廣東科學技術職業學院
GUANGDONG POLYTECHNIC OF SCIENCE AND TECHNOLOGY

毕业设计（论文）

基于符号表解析与 Optick 的图形渲染引擎性能检测工具

郭智诚（0113190324）

指导教师：**姜晔**

学院名称	**计算机工程技术学院**	专 业 名 称	**云计算技术与应用**
提交日期	年 月 日	答 辩 日 期	年 月 日

摘　　要

3D 游戏项目上线运营，需要面临各种复杂的软件、硬件环境，游戏性能表现差不齐。这种情况下，是否能做到实时分析用户的图形渲染引擎的性能与优化，是一件非常有挑战也非常有必要的事情。

以市面上常用的分析方式来举例，大多是通过用户或开发人员安装外部通用工具来进行，如安装 Intel VTune。这种既需要用户的授权同意、也需要购买性能优化软件提供厂商的授权认证，而且无法做到精确地分析定位引擎内部特 的私有数据，更做不到针对不同项目进行深度的定制开发和分析。

另一种常见方式是通过在游戏内自己写一套性能采集的内嵌代码，对原有代码进行插入修改并采集数据。这种方式具有便利性与灵活性，但往往以降低原有代码的完整性与可维护性为代价，而且不具备跨项目、跨平台的移植性。

基于上述传统性能检测方式存在的问题，本文对两个方向进行了研究改良。

对于外部通用工具存在的授权、定制等局限性，本文采用开源软件 Optick 进行补充。Optick 是一款用于游戏的超轻量级 C++性能分析器，提供计时采样、项目数据标签定制等功能。其开源特性可以较好地支持软件代码结合至现有项目中，提供私有数据分析与深度定制功能。

对于内嵌分析代码的缺陷，本文采用动态二进制插桩的形式进行弥补。将原先需要嵌入生产环境中的代码，改为程序运行时动态注入的形式。在保持灵活性与深度采集数据的同时，减少对原有代码的破坏。

最后，本文在改良基础上研发了一种基于符号表解析后动态插桩的二进制注入技术，在 Optick 基础上开发了一款性能分析和优化图形渲染引擎的检测工具。

本文研究成果已在金山软件旗下子公司，西山居游戏，上线项目之一《剑侠情缘网络版叁重制版》中获得应用，并对数十万用户的实际体验进行了较大的改善。

在上述研究成果的支持下，项目对引擎渲染线程和个别性能较差的场景进行了针对性分析，改善了产品质量。

关键词：测量；性能优化；图形渲染

A graphics rendering engine profiler based on symbol analysis and Optick

Guo Zhicheng

(Computer Engineering Technical College (Artificial Intelligence College), Guangdong Polytechnic of Science and Technology，ZhuHai 519090，China)

Abstract： The online operation of 3D game projects needs to face various complex software and hardware environments, and the game performance varies. In this case, it is very challenging and necessary to analyze the performance and optimization of the user's graphics rendering engine in real time.

Taking the common analysis methods on the market as an example, most of them are performed by users or developers installing external general tools, such as Intel VTune. This requires not only the authorization of the user, but also the authorization and certification of the manufacturer to purchase performance optimization software, and it is impossible to accurately analyze the specific private data inside the positioning engine carry out in-depth customized development and analysis for different projects.

Another common way is to insert and modify the original code and collect data by writing a set of embedded code for performance acquisition in the game. This method is convenient and flexible, but it often comes at the cost of reducing the integrity and maintainability of the original code, and it does not have cross-project and cross-platform portability.

Based on the problems existing in the above-mentioned traditional performance detection methods, this paper has carried out research and improvement in two directions.

For the limitations of authorization and customization of external general tools, this article uses the open source software Optick to supplement. Optick is an super-lightweight C++ performance analyzer for games that provides timing sampling, item data label customization, and more. Its open source feature can better support the integration of software codes into existing projects, and provide private data analysis and deep customization functions.

For the defects of the embedded analysis code, this paper adopts the form of dynamic binary instrumentation to make up for it.

Finally, on the basis of improvement, I have developed a binary injection technology

based on dynamic instrumentation after symbol table parsing, and develops a detection tool for performance analysis and optimization of graphics rendering engine based on Optick.

The research results of this paper have been applied in "JX3", one of the online projects of Season Games, a subsidiary of Kingsoft Inc., and the game experience of many users has been greatly improved.

With the support of the research results above, the project has carried out targeted analysis on engine rendering threads and individual scenes with poor performance, and improved product quality.

