jamk

Impact of Optimization on Visual Effects Tools in Unity 6

A Performance and Fidelity Study

Jeremy LeBlanc

Bachelor's Thesis
November 2025
Business Information Technology

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Description

LeBlanc, Jeremy

Impact of Optimization on Visual Effects Tools in Unity 6: A Performance and Fidelity Study

Jyväskylä: JAMK University of Applied Sciences, September 2020, 40 pages.

Degree Program in Business Information Technology. Bachelor's thesis.

Permission for open access publication: Yes

Language of publication: English

Abstract

In recent years, the sophistication of real-time VFX has increased, and with it, several performance challenges for real-time rendering began to present themselves alongside the improved visuals. Efficient optimization in various forms is necessary in order to keep performance stable and visual fidelity consistent across the wide range of hardware platforms. The aim was to evaluate the effect of three optimization methods: overdraw reduction, geometry simplification, and culling on the performance of VFX in the Unity6 Universal Render Pipeline.

An experimental approach was adopted whereby each method of optimization was tested against the same test scene. Results were recorded using the profiling functionality available within the Unity framework through various metrics of performance. A visual comparison of quality was also achieved through a side-by-side review.

The best performance enhancements came from the overdraw reduction technique, which helped cut the memory usage by about 41% and boost the frame rate by approximately 5%. The geometry simplification method provided medium performance enhancements, though the culling optimization technique achieved the highest average frame rate along with a slight increment in the GPU frame time. In no scenario did significant degradation of visuals occur, though it is a qualitative assessment rather than quantitative.

Testing results have shown that targeted optimization can greatly improve runtime performance while retaining visual quality. The findings provide guidelines for achieving acceptable VFX production in real-time rendering environments, with specific applicability to game development, virtual production, and other performance-critical use cases.

Keywords

Visual Effects (VFX); Optimization; Unity 6; Universal Render Pipeline (URP); Real-Time Rendering; Game Development; Performance Benchmarking; Experimental Study; Sustainable Development

Miscellaneous

Confidentiality: None, all research data was gathered using publicly available Unity tools and the Fantasy Kingdom URP demo project, which is free to access for educational and research purposes.

Contents

Figures

Tables

Abbreviations and Glossary of Terms

Table 1

Abbreviations

Abbreviation	Definition
VFX	Visual Effects – The use of computer-generated imagery (CGI) to enhance visual elements in real-time or pre-rendered media.
LOD	Level of Detail – A rendering technique that reduces the complexity of 3D models based on their distance from the camera to optimize performance.
ICVFX	In-Camera Visual Effects – A technique where real-time rendered digital environments are displayed on LED walls during live-action filming to capture effects directly in-camera.
URP	Universal Render Pipeline – Unity's scriptable render pipeline designed for performance scalability across different hardware platforms.
GPU	Graphics Processing Unit – Specialized hardware designed to accelerate rendering, physics, and complex visual computations.
CPU	Central Processing Unit – The general-purpose processor that manages logic, physics, and computational tasks in a game engine.
BSP	Binary Space Partitioning – A technique used to optimize rendering by organizing 3D environments into hierarchical structures for efficient visibility determination (Widmer, 2022).
RAM	Random Access Memory - a type of computer memory that can be searched in any order and changed as necessary
VRAM	Video Random Access Memory - dedicated memory on a computer's graphics card or graphics processing unit (GPU) that stores and manages data related to graphics and video processing.
AAA	Triple-A - a buzzword used to classify video games produced or distributed by a mid-sized or major publisher, which typically have higher development and marketing budgets than other tiers of games

Table 2.

Glossary of Terms

Term	Explanation
Real-Time Rendering	The process of generating graphics dynamically, as opposed to pre-rendering visuals in advance. Used in video games and real-time film production.

Particle System	A technique used to simulate effects like fire, smoke, and explosions by rendering thousands of individual particles that follow physics-based behavior.
Shader	A program that runs on the GPU to determine how objects appear based on lighting, materials, and post-processing effects.
Instancing	A rendering technique where multiple copies of a model are drawn using a single draw call, reducing CPU overhead and improving performance.
Overdraw	A performance issue where multiple layers of transparent textures are rendered on top of each other, causing excessive GPU load (NVIDIA, 2021).
Render Pipeline	The series of computational processes involved in generating an image on screen, including geometry processing, shading, and post-processing effects.
Pre-Rendered Graphics	Visuals that are computed offline and stored as static images or videos, typically used in CGI-heavy film production rather than real-time applications.
Optimization	The process of improving software or rendering efficiency by reducing computational costs without sacrificing significant visual quality.
Binary Space Partitioning (BSP)	A technique for visibility determination in 3D environments, where complex scenes are divided into a hierarchical structure to improve rendering performance (Widmer, 2022).
Frame Rate (FPS)	The number of frames displayed per second, directly affecting smoothness and responsiveness in real-time applications.
Frustum Culling	The process of removing from rendering processes those objects which lie completely outside the viewing frustum or are otherwise not visible.
Occlusion Culling	A process which prevents a videogame engine from performing rendering calculations for GameObjects that are completely hidden from view (occluded) by other GameObjects.
Bounding Box	An axis-aligned bounding box, or AABB for short, is a box aligned with coordinate axes and fully enclosing some object. Because the box is never rotated with respect to the axes, it can be defined by just its center and extents, or alternatively by minimum and maximum points.
Texture Atlas	A Texture atlas is an image that contains data from several smaller images that have been packed together.
Fill Rate	A broad term referring to the speed at which the GPU can draw fragments which have not been omitted by other processes.
Micro-stutter	A quality defect that manifests as irregular delays between frames rendered by a graphics processing unit (GPU)

Introduction

Real-time VFX are an important part of modern game development, and due to their increasing complexity performance-based challenges need to be addressed in order to ensure smooth user experiences on a wide range of hardware platforms. With more advanced game engines and ambitious visual goals, the demand for higher quality real-time rendering grows with every year, which puts the developers under pressure to balance visual quality with performance.

Optimization has, therefore, become an important part of the development process: it can no longer be a last step but is considered instead as an ongoing design and production issue. Techniques for managing workload, memory usage, and rendering complexity of the GPU have been applied since the early days of 3D games; however, both modern hardware diversity and multiplatform releases make performance tuning a much more complex challenge than it was before. In this respect, Unity 6's Universal Render Pipeline offers a modern rendering environment that also provides flexible ways of creating VFX, but careful management of particle systems, shaders, and other GPU-intensive effects are expected of a developer.

The constantly growing need to apply targeted optimization within these tools creates a highly relevant question for both industry and academic research: **How can developers achieve stable performance without compromising visual quality?** In this sense, the focus of the study is on evaluating three core optimization techniques: overdraw reduction, geometry simplification, and culling optimization. Each is chosen because of its common industry use and broad application to real-time effects. Rather than providing a deep explanation of methods and results here, this introduction sets up the professional and technical relevance of the topic, further developed in the sections that follow.

1 Impact of Optimization on Visual Effects Tools in Unity 6

1.1 Justification of the Choice of Topic

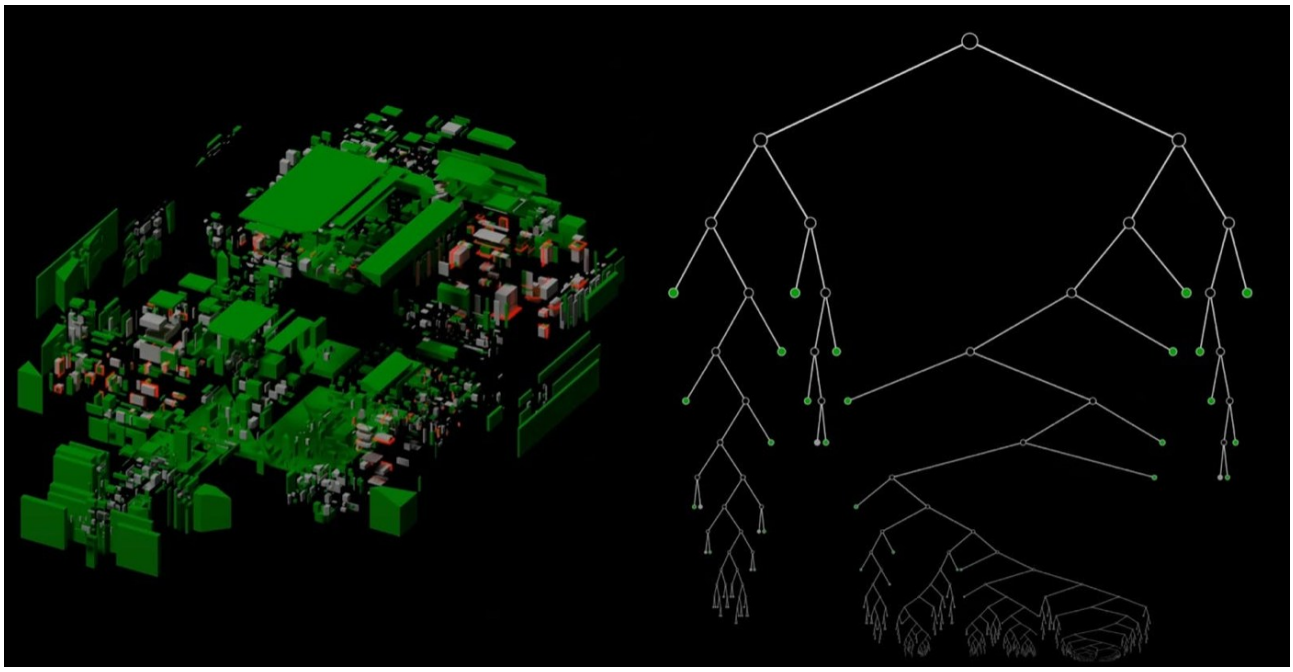
Visual Effects (VFX) have become crucial to modern-day game development techniques and have further added to immersion and storytelling capabilities through particle effects, shaders, and environmental effects. Due to advancements in hardware and software technology, VFX have become increasingly commonplace and now use real-time rendering capabilities to render more complex and interactive visual effects which at one time were considered impossible to do (McCullaugh, 2024). In today's development environment engines such as Unity 6 and Unreal Engine 5 have enhanced their capabilities to create and integrate visual effects into a given project. As a result, modern day games have become richer with particle effects, lighting techniques, and physics simulations to create a truly immersive gaming experience for gamers around the globe.

Nevertheless, as these processes become more frequent and complex among VFX in gaming environments, new difficulties arise, especially regarding optimization techniques. Unlike movies and TVs' use of pre-rendered processes for graphical functions through offline rendering within render farms, interactive and dynamic gaming processes need to generate intricate visualizations for real-time analysis to avoid lag on different machine configurations. Notwithstanding the preceding points on steadily increasing difficulties regarding optimization for game development processes, it is not really a new challenge but rather one of its kind since its inception.

Over the years many game developers have relied on innovative optimization techniques to overcome the limitations of their hardware and use them to their full capabilities. Using Quake (1996), id Software implemented optimizations to facilitate fast 3D rendering using Binary Space Partitioning trees to improve visibility calculations to minimize the number of polygons required to create each image (Lesiv, 2024). Another example is Crash Bandicoot (1996), which implemented fast polygon visualization by insuring high detail around areas where the camera is focused while discarding any detail around areas unseen by the user. In another example, Silent Hill (1999) implemented its signature high-quality fog to reduce draw distances for smooth hardware performance (Schultz, 2024).

Figure 1

A visual representation of spatial partitioning in real-time rendering in games



Note: Adapted by author from Earl, 2023

The examples presented above are illustrations to demonstrate how optimization is always a challenge for game development, but new challenges have appeared in recent years. Those past optimizations were designed within their limitations for hardware at the time, but now modern games need to deal with scalability on a number of platforms that range from high end PCs to mobile devices. The arrival of highly graphical real-time visual FX techniques such as volumetric lighting and fog, physics simulations for particle systems, and fluid simulations makes optimization also no longer just simply reducing polygons but rather have to deal with effective shader management, GPU load balancing techniques, and clever memory allocation strategies (Chornyi, 2024). The modern-day video game is one which is intended to create high quality dynamic graphical visual effects all within real-time solely within the capabilities provided for within any end-user hardware. This shift is indicative of how important it is to also factor optimization for game development at all stages.

Beyond its technical and artistic values, optimization is also related to contemporary sustainable development strategies. Since optimization helps minimize GPU load, memory consumption, and draw calls, it is possible for a developer to directly minimize development machine energy consumption and user device energy consumption, ensuring ecologically sustainable development by placing less demands on electricity. From an economic perspective, optimized features and effects may also extend the lifespan of development hardware through e-waste reduction and cost reduction for upgrades for development studios and users alike.

This is no longer a challenge faced only by the gaming industry. Conventional movies and television program industries have traditionally depended on render farms consisting of hundreds of workstations for executing high-quality visual processing for several hours at a time. Disney's *Toy Story* (1995) featured 117 workstations for rendering its visuals, which required several hours for each pixel (Yao, Pan, & Zhang, 2009). But recently Hollywood's movie and television program manufacturers have started adopting gaming engines such as Unreal Engine and Unity to improve their own workflows (Farris, 2020). These engines, developed for simulating real-time applications for gaming, have already implemented prewritten functions for digital asset management, environmental setup, and real-time visualization. This development resulted in real-time visualizations becoming folded into movie making for techniques such as In-Camera Visual Effect (ICVFX), where dynamic digital environments are projected inside Light Emitting Diode (LED) Volumes and are recorded during actual live-action photography (Livingstone, 2024).

The current use of game engines for modern movie making has resulted in real-time rendering techniques being implemented for instant rendering of complex scenes and interactive environments. This new development not only enhances the efficiency of movie making but also demands optimization to achieve a good balance between performance and image quality, just like game development scenarios. Based on current advancements in efficiency optimizations of image rendering processes for multimedia environments, sustainability is also addressed as fewer energy-demanding rendering tasks are executed because of pipeline optimizations for sustainable environments. Consequently, optimizations have become crucial for both videogame and media industries to deliver high-quality visuals without exceeding hardware constraints or slowing down project development processes.

1.2 Delineation of the Topic

Because of the increasing complexity of real-time visual effects in videogames, developers continue to place emphasis on optimization techniques for improvement. Unlike many videogames developed before now which needed different hardware specific optimizations for efficiency, developers experience different challenges today. It is no longer only about optimizing for one machine but now for many types of hardware configurations ranging from gaming PCs to smartphones (Forbes Technology Council, 2023). The complexity of visual effects also raises requirements for balancing visual fidelity and efficiency for performance.

This thesis examines how real-time VFX efficiency and visual realism have been affected by optimization methods within the Unity 6 Engine and using the Universal Render Pipeline (URP). Although there is already existing knowledge on GPU performance and real-time rendering, very little information is currently being produced regarding its actual application concerning practical efficiency by optimization techniques within Unity's VFX software capabilities, and this is what this thesis hopes to cover. It should also be noted that this gap of information could simply be about proprietary business policies of studios rather than no actual information being provided for such a subject.

For effective insights, this paper will concentrate on:

- Studying basic optimization techniques within the Unity 6 ecosystem.
- Identifying best practices for optimizing real-time VFX in Unity's ecosystem.
- Evaluation of the performance influence of optimization strategies by employing Unity's Profiler and Statistics system.
- Evaluation of optimization impacts on frame rates, GPU utilization, memory consumption, and rendering efficiency.
- Comparing Trade-offs between optimized and non-optimized VFX

For these purposes, this study uses an experimental method of testing basic VFX optimization strategies outlined by official Unity documentation and best practices for developers within the field. This involves techniques such as Level of Detail (LOD) settings, use of instancing techniques, GPU particle systems, reduction of particle complexity, and management of texture data. These strategies will be tested for their effectiveness against performance and quality to gain an understanding of how developers should optimize VFX for real-time rendering while maintaining quality.

The experiment will consist of a comparison between optimized and unoptimized VFX implementations through the use of Unity's Profiler and Statistics features for measurements of frames per second, render time, memory allocation, and GPU utilization.

Other than gaming, requirements for optimization have also risen within the movie and television industries because many studios have started employing game engines for real-time rendering capabilities. Methods for real-time rendering such as In-Camera Visual Effects require efficient optimization techniques to maintain smooth performance during live-action shooting. These requirements also meet the requirements for sustainable development. Efficient optimization reduces any strain on computation, decreasing power consumption and also extending the life of any production hardware. This makes it ecologically sustainable because it reduces environmental footprints associated with rendering processes but is also economically sustainable because it opens up high-quality real-time pro-duction workflows to smaller pro-duction facilities or individuals.

With this approach to structuring, this study is able to offer not only empirical performance information but also development insights. This information is particularly helpful for game developers and real-time production crews seeking to determine which optimization strategies result in the best performance and visual detail quality being achieved, as it points to areas of interest for best practice strategies for real-time VFX optimization within Unity environments.

1.3 Definition of Objectives

The purpose of the thesis is to evaluate the impact of Unity VFX Graph optimizations concerning various aspects of real-time performance and quality. The level of complexity involved in the real-time rendering of Unity game development and media is increasing day by day. This work aims to apply the concept of data analysis to create guidelines that can allow maximum performance usage without the requirement of incorporating quality enhancement and optimization methods devoid of resultant impact. It also seeks to utilize findings for sustainable development cycles by finding means to reduce energy spending while at the same time ensuring accessibility to hardware for varied development environments for sustainable development efforts for environmental conservation and economically sustainable development processes.

For achieving these aims, this study concentrates on the following:

- Evaluation of basic techniques for VFX optimizations as mentioned in Unity documentation, such as techniques like adjusting Level of Detail, Instancing, reduction of shader complexity, and GPU particle systems.
- Calculating the impact of optimization techniques on Frame Rates, memory use, GPU load, and rendering efficiency through use of Unity Profiler and Statistics features.
- Optimized vs. Non-optimized VFX Implementation Comparisons for identifying performance differences related to different approaches.
- Assessing whether specific techniques of optimization result in any visual degradation becoming apparent and thus learning how to implement these techniques during development.
- Benchmarking the use of Unity's native optimization strategies against different strategies for manually optimizing VFX to determine their efficiency for different VFX applications.

