Mastering GFXReconstruct

Sept. 18, 2025

LunarG, Inc.

Introduction

This document contains formatted versions of LunarG blog posts describing how to use the popular GFXReconstruct tool that were published on LunarG's website starting in September 2025. This multi-part series was designed to give developers a comprehensive understanding of how to use GFXReconstruct and appreciate the value it brings to graphics development workflows.

Chapter 1:

A Powerful Open-SourceTool for Graphics API Capture and Replay

Welcome to the first post in our comprehensive series on GFXReconstruct, an open-source tool designed to capture and replay graphics API calls. Whether you're a graphics developer debugging a complex rendering issue or optimizing performance for a new platform, GFXReconstruct is a powerful ally in your toolkit. In this foundational post, we'll explore what GFXReconstruct is, its evolution, its licensing, and its core value propositions for developers. We'll also dive into its primary use cases and the APIs and operating systems it currently supports. By the end of this post, you'll have a clear understanding of how GFXReconstruct can enhance your graphics development workflow and why it's a critical tool in the graphics ecosystem.

Background

A possible way forward starts with tracking the resources during capturing. This allows additional content to be stored in the capture file that will assist detection of items (such as buffer addresses) during replay. Then, during replay, these handles and addresses will be fixed to generate the original desired result of the captured content.
The concept of capturing and replaying graphics workloads has roots stretching back to the early days of interactive graphics. In the 1980s and '90s, Silicon Graphics (SGI) provided tools like GLdebug for its Iris Workstations running Iris GL. GLdebug could log every API call—along with parameters and warnings—and even generate replayable C code from a captured session.[1] Around the same time, the X Window System introduced the RECORD extension, enabling low-level capture and replay of X11 protocol events, including drawing commands.[2]

As real-time 3D graphics evolved, more sophisticated tools emerged: GLIntercept (~2002) provided the first open-source OpenGL call interceptor with basic replay[3], while PIX from Microsoft introduced robust Direct3D capture and replay on Xbox and Windows platforms. [4]

Graphics APIs like Vulkan and DirectX 12 are now the backbone of modern, high-performance graphics applications, enabling developers to harness the full power of GPUs. Debugging and optimizing these applications can be challenging due to the complexity of API calls and their interactions with hardware. Previous capture/replay tools successfully laid the groundwork for modern, cross-platform tools like RenderDoc and GFXReconstruct, which bring precision, automation, and open-source, cross-platform flexibility to graphics debugging and performance analysis.

GFXReconstruct's Place in the Ecosystem

Developed by LunarG in collaboration with partners like AMD's GPUOpen team, GFXReconstruct began as a Vulkan-focused tool aimed at improving the quality of Vulkan applications. Initially integrated into the Vulkan SDK in 2020 (version 1.2.141), it replaced earlier tools like Vktrace/Vkreplay, offering enhanced capture and replay capabilities. Over time, GFXReconstruct evolved to support DirectX 12 (D3D12) and DirectX Raytracing (DXR), announced in January 2023, reflecting its growing relevance in the Windows gaming ecosystem, where D3D12 is a dominant standard. This expansion demonstrates GFXReconstruct's adaptability to meet the needs of developers working across multiple graphics APIs.

Licensed under the MIT License, GFXReconstruct is freely available for use, modification, and contribution, fostering a collaborative community of developers who can enhance its functionality or tailor it to specific needs. Its open-source nature ensures accessibility and encourages contributions, such as bug fixes or support for additional APIs, making it a living project that evolves with the graphics industry. As part of the broader Vulkan ecosystem, with sponsorship from Valve and with contributions from industry leaders like AMD, GFXReconstruct plays a pivotal role in providing cross-platform, high-efficiency tools for graphics development.

Looking forward, while currently focused on Vulkan and DirectX 12, GFXReconstruct's API-agnostic container format can support additional APIs in the future, such as OpenXR or Metal, as community or industry needs arise. This extensibility positions GFXReconstruct as a versatile tool for the evolving graphics landscape.

Unlike full-featured GPU debuggers like RenderDoc or Nsight, GFXReconstruct is laser-focused on capture and replay—making it especially useful for automation and integration into regression testing systems, CI pipelines, and cross-platform validation.
Its modular CLI toolset allows for:

- Capturing workloads using Vulkan or DirectX.
- Replaying .gfxr files to reproduce visual results.
- Converting or trimming captures for performance and debugging.
- Inspecting captured data for analysis or tooling.

Understanding API Capture and Replay

API capture involves intercepting and logging the sequence of graphics API calls made by an application, creating a capture file that records the application's rendering commands and state. Replay, on the other hand, allows developers to re-execute these captured commands on the same or different hardware, enabling analysis, debugging, or optimization without needing to run the original application.

Programming modern GPUs is inherently complex, especially when trying to understand what happens between your application's API calls and what eventually shows up on screen. Whether you're diagnosing a rendering bug, optimizing performance, or validating behavior across hardware, you often need to freeze time and replay a precise sequence of GPU commands.

At its core, GFXReconstruct operates like a graphics black box recorder. It intercepts API calls—such as Vulkan commands or DirectX 12 draw calls—and logs them into a capture file. That file can then be replayed independently of the original application. This provides a snapshot of your application's graphics behavior at a specific moment in time.

