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
new solution

Configure & Sequence complete shows in Real Time
Support 100+ instances
Run on low spec hardware with integrated GPU
With or without the physical installation attached
Accurately pre-visualize movement
Ensure high data throughput for real-time control
Beckhoff Twincat 3

Industry Standard Automation Software

Only suitable for developers

Completely replaced with our own RT solution

SOEM C library

simple open ethercat master
Shylight
General Structure
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
Rendering Shylights

Controller

- get state
- render thread

Render Installation component instance
- draw()
- selectModel()

ShylightApp::Render()

Constant Material
- hard coded resource
- UniformOverrides

Render Abstract component instance
- onDraw()

Render Physical component instance
- onDraw()

Physical Material
- resource
- UniformOverrides

Physical Shader
- resource
- VertShader
- FragShader

physical vert file

physical frag file

ShylightMesh
- resource
- getInstance()

install.fbx

lamp.mesh file

suspension.mesh file

led.mesh file

Vulkan

convert

load

load
app::Render()

eample
nap::RenderableComponent

ShyightApp::Render()

Render Installation
component instance

draw()
selectModel()

onDraw()

Render Physical
component instance

onDraw()

onDraw()

RenderService::getOrCreatePipeline()

vkCmdBindPipeline()

MaterialInstance::update()

vkCmdBindDescriptorSets()

vkCmdBindVertexBuffer()

vkCmdBindIndexBuffer()

vkCmdDrawIndexed()
Conclusion

Key Takeaways

Vulkan simply **works** cross platform

Performance Increase of ~10-40%

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

No noticeable difference for developers

*Render interface almost the same*

Debug layers are a **godsend**

Headless rendering is now an option

Explicit = Good
Official Support for Raspberry Pi
- Very promising initial results
- Good for pre-visualization purposes

Vulkan Compute

Web Client
- Data-driven web-portal
- Viewport streaming (0.6+)

Realtime Audio Sequencing
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
Klaas van der Molen
Jelle van der Beek
Ritesh Oedayrajsingh Varma
Bas Korsmit
Chris Heywood
Tim Groeneboom
Lesley van Hoek
Dodo Kis
Thijs Krooswijk

All of our Users & Clients
The Khronos Group