Key words： Measurement　Performance Analysis　Graphics Rendering

目　　录

1. 前言

随着 3D 游戏研发技术的发展，用户对游戏软件产品的品质要求正在逐渐变高。游戏画面的精细化发展与游戏引擎系统的研发复杂化，使性能测量与优化技术的重要性在不断上升。在此背景下，如 Intel、NVIDIA、AMD 等硬件设计厂商与 Unity、Unreal 等游戏引擎软件研发厂商，均开发了种类繁多的通用图形性能分析软件。同时，对于不同项目的特殊需求，大型游戏研发厂商均各自组建质量保证部门，通过项目内嵌分析代码、研发内部性能工具与使用外部通用图形性能分析软件等方式，针对性地进行质量保证工作。

在此背景下，本文对传统性能检测中使用外部通用工具与内嵌分析代码的两种方案，在特定环境下的分析原理与技术局限性进行了研究。基于上述传统性能检测存在的问题，本文基于 Optick 与动态二进制技术分别做出改良，并研发了一种使用符号表解析后动态插桩的二进制注入技术，基于开源软件 Optick 二次开发，完成了一款图形渲染引擎性能检测工具。

2. 前期调研

2.1. 调研内容

2.1.1. 已上线项目性能分析需求

图形硬件的进步推动了交互计算机图形领域研究的爆炸式增长[1]。同时，用户对游戏软件产品的高要求，推动了游戏画面的不断提升。实时渲染算法的高效率运转与硬件的充分利用，成为了保证画面效果的重要因素。对上述两点的保障需求，推动了性能测量与优化技术的发展。从传统的使用计时代码进行测量，到基于硬件事件的深度分析，性能测量技术越发深入。因此，对实时渲染算法的充分优化是可行且非常有必要的。

根据大型 3D 游戏研发公司中的实际研发经验，上线游戏项目的性能优化需求可以按照相关算法所负责的工作职能，分为渲染代码优化与逻辑代码优化；按照优化时间节点，可分为上线前的开发环境优化与上线后的生产运营环境优化。其中，渲染部分的代码性能，通常为程序中资源开销最大的部分。同时，上线后的生产运营环境优化，与开发环境不同，部分性能问题仅能在用户机器上进行复现，复现难度较大。

2.1.2. 业界性能测量常用技术

目前而言，游戏开发业界最常使用的仍为以函数耗时统计与系统性能数据获取为核心的测量技术。基于上述两种测量技术，拓展出众多性能分析领域的集成开发环境。以网易游戏雷火工作室为例，在业务实践中，使用与自研开发了数个工具，实现了完善的压力测试与性能分析集成环境[2]。

其中，Windows 操作系统环境下较为常用的通用测量工具为 Intel VTune Performance Analyzer。Intel VTune 是一款用于单线程与多线程应用程序的性能分析工具，可用于定位应用程序或整个系统中最为耗时的代码片段[3]。

2.2. 调研分析

2.2.1. Intel VTune 软件应用

2.2.1.1. 算法分析功能原理解析

在工程实践中，对于 Intel VTune，最为常用的功能是 Algorithm analysis 算法分析分类下的 Hotspots 热点功能。热点功能基于 VTune 内置的用户模式采样与跟踪收集功能实现。

该功能基于用户模式采样与跟踪收集器进行采集。用户模式采样与跟踪收集器通过中断进程，收集所有活动指令地址的值，并为每个样本捕获一个调用队列。采样指令指针与它们通常为堆栈的调用队列，被保存到收集数据文件中。统计带有调用队列的 IP 样本可在软件内置浏览器中显示调用图或最耗时的调用路径。使用该数据可以了解具有统计重要性的代码片段的控制流[4]。

基于该功能生成的函数调用图或调用路径，可以清晰展示软件中耗时较长的局部函数，定位到性能热点片段，从而针对性地对软件热点进行性能优化。

2.2.1.2. VTune 在特定分析场景下的局限性

一般而言，在工程内部网络的开发环境中，可以较好地通过 VTune 等通用工具进行软件性能测量，快速发现性能热点。但在外部网络与面向用户的生产运营环境中，当性能问题仅能在用户机器上复现时，大型通用工具通常难以使用。开发人员远程调试部署与指导用户使用均具有较高的时间与人力成本。

除生产环境的局限外，通用软件在图形渲染引擎的私有性能数据采集方面也有缺陷，通常无法获取内部数据。

以 LOD (Levels of Detail, 细节层次技术) 技术为例。LOD 是一项通过对模型进行细节分级加载，从而减轻渲染压力的优化技术。在部分性能分析场景中，需要对引擎的各层级模型数量进行统计，确定各层级的数量是否有过多或超出引擎渲染能力范围的情形发生。

以文件打开性能测量为例，部分场景下，需要对引擎内各个模型文件的载入时间进行测量。由于较多游戏采用自身独有的数据打包形式，通用工具无法获取专有算法压缩与解压的文件情况。

2.2.2. 内嵌性能分析代码技术

2.2.2.1. 基于时间的函数耗时统计原理与实现

基于时间的函数耗时统计是性能测量领域最为通用的方式之一。其实现方式一般为，通过获取系统时间或硬件支持的高精度计时器数据，在函数调用前进行记录，并在函数调用完毕后进行第二次记录，通过计算差值获取函数的实际运行时间。