GFXReconstruct excels at this process, providing a robust framework for capturing and replaying API calls with precision, making it easier to diagnose issues, optimize performance, or test compatibility across platforms.

The benefit? Developers can:

- Reproduce rendering issues even without access to the full application.
- Analyze the API usage offline.
- Test across hardware platforms.
- Profile GPU workloads in isolation.

Capture and replay workflows are foundational in modern graphics debugging, and GFXReconstruct delivers a clean, deterministic approach to doing just that.

Use Cases for Developers

GFXReconstruct is a cornerstone tool for graphics developers, performance engineers, QA teams, and platform validation labs, offering powerful capabilities for debugging, optimization, and testing. Its versatility stems from four key design goals, each supporting critical use cases in GPU software development:

- **Fidelity**: Capture and playback on same device with identical results
- **Integrity**: Optimizations that stay true to application behavior
- **Portability**: Playback across a broad range of devices with variable fidelity
- **Performance**: Deliver the performance required for usability and interactivity.

Important GFXReconstruct use cases include:

- **Defect Reproduction and Debugging:** Debugging visual artifacts, crashes, or incorrect rendering in graphics applications can be challenging, especially when issues are intermittent or hardware-specific. GFXReconstruct captures a complete sequence of API calls, creating a reproducible test case that can be replayed on the same or different hardware. This enables developers to isolate defects by analyzing the exact application state at the point of failure without needing to recreate the issue in the live application. For bug reporting, capture files provide a standardized, reproducible format that improves communication with driver vendors or API maintainers, accelerating resolution times.

- **API Usage Analysis**: Understanding how an application interacts with graphics APIs like Vulkan or DirectX 12 is essential for ensuring correct and efficient implementations. GFXReconstruct enables developers to inspect API call frequencies, state changes, resource usage, and potential misuses. This analysis helps identify inefficiencies or errors, making it a powerful tool for optimizing API usage and ensuring compliance with best practices.
- **Performance Profiling and Optimization:** Optimizing graphics applications requires detailed insight into rendering performance. GFXReconstruct supports performance profiling by capturing API call traces that can be analyzed to pinpoint bottlenecks or inefficient resource allocation. Developers can use these insights to optimize critical code sections, ensuring better performance on specific hardware.
- **Regression Testing and Platform Bringup:** As graphics applications and drivers evolve, ensuring that updates don't introduce regressions is critical. GFXReconstruct supports regression testing by enabling developers to capture known-good rendering outputs and replay them after code or driver updates to verify consistent behavior. This is particularly valuable for platform bringup, where developers validate API compatibility and performance when porting applications to new hardware or operating systems. By replaying captures across diverse environments, GFXReconstruct ensures smooth operation and reduces the risk of regressions, making it indispensable for driver development and application stability.

As you can see, GFXReconstruct serves as a versatile tool for developers working on Vulkan and DirectX 12 applications, enabling them to build robust, high-performance software across Windows, Linux, and Android environments.

Current OS and API Support

GFXReconstruct is designed for cross-platform compatibility, supporting the following operating systems and graphics APIs as of its latest releases:

- **Operating Systems:**
  - **Windows**: Fully supported for both Vulkan and DirectX 12, leveraging the Windows 10 SDK (version 10.0.20348.0 for D3D12).
  - **Linux**: Robust support for Vulkan, with tools for capturing and replaying API calls.

    - **Android**: Supports Vulkan capture and replay, with specific configurations for Android 10 and newer, including permissions for external storage access.
    - **macOS**: Supports Vulkan via KosmicKrisp, a Vulkan-on-Metal driver developed by LunarG
- **Graphics APIs:**
    - **Vulkan**: Comprehensive capture and replay support, including experimental OpenXR support for developer evaluation.
    - **DirectX 12 (D3D12)**: Full support for capturing and replaying D3D12 applications, including DirectX Raytracing (DXR), introduced in 2023.
    - **OpenXR**: Initial support for OpenXR was demonstrated by LunarG at the AWE 2025 event.

While Vulkan and DirectX 12 are the primary focus, the API-agnostic design of GFXReconstruct's capture file format lays the groundwork for potential future support of other APIs, such as OpenXR or Metal, depending on community contributions and industry demand.

GFXReconstruct's open-source nature, MIT License, and support for Vulkan and DirectX 12 make it a cornerstone tool for modern graphics development. Its ability to reproduce defects, analyze API usage, profile performance, and support regression testing empowers developers to build robust, high-performance applications across Windows, Linux, and Android.

References

1. Silicon Graphics Inc. GLdebug Debugger User's Guide, 007-1489-030. SGI, 1993. https://irix7.com/techpubs/007-1489-030.pdf

2. X.Org Foundation. RECORD Extension Protocol Specification, X11R7.6. https://www.x.org/releases/X11R7.6/doc/recordproto/record.html

3. Damian Trebilco. GLIntercept: OpenGL Function Interceptor. SourceForge, 2002–2020. https://github.com/dtrebilco/glintercept

4. Microsoft. PIX on Windows – Performance Tuning and Debugging for Games. https://devblogs.microsoft.com/pix/