This investigation looks to address questions such as:

1. How do different optimization techniques influence the performance of Unity's VFX features such as Particle System, VFX Graph, and Shader Graph?
2. In what way do these optimization techniques influence visual realism for real-time rendering?
3. What are the best optimization techniques for balancing performance and visual quality?
4. Are particular strategies for optimization more relevant to specific situations involving visual effects or to situations involving shaders?
5. How do the native optimization capabilities of Unity compare to manual optimizations for VFX?How do Unity’s built-in optimization tools compare to manual VFX optimizations in terms of performance gains?

1.4 Hypotheses

This study is exploratory in nature but is conducted within a structured hypothesis-driven framework to examine how real-time VFX optimization methodologies are related to performance and graphical fidelity in Unity environments. By definition, exploratory studies are executed to discern patterns, relationships, or possible causative factors rather than to validate any particular hypothesis or theory already developed. Nevertheless, for the purpose of this particular study, major hypotheses have been developed considering best practices and documented information on VFX-related optimization methodologies for game development environments.

The following hypotheses will be tested:

- Built-in optimization strategies should also result in slight performance optimizations for Unity's VFX editors without any major impact on graphical detail.
- Some techniques for optimization will have stronger performance effects than others do for example, instancing and level of detail adjustments.
- A unity optimization toolkit will have less effectiveness as compared to manual optimizations because developers have the option to tweak settings beyond unity settings.
- Optimizing transparent textures to reduce overdraw and optimizing complex visual effect systems will give the most benefit to performance because it is very computationally expensive to have lots of overdraw and to do complex GPU math.

These hypotheses have been derived from current technology knowledge of the author regarding real-time rendering optimization techniques developed by Unity and best practices adopted by developers for OpenGL rendering techniques. It is expected that some performance benefits will come from using built-in optimization techniques offered by Unity Engine itself but may not be very effective as those implemented manually by developers to overcome default settings of Unity Engine.

Some of the most powerful optimization techniques include instancing, which can cut down on draw calls by letting several objects draw at once, and adjusting the Level of Detail to automatically reduce objects to draw them quickly at distances away from the viewer. Other than these optimizations for structure, even more specific performance-related issues arise within the rendering pipeline itself. Overdraw, resultant from the overlap of transparent screens, and shader difficulties bringing additional load to GPUs, are some of the most problematic areas for real-time rendering (Unity Technologies, 2024). Methods for resolving overdraw, like transparent effect optimizations or optimizations for complicated visual systems, are expected to offer the most substantial performance boosts.

Besides performance and display aspects, these hypotheses also implicitly verify another significant principle of sustainability: optimization techniques leading to decreased computation requirements would similarly contribute to decreased energy expenditure during development as well as at user level execution. It helps to maintain sustainability on the eco-front through decreased energy consumption and on the economic front through decreased hardware stress and overall increased lifespan of hardware devices.

By evaluating these hypotheses, this project will create measurable data on how techniques for optimization influence performance and visualization outcomes. The result of this project will thus have direct utility for developers seeking to learn which approach to optimization is most efficient within real-time VFX software implemented in Unity while also considering sustainable development initiatives for real-time software development. This project meets sustainable development objectives while considering effectiveness and practical visualization techniques for real-time software development influenced by optimization techniques.

2 Knowledge Base

2.1 Information Retrieval

A broad approach was taken when gathering relevant literature on real-time VFX optimization in Unity. This data collection process involved the usage of academic databases, game development industry sources, manufacturer white pages, and technical documentation to ensure thorough coverage of the topic.

Table 3

Search Strategy Overview

Category	Sources	Example Websites
Academic Databases	Peer-reviewed papers, conference proceedings, and technical journals	Google Scholar, IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect
Industry Reports	Whitepapers, technical documentation, and industry analyses	Unity Documentation, Unreal Engine Blog, NVIDIA & AMD whitepapers, GDC Vault
Game Development Blogs	Developer insights and technical articles	Unity Developer Blog, 80 Level, Real-Time VFX Forum
Benchmark & Case Studies	Performance comparisons and real-world optimizations	Digital Foundry, Unity Performance Benchmarks, AMD/NVIDIA GPU performance studies

To ensure full coverage, a variety of different keywords and phrases were used across the various sources. The following searches were adapted for use based on the different database-specific requirements:

Table 4

Keywords and Search Terms

Primary Keywords	Secondary Keywords	Related Concepts
Real-time, VFX optimization	Unity, Performance	Shader Graph, Particle System, Shuriken, VFX Graph
GPU Optimization, Real-time rendering	Game Engine, Performance tradeoffs	Render pipelines, efficiency
VFX Best Practices, Game development	Performance impact	Real-time rendering vs pre-rendered

In order to be relevant and reliable, sources were chosen according to the following criteria:

- **Date of publication**: Priority was given to sources no more than 5–10 years old, unless basic texts necessitated the use of earlier sources.
- **Relevance to Research Topic**: Sources must be relevant to the research topic because the research involves real-time rendering optimization in Unity and discussions about the VFX toolkit and the performance trade-offs of game engines.
- **Academic & Industry Credibility**: Preference will be given to peer-reviewed articles, tech white papers, and documents from **Unity**, **Unreal Engine**, **NVIDIA**, and **AMD**.
- **Case Studies & Benchmarks**: Studies involving quantitative performance data, case studies, and/or real-world performance benchmarks were preferred.

Although the review has been extensive, there are certain limitations:

- **Access to Paywalled Research**: Most of the relevant research regarding the GPU optimization and real-time rendering topics was behind paywalls. Pre-prints and open access articles from the publication of this research were preferred as sources when possible. The university library served as a resource when it came to alleviating research paywalls.
- **Limited Academic Focus on Game-Specific VFX Optimization**: Although the performance of the GPU and the efficiency of the rendering phase are already widely investigated in the field of academia, there are limited studies about the Unity-specific VFX optimization.

2.2 Source Material Analysis

The sources used in the study were chosen due to their relevance to the topics of real-time rendering optimization and the history of game development. The study utilizes sources from a combination of academic writings, industry reports, and technical documents to create a balanced knowledge base.

The usage of peer-reviewed research work helps guarantee the scientific accuracy of this research work. The paper presented by Wang & Zhang (2021) lays the groundwork of understanding the concept of GPU-based real-time rendering optimization. Likewise, the paper presented by Yao, Pan, & Zhang in 2009 helps understand the concept of the distributed rendering farm system. This particular work is relevant because it helps understand the concept of rendering. However, the work does relate to a different aspect of rendering in the field of video games.

As real-time rendering optimization remains a constantly developing field of study, relevant industry articles and professional manuals are also invaluable. Unity Technologies' manual (2024) presents direct professional advice on rendering optimization in the available Unity visual effect software to ensure the research conforms to the best industry practices available at the relevant point in time. The articles of Farris (2020) and of Livingstone (2024) also illuminate the wider applications of real-time rendering software in contexts beyond gaming. Although non-peer-reviewed articles are non-academic texts, they are direct professional sources from reputable professional bodies in the field.

To delve deeper into the history and technological development of game optimization, certain editorial articles from reputable gaming publications were used. Lesiv (2024) focuses on the history of the gaming industry. Schultz (2024) provides information about the role of technological constraints in the optimization of game development in the release of certain games such as Silent Hill and Crash Bandicoot. Though non-peer-reviewed articles, reputed writers of the gaming sector have written them.

Contemporary trends in the field of tech were complemented with reliable internet sources. McCullaugh & Chornyi (2024) showcase trends and optimization methods regarding the hardware part of the problem. Lastly, Earl (2023) demonstrates BSP trees and rendering methods through videos. Even though the above sources are secondary materials, they can be used to highlight the best practices of the industry when considering the challenges of contemporary implementation.

By incorporating research articles from peer-reviewed journals, white papers from the industry of interest, technical documents, and professional journalistic writing, this study provides a balanced viewpoint regarding the optimization of real-time rendering. The incorporation of both primary and secondary sources provides the ability to address the subject from various perspectives.

2.3 Base Concepts

Table 5

Key Technical Terms and Concepts

Term	Definition	Source
Real-Time Rendering	A field of computer graphics focused on analyzing and producing images in real time.	Autodesk, n.d.
Frame	A single image from a sequence of images that represent moving graphics.	Unity Technologies, n.d.
Frames Per Second (FPS)	The frequency at which consecutive frames are displayed in a running game.	Unity Technologies, n.d.
Visual Effects (VFX)	Images in a film or entertainment that are created using computers or models, and that can be mixed with live action	Cambridge University Press, n.d.
VFX Graph	A package in Unity which simulates particle behavior on the GPU.	Unity Technologies, n.d.-b
Particle System	A component that simulates fluid entities such as liquids, clouds, and flames by generating and animating large numbers of small 2D images.	Unity Technologies, n.d.
Shader	A program that runs on the GPU	Unity Technologies, n.d.
Shader Graph	A visual shader editing tool in Unity which uses a visual node-based structure.	Unity Technologies, n.d.-b
GPU Instancing	A draw call is an optimization method that renders multiple copies of a mesh with the same material in a single draw call.	Unity Technologies, n.d.-b
Draw Call	Tells the graphics API what to draw and how to draw it. Each draw call contains all the information the graphics API needs to draw on the screen, like information about textures, shaders, and buffers.	Unity Technologies, n.d.-b
Draw Call Batching	A draw call optimization method that combines meshes so that Unity can render them in fewer draw calls.	Unity Technologies, n.d.-b
Level of Detail (LOD)	An optimization that reduces the number of triangles that Unity must render for a GameObject when its distance from the Camera increases.	Unity Technologies, n.d.-a
Overdraw	Overlapping transparent elements, such as UI, particles, and sprites.	Unity Technologies, n.d.-b

Render Pipeline	A series of operations that take the contents of a Scene and display them on a screen.	Unity Technologies, n.d.-a
URP	A prebuilt Scriptable Render Pipeline, made by Unity.	Unity Technologies, n.d.-b
Scene	A Unity asset which contains all or part of a game or application.	Unity Technologies, n.d.-b
Graphics Processing Unit (GPU)	A special type of processor used in computers and other devices that is designed to make them able to deal more quickly with graphics.	Cambridge University Press, n.d.
Central Processing Unit (CPU)	The part of a computer that controls all the other parts.	Cambridge University Press, n.d.
Profiler	A Unity tool which supplies performance information about a game or application within the Unity Editor.	Unity Technologies, n.d.-b
Optimization	The process of refining a video game to run as efficiently as possible on its intended platform(s)	Polydin, 2023
Control Group	The standard to which comparisons are made in an experiment.	Britannica, n.d.

In the case of traditional media like movies and TV shows, the images are produced through a technique known as offline rendering, which involves the processing of each image, also known as a frame, through the power of computers over a long period of time. The system used in this fashion can be known as a render farm and can take hours to render a single frame. However, the effect achieved through this technique is of the highest level of visual quality (Yao, Pan, & Zhang, 2009). Interactive graphic media like video games require real-time rendering of images. The images are rendered at a quick pace of about sixty frames per second or higher through limited hardware components like gaming consoles, smartphones, and basic computers. This requirement makes video games the toughest software applications when it comes to the performance needs of images being rendered at the speed of a few milliseconds without affecting interactivity.

Real-time rendering also necessitates the immediate display of each visual effect (VFX). This necessitates the processing of required tasks involving lighting, textures, and animation within a few milliseconds. This presented a challenge to the developer because even the slightest lag could cause severe problems such as the dropping of frames and overheating. To eliminate this problem, the developer makes use of various optimization techniques. In this case, the meaning of this term

involves the enhancement of the visual processing and display of the game. This helped ensure the final product looked good and ran smoothly. However, this did not happen because of the absence of this concept.

To address the needs involved in real-time rendering, game engines today allow the usage of specialized components like Unity's Universal Render Pipeline (URP). The Universal Render Pipeline is a rendering pipeline, which refers to a series of tasks involved in taking the scene (which holds the totality of the game's contents). The rendering pipeline transforms the scene data to the images displayed on the screen. The Universal Render Pipeline scales properly from mobile phones to high-performance PCs because of its ability to efficiently handle the rendering of lighting, shadowing, and transparent materials. Though the Universal Render Pipeline provides performance enhancement tools, it doesn't necessarily optimize visual effects automatically. This remains the developer's task.

Unity provides two main tools when it comes to the implementation of visual effects: "Particle System" and "VFX Graph." The "Particle System" is the older of the two and utilizes the CPU to create simple visual effects such as smoke, fire, or sparks through the animation of thousands of small images known as "particles" in a two-dimensional environment. The "VFX Graph" utilizes the graphics processing unit of the computing device to create complex visual effects involving thousands of "particles" rather efficiently. Although this assists the developers greatly when implementing visual effects in the Unity environment, the number of particles to be "simulated" and the "renders" of said visual effect can affect the "FPS" of the game.

The most common optimization strategies relevant to this thesis involve reducing the amount of work the computer must perform in each frame, without significantly lowering visual quality. These strategies include:

- **Instancing and Batching**: Every time the game engine tells the computer to draw something, it makes a draw call. Too many draw calls can slow things down. GPU instancing is a method that allows the engine to draw many identical objects like leaves, sparks, or raindrops in a single call. Similarly, draw call batching groups objects together to reduce the number of instructions the computer must send to the GPU (Chornyi, 2024). These techniques are especially useful in scenes with repeated elements or effects.
- **Level of Detail (LOD)**: LOD techniques reduce the complexity of objects that are far from the camera. For example, a detailed explosion effect might use hundreds of particles up close but switch to a simple animated texture when far away. This reduces the amount of data the engine must process while maintaining a convincing appearance at different distances (Unity Technologies, 2024).
- **Overdraw Reduction**: Many visual effects use transparent elements like smoke, fog, or glowing particles. These effects often stack on top of one another, causing what is known as overdraw, where the GPU renders multiple layers in the same screen space. By reducing the number of transparent elements, limiting how often they overlap, and using optimized shaders, developers can reduce GPU strain (McCullaugh, 2024).
- **Shader Optimization**: Shaders are little programs that decide how an object will be displayed when being rendered, reflected in its lighting, color, and texture. The computation of complex shaders can be expensive, especially when combined with lighting and sampling textures. Solutions such as the Unity Shader Graph allow programmers to create the shaders and test them visually to remove unnecessary computations that can be optimized through alternative algorithms that are less expensive (Wang & Zhang, 2021).

In this thesis the effectiveness of the above-mentioned optimization methods will be checked using the Unity's Fantasy Kingdom Demo Project which was created under the Universal Render Pipeline (URP) and can be found at Unity Asset Store. This Demo contains a complex environment rich enough to be used for performance-testing work. The same scene will be split into two groups: the control group which remains unoptimized and the test group which will be optimized.

The impact of each technique will be tested singularly to assess its direct effect on system performance. System performance will mainly be measured through the number of images the system can generate per second, which is known as the number of frames per second (FPS). A method that substantially increases the number of FPS without impacting the image quality will be regarded as being effective. However, those methods which might cause the number of images per

second to reduce along with the quality will be considered ineffective or costly. This will permit the research to also identify the limitations of each technique in the context of real-time image rendering.

By integrating the above mentioned basic principles along with the knowledge of Unity's rendering solutions – Universal Render Pipeline (URP), Particle System, VFX Graph, and Shader Graph – a robust platform has been developed that helps to identify the research issues being raised in this study. All of the above-mentioned solutions add to the graphical complexity of a real-time application but also contribute to the performance issues at the same time. For instance, the Particle System provides functionality related to smoke and sparks through the simulation of thousands of small particles, while the VFX Graph shifts the above-mentioned computation tasks to the GPU to allow improved performance. The role of the Universal Render Pipeline (URP) becomes that of the basic render pipeline allowing the management of the processing of various objects, lighting, and graphical effects being displayed upon the screen. The knowledge regarding the functionality of the above-mentioned rendering solutions and the ways of improving their performance through the implementation of concepts of batching, instancing, Level of Detail (LOD), and shaders becomes integral in this regard.

To understand the trade-offs involved in the above knowledge, there has to be both technical understanding and hands-on experience. In Re-al-time graphics, there has to be efficiency without compromising the visual effect. An unoptimized shader effect can immediately bring down the performance of the device. This can be optimized using Unity's Profiler and Frame Debugger. However, the understanding of the performance information involves understanding the functionality of the engine's draw call, instancing, or transparency effect on the GPU. All this information directly affects the performance of the game regarding whether the game runs smoothly or lags and whether the battery drains quickly or whether the effect will be rendered properly in the mobile device.

Figure 2

Unity Profiler Tool Window by Jeremy Leblanc

As the role of real-time visual effects increases across various sectors, the importance of this research also increases. The same performance considerations which ensure the smooth operation of the game at 60 FPS also apply to the rendering of digital environments in real time using LED volumes, as well as in the context of the development of museum installations through the usage of real-time engines. Issues concerning GPU instancing, Level of Detail (LOD), and the complexity of shaders remain basic and universal, regardless of context. As this research work has been conducted at the practical level of Unity implementation, it benefits not only the game development community but also anyone who focuses directly on real-time rendering. Optimization has ceased to be the specialty of the field, it has become the standard required in performance-critical applications (Farris, 2020; Livingstone, 2024).

2.4 Further Hypotheses

The layout of the thesis will be composed of a short introductory chapter which sets the context of the research work along with the objectives and hypotheses of the research work. Alongside the research objectives mentioned in Section 1.3, this section will list the targeted objectives of the particular research work being done in this case. The following points will direct the research work properly:

- Evaluation of the CPU efficiency of the optimization algorithms applied to the Unity VFX modules (Particle System, VFX Graph, Shader Graph). This will be done using the Unity's built-in Profiler & Frame Debugger.
- Evaluation of the effect of real-time rendering optimizations on the frame rate performance and GPU utilization and memory usage of different hardware setups, from high-end machines to low-end ones.
- Exploring the impact of various rendering methods on visual quality and the extent to which certain optimizations cause visual artifacts.
- The efficiency of the internal Unity optimization tools against manual optimization strategies and when to adopt which technique.

