Bachelor's thesis

Information and Communications Technology

2023

Nico Aho

Automatic Optimization of Program Settings for Performance

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Bachelor’s | Abstract

Turku University of Applied Sciences

Information & Communications Technology

2023 | number of pages 58

Nico Aho

Automatic optimization of program settings for performance

The purpose of this thesis was to determine if it is possible to create an easy-to-use system that automatically optimizes the settings of a video game for improved performance. The reason this kind of system is needed is because optimizing the settings manually can be quite challenging and time-consuming for the average user.

The objectives of the thesis were achieved by first researching the settings used in video games. Then the acquired knowledge was used to develop several possible ways to achieve the system goals. Finally, a system was built to test the developed methods in practice.

When testing the system, it was able to automatically recognize when there was poor performance and improve it by changing the settings in the intended order. Additionally, the system was simple to use.

Based on the test results. it can be concluded that it is indeed possible to create a system that automatically optimizes the performance of a video game by changing its settings and that is user-friendly.

Keywords:

Optimization, program settings, automation

Opinnäytetyö (AMK / YAMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tietotekniikka

2023 | 58 sivua

Nico Aho

Ohjelman asetusten automaattinen optimointi suorituskyvyn parantamiseksi

Tämän työn tavoite on selvittää, onko mahdollista luoda helppokäyttöinen järjestelmä, joka automaattisesti optimoi videopelin asetuksia sen suorituskyvyn parantamiseksi. Järjestelmälle on tarve, koska asetusten manuaalinen optimointi voi olla keskiverto käyttäjälle hankalaa ja työlästä.

Työn tavoitteiden saavuttamiseksi ensin tutkittiin videopeleissä käytettäviä asetuksia. Sitten asetuksista hankitulla tiedolla kehitettiin joukko mahdollisia keinoja toteuttaa työn tavoite. Lopuksi rakennettiin järjestelmä, jolla kehitettyjä keinoja voitiin testata käytännössä.

Kun järjestelmää testattiin, se pystyi automaattisesti tunnistamaan huonon suorituskyvyn ja parantamaan sitä muuttamalla asetuksia halutussa järjestyksessä. Lisäksi järjestelmä oli helppokäyttöinen.

Testituloksista päätellen oma johtopäätökseni on, että on mahdollista rakentaa järjestelmä, joka automaattisesti optimoi videopelin suorituskyvyn, sen asetuksia muuttamalla ja jota on helppo käyttää.

Asiasanat:

Optimointi, ohjelmisto asetukset, automaatio

Content

1 Introduction

The goal of this thesis is to determine the feasibility of developing a simple to use system that is capable of automatically optimizing the performance of a video game on Unity by modifying its settings. In this thesis, the phrase “to optimize game performance” means that the settings of the game are set to values that result in the video game running at the desired frame rate, while minimizing their impact on its other functions. The settings that are most likely to be changed for performance gain are graphics settings. This is because graphics settings tend to have high impact on performance, while also having little impact on game functionality. Different graphics settings have varying impact on the frame rate and graphical quality. Common settings that are most likely to have high performance impact are the ones affecting things like textures, light and shadows. The settings impact on graphics quality tends to vary between games.

The goal and focus of this thesis, is to see if it is possible to develop a system that can automate the optimization of a video game performance which will include building a functional test system. While the goal is to allow the results to be usable across multiple development platforms and target devices, to keep the subject matter at a manageable level, the development and research will focus on using the Unity engine and targeting a computer-based video game. The goal of this thesis is not to provide a complete ready use system; instead, it is meant to demonstrate that it is possible to automate performance optimization and to give developers a possible starting point for their own automation systems.

The focus of this thesis is on the automation of a video games performance using settings and values that a user of a video game could be expected to otherwise modify themselves. Unless it is needed for automation purposes, this thesis will not focus or go in dept at optimization of video game performance in general. It should also be noted that graphics settings are likely to be featured

heavily in this thesis, simply because most of the settings that involve video game performance seem to be tied to graphics.

1.1 Reasons why automatic optimization of a game performance might be needed?

There are several real reasons why, “a simple to use system that is capable of automatically optimizing the performance of a game at runtime” might be needed. Automatic optimization is needed because the optimization of a game performance often requires a significant amount of time and knowledge that a user simply might not have, and some devices might not be able to run the game without at least some optimization. A simple-to-use (user-friendly) system being is needed because of the assumption that many of the users needing this kind of system might not have deep knowledge when it comes to using computer games.

1.2 Challenges of optimizing performance manually

Reading settings guides for the game Elden Ring released by Fromsoftware should showcase the potential problems users might face when trying to optimize game performance.