在实际生产环境中，基于编程语言本身或系统提供的普通接口，通常仅能获取到毫秒级的精度数据。而在渲染性能分析中，以帧为单位的统计常为毫秒级，其单帧内的较

多函数调用以微秒级计算。在此背景下，本文使用 Windows 系统环境中的 QueryPerformanceCounter 函数，实现了一个简单的计时器用于展示。

该函数基于硬件高分辨率性能计数器，在调用时返回一个独立于外部时间参考系的高精度时间戳，可用于函数间隔时间测量。其精度根据硬件而细微变化，在循环时间为 3 GHz 的处理器上，其精度大致为 360 纳秒[5]，可用于微秒级数据测量。

```cpp

#include <iostream>
#include <Windows.h>

/*! @fn int main();
 *  @brief      To show an elapsedMicroseconds Counter.
 *              Based on QueryPerformanceCounter(QPC).
 *  @param[in]  None
 *  @param[out] None
 *  @return None
 */
int main()
{
    LARGE_INTEGER n_start, n_end, n_elapsedMicroseconds;
    LARGE_INTEGER n_frequency;
    QueryPerformanceFrequency(&n_frequency);
    QueryPerformanceCounter(&n_start);
    std::cout << "Hello World!" << std::endl;
    QueryPerformanceCounter(&n_end);
    n_elapsedMicroseconds.QuadPart = n_end.QuadPart - n_start.QuadPart;
    n_elapsedMicroseconds.QuadPart *= 1000000;
    n_elapsedMicroseconds.QuadPart /= n_frequency.QuadPart;
    std::cout << "Cost time: " << n_elapsedMicroseconds.QuadPart << " us" << std::endl;
}
```

图 1 高精度计时代码

图片来源：作者提供

2.2.2.2. 内嵌性能采集代码在生产环境中的缺陷

在上述实现中，本文采用了在函数调用前后写入耗时计算代码的方式进行统计。在开发环境中，引擎内部数据与少量函数耗时均可以使用嵌入代码并输出日志的形式进行监测。该代码均为开发人员插入，获取数据方面具有较大的灵活性。

与开发环境不同，在大型上线项目的生产环境中，需要对原有代码进行插入修改，降低原有代码的完整性与可维护性。大量的日志与测试代码会影响业务代码本身的可读性，如图所示，大量无效日志对性能分析产生了较大干扰。

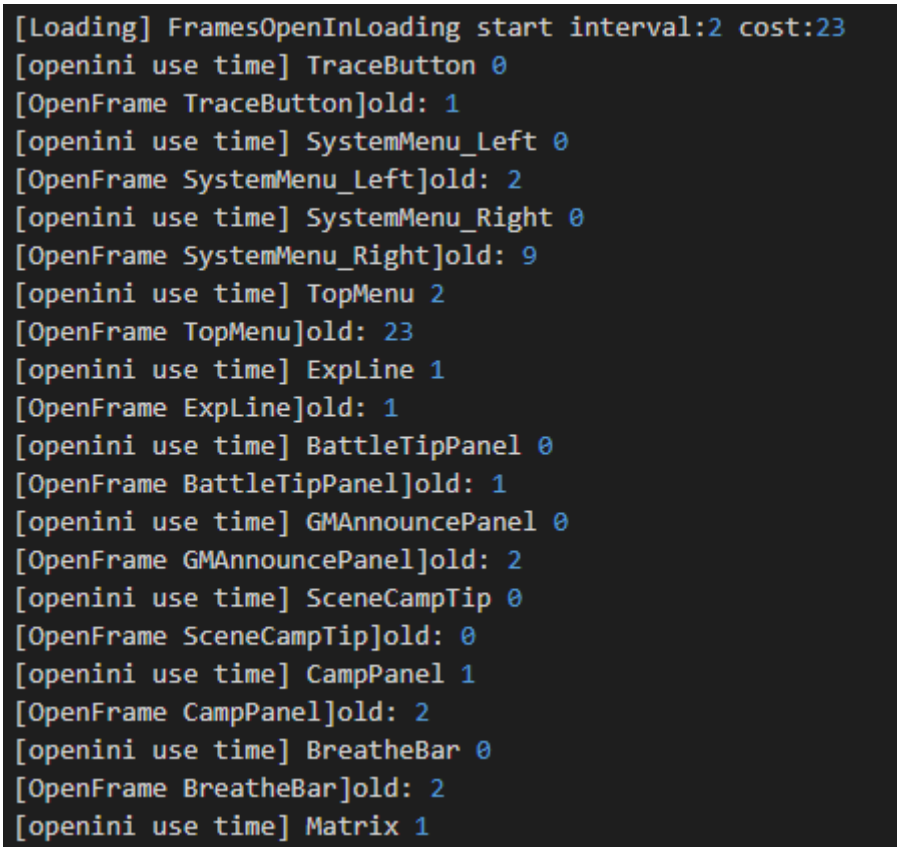

图 2 无效日志

图片来源：作者提供

除对原有代码的干扰外，生产环境中插入的性能采集代码，会在一定程度上影响图形渲染引擎的工作效率。以 VTune 自身的数据展示为例，其性能开销程度在 3-5% 。通用采集工具的大规模采集性能影响较大，而小规模的测试代码重复插入会使开发人员难以维护，对于每个测试点都要进行手动嵌入测试代码，其工程量较大。

