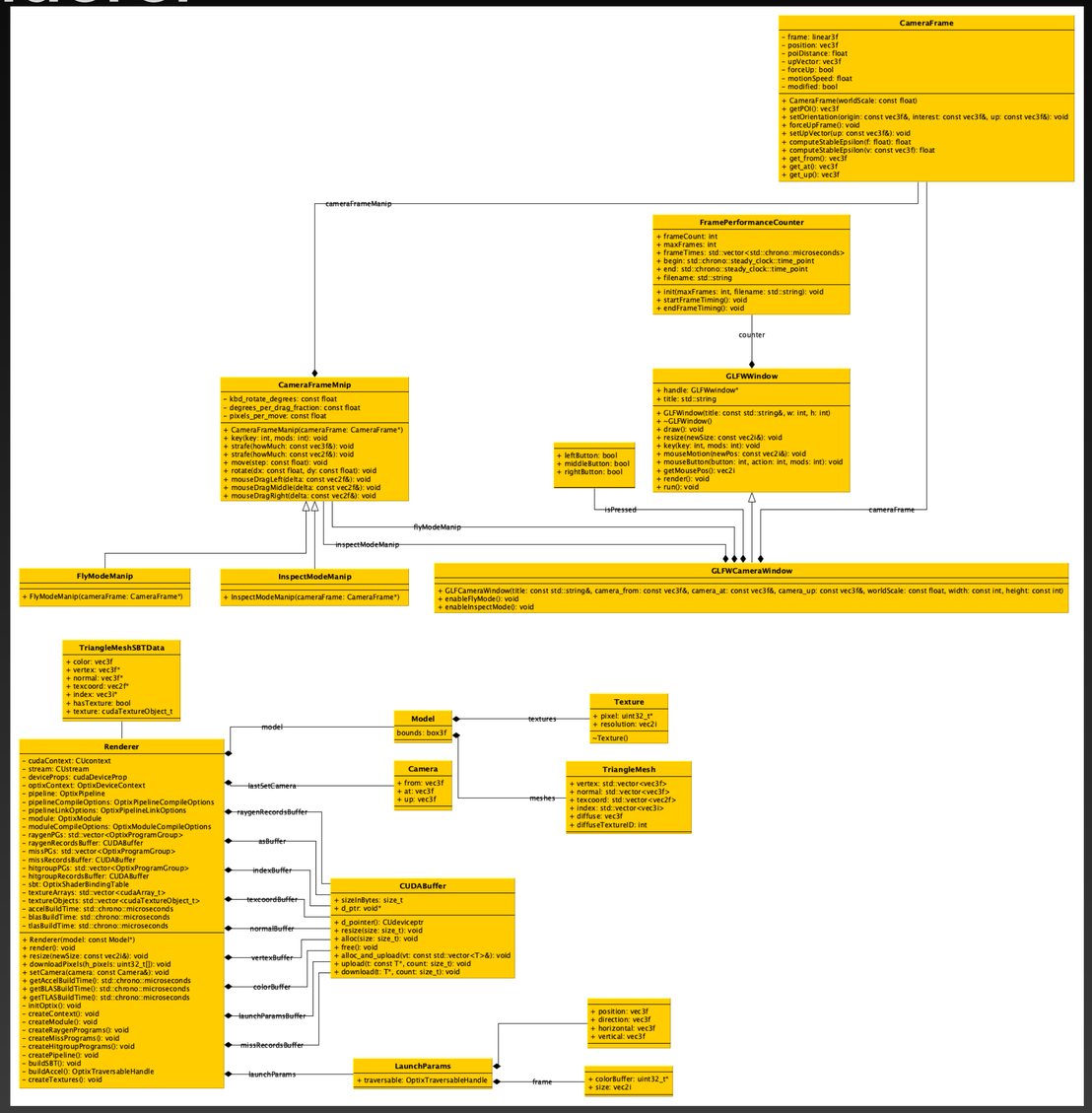
Design
Vulkan Renderer

- Ray Traced version
  - **Nvvk** library
    for memory management
  - Reused class
    FramePerformanceCounter