In his guide “Elden Ring PC performance and best settings guide” (Archer 2022) shows the minimum PC recommendations for the game as released by the developer and follows it by stating that using the correct settings, it should be possible to still run the game even on a PC that does not fully meet the minimum requirements. Later Archer goes through the various graphics settings describing their effects on the frame rate as shown by his tests. The article finishes with Archer giving his recommendation for the settings.

James Roach in his article “The best settings for Elden Ring: PC performance guide” (Roach 2022) tells the reader not to trust the minimum recommendations by the developer and that the game can be still run without fully meeting them. Roach also gives his recommendations for the settings based on his tests, though he does not tell what effect a specific setting has on performance.

When the articles are compared, certain similarities and differences comes up. Both articles advise not to trust the minimum requirements and that they obtained their recommended settings through testing. Where they differ are the recommendations themselves, where Roach recommends far lower settings than Archer. While it seems like the differences come from Roach targeting lower end PCs with his recommendations compared to what Archer does. What is curious that Roach recommends setting the “grass quality” setting to high (Roach, 2022), yet Archer specifically claims it having the greatest impact on performance (Archer, 2022).

Ultimately these articles (Roach 2022; Archer 2022) prove is how difficult it can be to figure out the best values for each setting and that it can be hard to find reliable information about it. The fact that there are actual guides about the subject matter alone, should show how difficult it can be, in addition to this the fact that not all video games have guides written about their settings and even an above average user might wind themselves struggling to find even somewhat optimized settings for a video game.

1.3 The goal for the system

While the previous sections of this chapter outline the goal for this thesis as whole. There should be separate goals set for the system that will be built during the thesis.

An ideal goal for the system would be one that is capable of automatically figuring out the best settings to change and adjust them to their optimal values and as such obtaining the best possible level of optimization of the settings that is possible. However, in practice an ideal system is not a practical goal for this thesis due to both time and technical constraints.

A more practical goal for the system would be somewhere between the ideal system and just systemically lowering the settings until a desired frame rate is reached. As such the actual goal for the system built in this thesis is as follows: a system that can adjust the settings in a logical manner to reach a level of

optimization that can match or exceed the one that an average user or player can achieve without help.

To clarify the goal, adjusting the settings in a logical manner means that the game uses a logical approach to change them instead of doing it completely randomly and as stated previously. Optimizing the settings means that the settings of the game are set to values that result in the game running at the desired frame rate, while minimizing their impact on its other functions. Lastly, in the context of this work the average user refers to someone who can carry out basic adjustments to the settings without help but lacks the more in-depth knowledge required to understand what each of the settings does.

1.4 Terms used interchangeably within the work

In this subchapter several common terms that are used throughout the work interchangeably, are described. Note that the description of the terms is meant to only apply in this specific work.

Developer

Developer revers to the person or persons, who create and develop programs and video games.

Game and Video Game, Computer Program and Program

While this work mostly uses the terms game and video game to refer the type of application it seeks to optimize, the words program and computer program may also be used occasionally to refer for similar entities. Within the context of this work all of these terms are mostly used to refer to graphics heavy applications that involve animations and movement that are not prerecorded ahead of time.

User and Player

User and player are terms used interchangeably to refer people, who either use a program for its designed purpose or play a video game.

2 Game settings

Before building a system to optimize the settings, it makes sense to look at the settings to figure out what their effects are.

2.1 What are the settings with most impact?

For the most part the performance affecting settings offered by various programs and games are tied to the graphical ones. The settings, that aren't tied to the graphics seem to affect things like player/user controls and other things, that should be left for the player preferences and as such this section will focus on mainly on the graphics settings.

When the question "What settings affect FPS the most?" was asked at www.quora.com, the answers given and collected by Jon B. Anderson were Ray Tracing, Resolution, FLSS/FSR, Tessellation, Game Physics, Antialiazing, Shadow Quality, Anistropic Filtering, Effect Quality, Ambient Occlusion, Bloom, Shadow Distance, Texture Quality and Depth of Field (Anderson 2022).

A further look at some other guides like the "Gaming Graphics & Optimization | What Settings Affect Performance the Most?" (Kozlowski 2022) gives us the settings: resolution, anti-aliasing, supersampling, volumetric lighting/fog, shadows, lighting, ambient occlusion, and a draw distance.

The guide "Understanding graphics settings for better frame rates" (Kalbere 2022) on the other hand gives us the settings: anti-aliasing, DLSS, adaptive resolution, motion blur, field of view, v-sync, anisotropic filtering, tessellation, ambient occlusion and bloom, ray tracing and HDR.

