Lukas Rennhofer 16.4.2025 **V:2025:0:1**

# Render Engine Architecture: Multi Render Hardware Interface (MRHI)

## 1. Abstract

This paper presents a novel architecture for a Multi-Render Hardware Interface (MRHI) designed to support multiple rendering backends such as OpenGL, Vulkan, and DirectX across different platforms including Windows, Linux, and Nintendo Switch. The MRHI enables seamless runtime or compile-time switching of rendering APIs through a unified interface, minimizing coupling and maximizing flexibility. The architecture includes shader abstraction, resource management, a dynamic pipeline system, and modular render paths, providing a scalable foundation for next-generation graphics engines.

---

## 2. Introduction

As modern game engines grow increasingly complex, the need for flexible and portable rendering systems has become paramount. Supporting multiple graphics backends—such as OpenGL, Vulkan, DirectX, and platform-specific APIs like NVN (Nintendo Switch)—is essential for ensuring broad hardware compatibility. However, most rendering abstractions today are either tightly coupled with specific APIs or rely heavily on runtime polymorphism, which can be costly and unsuitable for low-level or console development. Modern engines like Unreal Engine heavily rely on a complex RHI (Render Hardware System), which is very flexible and can scale enormously, but resources like shaders or API specific content need to be handled individually, which also impacts performance and development. The reason for this approach is to find a new way to abstract this context and find a way to compile a renderer with a unified interface that supports multiple low-level graphics APIs while preserving performance, platform compatibility, and to use as little storage space as needed.

---

## 3. MRHI Architecture Overview

The Architecture is split into several key modules:

- RenderAPI Selector**:** Determines active API based off of the platform at runtime or compile-time. This enables switching APIs during runtime or compiling the renderer with only one specific API independently.

- Device Abstractions**:** Encapsulates command buffers, pipelines, shaders, and buffers.
- Shader Interface (RShader)**:** Unified representation of shaders, independent of source language (GLSL, HLSL, SPIR-V).
- ResourceManager**:** Handles textures, and cross-API compatible assets.
- RenderPath_Interface**:** Abstract base for systems like Forward, Deferred, or custom pipelines (e.g., RenderPath3D).
- Pipeline State Cache: Tracks and reuses render states efficiently.

---

## 4. Shader Abstraction Layer

A possible shader translation and compilation system would allow the use of a unified shader language or preprocessed macros to generate platform-specific binaries. The system would support:

- GLSL and SPIR-V for Vulkan/OpenGL
- HLSL for DirectX
- Optional cross-compilation via shader transpilers
- Unified reflection for resource binding
- Hot-reloading capability

The shader layer is composed of three primary parts:

1. BaseShader (Abstract Class)
2. BackendShader (Concrete Implementation)
3. ShaderCompiler / Translator (Preprocessor and Metadata Extraction)

For Example:

```
class UnifiedShader : public RShader {
public:
    void Use() override {
        backend->Use();
    }
    void SetInt(const std::string& name, int value) override {
        backend->SetInt(name, value);
    }
private:
    std::unique_ptr<RShader> backend; // OpenGLShader, VulkanShader, etc.
};
```

---

## 5. Evaluation and Benefits

- **Scalability:** Add new backends or features with minimal change.
- **Performance:** Optimized virtual interface usage, GPU-friendly layout.
- **Runtime Flexibility:** Switch between renderers or platforms.
- **Maintainability:** Cleaner structure and modular expansion.
- **Cross-Platform Compatibility:** Shared logic across devices.
- **Theoretical Consistency:** Based on proven software architecture principles.

---

## 6. Conclusion

The MRHI architecture provides a flexible, scalable foundation for modern rendering engines, enabling cross-platform graphics with minimal boilerplate. It bridges the gap between low-level API intricacies and high-level engine workflows, promoting maintainability, consistency, and rapid iteration.

This architecture is currently being implemented within the *CHIFEngine*, a custom-built, low-level real-time engine designed to support advanced rendering capabilities across multiple platforms. The MRHI as described herein should be viewed as a theoretical and architectural framework intended to guide the structured evolution of such engines.

---

## 7. References

[1] Epic Games, "Unreal Engine RHI Architecture".
[2] Khronos Group, Vulkan Specification
[3] Microsoft, DirectX 12 Documentation
[4] The Switchbrew Team, Nintendo Switch Development Wiki
[5] MoltenVK, Vulkan Portability on macOS and iOS
[6] CHIFEngine, Lukas Rennhofer

---