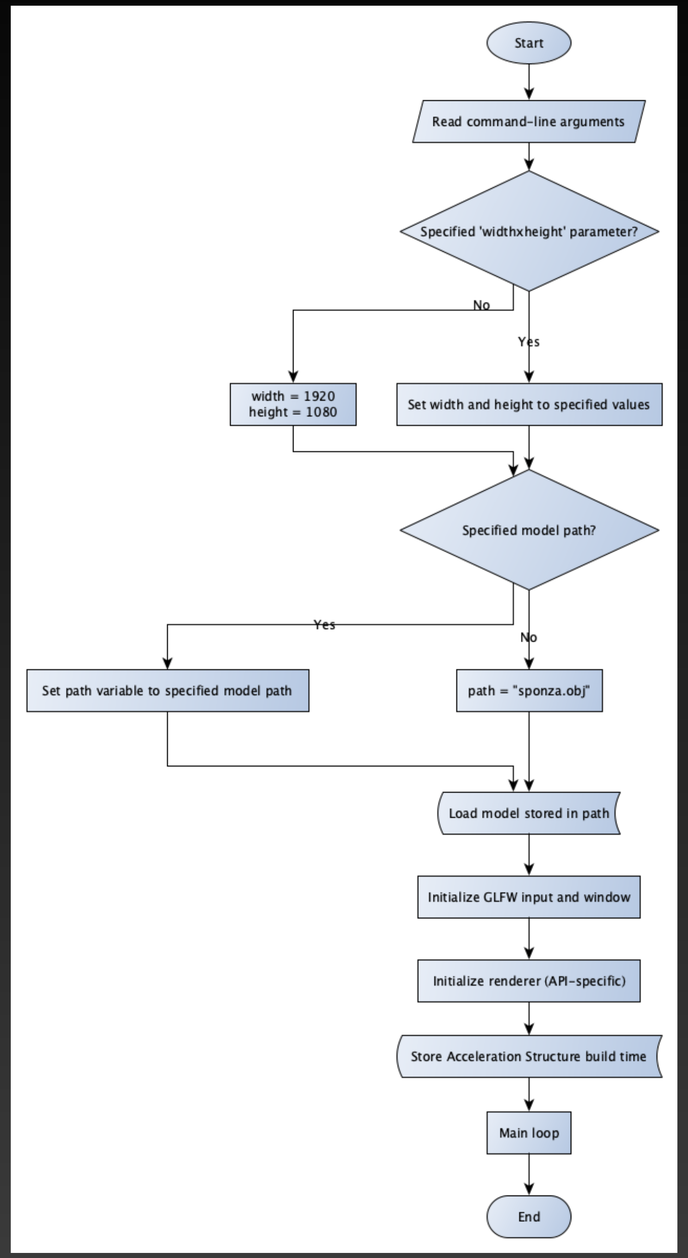


Design
OptiX Renderer

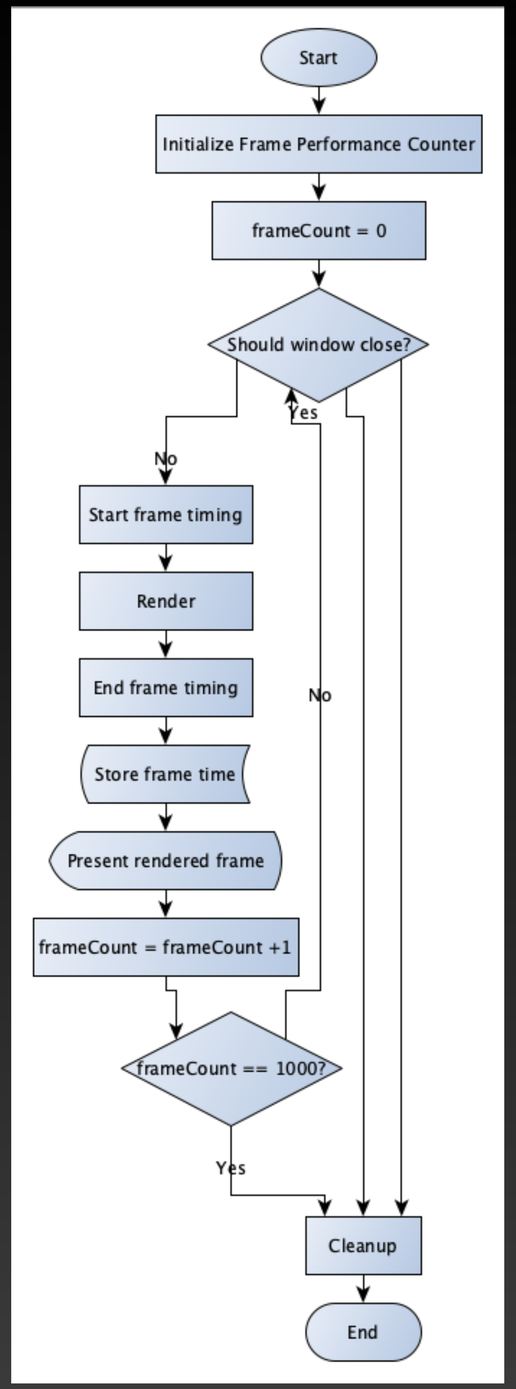
- Main **Renderer** class
- Further **encapsulated** functionality
  - Window/WindowCamera hierarchy
- Camera system from
  2019 SIGGRAPH OptiX Course
- Reused **FramePerformanceCounter**



Design
Flow Charts



- Common to all renderers



Development
Implementation

- **Two** renderers with as close as **functional parity** as possible
- Python scripts for experiment **automation** and **reproducibility**
- Python scripts for **plot generation**

Development
OptiX Renderer

- Based on the 2019 SIGGRAPH OptiX 7 Course
  - Ray Generation program that computes pixel colors
  - OptiX Pipeline handling the *kind* of programs to run
  - Shader Binding Table (SBT) handling the programs' run configuration
  - Frame Buffer to store generated image