Lastly in his guide "5 graphics settings you need to change in every PC game" (Roach 2022), Jacob Roach gives us the following settings: v-sync, motion blur, field of view, shadow and lighting quality, ambient occlusion, reflections and ray tracing, anti-aliasing, DLSS, FSR and dynamic resolution, texture quality, geometry quality, anisotropic filtering, and post processing.

To make inspecting the settings easier, they should be condensed into a single list, with duplicates removed and some of them placed under wider categories. As such the new list will be as follows: Resolution, Antialiazing (with the subcategories of Supersample Anti-Aliasing (SSAA), Multisample Anti-Aliasing (MSAA) and Fast Approximate Anti-Aliasing (FXAA), Lighting and shadows (with the subcategories of Lighting, Shadows, Shadow distance, Ray Tracing, Ambient Occlusion, Volumetric Lighting/Fog, Bloom and High-dynamic-range rendering), Tessellation, Texture Quality, Draw Distance, Field of View, Effects (with the subcategories of Motion Blur and Post processing effects), Game Physics and Performance optimization techniques (with the subcategories of V-Sync, Anistropic Filtering, DLSS/FSR, Adaptive/dynamic resolution and High-dynamic-range rendering(HDR).

2.2 Impact of the settings

Here the impact of the setting refers both to the impact on performance and also to the impact on the game e.g., how noticeable it’s graphical effects are or how much effect it has on the game play.

In this section I’ll be going through each setting listed in the previous subchapter and give a brief description of what they do and the impact they have. Under each setting I have also described how they should be handled by the automated system build in this work.

Resolution

Resolution affects the number of pixels that are send to the monitor at once. As resolution essentially the whole screen, changing it has a noticeable effect on graphics quality. Sebastian Kozlowski says that the “resolution is one of the most performance damaging settings there is” and that even relatively small changes can have huge effect on frame rate. (Kozlowski 2022.)

Antialiasing

Antialiasing is used to smoothen the jagged edges that occur on digital images and graphics. While it can have a noticeable effect on graphics quality it does come at a cost to performance. There are various antialiasing methods that have different effect on performance and graphics discussed below. Each method can come with a separate setting to change level of quality they work at, with higher levels offering better quality at the cost of performance. (Kozlowski 2022.)

Supersample Anti-Aliasing (SSAA) is a method of antialiasing in which the game is rendered in a higher resolution to grab extra detail and then downscaled to the actual resolution. While it can offer high levels of quality it can an have extremely high impact on performance. (Syed 2022.)

Multisample Anti-Aliasing (MSAA) only renders the edges of the scene at a higher resolution while otherwise working similarly to SSAA. While it can offer similar levels of quality as SSAA, it tends to have a lot less impact on performance. (Syed 2022.)

Fast Approximate Anti-Aliasing (FXAA) uses the pixels surrounding an edge to essentially guess the information needed to smoothen the edges. While it may offer smaller level of quality compared to other methods, it also tends have the lowest impact performance. (Syed 2022.)

Lighting and shadows

Lighting and shadows refer to various methods and settings that impact how the game handless lighting and shadows. According to Jacob Roach “By far the most demanding task for your graphics card is shadows and lighting”. Because they work together, they are handled together here, despite some gamess offering them as separate settings. (Roach 2022.)

Lighting works by rendering a lightmap that determines the brightness of objects in a scene, while shadows work by doing the same with a shadow map that is rendered by using the lightmap. As both features essentially require the rendering of second scene it, becomes clear why they might have such a heavy impact on performance. Lowering the lighting quality tends to result in less detailed lighting effects, with less lighting sources. Lowering the shadow quality on the other hand typically results in less detailed shadows and additionally games often offer the option change the distance in which shadows are shown. (Roach 2022.)

Ray Tracing is an advanced method of generating lighting and shadow effects by drawing individual rays of light from a lighting source. While this method can generate realistic shadows and reflections as Jacob Roach buts it “the problem is that ray tracing is insanely demanding”. (Roach 2022.)

Ambient Occlusion deals with shadows that objects cast on each other. The effect is often subtle, yet it can have a large impact on performance. (Roach 2022.)

Volumetric Lighting/Fog is a method of simulating the effect of light cast through windows, fog and other mediums. The effect can be quite demanding on performance. (Kozlowski 2022.)

Bloom setting controls the luminosity of light sources in games, when implemented well it adds to the realism of the game, however poor implementation may result in oversaturating and /or blinding lights. (Anderson 2022.)

High-dynamic-range rendering (HDR) is way to add more detail to the lighting of the scene by doing the lighting calculations with a higher range of values than the standard (0.0 - 1.0). Without using the HDR details of the scene can be lost in especially bright or dark areas. It should be noted that while using a non HDR capable monitor the values need to be converted back to the standard range, but if it is done using proper algorithms, the end result will still have more detail than not using HDR. (Vries n.d.)