To further develop this research according to the standard of academic literature the hypotheses below have been extended and improved:

- The Unity built-in optimization tools do optimize performance but do not allow the same level of control that can be achieved through manual optimization.
- Particle systemoptimization (such as overdraw reduction and GPU-based particle simulation) has been found to provide asignificant boost in performance over static meshoptimization.
- LOD adjustments and instancing: These help to achieve a middle ground of image quality and performance. However, this also depends upon the rendering conditions.
- Overdraw and the complexity of shaders are the top performance-critical factors in real-time VFX rendering, and optimizing them gives the best results (Unity Technologies, 2024).

3 Research Implementation

3.1 Research Methods

This research adopts an experimental and comparative research method that will enable the examination of the performance of various optimization algorithms used in Unity's real-time VFX rendering (Creswell & Creswell, 2018). The research method involves performance measurement through the application of several optimization algorithms to the Unity Particle System, VFX Graph, and Shader Graph tools. This will be achieved through profiling results offered by the respective tools.

Instead of using the method of theory description and qualitative research assessments, this research lays emphasis on the collection of empirical observations. The best possible performance can be achieved only when the optimized and non-optimized code implementations are used to compare the objectives of performance enhancement against the visual quality of the methods used.

The performance information is collected from profiling utilities that enable the measurement of rendering efficiency. The information mainly consists of the following:

- **Frame Rate (FPS)** – Measures rendering performance and stability across different optimization techniques.
- **GPU Utilization (%)** – Tracks how much graphical processing power is being consumed.
- **Memory Allocation (VRAM & RAM Usage)** – Evaluates whether optimizations reduce memory overhead.
- **Draw Calls & Overdraw** – Determines how efficiently assets are being rendered.

Tools Used:

- **Unity Profiler** – Measures CPU & GPU workload.
- **Frame Debugger** – Analyzes rendering performance and shader efficiency.
- **GPU Frame Timing** – Tracks latency and rendering bottlenecks.

A number of controlled test environments are employed to test different VFX optimization methods. The various tests are also carried out repeatedly to obtain statistically valid results.

Test Cases:

- **Baseline (Non-Optimized Implementation)**: Using default Unity settings without applying any optimization.
- **Optimized Implementation**: Several methods employed: Instancing, Level of Detail, overdraw reduction, and Shader Optimization.

By analyzing the similarity of frame rates of the above two scenarios regarding the usage of system resources and the efficiency of rendering the images, the effect of the rendering algorithms used will be identified.

Once the performance data has been gathered, the statistical technique can be used to evaluate the results of the various optimization methodologies.

- **Measures of Center**: Average FPS, GPU usage, and memory usage through various test sessions.
- **Variability Analysis**: Standard deviation of the frame rate to evaluate performance.
- **Rendering Bottleneck Identification**: Frame Debugger's analysis of shader complexity & overdraw.

This approach allows for objective comparison and ensures that conclusions are based on measurable performance differences rather than subjective judgment. This makes possible the objective comparison and the drawing of conclusions that are based solely upon measurable differences rather than subjective opinions.

The above data supports answering the following core research questions:

Table 6

Core Research Questions and Analysis

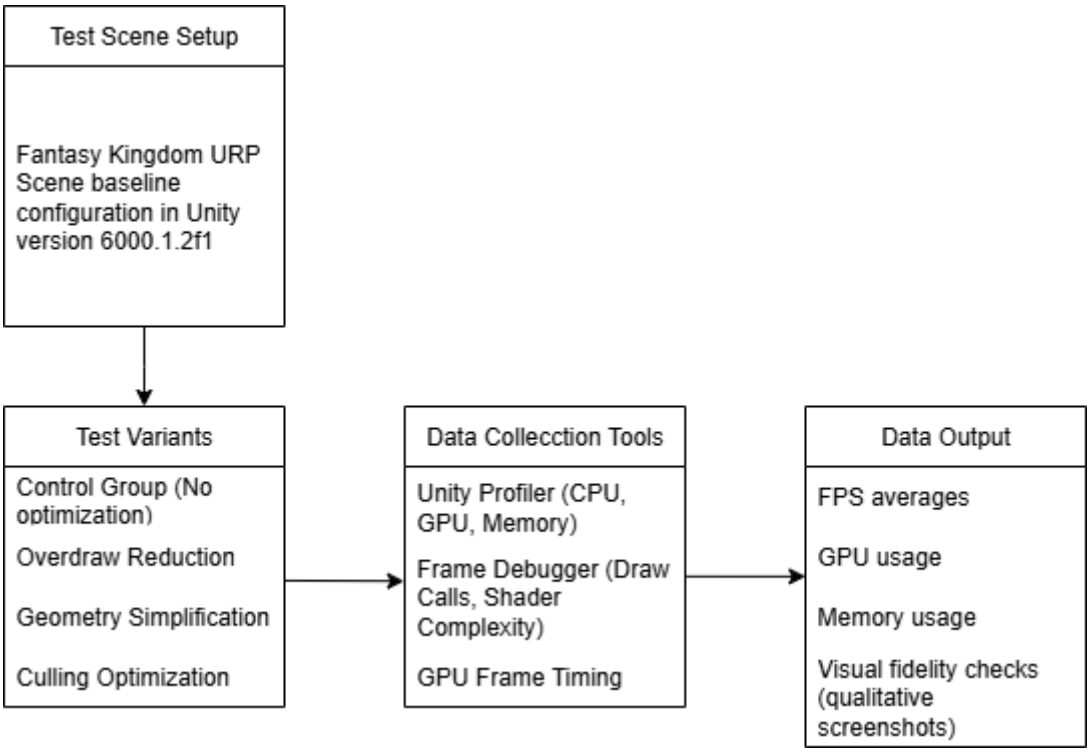
Question Number	Research Question	Analysis
1	How do different optimization techniques affect performance?	FPS and GPU utilization data show whether optimizations reduce computational load.
2	What impact do optimizations have on visual fidelity?	Side-by-side comparisons reveal whether certain techniques introduce noticeable degradation.
3	Which methods provide the best balance between performance and quality?	The comparative analysis highlights which techniques are most effective without major trade-offs.
4	Are certain techniques more effective for specific VFX types?	By testing different effects (particle-heavy vs. shader-based), we determine the best optimization strategies for different use cases.
5	How do Unity's built-in optimization features compare to manual optimizations?	Performance data from automated vs. manually optimized implementations provides a direct comparison.

This methodology was selected because:

- It provides objective, repeatable results. Performance profiling eliminates subjectivity and allows direct comparison of optimization techniques.
- It aligns with real-world industry practices. Game developers rely on data-driven optimization workflows rather than qualitative assessments.
- It identifies the impact of each optimization technique. By running controlled tests, external factors are minimized, making it possible to assess performance gains accurately.
- It ensures practical relevance. The findings will be applicable to game developers, VFX artists, and technical artists working in real-time rendering environments.
- This structured approach ensures that the study produces actionable insights into best practices for optimizing real-time VFX in Unity.

Figure 3

An experimental test workflow for a Unity VFX optimization study (Fantasy Kingdom URP Scene in Unity 6000.1.2f1).



It also becomes crucial to state the limitations of the chosen methodological approach. The chosen optimization methods to be tested: LOD adjustments, overdraw reduction, instancing, and Shader Optimization, were selected because of their ubiquity within the gaming industry and their ability to be utilized within the Unity environment. This has resulted in the exclusion of certain methods, which might also be of significance in this study. In addition to this, the chosen study will be limited to only one type of hardware. This becomes an important point because the results might be different in various environments.

3.2 Development Methods

In this paper, the research method involves experimental testing to measure the effect of various optimization methods used in the Unity VFX suite. The test environment has been constructed using the Fantasy Kingdom Demo project available as the Unity version 6 example scene. This provides a uniform and practical environment to compare the effect of different rendering optimizations.

The study will compare the optimized and non-optimized code to evaluate the benefits of performance and image quality. Three main test setups are defined:

- **Control Group**: In the Fantasy Kingdom scene, the default options are used without the application of any optimization technique.
- **Single Optimization Tests**: The different methods of optimization are used singularly, including Culling algorithms (frustum culling and occlusion culling)
- **Level of Detail adjustments Reducing overdraw**: Methods involving transparent sorting and reducing fill-rates.
- **Combined Optimization Tests**: A combination of various optimizations is used to ascertain the combined impact and possible points of performance bottlenecks.

In order to provide consistent results from the tests, the performance profiling of the work will be carried out using the default Unity6 settings without capping the frames per second rate.

Although thirty simulation runs are suggested in the simulation study of statistical modeling (Law & Kelton, 2007, Ch. 9), this Bachelor level thesis will conduct only ten simulation runs per test. This will provide only a general insight into the methods and their performance results without analyzing them in-depth. Performing many simulation runs helps to minimize the effect of outliers as well as randomness in the results. The performance results recorded are:

- **Frame Rate (FPS)**: This evaluates the performance of rendering images.
- **GPU Utilization (%)**: Indicates usage levels. • Memory Usage (RAM/VRAM): This determines the effectiveness of memory-optimization methods. • Draw Calls & Overdraw: Represents the rendering pipeline’s efficiency.

3.3 Data Procurement Design

The test data comprises Unity VFX in a specific scenario, and the performance profiling has been carried out under three settings:

- **Baseline (Control Group)**: Default scene without any optimizations.
- **Isolated Optimization Tests**: One optimization method applied at a time.
- **Combined Optimization Tests**: Stacking together various optimizations.

A run needs to be done ten times for each case because of best practices when doing simulations (Law & Kelton, 2007).

This data is collected using Unity's profiling tools to guarantee the information's accuracy. The main tools used are:

- **Unity Profiler**: This captures data about the timing of the frame, the CPU and GPU performance, and
- **Frame Debugger**: This provides information about the details of rendering each frame.
- **GPU Frame Timing Tool**: Analyzes the performance of each frame's latency.

In each test run, there is a standard duration of the observation period where data can be collected. The data collected is exported and dealt with through the spreadsheet tools available in Microsoft Word to create summaries. To calculate the effect of the optimization methods used, the following statistical measurements are employed:

- **Mean (Average)**: This determines the average value after several test runs.
- **Standard Deviation**: Indicates the dispersion of performance gains.
- **Comparative Analysis**: It examines the optimized result against the non-optimized result.

Frame time variance means variations in the frame rendering times that can affect the level of performance being achieved even when the average number of frames per second is high (Digital Trends, 2022). Although this is important in real-time rendering the paper will concentrate on the general efficiency of rendering rather than the level of user experience performance. Future research should continue to develop from this point.

3.4 Justification of Implementation(s)

The approach outlined above was chosen based upon its practicality and relevance to best practices as used in the industry.

- Performance profiling has become the norm. Gaming companies employs performance profiling tools and test cases in order to judge the effectiveness of optimization.
- The Unity Profiler and Frame Debugger are native tools. Using Unity's built-in profiling tools helps ensure that results will be relevant to people developing inside the Unity environment.
- The Fantasy Kingdom project allows real-world testing. This demonstration scene involves real-world rendering rather than the simplified scene of a typical test environment.
- Thirty test runs are statistical best practice. For this thesis, ten tests produces a large enough sample size reduces variations in the results (Law & Kelton, 2007).

Previous research regarding real-time rendering optimization tends to target issues related to the performance of the GPU rather than the overall VFX performance in Unity. Though the previous research work had explored the topics of the research project in general, fewer studies applied the comparative experimental method to various rendering optimize techniques in the commercial game engine environment. This research helps to close this research gap.

3.5 Research Ethics Review

The research follows the principles of research integrity, which promotes the collection of data in a clear and unbiased manner (Jamk University of Applied Sciences, n.d.). Even though the research involves neither performance benchmarking nor the usage of the Unity software directly related to humans, the prime concerns regarding research ethics form part of the above considerations: research integrity.

All the sources of information used from the manuals, research essays, and industry reports are properly cited in APA 7th edition format. The research also adheres to the principles of ethics at Jamk and ensures that:

- There are no proprietary software modifications used here that require confidentiality. Only the software tools available in Unity are used.
- The research neither utilizes unverified software for benchmarking that might create discrepancies.
- The data collected and presented remains unaltered to ensure its accuracy and reproducibility.
- The work makes no use of AI-generated materials or plagiarism.

The chosen test environment as the Fantasy Kingdom Demo project is a free Unity asset store project specifically made to be used in educational and research purposes to comply with the standards of fair use and research.

To eliminate bias when gathering and analyzing information, the following steps can be followed:

- **Identical testing conditions**: All the scenarios are exercised under the same environment of hardware and software configurations to eliminate the effect of any external factors.
- **Repeated testing (10 iterations per scenario)**: This helps to eliminate random variations which might bring about erroneous results.
- **There are no prior expectations that affect the outcome**: The effectiveness of the optimizing algorithm will be strictly measured according to numeric results rather than personal inference.
- **Objective comparison of various methods of optimization**: The paper will compare the effectiveness of various methods without considering which method would be the best approach at the outset.
- **No manual input during the test run**: During the test run itself, the performance data will be recorded without the manual input of the researchers.

As there are no human participants, personal information, or AI results involved in this research, the usual concerns of informed consent and the right to privacy do not arise. The following are the ways the research maintains scientific validity:

- **Transparency in Reporting**: Every methodology has been described in depth so that in theory, the same researcher could reproduce the results.
- **No Selective Reporting**: All the data points available, even when an optimization method performs sub optimally compared to expectation, are used in the results.
- **Refraining from the misrepresentation of results**: No data manipulation or "cherry-picking" of results occurs.

Final Ethical Compliance

This research follows the best practices of academic studies concerning research ethics:

- Ethical guidelines of JAMK University of Applied Sciences
- Standard research practices for reproducible benchmarking and performance testing
- Applying APA 7 citation and reference requirements to avoid plagiarism. In this way, the study’s transparency, objectivity, and integrity guarantee that the results achieved are credible, unbiased, and valuable to game developers and VFX artists utilizing rendering capabilities provided by Unity.

4 Results

4.1 Abstraction of Results

This section will give a brief overview of the results of the tests described above concerning descriptive statistics of the average results of the ten runs of each test scenario. Tables are employed in this work to give the performance characteristics observed in the Control Group and the three optimization scenarios.

There were four test groups used in performance testing:

- Control Group (non-optimized baseline),
- Overdraw Reduction (Reducing transparency overlap)
- Geometry Reduction (Replacing octagon with a quad)
- Culling Optimizations: (frustum culling, occlusion mapping, bounding boxes of VFX systems.)

The performance results from the ten runs within each group are collated together in the following table, highlighting the arithmetic mean of the selected performance characteristics:

Table 7

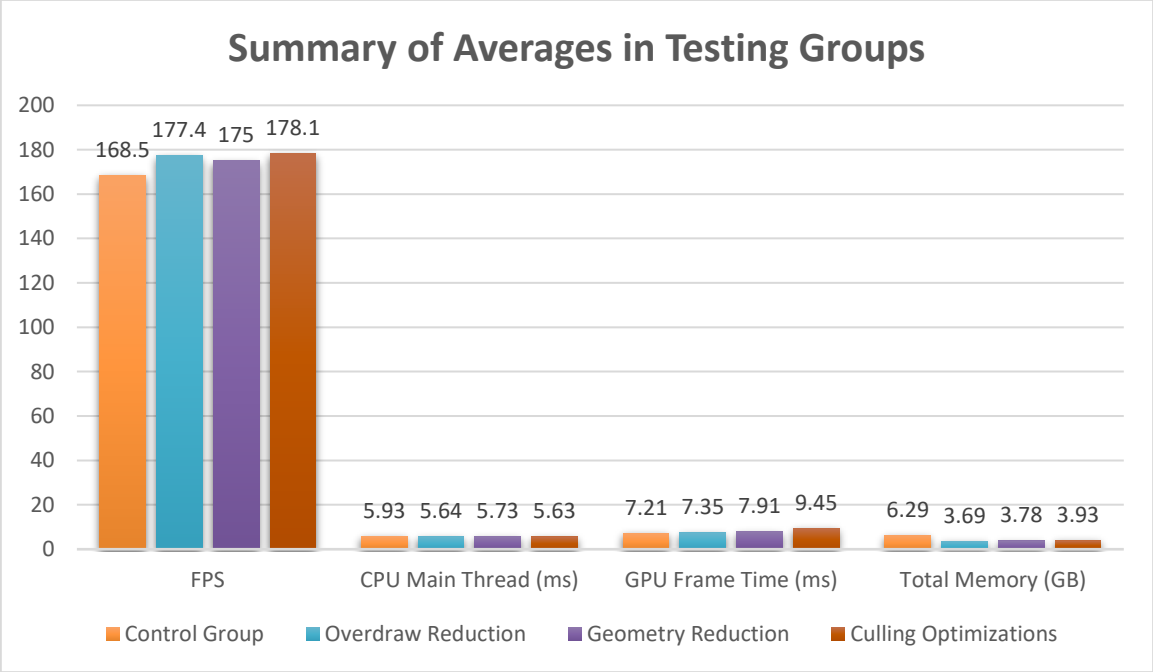
Summary of Averages in Testing Groups

Test Group	FPS	CPU Main Thread (ms)	GPU Frame Time (ms)	Total Memory (GB)
Control Group	168.5	5.93	7.21	6.29
Overdraw Reduction	177.4	5.64	7.35	3.69
Geometry Reduction	175.0	5.73	7.91	3.78
Culling Optimizations	178.1	5.63	9.45	3.93

The differences in performance between optimization scenarios and the control group are summarized visually in Figure 4, showing the average results across FPS, CPU, GPU, and memory metrics.

Figure 4

Clustered column chart of Testing Group Results (Averaged)



The control group, which served as the non-optimized baseline, was able to support an average of 168.5 FPS at a CPU main thread time of 5.93 ms and a GPU frame time of 7.21 ms. The total memory used was 6.29 GB. This setup served as the reference point against which the impact of each of the various optimization techniques was assessed.

The second group, also known as Overdraw Reduction, dealt with the reduction of the number of active particles and the modification of their behavior to optimize the reduction of overdraw. This reduction resulted in performance gains due to the average FPS rising to 177.4, which was about a 5% improvement from the control group. CPU main thread time reduced slightly to 5.64 ms. More crucially, however, GPU frame time actually stabilized at 7.35 ms, which suggests a reduction of

the GPU workload due to the benefits of reduced overlapping particles. Memory usage improved the most, reducing to 3.69 GB (41% less than the control group).