3. 研究与实践

3.1. Optick 对于 VTune 的工程实践补充

3.1.1. 实践改进设计

本文采用开源软件 Optick 对 Intel VTune 进行了补充，对针对性性能分析提供了较好的实践。Optick 是一款用于游戏的超轻量级 C++ 分析器，提供了包括采样在内的高效性能分析与优化所需的必要工具[6]。

Optick 基于时间戳进行耗时统计，但与 VTune 的通用采集不同，其采集需要在代码内调用相关接口。在实践中，接口以 C++ 代码宏在需采样函数中插入的形式调用。与通用采集和插入性能采集代码相比，降低了通用采集时的性能开销，仅针对指定函数进行采样。同时，仅需一行代码的调用宏，也提高了测试代码的可维护性。

除具有性能与可维护性优势以外，Optick 也针对工程内部参数提供了 Tags 标记的功能。开启标记功能后，可将自定义数据写入当前宏的采集数据中。对于通用工具难以采集的引擎私有数据，可使用该功能进行数据记录。

3.1.2. 函数耗时统计图表

3.1.2.1. Optick 耗时统计原理

Optick 程序基于内置的事件系统进行统计通知，在代码内渲染线程主循环嵌入其导出的 OPTICK_FRAME 宏，可开启统计标记。随后调用 OPTICK_EVENT 宏可通知程序对嵌入代码所在函数进行描述记录与计时。

在记录发起时，程序基于内置的 Platform 结构体，对不同系统进行了适配，先对系统环境、线程编号、进程编号、函数名、代码行数、图标染色颜色等信息进行记录，随后通知内置计时器启动调用计时。在函数的调用入口与出口分别调用系统高精度计时器进行时间间隔计算。

程序在 Platform 代码中，在 Windows 系统环境下使用 QueryPerformanceCounter 函数；在 Linux 系统环境下 clock_gettime 函数进行时间间隔统计，精度均为微秒级。

在启动与结束计时按钮点击后，程序传递停止计时的事件信号，所有数据保存至内存中，并基于记录的函数信息与染色信息等，进行函数调用图生成，最终通过附带的数据浏览器显示性能耗时数据。

3.1.2.2. 实践效果——以全局耗时数据为例

在工程实践中，本文根据业务需要在原 optick 代码上增加 GRANULARITY 宏，根据传入的 uint32_t 类型值进行分类，用于对不同粒度的性能分析进行控制。在 Optick 事件系统中的 DumpEvent 函数执行时，会将记录的数据传入内存。此时，程序会根据记录软件中选择的粒度等级所记录的值进行判断，使用传入值与记录值进行比较，判断条件成立则记录数据。

在工程项目中，可根据图形渲染引擎中不同的控制层级，对代码进行埋点。实践中主要为 SUMMARY、DETAIL、EXTEND 层。如在一般的如渲染主循环、渲染数据准备与执行渲染等大范围函数中采用粗略的 SUMMARY 层，进行初步性能统计。在需要时使用 EXTEND 层，对各渲染模块中的具体函数进行耗时统计，粒度可达到 DirectX 等图形 API (Application Programming Interface, 应用程序编程接口)的入口上层，如 ID3D11DeviceContext 结构体中的 Map 与 DrawIndexed 耗时统计。前者用于显存读写，后者用于向图形硬件发送渲染命令。