Development
OptiX Renderer

- Extra Shadow Ray type (own *closest hit* program)
  - Hitgroup Program Group has 2 records, SBT has 2 records per mesh
- Assume all surfaces are opaque (kill ray on first occlusion)
- Ported from OptiX 7.3 to 7.5 (minimal changes)
- Frame Buffer gets displayed with an OpenGL quad
- Instrumented code to register performance data
  - std::chrono
  - cudaMemGetInfo

Development
Vulkan Renderer

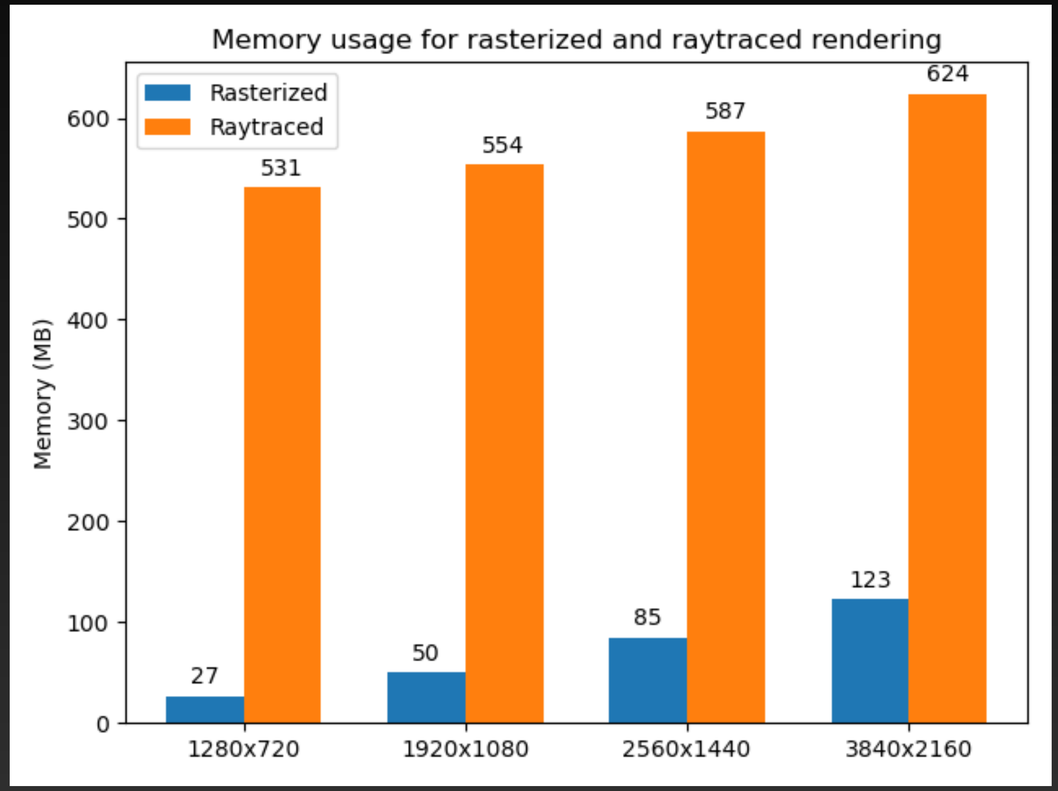
- **Rasterization renderer** took the longest to get working, highly explicit
- Built **2 versions**, with a rasterization one
  for comparison between techniques
- DirectX Graphics Infrastructure for querying memory usage

Development
Vulkan Renderer

- **Ray Tracer** based on Nvidia's Vulkan Ray Tracing Tutorial
  - Explicitly defined **Acceleration Structure**, divided in BLAS and TLAS
  - RT **Descriptor Set** to reference external resources used by shaders
  - RT **Pipeline** for accessing all shaders at runtime
    - More similar to Compute Pipeline than to Rasterization Pipeline
    - Core component is **SBT**, manually built
    - Entry point is Ray Generation Shader, supported by Intersection and Anyhit shaders
- **Nvvk** library for initialization and memory management in ray tracer