The third group optimized geometry involved simplifying the mesh of the particle system's geometry to a quad mesh. This resulted in an average of 175.0 frames per second. Although this value was better than the control group's results, the improvement was not as dramatic as in the case of overdraw reduction. CPU main thread time was approximately 5.73 ms, but the GPU frame time increased slightly to about 7.91 ms. The memory usage dropped to 3.78 GB. The above findings indicate that geometry simplification has the capability of delivering moderate performance gains but not as dramatic results as the reduction of the number of particles.

The addition of culling methods, the enabling of frustum culling in the VFX output node, the usage of a baked occlusion culling map, and the constriction of the bounding box of the particle system resulted in the highest average FPS of 178.1, which was the best performance across all groups. The CPU main thread time was the shortest recorded at 5.63 ms. The GPU frame time lengthened to 9.45 ms, which was the longest recorded in all groups. This indicates that although the number of rendered particles reduced, the processing costs of the culling computation (CPU and GPU combined) resulted in additional processing time. The memory usage reduced from the control group to an average of 3.93 GB.

The findings indicate that each of the tested optimization methods resulted in noticeable performance enhancements against the non-optimized control group. The method of overdraw reduction was the most effective technique in minimizing GPU loading as well as memory consumption. The method of geometry reduction resulted in moderate performance enhancements. The methods of culling resulted in the maximum enhancement in FPS performance but entailed additional GPU processing time. The findings emphasize the significance of picking the right methods of performance enhancement suited to the performance bottlenecks of the project.

4.2 Collected Data

For this study, the quantitative performance data was collected through Unity 6 and the Universal Render Pipeline rendering engine. The performance information was collected from four different scenarios:

- Control Group (non-optimized baseline),
- Overdraw Reduction,
- Geometry Reduction (replacing the octagon mesh with the quad mesh).
- Culling Optimizations (occlusion culling, frustum culling, bounding box adjustment).

In each test case, the experiments consisted of ten independent runs to account for the natural variance of the runtime.

The measurements described below were taken at each run:

- **Frames Per Second (FPS)**: This value directly reflects the number of rendered frames per second and hence the performance of the runtime environment.
- **CPU Main Thread Time (ms)**: This value reflects the average CPU main thread processing time of a frame and hence the CPU load involved in the rendering of the scene.
- **GPU Frame Time (ms)**: This shows the average time the GPU takes to render each frame of the graphics. This value will depend on the level of graphics work being handled when the frames are being rendered.
- **Total Memory Usage (GB)**: This shows the memory usage when the environment runs. This will provide information about the memory usage of each case.
- **Batches**: The number of draw calls being submitted at the same time to the GPU per frame. This reflects the efficiency of submitting the renders.
- **SetPass Calls**: The number of state changes during rendering, which reflects the state change overhead of rendering.
- **Triangles**: The number of triangles being processed per frame. This number provides an indicator of the level of geometric complexity being handled.
- **Vertices**: The number of vertices being handled per frame.

  The above metrics were selected because they play a significant part in the performance and usage of real-time graphic applications. They are also standard benchmarks used to measure the effectiveness of the optimization techniques chosen.

The data was captured by the Statistics Window in Unity, in combination with the Unity Profiler. To provide for easy comparison of the data, the values for each of the parameters measured were systematized in tables. Calculated means for each test group also aided in summarizing the data.

Thus, the approach used ensured that the data obtained was not limited to instantaneous performance snapshots but in fact was broad enough for the determination of the relative effect of the various optimization methods being used.

4.3 Data Analysis

The analysis performed in the current research used descriptive statistical analysis methods, which suit the nature of the research questions being asked. This research aims to assess and compare the performance capabilities of different VFX optimization methods on the Unity platform.

Each test condition was evaluated through ten runs. To make the runs representative of the whole process despite the time taken by the runs, the approach used to analyze the results was the calculation of the average in the context of each metric. This means for each metric (the Frames Per Second (FPS), the CPU Main Thread Time (ms), the GPU Frame Time (ms), the Total Memory Usage (GB), the Batches, the Set Pass Calls, the Triangles, and the Vertices), the values from the runs are added together and divided by ten.

Additionally, the method used for the data analysis of the test groups allowed for the determination of representative measures of central tendencies for the groups. By comparing the average values for the different groups in the different scenarios, the analysis was able to provide direct comparisons. Additionally, the analysis was not geared towards the determination of statistical significance for any observations made. By the nature of the analysis, no statistical hypothesis was involved. Thus, no statistical procedure like the t-test was applicable in the analysis.

It must be recognized, however, that the approach taken does not adjust for the variance within the runs that might occur beyond the mean in either distribution, and no measures of dispersion were used. It was also noted that the approach was largely within the scope of the work in terms of comparing performance rather than strictly hypothesis-testing statistical analysis. On the whole, the approach used in the analysis of the data used in the work ensured that the results are clear and relevant to the specific comparison being made in the context of the work.

4.4 Suitability, Reliability, and Validity of the Data

The performance data gathered in the study was relevant for tackling the research objective of analyzing the effect of various optimization methods on the per-frame performance of the VFX solutions in the Unity platform (Saunders, Lewis, & Thornhill, 2019). Variables like Frames Per Second (FPS), CPU Main Thread Time, GPU Frame Time, Total Memory Usage, Batches, Set Pass Calls, Triangles, and VERT were chosen because they are all industry-standard performance and computational workload measures. Thus, the chosen parameters tended to cover the entirety of system resource utilization and performance aspects.

Reliability speaks to the ability of the data gathered to be consistent from one observation to the next. Each test group was performed for a period of ten runs in identical scenarios on the same hardware platform. Thus, the results clearly indicated less variance per test group. Nevertheless, one must note that reliability may depend on the hardware environment. Factors like the frames per second rate and the frame rate are very much dependent on the hardware being used. Additionally, the reliability achieved through ten runs per test group could have been improved by carrying out additional runs.

Additionally, the reliability of the results might also depend on the composition of the VFX asset being tested. Strategies for optimization that succeeded in the scene and the particle system in the research might produce different results in more complex scenes.

The fact that the data gathered is valid is attested by the fact that the measures directly correspond to the performance aspects being investigated. FPS, CPU time, and GPU time all directly represent the speed of execution of the process during the metric gathering process. Memory usage also directly represents the resource utilization for each of the optimization scenarios being investigated.

Concerning the fidelity aspect, the approach used in the study was qualitative. To facilitate the discussion of this aspect, the screenshots taken during each individual test run are presented in Appendix 2. This serves to provide the reader with an opportunity to determine the perceived fidelity effect of each optimization strategy. Even though the user test was not carried out in order to determine the perceived fidelity quantitatively, the demonstration of the screenshots in the appendix enhances the reliability of the observations.

5 Discussion

5.1 Argumentation

The results achieved from the given research clearly indicate some identifiable tendencies in the effect of the application of the optimization methods on the performance in the VFX capabilities in the Unity platform. These results can therefore be explained in the context of the current levels of understanding pertaining to the GPU performance capabilities.

That the continuous improvement in performance achieved through the usage of optimization methods, especially those addressing overdraw and complexity of geometry in the context of transparent rendering and particle system overdraw, supports theoretical expectations that the aforementioned factors of excessive transparent rendering and the overdraw of particles in the system are major bottleneck areas in real-time GPU computing (Unity Technologies, 2024).

The overdraw reduction test group, for instance, showed some of the largest performance improvements in terms of higher frame rates and lower GPU frame times than the control test group. This lends credence to the hypothesis that the methods used for the reduction of overdraw will provide the largest performance improvement because the reduction of overlapping transparent geometry directly reduces the workload of the pixel shader and the GPU fill rate requirement. This agrees with current rendering theory that puts the reduction of overdraw first in the order of precedence for scenes dominated by transparent geometry.

However, the geometry reduction (Level of Detail) team showed some, but definite, improvement in performance. This is consistent with the theory but suggests that the improvement is perhaps scene dependent. Simplification of vertex complexity will decrease the amount of work done in the vertex shader and possibly lower the memory bandwidth usage. However, in many scenes featuring high levels of transparency and fill rate issues, the effect is usually dominated by fragment shading in the current environment.

Optimizations for occlusion culling resulted in small improvements over overdraw reduction. This indicates that while the main goal of occlusion culling is to remove views of fully occluded geometry, the VFX system being tested is comprised of semi-transparent particles facing the camera. This

is also true from the theoretical aspect because occlusion culling is not very effective for those VFX that are visually present in the scene in front of the camera.

Altogether, the experimental results demonstrate the insight that while the in-engine tools developed by Unity do contribute seriously towards providing some sort of performance boost if deployed effectively in the correct context, the applicability of each individual method is highly context-dependent on the specific scene elements being processed during the various rendering tasks. These observations tend to follow the same patterns witnessed in previous simulation work concerning the topic at hand (Unity Technologies, 2024).

To sum up, the above experimental results not only validate the various hypotheses presented in the previous section but also prove the significance of giving high priority to the optimization methods for pixel-shader-intensive scenarios (such as overdraw optimization) over vertex/geometry-intensive optimization. Moreover, the results also reaffirm the significance of context awareness in terms of geometry complexity levels & occlusion methods in scenes that exhibit heavy levels of trans parency.

Additionally, from the perspective of work life, the applicability of the results is immediate. Applying the methods of optimization presented within the context of the work is directly applicable in the context of video games. This could significantly lower the production costs associated with upgrading hardware used in the production process. Such applicability would prove extremely fruitful in the context of indie developers, VR/AR developers, and mobile developers.

In addition, the performance gains achieved through optimization also align with sustainable development principles. Lower GPU utilization and memory demand translate to reduced power consumption during both development and end-user runtime. Over time, this reduces energy costs and environmental impact, while extending hardware lifespan and reducing e-waste. In this way, effective VFX optimization not only benefits project performance but also supports ecological and economic sustainability within the digital content creation industry.

The above is based on changes made to one semi-transparent, camera-facing visual effect while the rest of the scene remained unchanged, except for occlusion-culling settings in the culling test. In production, these techniques are rarely used in isolation. Teams typically reduce overdraw, simplify geometry where it does not harm the look, and apply culling where occluders and bounds make it effective. When rolled out across many effects, bottlenecks can shift, so the order and strength of impact seen here should guide, not dictate, project decisions, with profiling used to confirm per scene and per effect.

5.2 Incidental or Unexpected Findings

What was unexpected in the control group was the first two runs' average frame rates being significantly lower than the values in runs 3 through 10. Since the control group represented the baseline scenario expected to perform optimally without the application of any optimizations, the values for the first two runs being lower than the values for runs 3 through 10 might point towards some interference in the profiling process. Such interference might have ranged from the behavior of the editor initializations to some background processes influencing the profiling period. After the initial two runs, the control group performed optimally.

There were no such anomalies in the groups for the optimization test (overdraw reduction, geometry reduction, and occlusion culling), where all the runs resulted in consistent outcomes. This indicates that the issue was not related to the data being processed but something outside the parameters of the test. Other than the one definite outlier, no dramatic contradictions of expectations in terms of hypothesis testing were noted. However, some subtleties existed.

Although the hypothesis was that the reduction of overdrawing would produce the largest improvement in performance, the occlusion culling group also showed equally good frame rates, contrary to expectations that it would rely on scene complexity parameters. However, the fact that the scene had a center object that was not occluded (the tree and fountain) does not seem to affect the success of the occlusion culling algorithm. Geometry (LOD) reduction was moderate in terms of improvement but did not go beyond expectations.

Briefly summarized, the one definite outlier occurred in the control groups in the very beginning. Otherwise, the results conformed to expectations but contained some small surprises concerning the strength of occlusion culling compared to other methods of optimization in this specific scenario.

5.3 Reliability of the Research

This work was performed in controlled conditions for one hardware setup and one scene in the Unity environment. This means that the degree of results generalization is restricted. Optimizations might work very differently on different platforms, different hardware setups, or different scenes. This is especially true for mobile scenarios or lower-end computers because the GPU capabilities used might vary.

Additionally, the research chose not to explore the combination of various methods in detail but rather sought to individually isolate the methods for the purposes of analysis. There are lingering questions regarding how various methods might play together in combination. Additionally, the research was limited in the scope of the various test runs per condition. While adequate for the determination of the means for each condition, having access to more test runs would provide for statistical significance in the results set.

Finally, while the process of carrying out the subjective evaluation of each test pass in terms of image fidelity was undertaken for each test pass for each test scene, the fact that the evaluations are subjective means that the point beyond which the effort being made in the pursuit of image fidelity becomes counterproductive is unclear. Also, the profiling performed in this work was on the build environment running on the platform being targeted by the build. Such profiling might produce different insights into the process by which image fidelity is lost.

5.4 Ethical Implications

This work's outcome verifies the importance of performance profiling & optimization in the context of real-time graphics processing, especially concerning the VFX aspects. By identifying the performance gain introduced by each of the methods for VFX graph optimization tested in the work, its outcome verifies the importance of considering performance aspects in the VFX production pipeline from the initial stages of its development. This also encourages developers to take preventive measures against the performance bottleneck problem in the VFX production pipeline from the very initial stages.

The results of the aforementioned imply the broader relevance of the research outcome for scenarios beyond the respective Unity scene used in the research. Hence, in scenarios concerning hardware-limited platforms like mobile phones or virtual reality headsets, the importance of the overdraw reduction aspect of VFX optimization cannot be overstated. Also, the applicability of geometry simplification and culling methods might seem to possess small relevance in comparison. However, the same methods also represent important aspects in the VFX world if implemented in the correct context. This clearly indicates the importance of the realm of performance profiling within the VFX community.

Looking ahead, the lessons learned from the work contained within this presentation relate to the importance of striking the perfect blend between artistic expression and the technologies used in the production process to ensure that optimization becomes an integrated function.

6 Conclusion

This research aims to explore the effect of the implementation of the three different optimization methods on the VFX components in the Unity platform: overdraw reduction, geometry simplification, and culling optimization. Initially, the intention was to explore how the implementation of the aforementioned methods would affect the performance aspects related to the frame rate of the system.

The outcome of the test was expected to confirm the achievement of the aforementioned goal. Each of the optimization methods was able to provide notable improvements over the control configuration. Overdraw reduction was the method that consistently achieved the most notable improvement in memory usage and framerate. Geometry simplification was the method that was able to provide moderate improvements, especially for high particle count effects. Culling optimization was the method that was able to provide the largest improvements in framerate but resulted in slight increases in GPU frame time.

These results clearly demonstrate the importance of pixel overdraw, geometric complexity, and redundant rendering in the process of optimization. This work clearly supports the importance of performance enhancement in real-time rendering. The results are applicable in the gaming industry for the development of better graphics. It will also help with the production of VR/AR content. Thus, the impact of the results is directly related to the industry. Implementation of the given process will help in decreasing the hardware expenses along with the enhancement of stability.

In the wider context, the scope also goes beyond the issue of performance to cover sustainable development. This is because the methods work to minimize the GPU workload and memory usage. Consequently, energy usage within the production process and by the end user ends up being reduced. By helping in the prolongation of the life of the hardware components by reducing the unnecessary workload on the system, the methods contribute towards ecological sustainability. Additionally, the methods also work towards economic sustainability by decreasing the frequency of the upgrade cycle.

Since all of these results are based on the testing of one graphical effect in one fixed scene, it is important to note that the lessons learned from them are applicable at the effect level rather than

the project level. In the normal world of project production, the techniques of overdraw trimming, simplification of meshes, and effect culling are used in conjunction with other project-level techniques such as instancing on the GPU, batching, texture atlases, and material simplification. By their very nature in the normal world of project production, all of these techniques work to affect the resource bottlenecks.

Practically speaking, one would begin by identifying where the bottlenecks in their project are through the use of Profiler or an equivalent analysis tool, followed by identifying the issue causing the bottlenecks in the project and the time and condition in which they occur.

6.1 Future

Even though the presented research fulfilled its tasks, some limitations have been pointed out. Firstly, the performance evaluation was done on one hardware configuration for the specific scene. Future research might therefore attempt to increase the scope of the study by evaluating various hardware configurations. Secondly, the research might also involve mobile platforms.

Additionally, other areas that could benefit from further research involve the exploration of other optimization methods like GPU instancing, draw call batching, creating texture atlases, and graph simplification. How different optimization methods work in combination could also provide insight into the best methods for different-sized projects.

Finally, though qualitative evaluations of image fidelity took place within the context of the current work, future research might profit from more thorough perceptual evaluations in order to better quantify the impact of image fidelity strategies on perceived image quality. It could also help in better comprehending the interplay between image fidelity and performance on one hand, and the impact of performance on the gaming experience on the other.

Also in the future, research might build on the theme of sustainability introduced in the current research by quantifying the energy savings of optimized real-time pipelines. Examining the ecological and economical impact of different optimization methods in production scenarios, for instance in the form of reduced energy consumption for render farms or lower average power in shipped solutions, would not only build support for the use of optimization methods from the perspective of best practices in the industry but also from the perspective of sustainable development.

Since the work done in this research involved one effect in controlled testing, the natural follow-on would seem to involve the validation of stacked methods for various effects and scenes in conjunction with project-level methodologies such as instancing, batching, atlases, and simplification. However, reality suggests that the process of optimization must by its very nature span the life of the project rather than being treated on an effect-by-effect basis.