While HDR adds extra steps to the lighting calculations, it only has a small impact on performance. (Vries n.d.)

Tessellation

Tessellation (also known as geometry quality) is a method of rendering smoother looking models and textures at the cost of increased demand for performance. (Kalbere 2022.)

Texture Quality

Texture quality determines the resolution of textures. While similar to the resolution setting discussed above, it only affects textures. While lowering it can improve performance, it may only do so at otherwise high resolutions or on graphics cards with limited video memory. (Roach 2022.)

Draw Distance

Draw distance affects the maximum distance object objects are drawn in a tree dimensional scene. While it’s effect may be quite noticeable at higher distances it can have a heavy impact on performance. (Kozlowski 2022.)

Field of View

Field of View (or FOV) affects the viewing angle of a player. Increasing it increases how much of the scene is seen at once. However, increasing it also increases the number of resources it takes to render the scene. Because it directly affects the user’s ability to see in the game, it may also have a direct effect on the user’s ability to play the game. (Kalbere 2022.)

Motion Blur

According to Jacob Roach there are two kinds of motion blur, which are camera motion blur and per-object blur. Camera motion blur tend to look terrible but has almost no performance impact. Per-object blur is fairly demanding on performance requires, but also looks more natural. (Roach 2022.)

Post processing effects

Post processing effects include things like depth of field, film grain and chromatic aberration and so on. They are applied to the scene after it has been rendered and typically don’t have a large impact on performance. (Roach 2022.)

Game Physics

According to John B. Anderson reducing game physics can have a significant effect on frame rate, however only some games offer it as a setting. (Anderson 2022.)

2.3 Performance optimization techniques

In this work, performance optimization techniques refers to methods and techniques that are intended to reduce the strain a game puts on performance. As this work mainly deals with optimization using pre-existing settings, a more detailed look and the actual implementation of these techniques is beyond the focus of it. However, as games often have these methods pre-implemented and offer settings relating to them, taking a brief look at them is still worthwhile.

V-synch

V-sync is method of preventing screen tearing by preventing graphics card from producing frames faster than the refresh rate of a screen can handle. (Roach 2022.)

Because the technique works by limiting the frame rate of the game, it may improve the performance, however it can also sometimes add extra input lag to the game. (Roach 2022)

Anisotropic Filtering

Anisotropic Filtering (also called AF) is a technique only the textures withing the players view are rendered instead of all of them. This not lonely lowers the performance impact, but also improves the rendering speed. (Kalbere 2022.)

Upscaling technologies

Upscaling technologies are a range of technologies, that allow the game to be set on a lower resolution and then later upscaled to what is effectively higher resolution, while still resulting in a lower performance impact.

Deep Learning Supersampling (or DLSS) technique that uses an A.I for upscaling. While it can provide a massive boost to performance it requires certain NVIDIA graphics cards to work. (Roach 2022)

FidelityFX Super Resolution(FSR) uses an established algorithm for upscaling. While DLSS tends to work better the FSR works on any graphics card and can still provide massive performance gains. (Roach 2022)

Adaptive/Dynamic Resolution

Adaptive or dynamic resolution (DRS) is technique, that seeks to maintain a certain frame rate by adjusting the resolution as needed. (Roach 2022.)

A proper implementation of DRS does not simply change the resolution setting directly or linearly but may for example adjust each axis separately. (Brookes 2021)

While dynamically adjusting the resolution setting is part of the system that this work seeks to build, a dedicated implementation of it is likely result in less noticeable drops in image quality. (Brookes 2021)

While adaptive resolution does something similar, to what this work seeks to do (which includes adjusting the resolution as needed), a proper implementation of it is likely to give better results. (Brookes 2021)

3 Potential approaches

3.1 Detecting the poor performance

Before a video game can try to “automatically optimise the settings” it needs a way to detect the need to do so first. How this is done depends on what constitutes as a need. If the goal is to simply optimize the settings before use, a simple prompt asking to do so when starting the game for the first time should be sufficient. If the goal is to optimize the settings as a reaction to the game running poorly, a more complex solution is needed.

To detect poor performance during play, requires that some sort of check is run continuously to keep an eye on something that indicates bad performance. As the performance checking will be done continuously during gameplay, it is important to keep its impact on performance as low as possible. While bad game performance can appear in gameplay in many ways with clear visual signs, which include things like continuous stuttering, lag, graphical glitches and low frame rate, many of these are not as obvious for a computer to detect as they are for a human. Of these perhaps the easiest to keep track is the low frame rate as shown by the many games that allow players to display their current frame rate during gameplay.