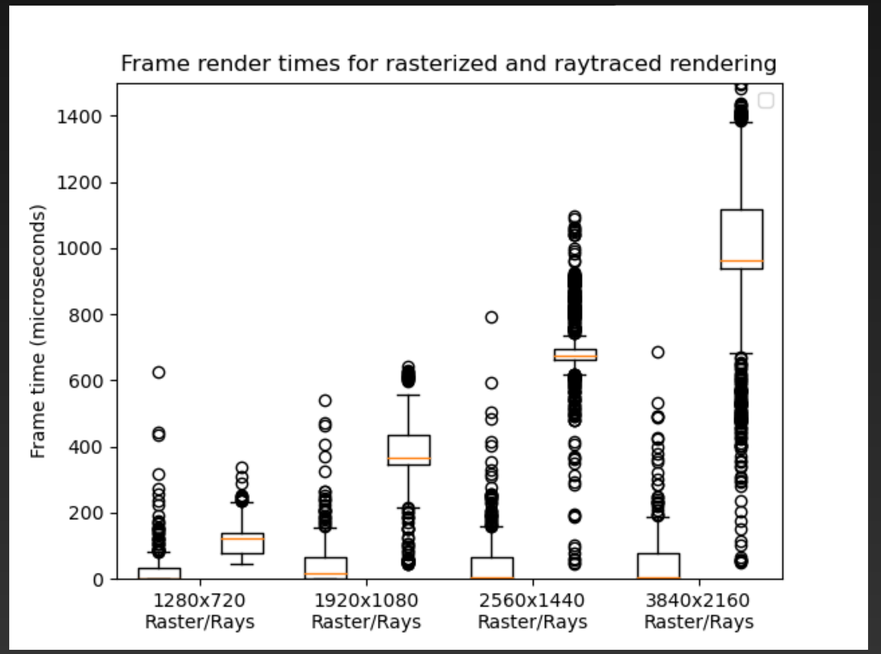
Results
Rasterization Baseline

- Cannot track AS build time
- Memory usage increases linearly with frame buffer resolution when using rasterization
- Memory usage increases exponentially with frame buffer resolution when using ray tracing
- Memory usage is an order of magnitude higher when ray tracing



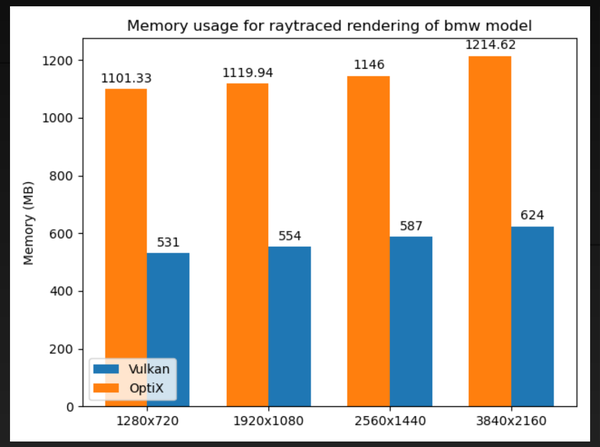
Results
Rasterization Baseline

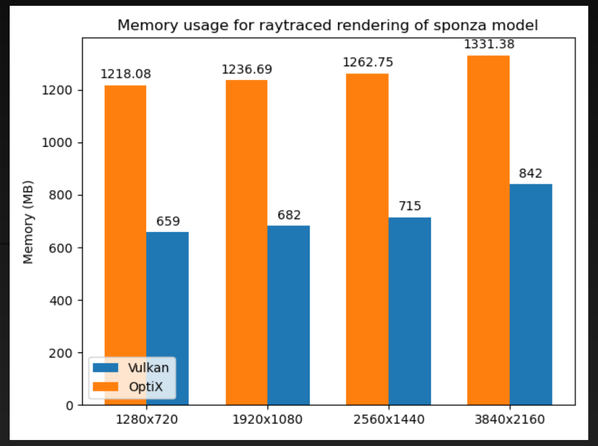
- Frame Time increases linearly with resolution in both techniques
- The increase is more pronounced when using ray tracing
- Frame times are unstable in both techniques

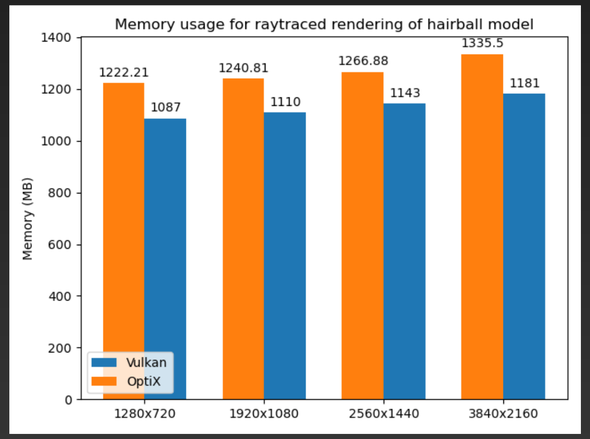


Results
Memory Usage

- Vulkan has a lower memory usage than OptiX
- Memory usage across different geometry amounts is more stable in OptiX
- Memory usage increases logarithmically with rendering resolution
- Memory usage tends to converge across both libraries as scene geometry increases