References

Autodesk. (n.d.). *What is real-time rendering?* Autodesk. May 15, 2025, https://www.autodesk.com/solutions/real-time-rendering

Britannica. (n.d.). Control group. In Encyclopedia Britannica. May 15, 2025, https://www.britannica.com/science/control-group

Cambridge University Press. (n.d.). *Visual effect.* In Cambridge Dictionary. May 15, 2025, https://dictionary.cambridge.org/dictionary/english/visual-effect

Chornyi, A. (2024, July). *Optimization techniques in game development*. Codefinity. https://codefinity.com/blog/Optimization-Techniques-in-Game-Development

Creswell, J. W., & Creswell, J. D. (2018). *Research design: Qualitative, quantitative, and mixed methods approaches* (5th ed.). SAGE Publications. https://spada.uns.ac.id/pluginfile.php/510378/mod_resource/content/1/creswell.pdf

Digital Trends. (2022). *What is frame time, and why is it so important in games?* https://www.digitaltrends.com/computing/what-is-frame-time/

Earl, M. (2023, March 15). *BSP Trees: The Magic Behind Collision Detection in Quake* [Video]. YouTube. https://www.youtube.com/watch?v=wLHXn8IlAiA

Farris, J. (2020, February 20). *Forging new paths for filmmakers on "The Mandalorian"*. Unreal Engine. https://www.unrealengine.com/fr/blog/forging-new-paths-for-filmmakers-on-the-mandalorian

Forbes Technology Council. (2023, September 12). *Advantages and challenges of building "everything apps" and all-in-one service suites*. Forbes. https://www.forbes.com/councils/forbestechcouncil/2023/09/12/advantages-and-challenges-of-building-everything-apps-and-all-in-one-service-suites/

JAMK University of Applied Sciences. (n.d.). Ethical principles. JAMK University of Applied Sciences. https://jamkstudent.sharepoint.com/sites/Eettiset-periaatteet-Elmo/SitePages/en/ethical-principles.aspx

Law, A. M., & Kelton, W. D. (2007). *Simulation modeling and analysis (5th ed.)*. McGraw-Hill Education. https://industri.fatek.unpatti.ac.id/wp-content/uploads/2019/03/108-Simulation-Modeling-and-Analysis-Averill-M.-Law-Edisi-5-2014.pdf

Lesiv, A.-S. (2024, November 6). *A short history of video games*. Every. March 4, 2025, https://every.to/p/a-short-history-of-video-games

Livingstone, T. (2024). *Game engines: Optimising VFX, reshaping visual media*. NECSUS European Journal of Media Studies. https://necsus-ejms.org/game-engines-optimising-vfx-reshaping-visual-media/

McCullaugh, J. (2024, January 9). *Exploring hardware and software trends in 2024*. VFX Voice. https://www.vfxvoice.com/exploring-hardware-and-software-trends-in-2024/

Polydin. (2023). Video game optimization: What is it and why is it important? May 15, 2025, https://polydin.com/video-game-optimization/

Saunders, M., Lewis, P., & Thornhill, A. (2019). Research methods for business students (8th ed.). Pearson Education Limited. https://www.pearson.com/en-us/subject-catalog/p/research-methods-for-business-students/P200000006534

Schultz, J. R. (2024, August 13). *How tech limitations actually made Silent Hill and Crash Bandicoot better games*. Polygon. March 4, 2025, https://www.polygon.com/playstation/24196061/silent-hill-crash-bandicoot-tech-limitations

Unity Technologies. (n.d.-b). *Unity user manual.* Unity. May 15, 2025, https://docs.unity3d.com/Manual/index.html

Unity Technologies. (n.d.-a). *Unity user manual: Glossary*. Unity. May 15, 2025, https://docs.unity3d.com/Manual/Glossary.html

Unity Technologies. (2024). *The definitive guide to creating advanced visual effects in Unity (Unity 6 edition)*. Unity Technologies. https://unity.com/resources/creating-advanced-vfx-unity6

Wang, S., & Zhang, J. (2021). Research and implementation of real-time render optimization algorithm based on GPU. *Journal of Physics: Conference Series, 2136*(1), 012059. https://doi.org/10.1088/1742-6596/2136/1/012059

Yao, J., Pan, Z., & Zhang, H. (2009). A Distributed Render Farm System for Animation Production. *In Lecture Notes in Computer Science: Vol. 5709. Entertainment Computing* – ICEC 2009 (pp. 264–269). Springer. https://doi.org/10.1007/978-3-642-04052-8_31

Appendices

Appendix 1. Test Results

Test Scenario 1: Control Group

Hardware Specs:

- CPU: AMD Ryzen 5 7600X 6-Core Processor

- GPU: NVIDIA GeForce RTX 4070 12gb

- RAM: Corsair 64GB (4 x 16GB) Vengeance RGB, DDR5 6400MHz, CL36, 1.35V

- Unity Version: Unity 6.1 (6000.1.2f1) Universal Render Pipeline

Table 8

Control Group Test Result Numbers

Run #	FPS	CPU Main Thread (ms)	GPU Frame Time (ms)	Batches	SetPass Calls	Triangles (million)	Vertices (million)	Total Memory (GB)
1	74.6	1.33	8.23	948	106	49.7	145.7	6.62
2	109.4	9.1	8.08	948	106	49.69	145.7	6.62
3	162.7	6.1	8.56	948	106	49.7	145.7	6.68
4	170.7	5.9	10.31	948	106	49.69	145.7	6.64
5	165.8	6.0	10.73	948	106	49.69	145.7	6.62
6	169.6	5.9	6.81	948	106	49.7	145.7	6.59
7	160.0	6.2	6.87	948	106	49.69	145.7	6.65
8	162.9	6.1	6.87	948	106	49.69	145.7	6.62
9	160.5	6.1	8.60	948	106	49.69	145.7	6.59
10	160.4	6.1	8.60	948	106	49.68	145.7	6.65
Average	**150.2**	**7.05**	**8.65**	**948**	**106**	**49.7**	**145.7**	**6.63**

Note: The "Average (Mean)" row represents the arithmetic mean of each performance metric across the ten test runs. Geometry and draw call metrics (Batches, SetPass Calls, Triangles, Vertices) remained constant across runs and therefore match the per-run values exactly.

Test Scenario 2: Reduced Overdraw

Hardware Specs:

- CPU: AMD Ryzen 5 7600X 6-Core Processor

- GPU: NVIDIA GeForce RTX 4070 12gb

- RAM: Corsair 64GB (4 x 16GB) Vengeance RGB, DDR5 6400MHz, CL36, 1.35V

- Unity Version: Unity 6.1 (6000.1.2f1) Universal Render Pipeline

Table 9

Reduced Overdraw Test Result Numbers

Run #	FPS	CPU Main Thread (ms)	GPU Frame Time (ms)	Batches	SetPass Calls	Triangles (million)	Vertices (million)	Total Memory (GB)
1	180.0	5.6	6.85	946	104	2.0	2.8	3.89
2	180.0	5.5	6.1	946	104	2.0	2.8	3.64
3	181.4	5.5	6.50	946	104	2.0	2.8	3.61
4	171.3	5.8	7.96	946	104	2.0	2.8	3.86
5	175.7	5.7	8.33	946	104	2.0	2.8	3.62
6	179.8	5.6	9.07	946	104	2.0	2.8	3.67
7	184.5	5.4	7.00	946	104	2.0	2.8	3.70
8	172.0	5.8	8.07	946	104	2.0	2.8	3.62
9	173.1	5.8	6.81	946	104	2.0	2.8	3.67
10	175.0	5.7	7.84	946	104	2.0	2.8	3.63
Average	**177.4**	**5.64**	**7.35**	**946**	**104**	**2.0**	**2.8**	**3.69**

Note: The "Average (Mean)" row represents the arithmetic mean of each performance metric across the ten test runs. Geometry and draw call metrics (Batches, SetPass Calls, Triangles, Vertices) remained constant across runs and therefore match the per-run values exactly.

Test Scenario 3: Geometry Reduction

Hardware Specs:

- CPU: AMD Ryzen 5 7600X 6-Core Processor

- GPU: NVIDIA GeForce RTX 4070 12gb

- RAM: Corsair 64GB (4 x 16GB) Vengeance RGB, DDR5 6400MHz, CL36, 1.35V

- Unity Version: Unity 6.1 (6000.1.2f1) Universal Render Pipeline

Table 10

Lower Quality Textures Test Result Numbers

Run #	FPS	CPU Main Thread (ms)	GPU Frame Time (ms)	Batches	SetPass Calls	Triangles (million)	Vertices (million)	Total Memory (GB)
1	178.6	5.6	7.23	946	104	2.0	2.8	3.80
2	172.3	5.8	7.86	946	104	2.0	2.8	3.76
3	176.4	5.7	8.25	946	104	2.0	2.8	3.74
4	175.0	5.7	8.09	946	104	2.0	2.8	3.80
5	161.0	6.2	8.25	946	104	2.0	2.8	3.74
6	177.4	5.6	8.20	946	104	2.0	2.8	3.74
7	170.6	5.9	7.48	946	104	2.0	2.8	3.81
8	176.4	5.7	7.84	946	104	2.0	2.8	3.78
9	188.8	5.3	8.39	946	104	2.0	2.8	3.75
10	173.8	5.8	8.48	946	104	2.0	2.8	3.82
Average	**175.0**	**5.73**	**7.91**	**946**	**104**	**2.0**	**2.8**	**3.82**

Note: The "Average (Mean)" row represents the arithmetic mean of each performance metric across the ten test runs. Geometry and draw call metrics (Batches, SetPass Calls, Triangles, Vertices) remained constant across runs and therefore match the per-run values exactly.

Test Scenario 4: Occlusion Culling

Hardware Specs:

- CPU: AMD Ryzen 5 7600X 6-Core Processor

- GPU: NVIDIA GeForce RTX 4070 12gb

- RAM: Corsair 64GB (4 x 16GB) Vengeance RGB, DDR5 6400MHz, CL36, 1.35V

- Unity Version: Unity 6.1 (6000.1.2f1) Universal Render Pipeline

Table 11

Occlusion Culling Test Result Numbers

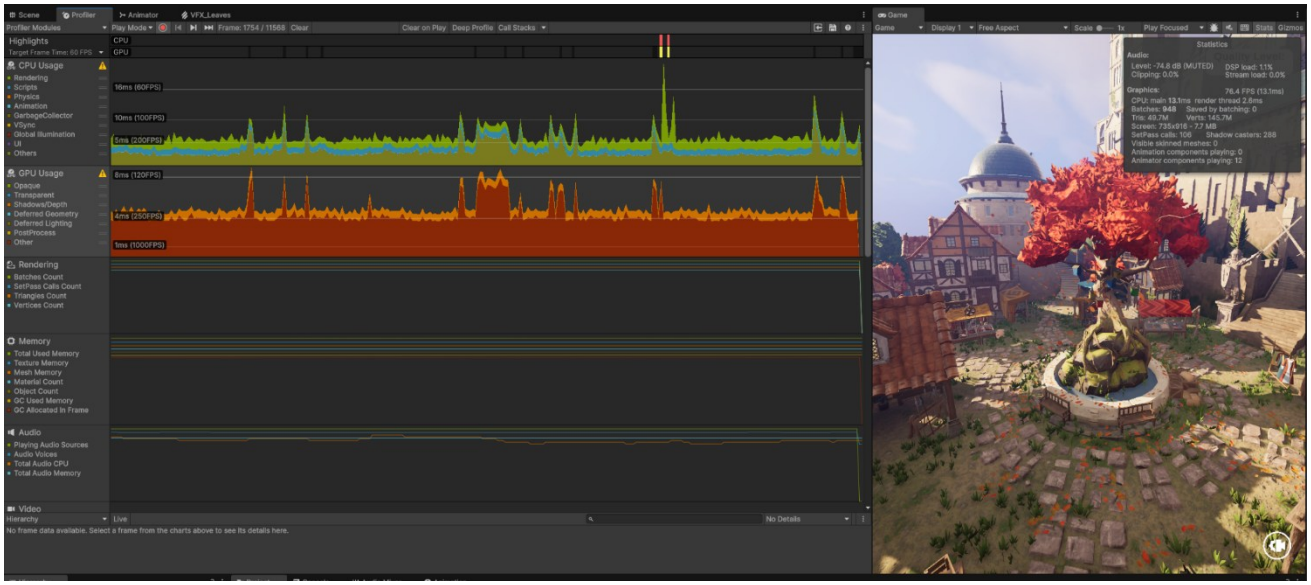
Run #	FPS	CPU Main Thread (ms)	GPU Frame Time (ms)	Batches	SetPass Calls	Triangles (million)	Vertices (million)	Total Memory (GB)
1	189.0	5.3	9.80	948	112	1.5	1.6	3.94
2	175.1	5.7	9.13	948	112	1.5	1.6	3.87
3	176.1	5.7	9.26	948	112	1.5	1.6	4.01
4	170.6	5.9	7.48	948	112	1.5	1.6	3.93
5	175.2	5.7	9.32	948	112	1.5	1.6	3.93
6	174.7	5.7	10.43	948	112	1.5	1.6	3.92
7	186.6	5.4	9.39	948	112	1.5	1.6	3.90
8	194.2	5.2	9.10	948	112	1.5	1.6	3.91
9	172.8	5.8	9.44	948	112	1.5	1.6	3.91
10	189.8	5.3	8.29	948	112	1.5	1.6	3.95
Average	**178.1**	**5.63**	**9.45**	**948**	**112**	**1.5**	**1.6**	**3.93**

Note: The "Average (Mean)" row represents the arithmetic mean of each performance metric across the ten test runs. Geometry and draw call metrics (Batches, SetPass Calls, Triangles, Vertices) remained constant across runs and therefore match the per-run values exactly.