To use the frame rate as an indicator of bad performance requires that the game knows what counts as bad frame rate. Having the system only keep track of the current frame rate and react immediately when it goes below acceptable levels might cause problems because sometimes there might be brief drops that don’t really affect the gameplay. These drops might be because during certain operations, like loading new areas or other assets, a game uses more resources than normal. There might also be brief bugs either on the game or the system it is running on, whose only result is a brief drop-in frame rate. If the drops aren’t common or disruptive to gameplay, there probably isn’t a need for optimization. To avoid unnecessary alerts, instead of tracking the current frame rate, it would likely to be better to track the average frame rate instead. When

the game is doing especially resource intensive operations, like loading a level, the resulting drops in fps might skew with the average frame rate value. By pausing the frame rate calculations during such points, could keep the average frame rate value closer to normal gameplay.

When poor performance has been detected, the system should ask the player permission to optimize the settings. Regardless of what kind of system is used to detect performance the player should have the option to manually start the optimization process anytime they want.

3.2 Figuring out the performance impact of a setting

While the previous chapter discusses the performance impact of the settings in general, it does not give the performance values in a manner that can be used in an automated system. To properly obtain the required values, a way to measure the impact each setting has on frame rate is needed. While this could be done manually by recording the fps with the default settings and then individually changing each setting and calculating the difference, it is both time consuming and adds a possibility of human error to the process. A better way to do things is to use an automated system.

While there are a variety of available tools, both free and paid, that are specifically made to obtain performance data. Going through all of them to find the most suitable one, is likely to take more work than what is worth for the purposes of this thesis. Because the test system build within this thesis will be done using Unity (as mentioned in the introduction chapter of this thesis), it is likely to be more efficient to handle the performance testing with a mix of custom code and the tools provided by Unity. It should be noted however, that it is possible that some other readymade tools might be of more use to others wanting to implement their own optimization systems.

There are several things that need to be considered when building the system that handles the performance tests.

Consistency and repeatability

The consistency and repeatability mean that the system must be able to get the same or similar results each time it is run with the same settings and hardware. Ideally, there is no more than 1% difference between each test. If a test shows inconsistent results, the scene and test should be analysed to see if said inconsistency is because of intentional randomness (etc. weather effects or different objects being spawned). In this case the best option might be to run the tests multiple times. If the scene lacks anything that might cause inconsistency, then there might be a bug in either within the test system or the scene. In this case testing the test system using multiple scenes is recommended. (Walton 2022)

Realistic test cases and a variety of situations

The test must be able to give results by using scenes and situations, that represent, what is happens in the actual game. An example of a bad test would be a fps test in a level with the other drivers removed. Another thing the tests must be able to handle, are all the various situations that can happen in the actual game. If one scene has lots of shadows and another has a lot of moving objects, then both scenes need to be tested. This does not necessarily mean that all the scenes need to be tested, only the ones that matter the most (etc. if testing shadow related settings, testing only the scene with most shadows should be enough). (Walton 2022)

Simple, convenient, and fast

When designing the tests, they should be as simple and convenient to use and fast to do, as possible (Walton 2022).

The simple to use means, that there should not be any unnecessary settings or steps, that the user needs to go through to run the tests (Walton 2022).

Convenient to do means, that there should be options for reducing the number of times the user needs to do things during testing etc. having the option to automatically run multiple tests in a row or automatically testing all the values, that a single setting might have (Walton 2022).

Fast to do means both that the that the tests can be set up quickly and only run if they need to, avoiding unnecessary delays and filler, etc. when testing water performance going through areas without water (Walton 2022).

Developer testing vs user testing

It is a well-known fact, that different hardware has different performance and as such, tests run on one machine are not necessarily applicable to another. This becomes even more clear, when one considers that a computer may have different programs running on the background that are taking up its resources (Walton 2022).

What this all means, is that a system using the results of performance tests can’t necessarily rely on values done only on the developer machines. As such it is important to give the user the ability to run the tests on their own machines. At the same time, it is probably not a good idea to just give the user the testing tools and tell them to test the game. A better idea is probably just to give the user a set of ready to use benchmarks, that they can just start to get the necessary values (Walton 2022).

Tests carried out only on developer machines are not necessarily useless however and might still give usable values or at least a general idea of a settings impact on performance. As such while including the option to allow the user to calibrate their values with ready to use benchmarks is a good idea, it might not be necessary to force them to do so. This is especially true if the system used does not care for exact performance values and for example only needs to know, if a setting has greater or lesser impact on fps than another (Walton 2022).

3.3 Subjective values.

Now, one might wonder why the graphical quality needs to be considered in a system used to optimize performance. While the desired frame rate could be obtained by simply lowering the settings based on their impact on frame rate, in practise it tends to result in worse graphical quality, than what is necessary.