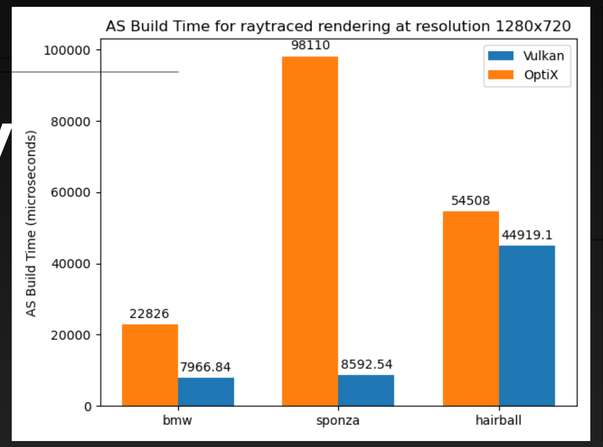


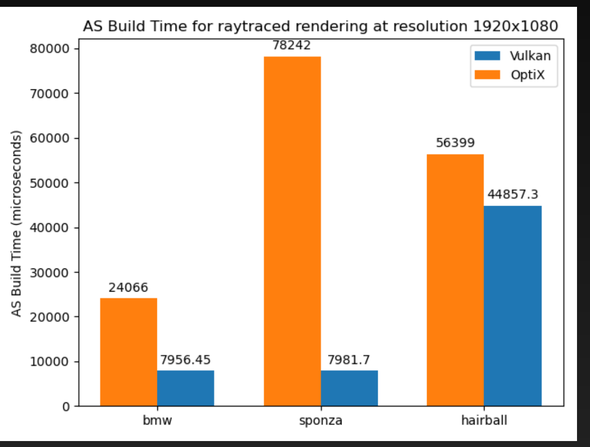


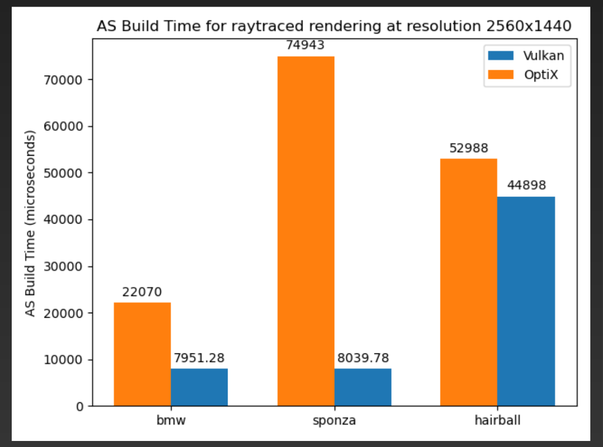
Results

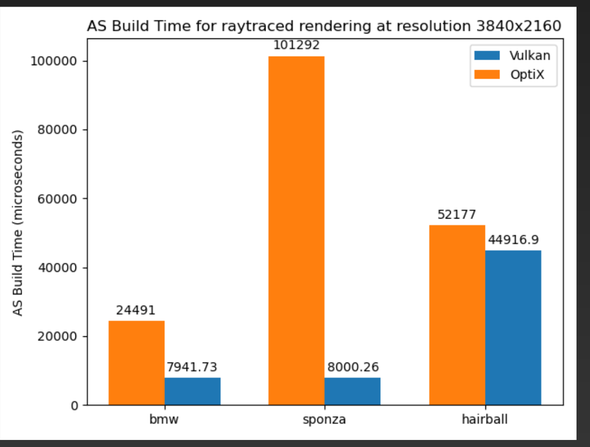
Acceleration Structure Build Time

- Vulkan increases ASBT **exponentially** with **triangle** count
- OptiX increases ASBT **linearly** with **BLAS** count
- ASBT is overall lower in Vulkan





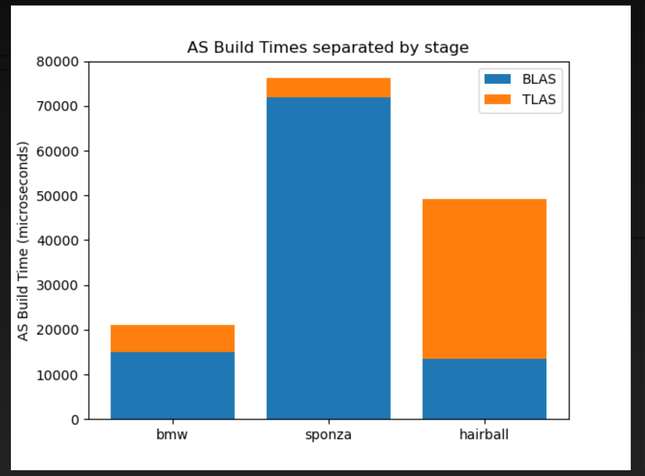


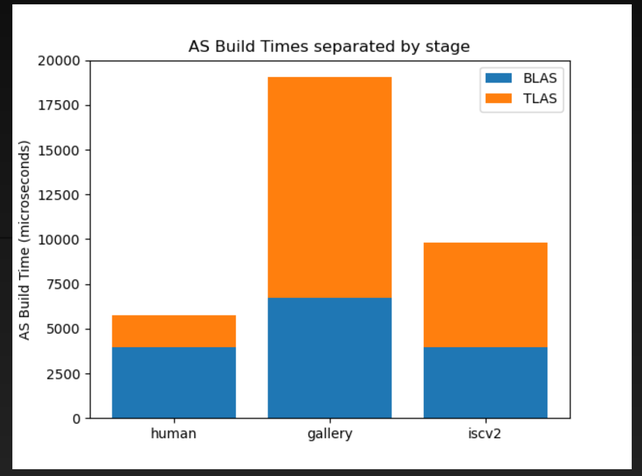


Results

Acceleration Structure Build Time

- Vulkan increases ASBT linearly with triangle count
- OptiX increases ASBT linearly with BLAS count
- ASBT is overall lower in Vulkan
- ASBT is independent of screen resolution