使用该技术后，全局数据收集时产生的巨大性能开销被极大改善，可用于许多通用场景的数据测量，同时保留了对个别深度函数的针对性分析。

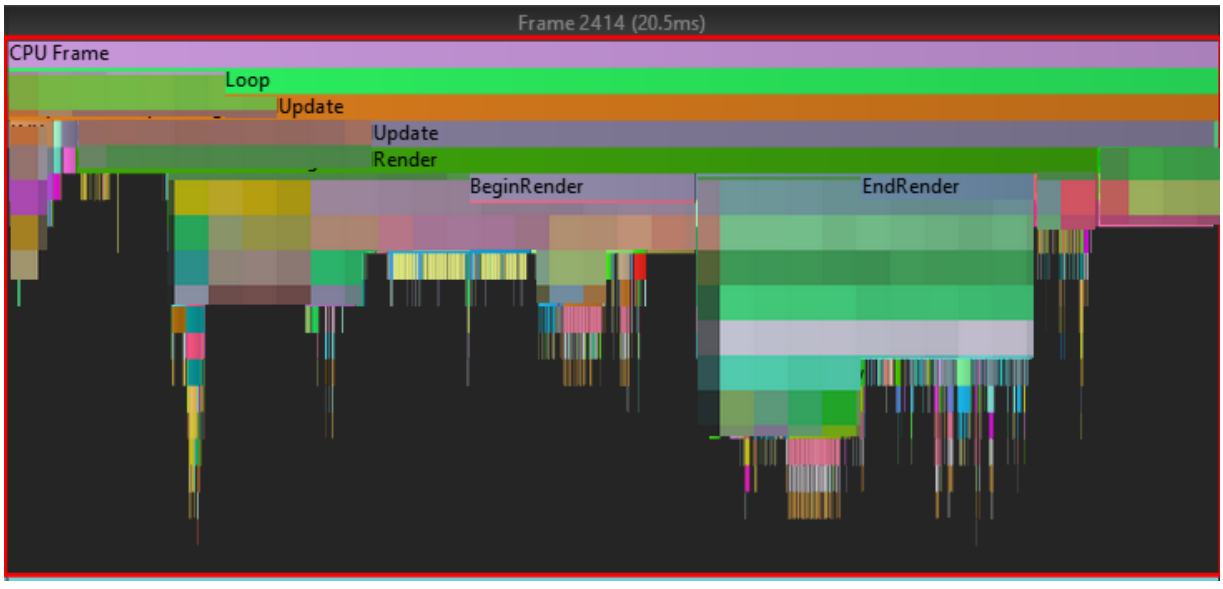

图 3 Optick 全局函数耗时统计

图片来源：作者提供

3.1.3. 项目标记数据统计

3.1.3.1. 标记数据统计原理

Optick 程序的标记功能在内置的事件系统中使用 DumpTags 函数向记录内存写入变量数据。变量数据通过内置的 Optick::Tag::Attach 函数进行传递，可传入 int32, uint32, uint64, vec3, string 五种类型的数据与描述字符串。数据与描述传入 Optick 内置的 MemoryPool 内存分配器所生成的内存空间，随后在 DumpTags 事件函数中向记录内存写入。

3.1.3.2. 实践效果——以文件打开路径统计为例

在工程实践中，该功能可以写入图形渲染引擎内部私有数据至特定函数中，并在数据浏览器中同时显示函数耗时与自定义的标记数据。

以 LOD 数据与模型文件加载数据为例，该功能可在 LOD 相关切换函数后附带显示各层级 LOD 模型数量统计，并显示该函数耗时。在该函数的文件读写中，可将长耗时文件的路径等信息写入标记中，统计文件打开函数的耗时，并显示文件路径。图为长耗时文件实际分析，可以看到该文件的读写在渲染线程中，并占用了较多的时间。

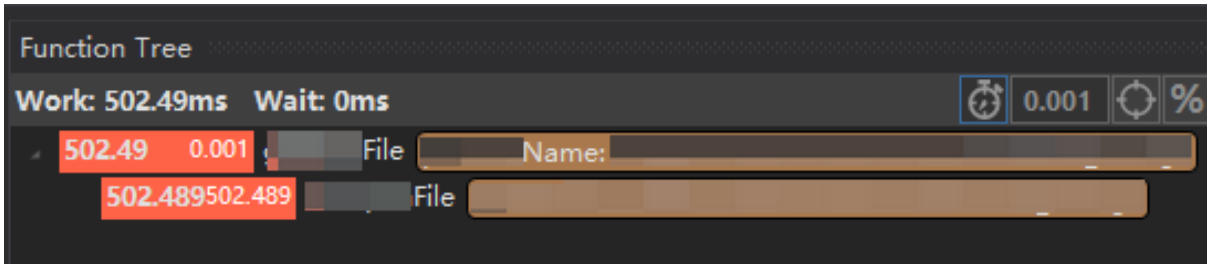

图 4　Optick 文件打开路径标记数据

图片来源：作者提供

3.2. 动态二进制技术对于静态内嵌代码的优化

3.2.1. 优化改进设计

在代码中嵌入测试代码进行性能测量的方案，在大型上线项目的生产环境中，不仅影响发布版本产品的性能，并且需要对原有代码进行插入修改，降低原有代码的完整性与可维护性。

在内部网络的开发环境中，虽然可以较为灵活的进行代码插入，但每一次的临时插

入都需要重新编译图形渲染引擎的相关代码，重新生成应用程序。对于改动引擎头文件的极端情况下，编译缓存复用将会失效，需要重新完整编译整个引擎项目，其开发效率将被极大降低。