As Christopher P Jones writes “A painting might be “beautiful” to one person and “ugly” to another, but the material object remains unchanged” (Jones 2019) so too can the graphics of a game be beautiful to one person and ugly to another. Art is subjective and whether video games can be considered as art or not, does not change the fact, that different people have different opinions, when it comes to the various graphical settings and the best values for them. What this means for this work is, that there are no practical automated tests or processes, that can be made to assign values for graphical quality and a human is needed for that. (Jones 2019)

When going through the various guides for optimizing game settings that are found on the internet, it quickly becomes apparent that many of them prefer to reduce multiple settings a bit over a large reduction on a single setting. This is because (as some of the guides explain) the impact a setting has on frame rate might be in disproportion to its impact on the graphics, with some settings having barely any visual impact yet huge effect on frame rate and others doing it in reverse. What this ultimately means, is that if the optimization system is going to be able work similarly to human perception, it needs to have some way of taking the graphics quality into account.

As a human is needed to determine the graphics quality, a human is needed to tell the system the impact a graphics setting has on graphics quality. One way this could be done is by having someone add a value to each setting that represents its impact on the graphics. Who this person is, is a matter of debate due to the subjective nature of graphics, but perhaps a developer could add the initial values based on how they intent the game to look but allow them to be changed later by the players.

3.4 Selecting the settings to change.

For the system, to properly select the settings to change, it needs the ability to consider both their impact on the performance and graphics quality. How this is done is dependent at the method, that is used to select the settings.

Regardless of the method however, it is important to decide where the optimization is started. Starting from the higher values, allows the settings to be lowered steadily, until the desired level of performance is achieved. Starting from the lower values requires finding the point, where the game runs poorly and then backtracking, making it likely less optimal.

It should also be considered if a setting should be touched at all if it has a very low or no impact on the performance. A good example is the “motion blur” setting, which many guides recommend turning off, yet the reason for it is rarely with its impact on performance, but instead on how it makes the game look. Ultimately ignoring the settings, that have no impact on the performance (in practise) should simplify and speed up the process of selecting the settings, that matter. Lastly, as someone might consider, it critical that certain setting has a specific value, giving the players the ability to exclude any setting they want from the selection process, should be considered.

Below are several methods, that might be used to select the settings to change. In cases, where a selected setting has multiple possible values, the method will lower them in steps, repeating the selection process between each step. The following methods should be considered untested. While this work seeks remedy that for some of the methods, there simply isn’t enough time to do so for all of them. Lastly it is possible and even likely, that there are selection methods that this work misses.

Selecting between presets

Instead of selecting the changed settings individually, the system could instead select between pre-set lists. In this method, the developer creates multiple lists, that each contain pre-set values for multiple settings. Using the pre-set lists, the system could for example start from the highest quality and most demanding list and then switch to lower quality lists in order until the desired performance level is achieved. For the system to work properly, it needs to know the order it should select the lists. This could be done either by simply placing the setting lists into an ordered list of their own, that is then given to the system. Alternatively, each list could have a separate value, that tells the system the order, it should pick them.

The advantages of this system are, that it allows the settings to be modified in a more controlled way as whole, compared to selecting them individually. In theory, this method allows the developer to select best possible values for different levels of performance, resulting in the best possible quality. Another advantage is, that this method has is that it is likely faster to go through a list of pre-sets, than it is to do so with the settings separately.

The disadvantages are, that doing multiple carefully optimized lists require extra work from the developer and to get the best possible lists the developer needs to really know what they are doing. Add in the subjective nature of graphics combined with potential performance issues raising from different hardware and the results might be worse than expected. The fact, that it simply isn’t possible to add a pre-set for every machine, might also result in some player having to use worse settings, than they need to. Lastly using the pre-set lists makes it harder to make use of player run testing to make the lists more accurate.

It should be noted that many of the games these days already provide the ability to manually select pre-sets to modify multiple settings simultaneously and this system could potentially be modified to also be used by the automated systems.

While giving the player the ability to make their own pre-sets would solve a lot of the problems, it defeats the purpose of using the system in the first place. A potential way to make use of player made pre-sets, is to allow them to be shared on the internet. Depending on how the sharing is handled, could theoretically allow the less knowledgeable players to find the best possible pre-sets for them. How much advantage this would have, compared to just using guides, is another question entirely.

Selecting the settings based on priority

One way to handle the selection process is to give each setting a priority value that determines the order they are selected. In this method the system starts with all the settings set to high values and then starts lowering them in order based of their priority. The value should be based on both their impact on quality and performance, with higher drops in quality lowering it and higher gains in performance raising it.