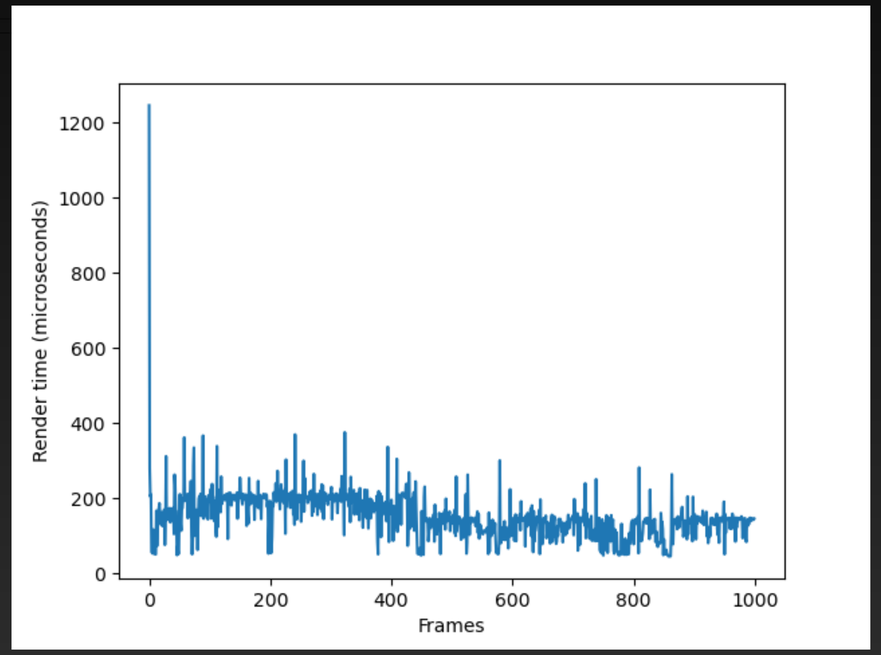




Model	BMW	Sponza	Hairball	Human	ISCV2	Gallery
BLAS Count	73	393	2	20	7	1

Results
Frame Render Time

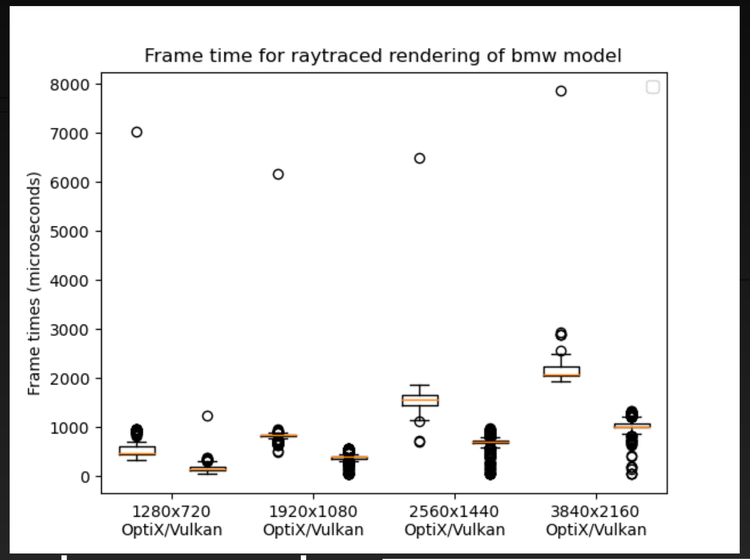
- Count time to render to frame buffer, not to display it
- In a modern OS, times are very unstable
  - Measure 1000 frames at a time
- Remove first frame time in Vulkan, always an outlier