因此，对于内嵌性能测量代码的上述缺陷，本文采用外部拦截注入的方式，进行动态二进制注入测量代码。将内嵌代码清除并转为外部注入，可以保持原有代码的可维护性，并保留嵌入代码的灵活性，根据需求进行实时注入。

3.2.2. Detours 库拦截注入

3.2.2.1. 拦截原理

Detours 库是一个在 Windows 系统环境中用于监视和检测 API 调用的软件包，可以拦截运行时代码的动态应用。Detours 将目标函数的前几条指令无条件跳转至用户提供的 Detour 函数，并将来自目标函数的指令保存在 trampoline 蹦床函数中。蹦床函数保存从目标函数中删除的指令与目标函数的其余部分。

当执行到目标函数时，控制流直接跳转到用户提供的绕行函数，并执行所需的拦截预处理。执行完毕后，detour 可将控制权返回给原函数或调用蹦床函数，该函数调用目标函数而不会被拦截。当目标函数执行完毕后，控制权将会移交至 detour 函数，并进行相应的后处理，处理完毕后将控制权返还给原函数[7]。

3.2.2.2. 实践效果——以 DrawCall 调用数拦截统计为例

根据 Detours 库提供的相关特性，可以通过在拦截预处理与后处理中执行性能测量代码，取代内嵌代码方案。所有的性能测量代码均在项目外部编写，并通过 Detours 库拦截目标函数。

以 DrawCall 调用数统计为例，可以很好的应用该特性。DrawCall 是指在图形渲染引擎中调用图形 API 的绘制指令，向图形硬件设备发送绘制命令的一次函数调用，如 DirectX 图形 API 中的 DrawIndexed 函数等。统计该函数的调用次数，可以通过拦截预处理，通过一个全局变量记录单帧中的调用次数，并在调用中累加。在使用 DirectX 图形 API 的图形渲染引擎中，单帧的结束以 IDXGISwapChain::Present 函数为标记。该函数向系统指示，将渲染图像结果呈现给用户。因此，拦截该函数后，在该函数的预处理中查询 Drawcall 的统计变量并清空，可以得到当前渲染帧的 DrawCall 调用数。

3.2.3. 符号表调用

3.2.3.1. 符号文件介绍

符号文件是将项目源代码中的标识符和语句映射到已编译应用中的相应标识符和说明。这些映射文件将调试器链接到源代码，以进行调试[8]。

在 Linux 系统环境下，符号文件中的符号表以编译器内嵌的方式，在编译时通过编译选项控制导出至目标应用的二进制编码中。

在 Windows 系统环境下，符号文件通常以.pdb 为后缀的程序数据库文件的形式存在，可在编译时生成，其内部保存了程序的符号信息。

3.2.3.2. 实践效果——以 PrintLog 函数外部调用为例

关于 Windows 系统环境下的符号表操作，可使用系统头文件 DbgHelp.h 进行相关操作[9]。在注入代码初始化时，需要调用头文件中的 SymInitialize 函数进行符号处理程序初始化。随后使用 SymFromName 函数进行函数名称查询，该函数返回以 SYMBOL_INFO 结构体类型的函数符号信息。通过该类型的数据，可以查询函数的虚拟地址，并基于此对函数进行调用。

本文在一款图形渲染引擎外部，通过远程线程注入的形式，将代码注入目标进程，并基于符号文件调用上述相关操作，向引擎原有的 PrintLog 函数调用日志写入功能，从外部注入代码执行。外部代码拦截原有的 d3d11.dll 运行库，使用 AMD AGS 库替换引擎原有的 D3D11CreateDevice 函数，通过 AGS 库获取显卡相关信息，并将返回信息通过引擎日志系统写入日志文件中。

3.3. 基于符号表解析后动态注入的性能数据采集技术

3.3.1. 综合改进设计

基于上述所有实践，本文在上述技术基础上进行了进一步改进，将以上技术结合使用，提出了一种基于符号表解析后动态注入的性能数据采集技术，以弥补各个改良方案中的缺点。

该方案通过动态注入的方式，规避了原有方案中嵌入代码的缺点。然后通过符号表解析的方案，进一步简化动态注入拦截操作，通过符号文件解析实现快速获取函数地址，将函数地址提供给 Detour 库。随后对 Optick 的事件系统流程进行分析，实现了动态注入后使用 Optick 的方案。通过 Optick 的函数耗时检测方案，实现了外部注入后对指定

函数进行针对性打点检测的成果。最后，对于通用工具中不能获取图形渲染引擎私有数据的缺陷，使用 Optick 的标记功能进行引擎数据写入，将私有数据附加到数据统计文件中，在 Optick 内置的数据浏览器中展示所有性能数据，完成了该论文的所有工作。