Ideally, the value would not be a static one, that is directly set by the developer, but instead consist off separate values representing both the quality and performance. By using separate values, it becomes simpler to use performance tests to automatically set the performance related priority values. Using the separated values also makes it easier to see what each priority values is based off. When the setting order is being selected, the values can simply be added together to get the final priority value.

The advantage of this method is, that it does not require the developer to make up multiple lists of settings and in theory will result in optimal performance and quality at any machine. It also allows the game to run performance tests on the player machine to improve the accuracy.

The disadvantage of the method is that it requires, that each setting has been set with their own priority values. Because each setting is changed separately, the system needs to test the performance between each change adding, quite a bit off extra time to process. To fully take advantage of the method, a user-

friendly performance testing system needs to be included with the game. Lastly, to fully optimize the results, requires that each separate value of a setting has their own priority. This is because some settings don’t operate linearly with, for example a setting might have low impact on quality and a high impact on performance at the highest values and the reverse on lower ones.

Miscellaneous

A potential way to speed up the testing is not to do it between each change, but instead make multiple changes at once before each test. In this method, the system first needs to calculate how many frames it is missing, additionally each setting needs a value, that tells how many extra frames they are expected to give. Once the system starts the selection process, it adds the values of the selected settings together until they are equal or exceed the missing frames and then tests the changes in practice. The problem of the method is that it relies that the frame values tied to each setting are accurate or it will not produce optimal results. The fact that many game guides tend to give potential frame gains or losses in way that leaves room for some variance (etc. the setting gives between 3 – 5 frames of performance), does raise some questions whether the method is worth using.

A potentially more effective way to speed up the selection process, would be to combine the above methods. First, the system uses the pre-set based method to select a pre-set, that is one step above the one, that gives the required performance. After picking the pre-set, it starts to use the priority-based method to get the required performance in a more optimal way. The problem of this method is that it requires that both methods are implemented, increasing the workload. It should be noted that the method could allow less optimized pre-sets to be used, as it handles the more precise optimization on its own.

When creating the tests for the project, the settings that will be considered and tested will be v-synch, resolution, texture quality, antialiasing method, antialiasing quality, shadow quality and shadow distance.

4 Test project APOT

The purpose of this section is not to give step to step instructions on how to implement an automatic performance optimization system.

The name of the test project is APOT (from the fords Automatic Performance Optimization Test)

4.1 Test Project Goals

There are two primary goals, when it comes to the project. The first goal is to transfer the theoretical ideas discussed in the previous chapter into a form, that shows, whether they work in practice. The second goal is to build a platform, that allows the actual testing of the ideas built during the first goal. To help make the explanations clearer at the later parts of the thesis, the first goal will be referred as the “performance optimization system” and the second goal as the “test environment”.

The requirements for the performance optimization system are as follows: a way to track the performance and test whether it is at acceptable levels, a way to let the system know of available settings, a way to determine the best order to change the settings and the amount they should be changed and lastly a way to automatically change the settings.

The requirement for the test environment are as follows: a test environments with sufficient amount of objects and graphical detail to, but some actual strain on the system, some way to move on the test environment in a way that simulates actual gameplay, a list of settings that can be changed within the test program, when it is running, a way to run the program with the performance optimization mode disabled to get a base performance data and lastly, a way to record the test results data for later analysis.

4.2 Tools and assets

Unity

Unity is game engine, developed by Unity Technologies, that allows the development of games and other programs. The engine comes with a large number of tools and assets, that are commonly used in the development of games and other programs leaving the developers free to focus on the development of the more unique features of their product. (Schardon 2023)

The main development tool and game engine for the project is Unity version 1.1. The reason for using Unity is mostly because of my familiarity with it and the reason for the chosen version is, because the client of the thesis (Turku Game Lab) recommended it.

Visual Studio

Visual Studio is an integrated development environment, developed by Microsoft, that can be used to develop edit and build code. (Microsoft n.d.)

The code needed by the project will be written with the Visual Studio 2019. My reason for picking it, is because of it being familiar to me and Unity's inbuild compatibility with it. The reason for using the version 2019 is simply because, that is the version that I have available and there isn't any actual need for another version. It should be noted, that while it does provide a lot of coding related features, in this project it is essentially only used as a text editor for writing code to Unity.

C#

C# (pronounced "See Sharp") is an object-oriented programming language, developed by Microsoft, that has its roots in the C family of languages. (Microsoft 2023)

C# is used as the main programming language for the project. This is because it is the main development language used in Unity and there is no specific need to use any other language. (Unity Technologies n.d.)

Unity Terrain - URP Demo Scene

Unity Terrain – URP Demo Scene is a free asset package, created by Unity Technologies, that contains various graphical assets and a scene, that uses them. (Unity Technologies n.d.)

