NAP Framework & Vulkan

Efficiently pre-visualize and control physical objects
Data Driven
Open Source
Real-Time
Low Overhead
C++
Cross Platform
Toolkit
For the Physical Space
The Problem

old solution

No real-time control over lights
No live sequencing and editing of shows
No live previsualization of complete setup in 3D
Solution did not scale well
Individual cables for Servo & LED
High latency due to inefficient communication protocol
Software was unstable, high maintenance cost
The Goal

Configure & Sequence complete shows in Real Time
Support 100+ instances
Run on medium to low spec hardware without dedicated GPU
With or without the physical installation attached
Accurately pre-visualize setup and movement
Ensure high data throughput for real-time control

new solution
Shylight

General Structure
Ethercat

Real Time Industrial Automation Protocol

Physical and Data Link layer only

Short cycle times (<100 us)

Low jitter (<1us)

Distributed clock

High-Precision Synchronization

High number of nodes (2^16)

CoE (CANopen profile over ethercat)
OS / Kernel

Considerations

Windows 10
- 8000 µs, 125 hz
- Dropped Frames

macOS
- 4000 µs, 250 hz

Linux

Ubuntu 20.04 LTS Generic Kernel
- 8000 µs, 125 hz
- Dropped Frames

Ubuntu 20.04 LTS Liquorix Kernel
- 1000 µs, 1000 hz

Realtime Kernel
- 250 µs, 4000 hz
Ethercat

Hardware Configuration
app::Render()

equivalent example:

- Start rendering a new frame
- Begin recording headless commands
- Draw video to texture using render target
- Stop recording headless commands
- Start recording commands for Window 1
- Start render pass Window 1
- Render background video
- Sort and render objects 1 and 2
- Render the GUI on top
- Stop render pass Window 1
- Stop recording commands for Window 1
- Done rendering frame
nap::RenderableComponent

ShyightApp::Render()
  → Render Installation component instance
    - draw()
    - selectModel()
    → onDraw()
  → Render Physical component instance
    → onDraw()

- MaterialInstance
  - RenderService::getOrCreatePipeline()
  - vkCmdBindPipeline()
  - MaterialInstance::update()
  - vkCmdBindDescriptorSets()
  - vkCmdBindVertexBuffers()
  - vkCmdBindIndexBuffer()
  - vkCmdDrawIndexed()
Descriptor Set

caching
Resource Management

Geometry & Texture Buffering Example
Conclusion

Key Takeaways

Vulkan simply *works* cross platform

Performance Increase of ~40%

- Compared to NAP OpenGL backend
- MacOS (MoltenVK), Windows & Linux

No noticeable difference for developers

- Render interface largely the same

Debug layers are a *godsend*

Headless rendering is finally an option

Explicit = Good
Coen Klosters | Founder & Lead Developer
coen@napframework.com
www.napframework.com
coen@naivi.nl
www.naivi.nl
Contactweg 47
1014 AN Amsterdam
The Netherlands
Special Thanks

Studio Drift

Jelle van der Beek

Ritesh Oedayrajsingh Varma

Bas Korsmit

Tim Groeneboom

Lesley van Hoek

Dodo Kis

Thijs Krooswijk

All of our Users & Clients

The Khronos Group