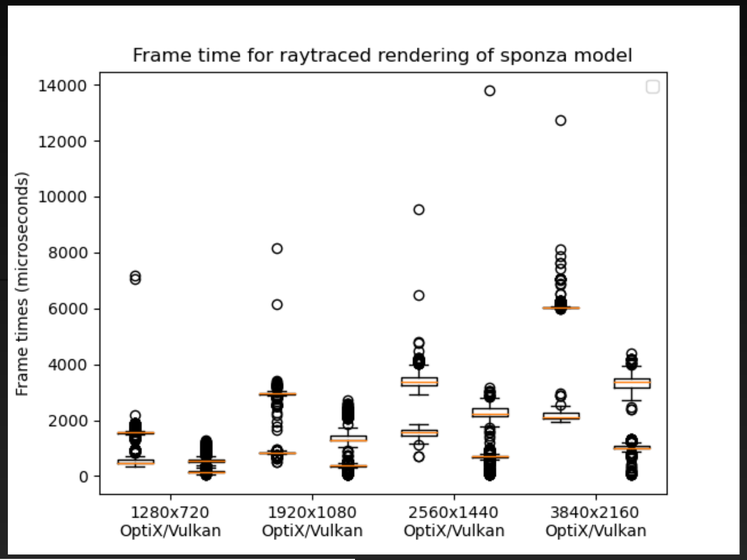


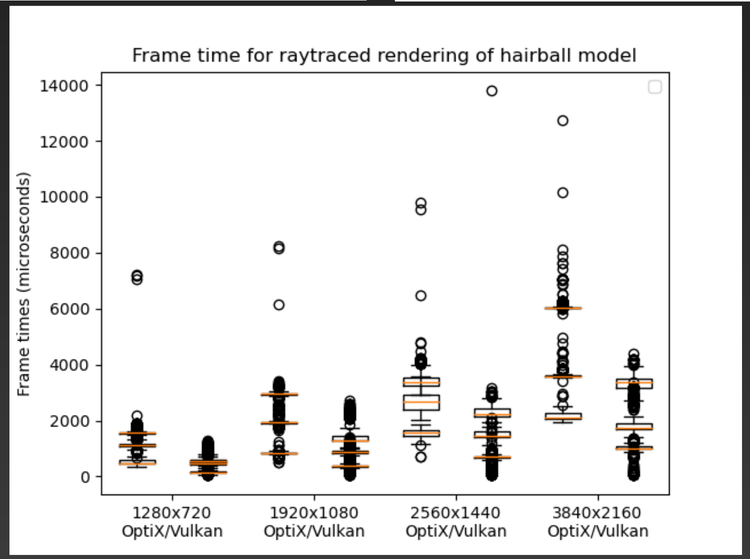
Results

Frame Render Time

- Render time increases with resolution
- Vulkan is overall faster than OptiX
- Not only the triangle count makes render times increase
- Both libraries increase render times with the number of meshes in scene