Appendix 2. Raw Data Screenshots from Testing

Control Group

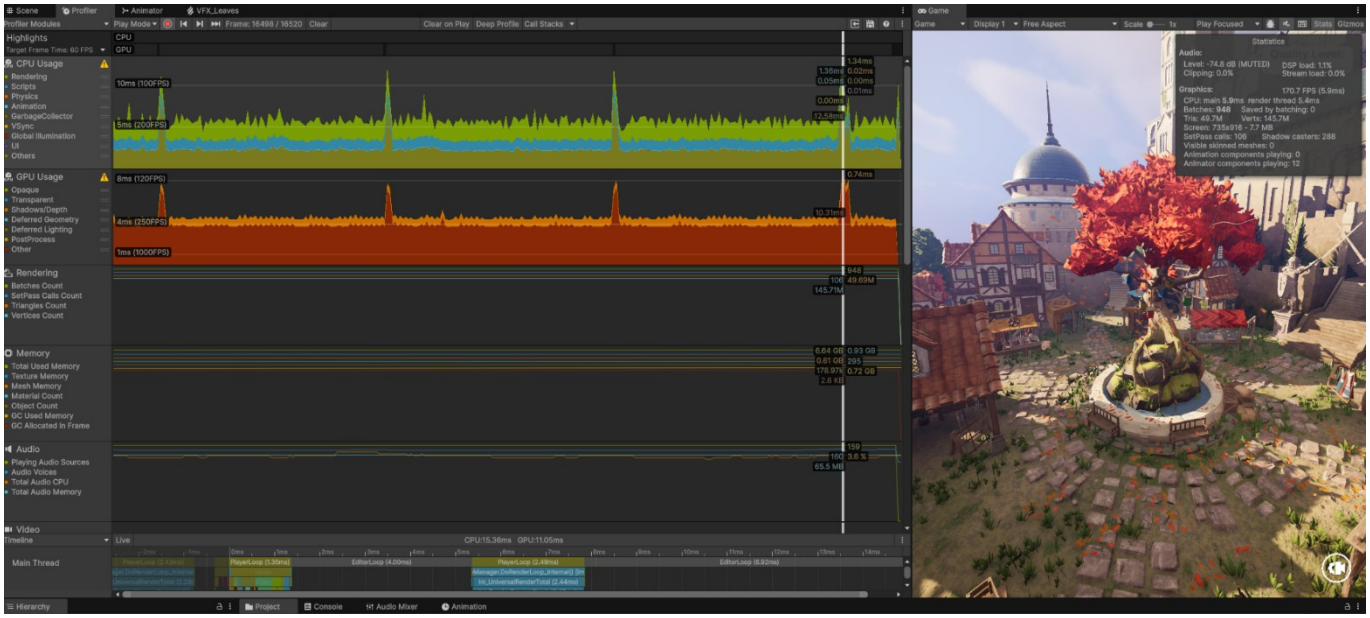
Control Group Test # 1



Control Group Test # 2

Control Group Test # 3

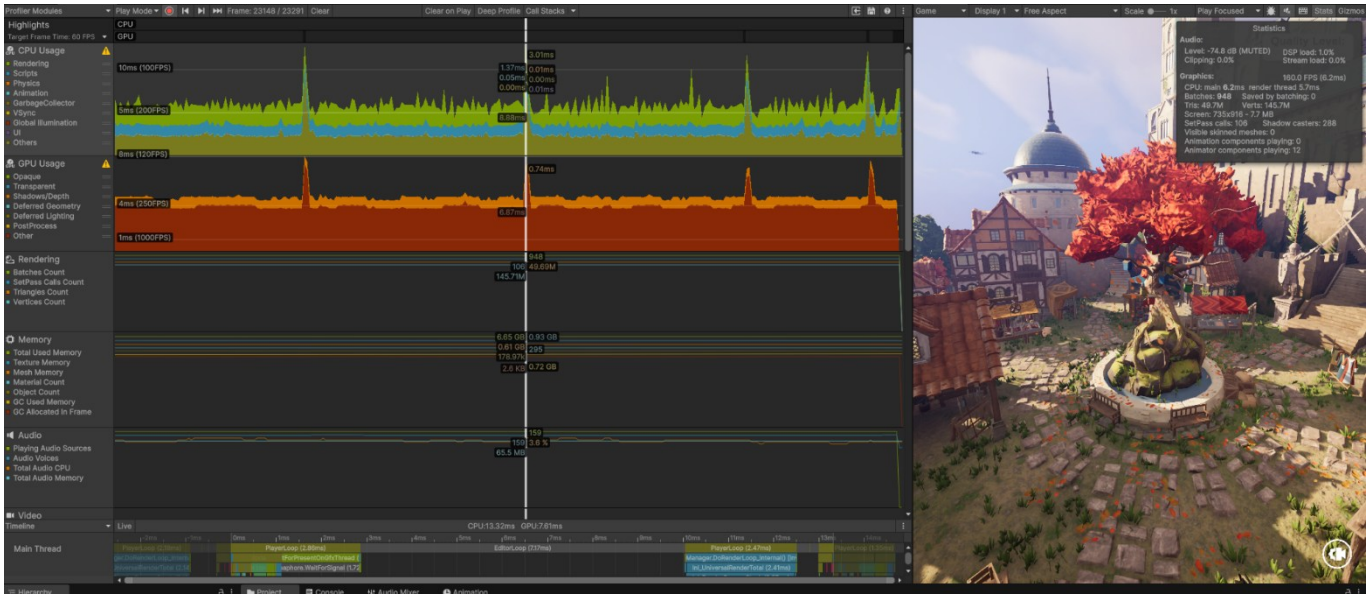
Control Group Test # 4



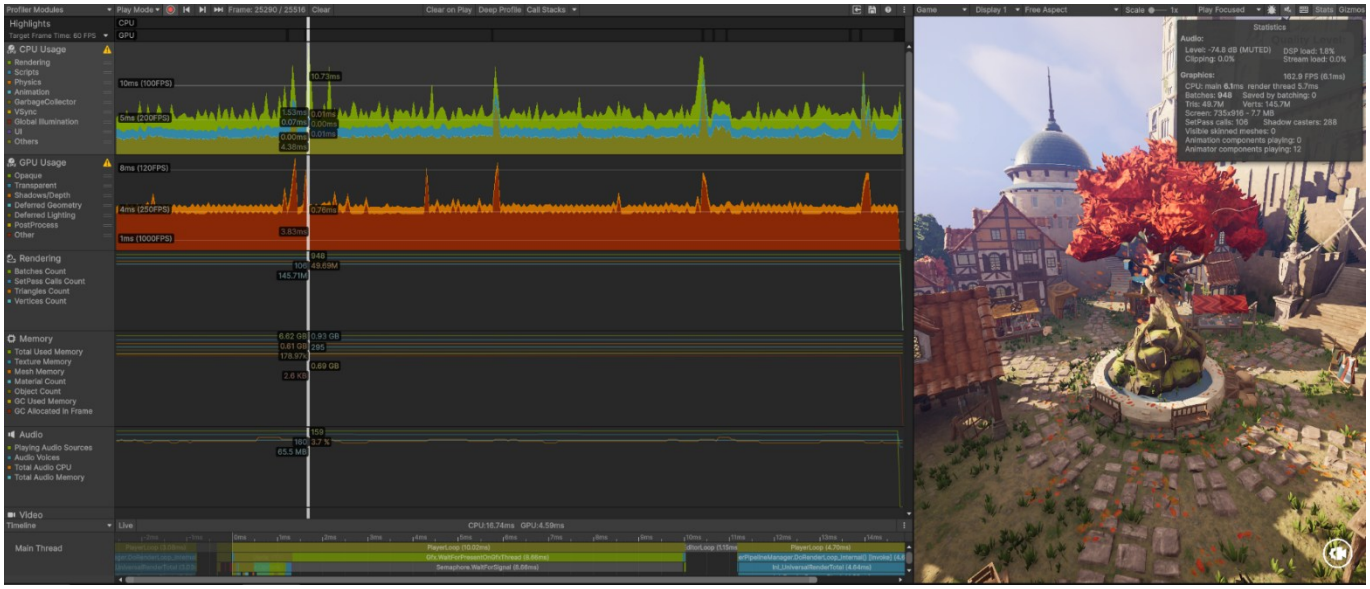
Control Group Test # 5

Control Group Test # 6

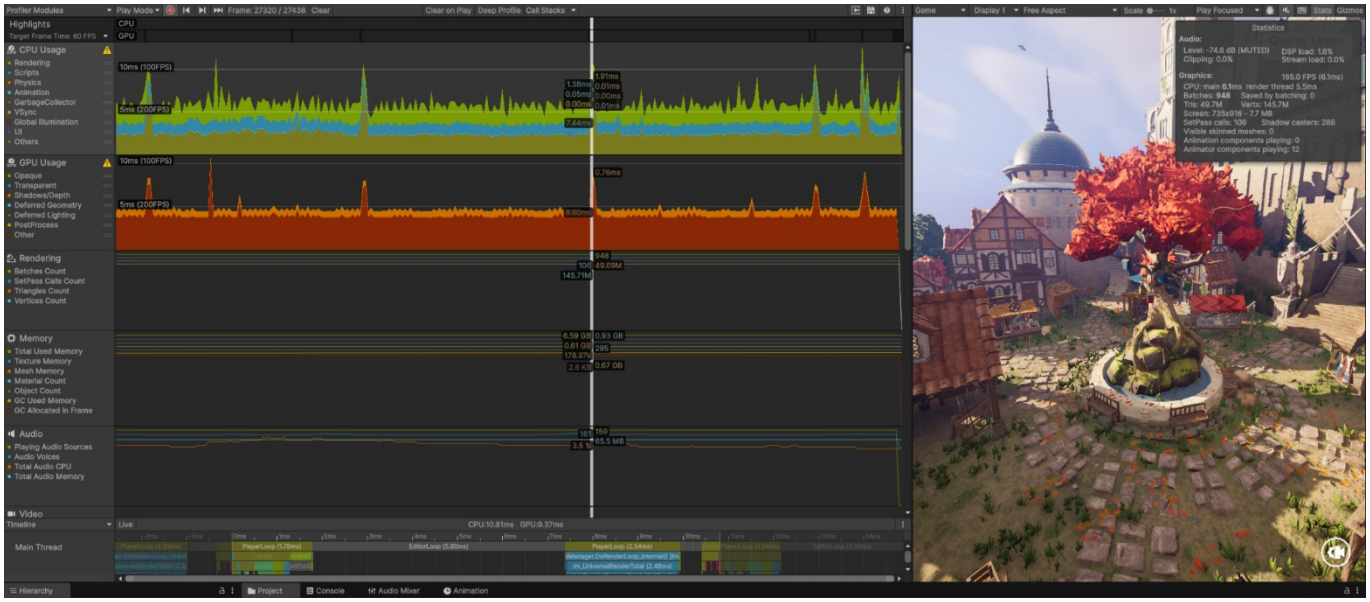
Control Group Test # 7



Control Group Test # 8



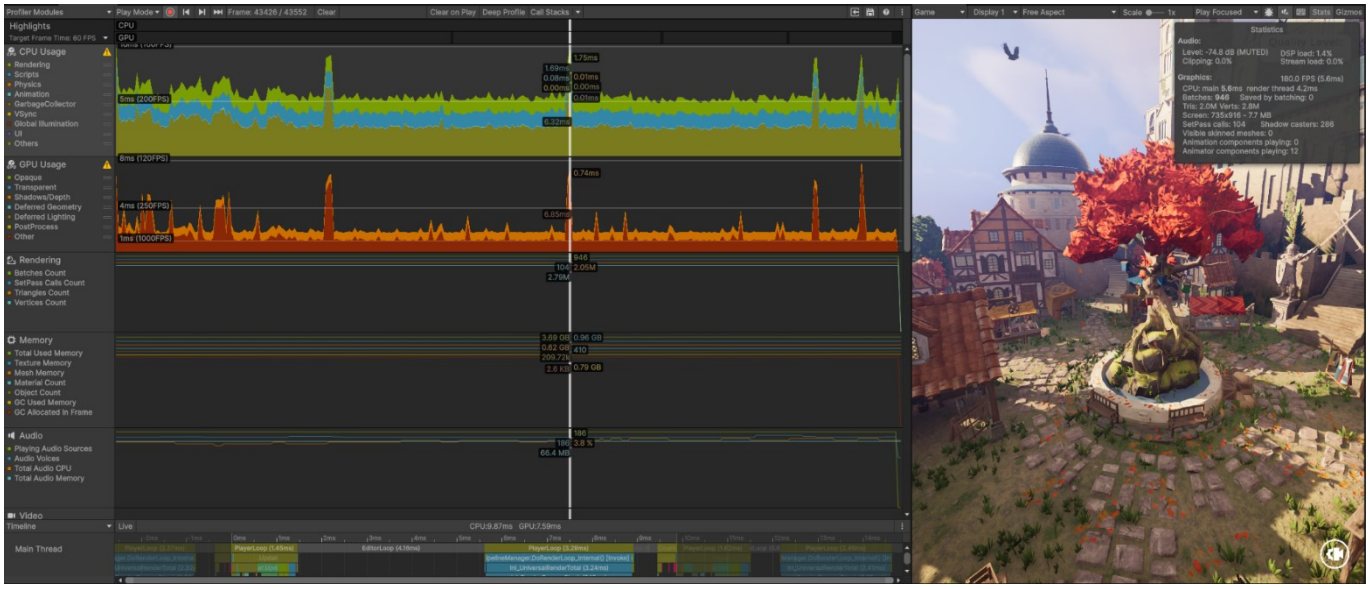
Control Group Test # 9



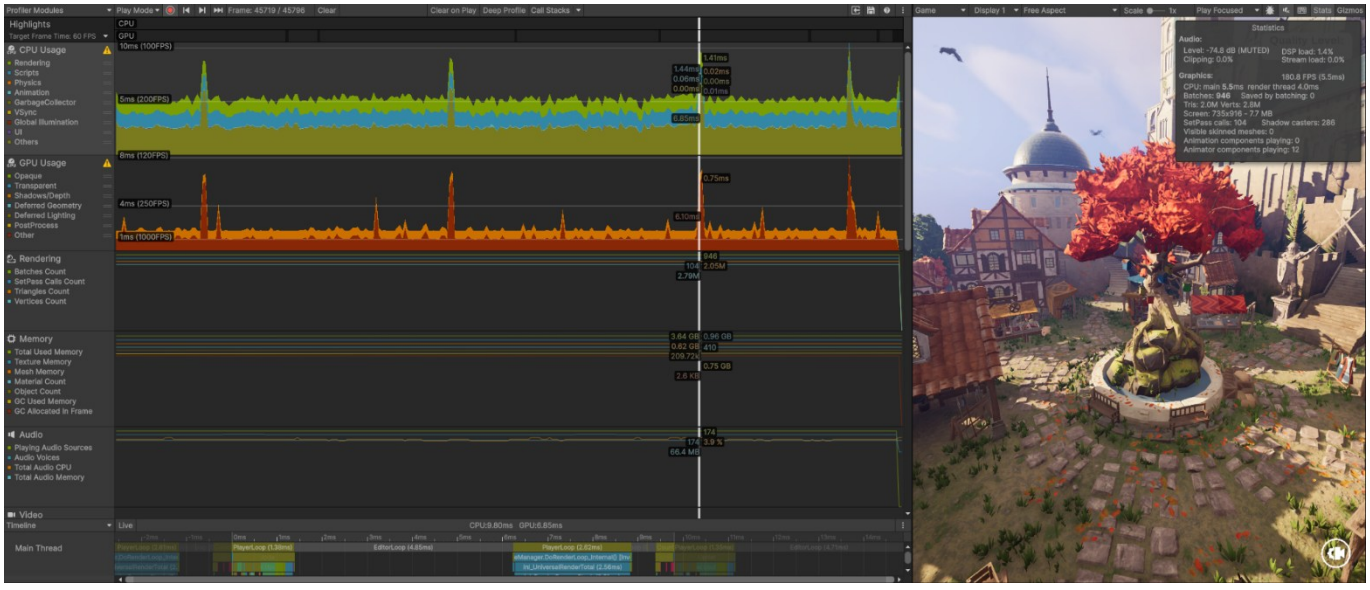
Control Group Test # 10

Overdraw Reduction

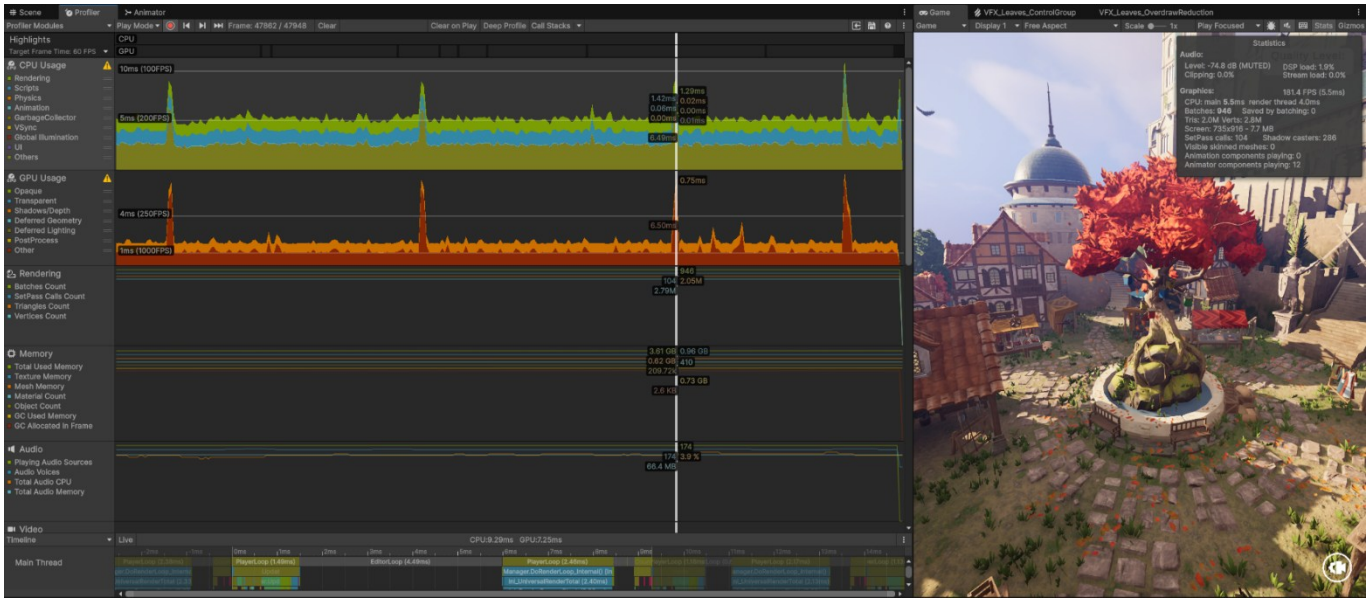
Overdraw Reduction Test # 1



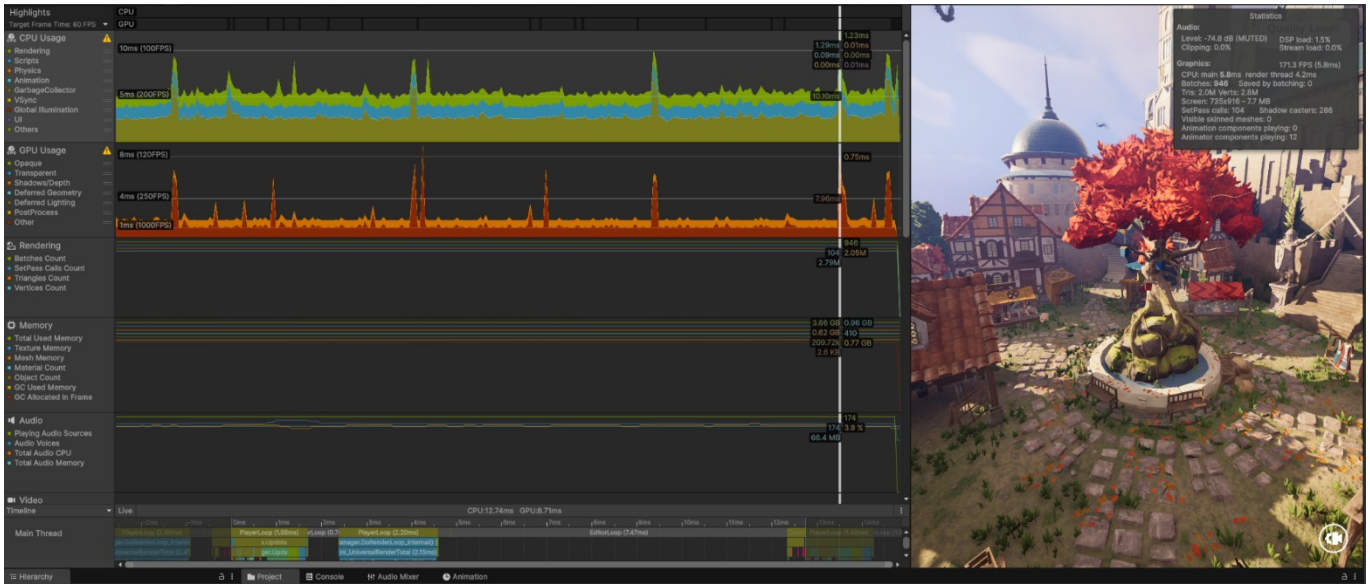
Overdraw Reduction Test # 2