3.3.2. 符号表解析与动态注入并结合统计技术

3.3.2.1. 结合原理

本文首先通过外部动态链接库远程线程注入后，基于 DbgHelp.h 头文件中的 SymFromName 函数获取需要拦截的目标函数列表的地址。随后通过 Detour 库中的 DetourAttach 函数对目标函数进行绕行拦截。通过对 Optick 的事件系统分析，可以了解到其通过 OPTICK_PUSH_DYNAMIC 与 OPTICK_POP 两个事件宏进行接口通知。因此，可以在 Detour 库拦截后，在进入函数时的预处理，调用 OPTICK_PUSH_DYNAMIC ，并在退出函数的后处理时调用 OPTICK_POP，从而对目标函数进行耗时统计。随后，对所需的私有数据，在调用 OPTICK_PUSH_DYNAMIC 后通过调用 OPTICK_TAG 写入该函数的统计信息缓存中。

3.3.2.2. 实践效果——以实时动态注入并统计为例

基于上述原理，本文实现了动态注入后进行性能统计的功能。在原有的图形渲染引擎基础上，不内嵌任何性能测量代码，仅通过编译时生成的 Pdb 符号文件，对引擎进行了注入，最终成功测量渲染主要循环函数耗时，并附带各渲染阶段的内部私有数据。

4. 研究与实践成果

4.1. 局部性能分析场景的应用

4.1.1. DirectX 内 map 函数耗时过长导致的低帧率分析

基于该技术，本文对西山居游戏旗下项目《剑侠情缘网络版叁重制版》进行了性能分析，并挑选了一个具有代表性的问题进行分析展示。下图为使用 Optick 后对项目渲染引擎中调用 DirectX 11 图形 API 的 ID3D11DeviceContext::Map 函数时耗时过长的问题。

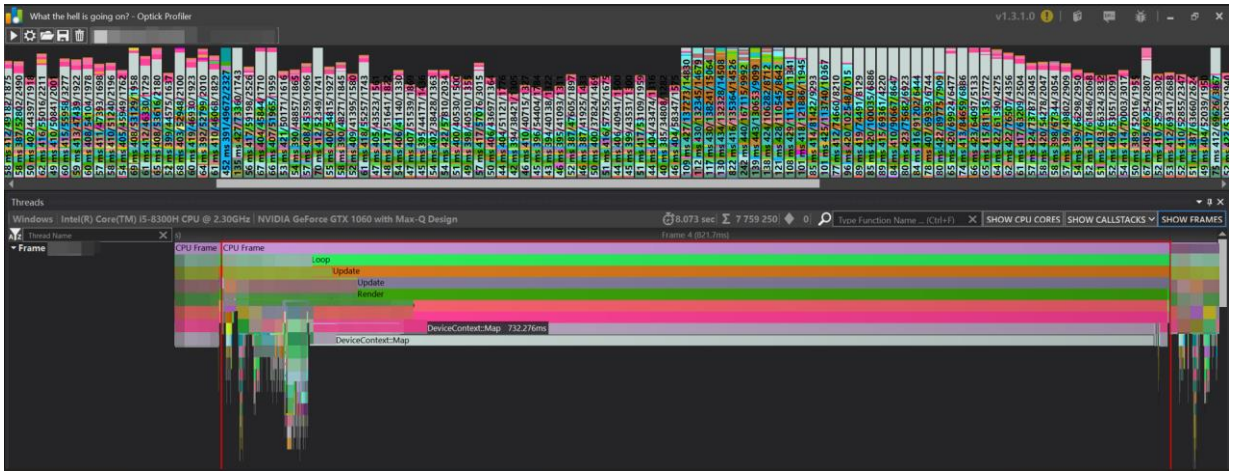

图 6 DirectX API 内 map 函数长耗时

图片来源：作者提供

通过 Optick 的数据浏览器，可以看到在渲染线程中，该 Map 函数执行超过 10 毫秒，且占据了渲染线程中较多的执行时间。同时，该问题在每帧中出现，每帧仅出现一次。根据与正常渲染数据的对比，可以判断该处出现性能异常现象，需要进行针对性分析处理。

经过测试与开发同事的协调研究，最终针对该处使用到的显存空间进行了一次性申请，改动原有重复申请释放的写入方式，解决了该问题。

4.2. 项目总体优化效果提升

4.2.1. 项目总体优化效果数据展示

通过本文实现的上述方案，《剑侠情缘网络版叁重制版》在项目中大量使用改进后

的 Optick 进行性能分析，并根据产出的性能结果与各研发部门进行协调优化，最终在 2022 年 4 月份上线的资料片《江湖无限》中极大改善了产品性能。根据 2021 年运营方公开披露的十二周年庆典时对于“大唐幻境”玩法的参与人数统计，预计改善数十万用户体验。图为产品性能提升展示。

图 7 剑网 3 用户数统计

图片来源：剑网 3 微信公众号. 我们又双叒叕创下新峰值啦！.[DB/OL].