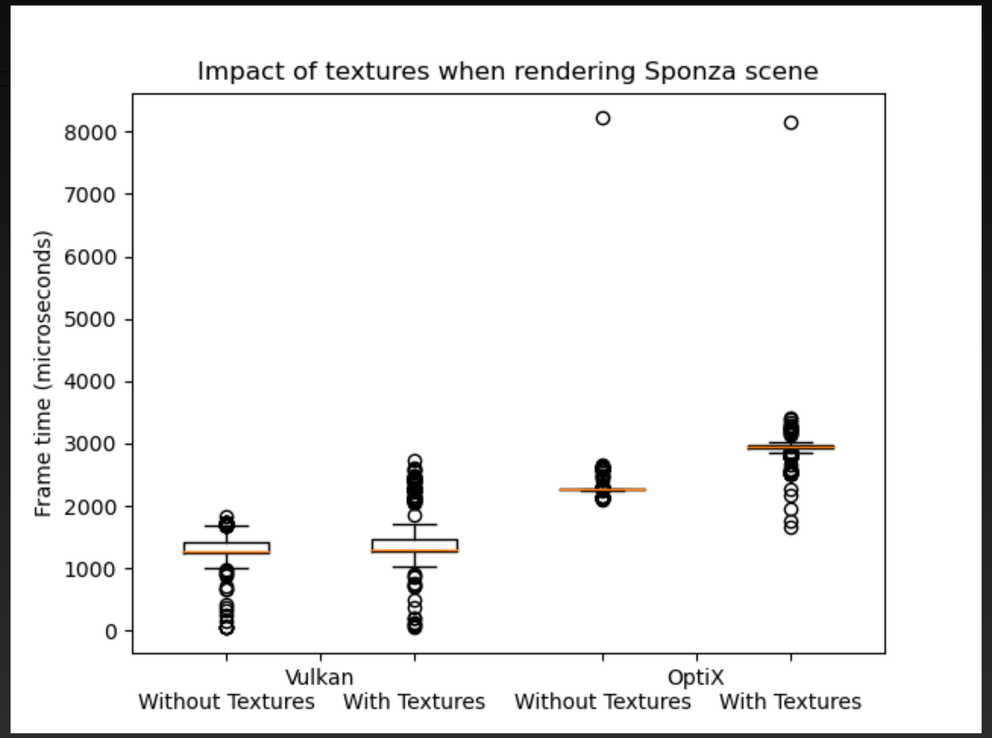






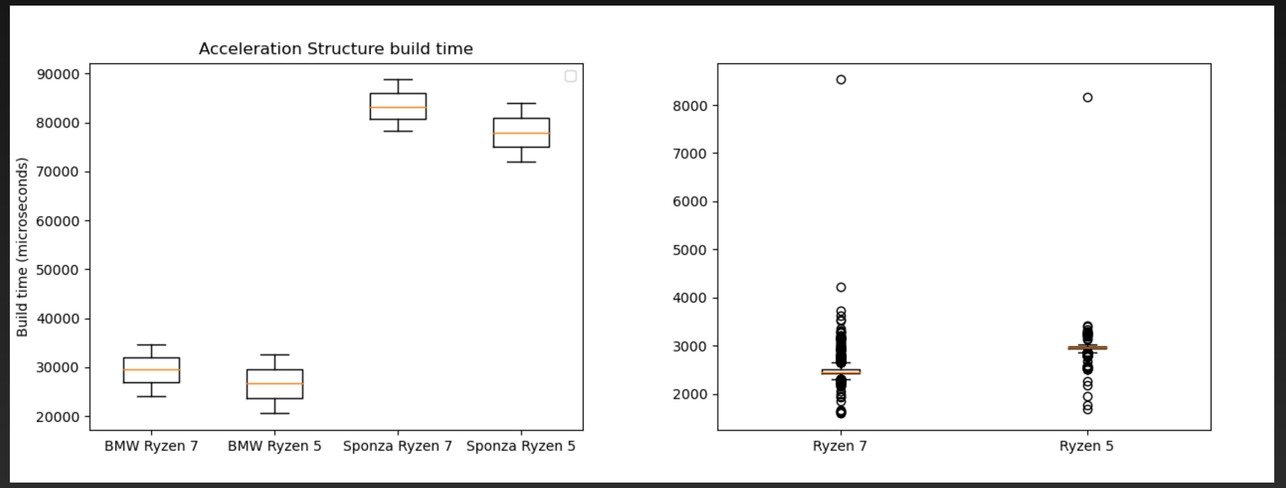
Results
Frame Render Time

- Vulkan is overall faster than OptiX independently of textures
- Vulkan is unaffected by the presence of textures
- OptiX increases render time when using textures



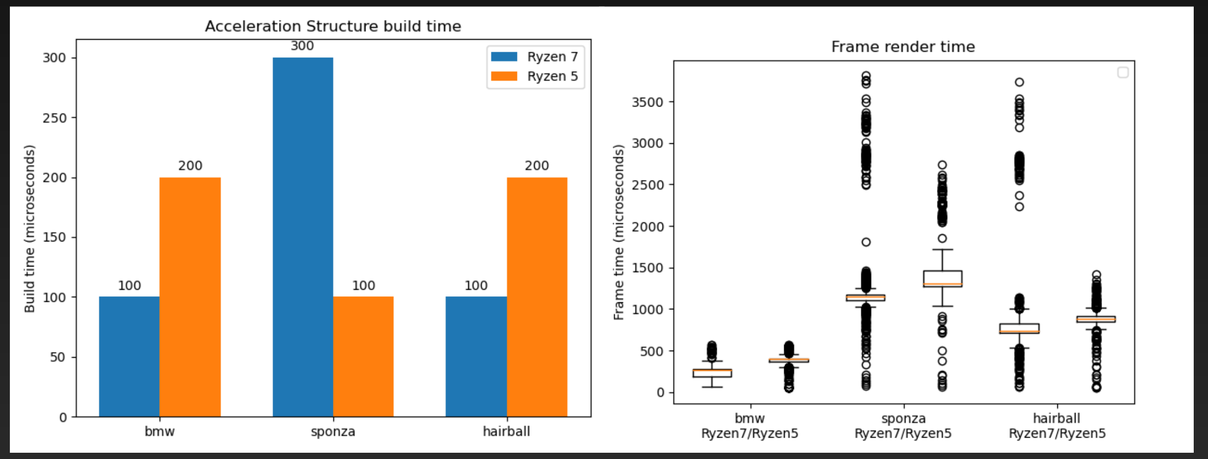
Results
Hardware Comparison

- For OptiX:
  - A faster CPU leads to marginally faster ASBT and render
  - Smaller than 10%, could be noise
  - Happens across all settings and models



Results
Hardware Comparison

- For Vulkan:
  - Less dependent on CPU than OptiX
  - Still relevant to performance



Results

Outlier Statistical Analysis

- Between and a third of data points are outliers
- Dispersion is different for each set
- Probably due to variable system load during testing

	Ryzen 7	Ryzen 5
Outlier percentage	41.7	31.8
Bigger than median	308	95
Smaller than	109	223

Results

Virtualization effects

- Comparison not as exact as others
- Only Vulkan worked under a VM
- ASBT is slower when virtualized
- Makes sense, CPU-bound to an extent

Hardware	Virtual Machine	Bare Metal
Time (microseconds)	400	100

Conclusions

Comparing Vulkan and OptiX

- Vulkan has longer development time and better performance than OptiX
- Ray Tracing is much slower than Rasterization, independently of screen resolution, sometimes by orders of magnitude
- Most tracked metrics when evaluating RT performance are render time, AS build time and memory usage
- Vulkan’s ASBT depends largely on triangle count, while OptiX’ depends on mesh count
- OptiX’ render time is sensible to the presence of textures, while Vulkan is not

Conclusions

Comparing Vulkan and OptiX

- Vulkan works without issue in a VM, while OptiX has driver issues
- Rendering is primarily GPU-bound, with the CPU being almost irrelevant

Conclusions
Future Work

- Tests with dynamic geometry
- Advanced RT implementations (reflections, other materials, etc.)
- Porting to the Nvidia Falcor framework

Comparison of Ray Tracing GPU Implementations

2021-2022

Universitat Politècnica de València
Dept. of Computer Systems and Computation

Master's Thesis
Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Author: Luis Carlos Catalá Martínez
Director: Francisco José Abad Cerdá