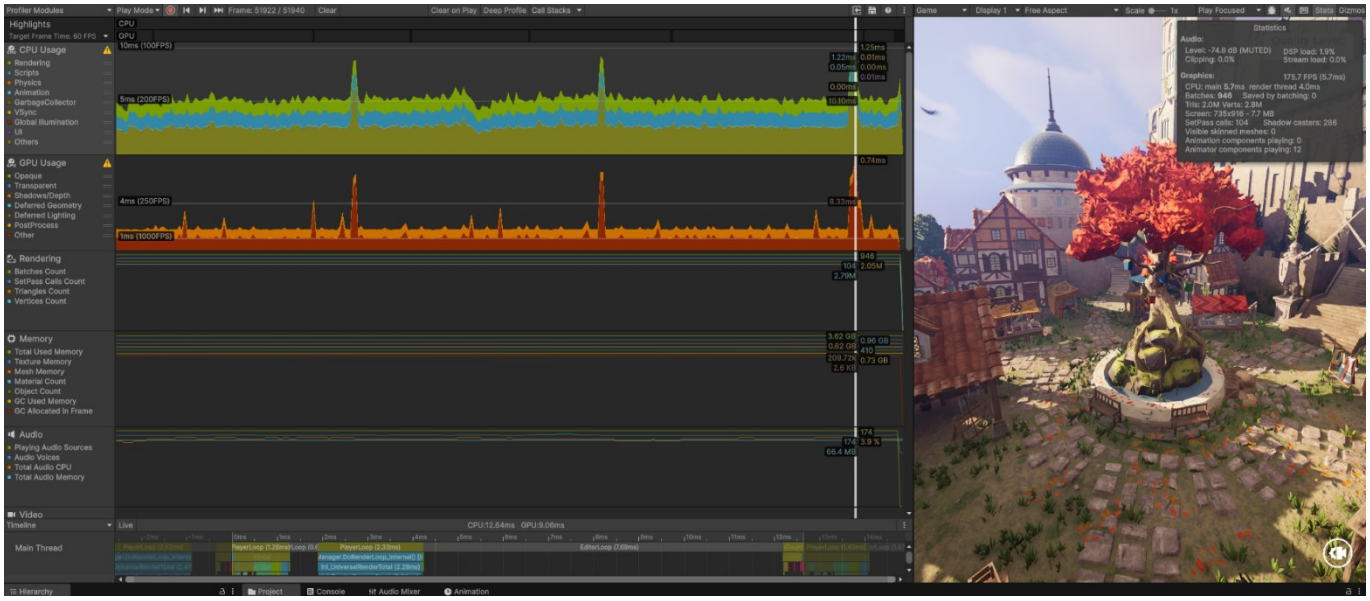
Overdraw Reduction Test # 3



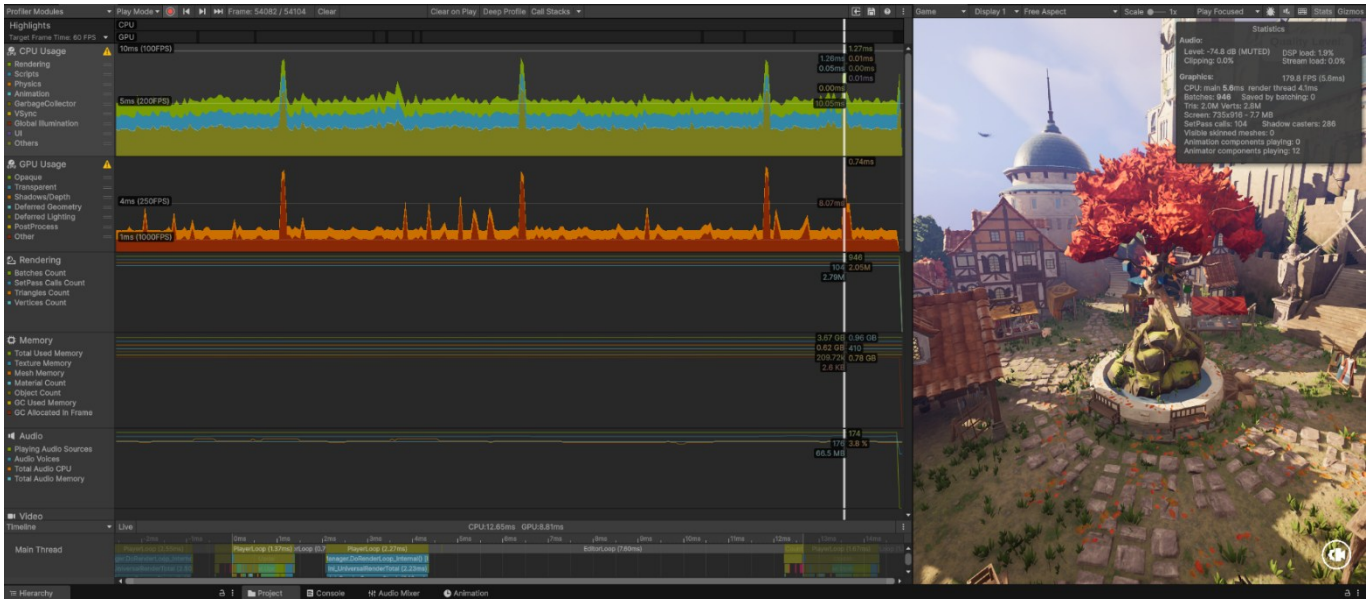
Overdraw Reduction Test # 4



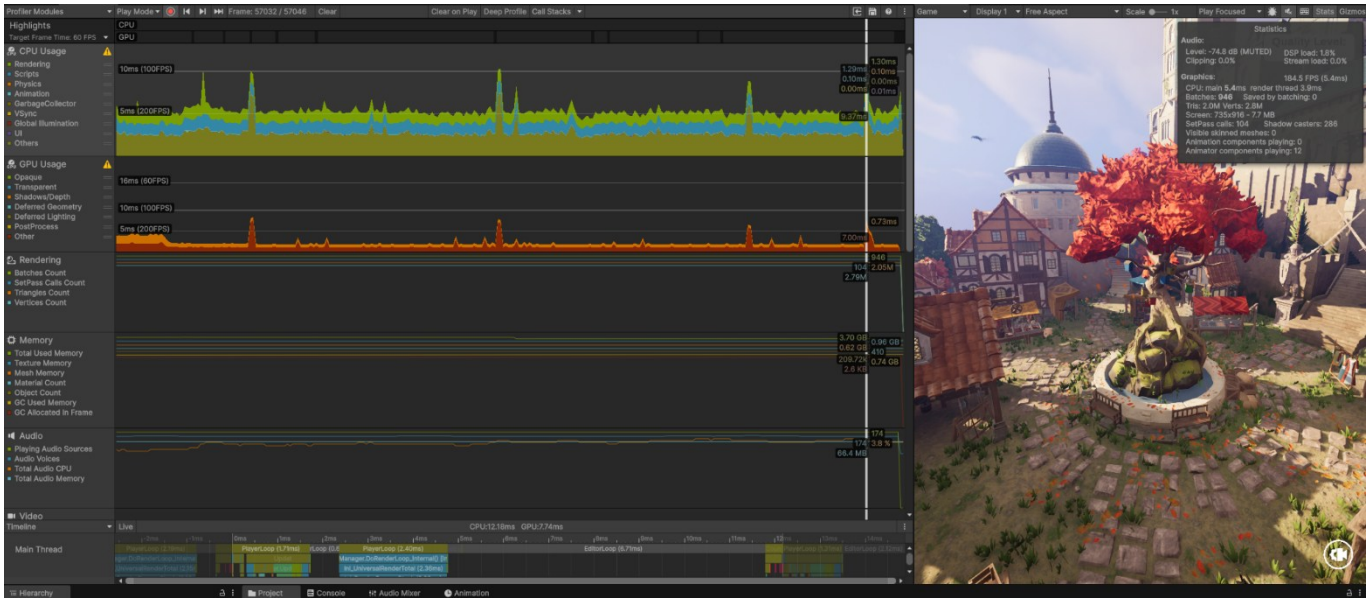
Overdraw Reduction Test # 5



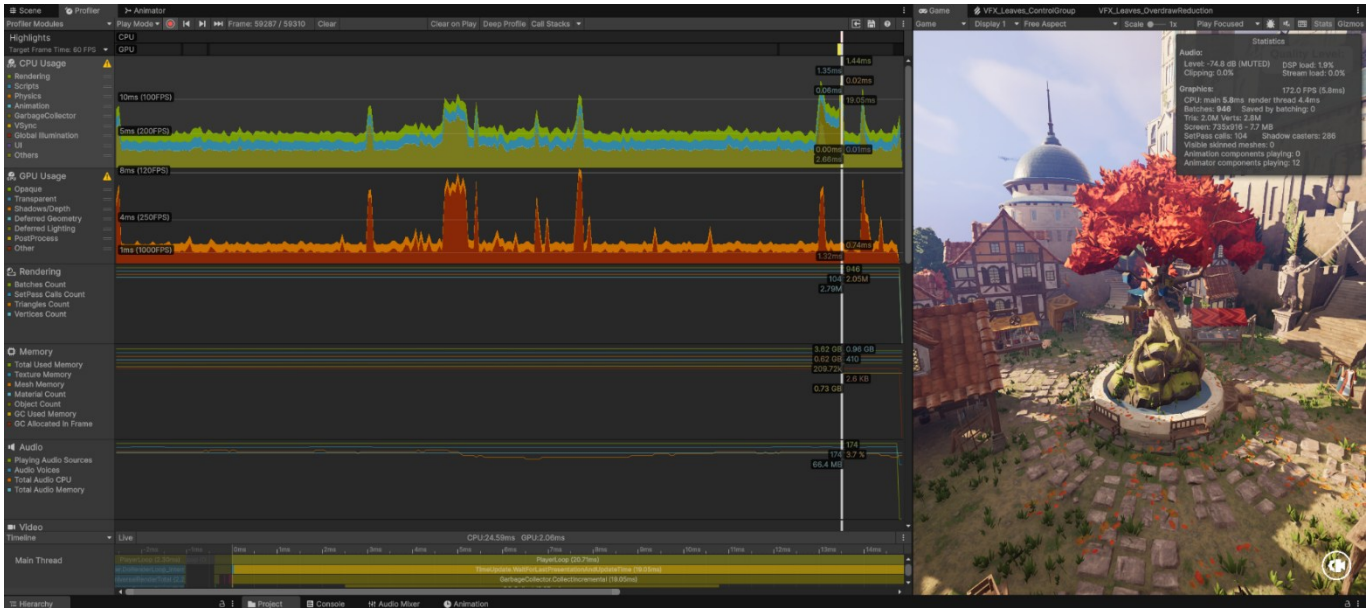
Overdraw Reduction Test # 6



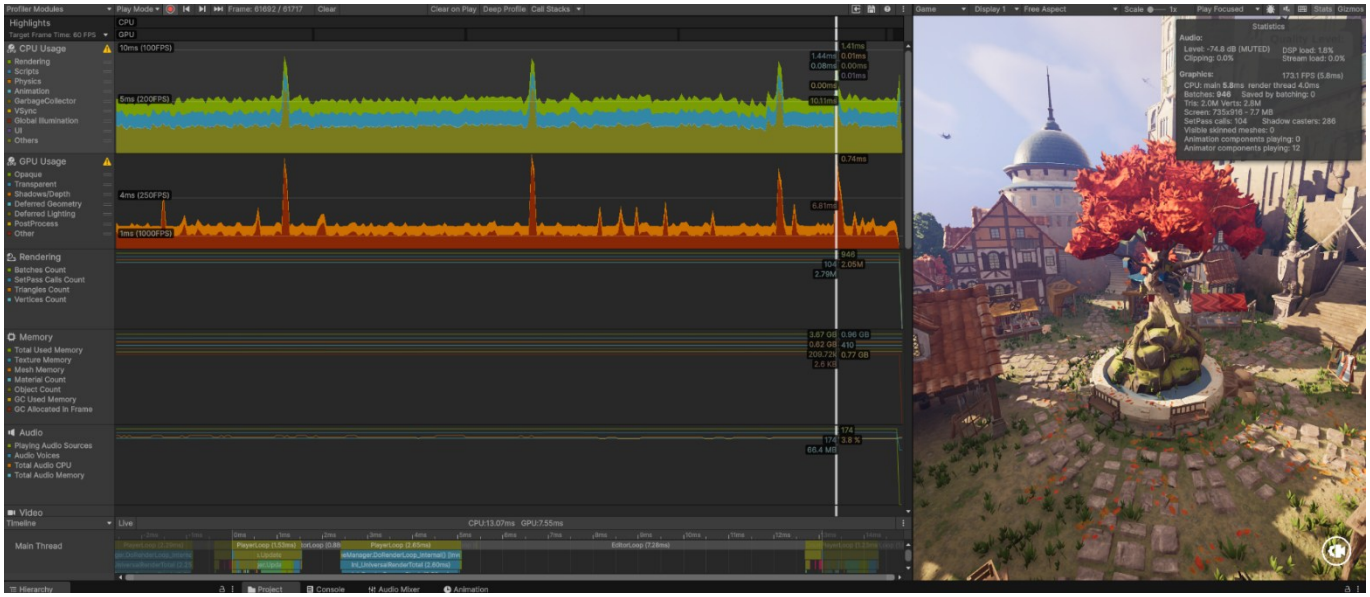
Overdraw Reduction Test # 7



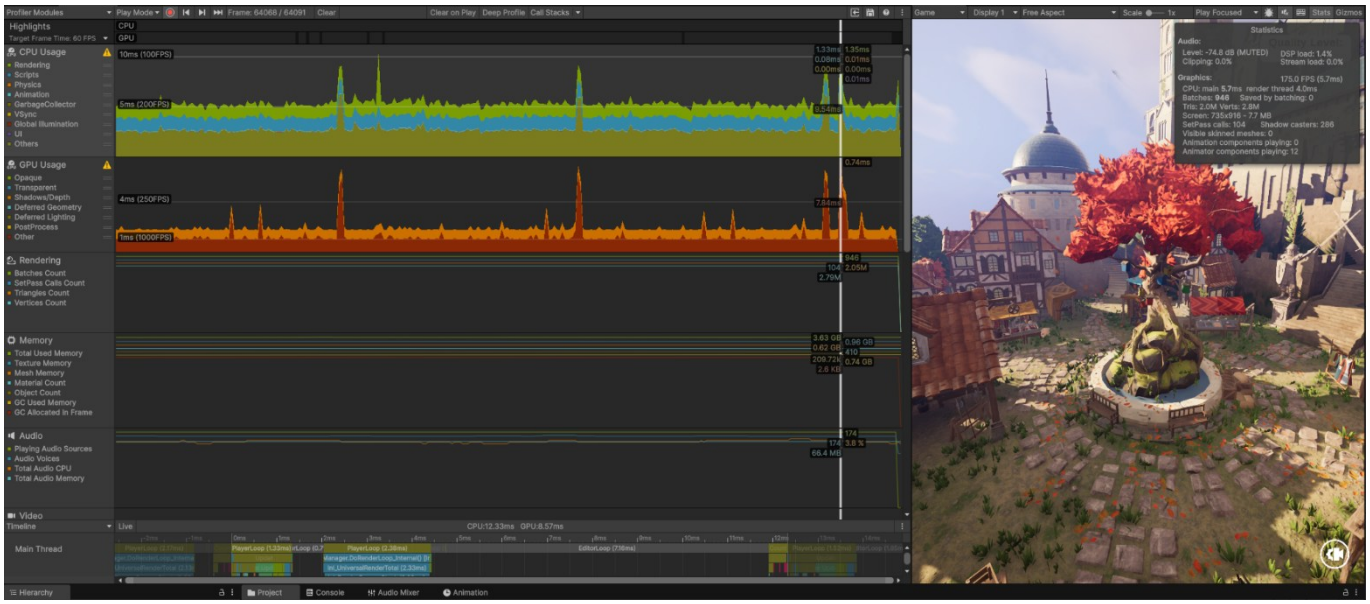
Overdraw Reduction Test # 8



Overdraw Reduction Test # 9



Overdraw Reduction Test # 10

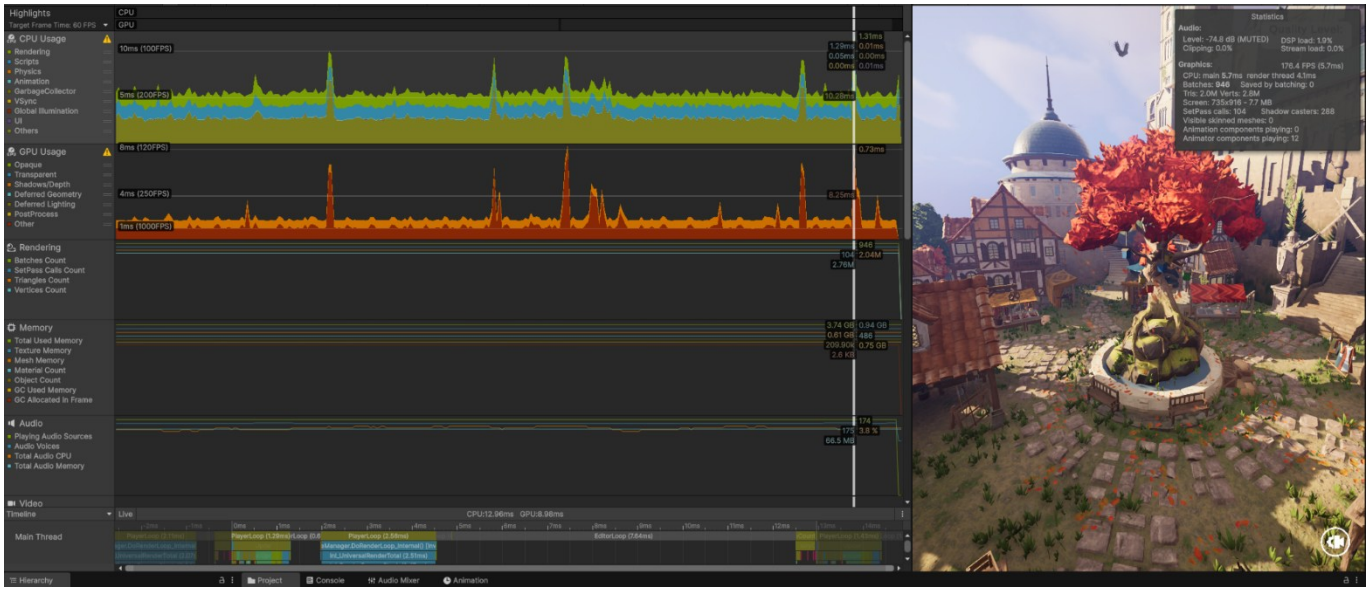


Geometry Reduction

Geometry Reduction Test # 1

Geometry Reduction # 2

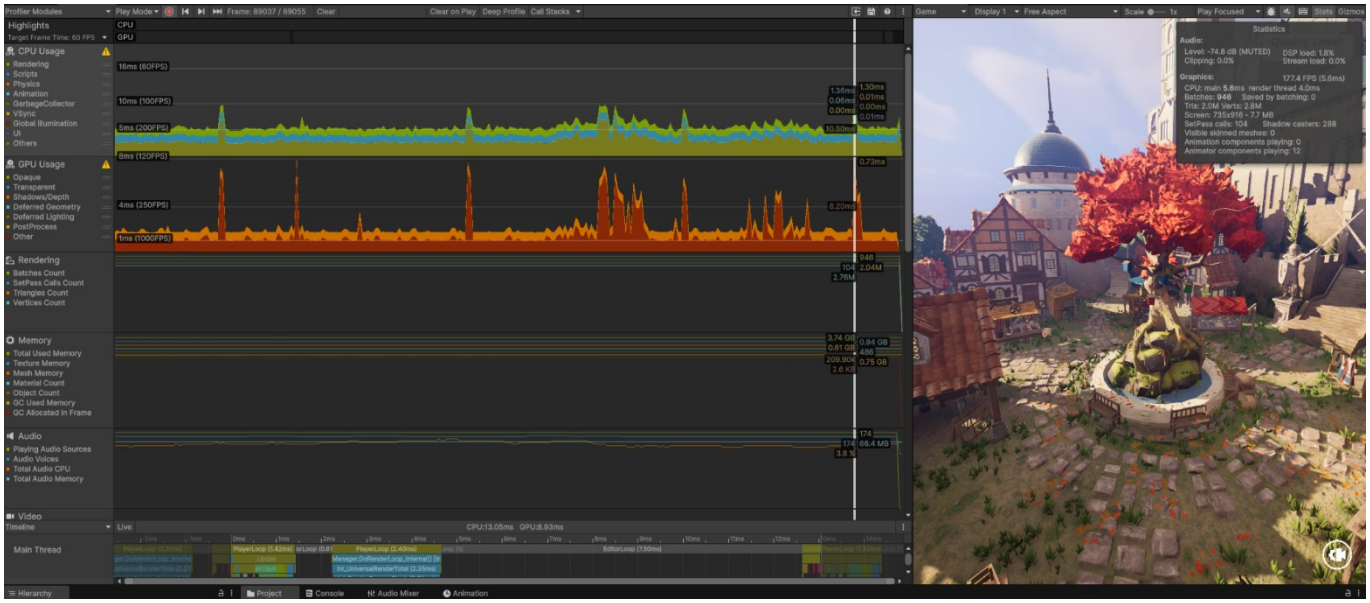
Geometry Reduction # 3



Geometry Reduction # 4

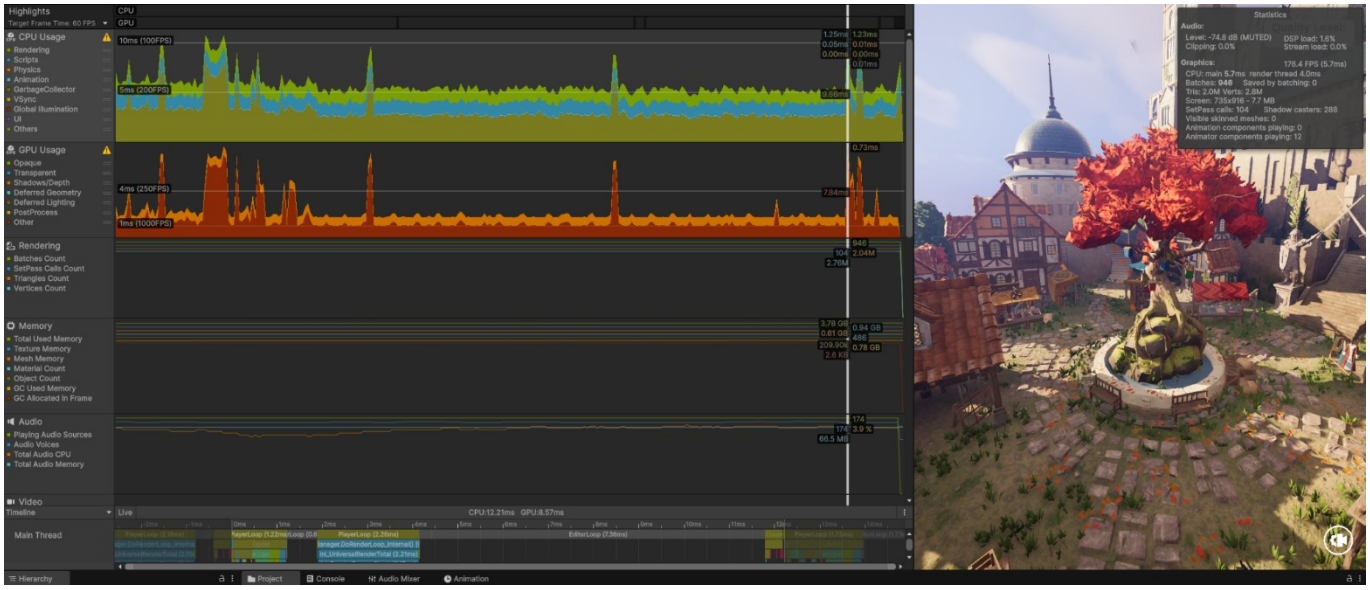