https://mp.weixin.qq.com/s/I-ggvTTadoajw8Al8-oTtQ , 2021-09-01

图 8 《江湖无限》游戏资料片优化性能数据

图片来源：剑网 3 微信公众号.【今日公测】新赛季全部更新汇总，快点进来看一看.[DB/OL].

https://mp.weixin.qq.com/s/7QxwQbhWwZii8qg2Z-pU8A , 2022-04-28

图 9 《江湖无限》游戏资料片优化技术手段

图片来源：剑网 3 微信公众号. 笔记本优化系统测评，居然还送电脑？！！.[DB/OL].

https://mp.weixin.qq.com/s/GtbCj7iBBq_a-MtSoGVx_Q , 2022-04-28

5. 结语

本文结合现有业界前沿方案，结合实际工程实践，在上线项目中完成了一款基于符号表解析与 Optick 二次开发的图形渲染引擎性能检测工具。根据文中所提各个方向方案，该工具仍可向硬件性能数据获取、生产环境自动化注入分析、研运一体化系统等方向发展，具有广泛的应用前景。

本文得到来自广东科学技术职业学院、金山软件公司西山居游戏、业界同行们的指导与大力支持，并在实际工程项目中得到了广泛应用。

参 考 文 献

[1]Tomas Akenine-Mo¨ller,Eric Haines,Naty Hoffman,et al.Real-Time Rendering Fourth Edition[M].Boca Raton,FL,USA:A K Peters/CRC Press,2018:P1-2.

[2]网易游戏雷火事业群.【游戏测试专题】端游的压力测试分享[R/OL].https://zhuanlan.zhihu.com/p/503044178 , 2022-04-29.

[3]Intel.Intel® VTune™ Profiler User Guide Introduction [DB/OL].https://www.intel.com/content/www/us/en/develop/documentation/vtune-help/top/introduction.html , 2022-03-22.

[4]Intel.User-Mode Sampling and Tracing Collection [DB/OL].https://www.intel.com/content/www/us/en/develop/documentation/vtune-help/top/analyze-performance/user-mode-sampling-and-tracing-collection.html , 2022-03-22.

[5]Microsoft.Acquiring high-resolution time stamps [DB/OL].https://docs.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps#resolution-precision-accuracy-and-stability , 2022-01-05.

[6]Bombomby.Optick: C++ Profiler For Games [DB/OL].https://github.com/bombomby/optick/blob/master/README.md , 2020-09-05.

[7]Brian Gianforcaro.Detour OverviewInterception: Interception of Binary Functions [DB/OL].https://github.com/microsoft/Detours/wiki/OverviewInterception , 2020-08-22.

[8]Microsoft.Specify symbol (.pdb) and source files in the Visual Studio debugger (C#, C++, Visual Basic, F#). [DB/OL].https://docs.microsoft.com/en-us/visualstudio/debugger/specify-symbol-dot-pdb-and-source-files-in-the-visual-studio-debugger?view=vs-2022 , 2022-04-30.

[9]腾讯游戏.腾讯游戏开发精粹[M]北京:中国工信出版集团/电子工业出版

附　　录

本文所用开源软件：

Optick - Github: https://github.com/bombomby/optick

致　　谢

又是一季柳絮飞，再不是曾经年岁。三年大学时光转瞬即逝，感谢学校给予的校企实习机会与栽培，感谢姜晔老师对本文提供的悉心指导。在学校与金山软件公司西山居游戏的辛勤培育下，我的技术水平与眼界均得到了飞速的成长。

本文的核心技术来自西山居游戏张强、孙国军、熊文娟、刘马良、马力、李邦戈、赖景茂、周星旭、罗言等众多前辈的教导，求学过程中得到了来自西山居游戏杨林、李晶晶、马希成、李琳与西山居其他同事的支持与鼓励。

本文产出过程中也得到了来自 Meta@Homomniverse 社群、腾讯游戏魔方工作室杨展鹏等众多图形引擎开发行业的朋友们；以廖越颖、疏悦、纪帅、焦一航、谭梦丛、郝奕博、向星宇、帖经明、袁月鸣、陈凯、唐佳乐、邓万睿、明恒光为代表的优秀策划朋友们的技术与精神支持。

感谢陈浩炫、歌宵、胡雪彦、小西洲、徐天行、黎恺毅、李泓德、苏泓昊、伦梓茹、李祎帆、邓梓豪、刘星榆、欧阳瑛珈、瑾颜、清梨、黄依伊、苏盏等与其他未在此处提及的众多家人与朋友们对本文作出的支持与贡献。

最后感谢全体剑网 3 有爱玩家在本文产出过程中给予的众多测试与帮助，这江湖，幸甚有你！

广东科学技术职业学院毕业设计（论文）答辩记录与成绩评定表