Geometry Reduction # 5

Geometry Reduction # 6



Geometry Reduction # 7

Geometry Reduction # 8

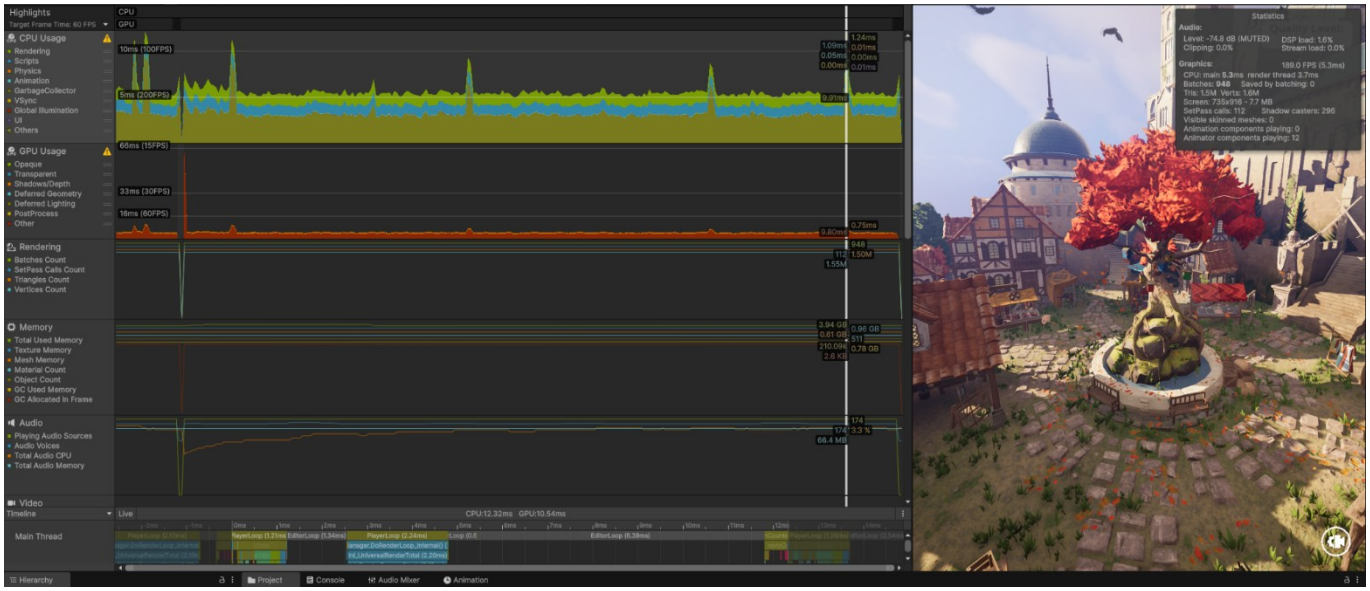


Geometry Reduction # 9

Geometry Reduction # 10

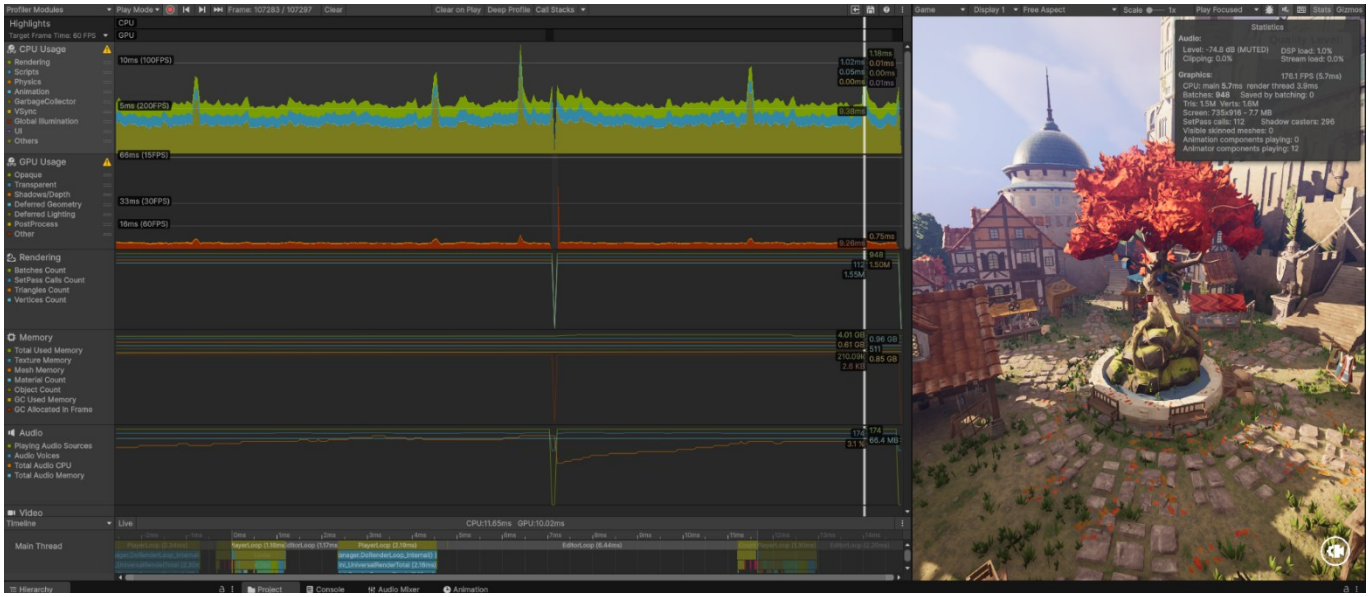
Culling Optimizations

Culling Optimizations Test # 1



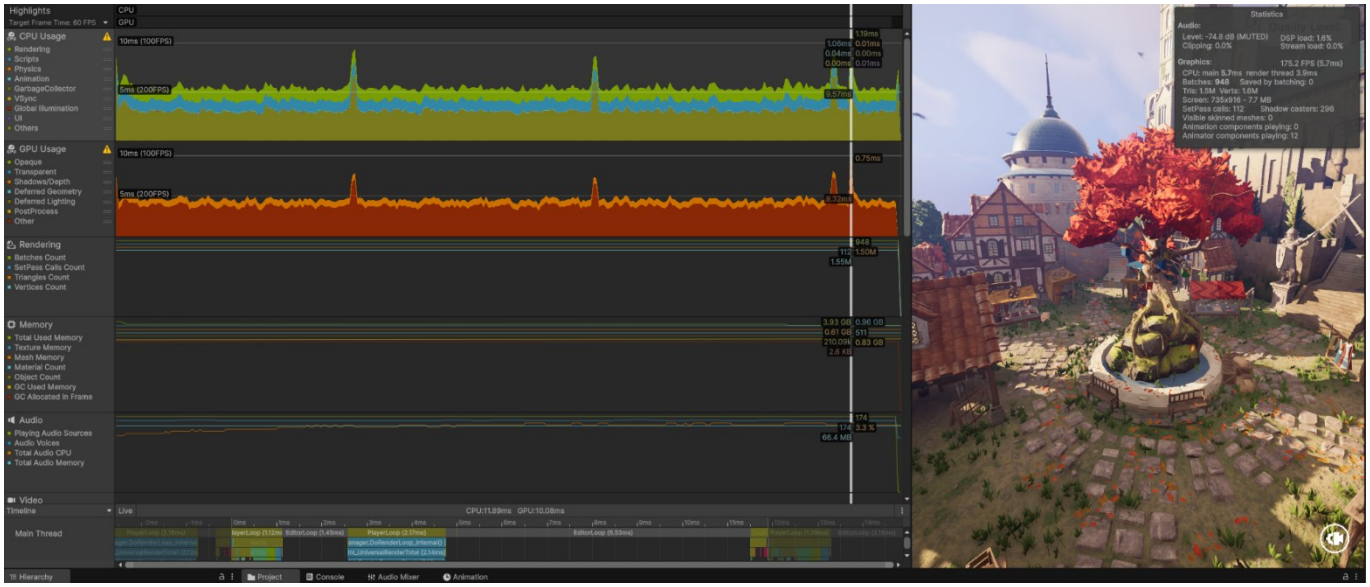
Culling Optimizations Test # 2

Culling Optimizations Test # 3



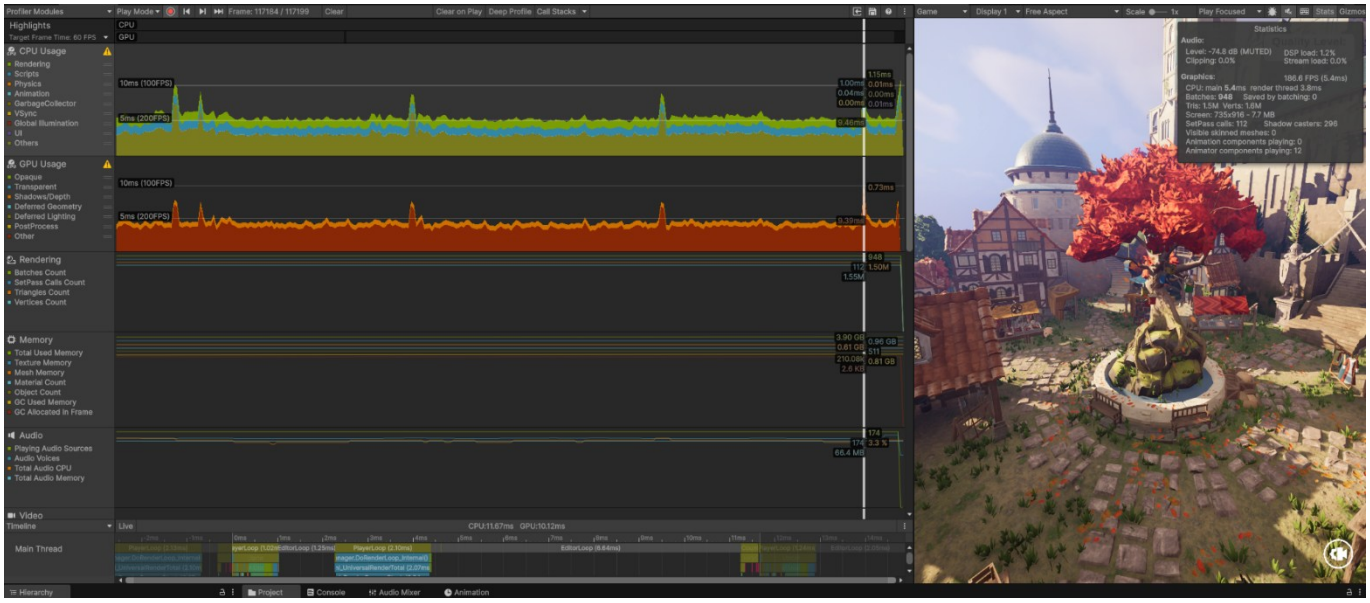
Culling Optimizations Test # 4

Culling Optimizations Test # 5

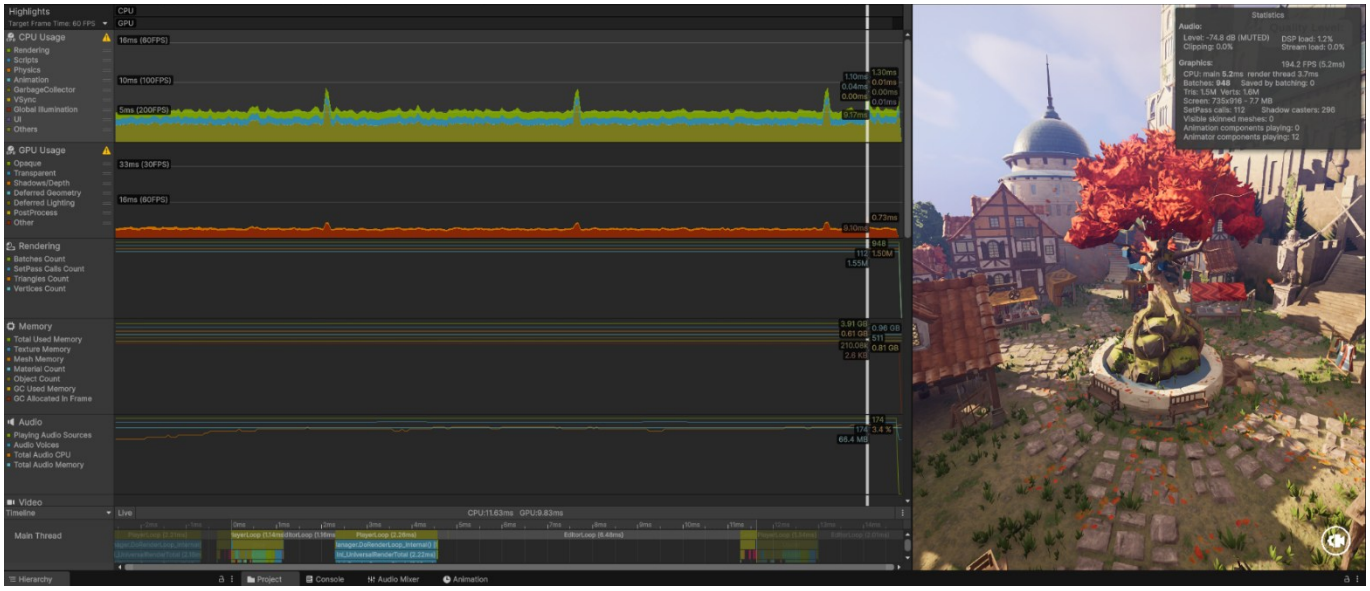


Culling Optimizations Test # 6

Culling Optimizations Test # 7



Culling Optimizations Test # 8



Culling Optimizations Test # 9

Culling Optimizations Test # 10