Unity Terrain - URP Demo Scene is included in the project to provide a ready to use graphics filled scene for testing the actual performance optimization system. It is used due to the lack of time and skill needed to create a custom environment and assets, that are sufficiently detailed to be of use in testing.

4.3 Writing the project

When starting to make the project, I initially started with the test environment, as otherwise there simply would not have been any way to tell, whether the other systems were working correctly.

I started by writing the movement system, using a simple scene containing, a small platform for testing. Initially, the movement system was a simple first-person view, where the user controlled the camera, using the keyboard and mouse. Later, I enabled collisions and gravity to the player, using Unity's CharacterController class. While I could have used a Rigidbody to enable physics, that would have increased the complexity of the system, which would have run the risk for additional bugs and I felt, that ultimately full physics system was not needed to simulate standard gameplay. I finished the movement system by adding the ability to jump.

After finishing with the movement system, I added a simple system for automatically moving objects between set point. Admittedly my original plans for the system were pretty vague, with some ideas to use it as an alternative to the user input-based movement system to provide more consistent test results and other ideas to use it to move non player objects to simulate gameplay more accurately. Initially, I did not spend much time with system, making only a simple version, that only allowed the most basic automated movement between several points, before moving to other tasks. My initial reason for not spending much time with the system was because I wasn't sure how necessary it was. Later, after realising actual need for more consistent movement during repeated tests, I would return to the automated movement system and develop it further.

After finishing the movement system (for now), I would focus on creating the graphical environment for the tests. As I neither had the time nor the skill set required for creating such an environment from the scratch I searched for a suitable environment from the Unity store, eventually settling with the "Unity Terrain - URP Demo Scene" (as mentioned in the "Tools and assets" chapter), which came with a suitable scene. While I did need to modify the provided scene to use my custom movement scripts, the package otherwise worked without issues.

After getting the test environment to a workable state, I finally started creating the performance optimization system. I started with the system used to track performance, as all the other systems handling the performance optimization would have had at least some need for it.

First, I got the system to track the frame rate by using the SystemInfo class that came with the Unity Editor. Originally, I intended to also track the memory use and other data related to program performance, but later I abandoned that plan partly due to the SystemInfo class not providing the necessary information and partly because said information ended up not being needed by the optimization systems.

As I wanted a way to see the tracked information in program, I added a simple HUD system to display it. At this point I also added the ability to pause the program and had the HUD show the available user controls.

Before starting the system used for the automated optimization, I added a main menu to the program. The reason I added the main menu was because I needed a place to switch between the performance optimization and normal gameplay and change the graphics settings manually. In addition to the main menu, I also added a simple pause menu to the test scene.

While the point of the project was to set graphics settings automatically, I had three separate reasons for having a place to change them manually in the project. Firstly, I wanted a way to test different graphics settings outside the automated systems. Secondly, I figured that having a menu for changing graphics settings would have more realistically simulated a real use case. Lastly, making the menu served as a way to choose the settings that the automated systems would handle, and later to easily see the changes that were made to them. Lastly, to make the system more closely mimic modern games, I finished the graphics menu by adding the option to change the settings by using a pre-set list of them.

My next task was to add a way for the system to select the settings and the order, in which it would change them. Before doing this, I needed to decide, which of the methods, discussed in “Potential Approaches” chapter of this work, I would try to implement. While ideally, I would have implemented and tested all of them, in practice I figured it would be unlikely, that I would have the time to do so. So, to ensure that I could test at least one of them, I decided to implement the “Selecting the settings based on priority” method first. I did also plan to implement other methods later if I had the time.

After selecting the method, I implemented a system, that placed each available settings into a list and then sorted it based on setting priority. Afterwards, during the optimization process, the system would select the first valid setting in the list and try to change it. I also had to tell the system. how much to change the

setting value by storing the change amount with the priority values. Of course, before I was able to test the automated setting optimization, I needed a way to test for bad performance.

To test for bad performance, I implemented a test mode where the player would move on a pre-set path (to ensure consistency) using the previously mentioned automated movement system while tracking the frame rate. After finishing the path, the system would figure the performance based on the final frame rate values. At this point, I also added the ability for the system to write the test results to a text file for later analysis.

At this point, I also realized that as implemented, the system could only change each setting once even if the setting would not be set to its lowest value. To fix this I had to modify the system so that each time after changing a setting, it would also add a penalty to its combined impact value and resort the setting list. This way the system could continue selecting the settings until either achieving the desired performance or having all the available settings set to their lowest values.

At this point, I had finished with all the needed components and was finally able to test them fully and somehow the code worked together without major issues. Because I did not need to spend too much time on fixing bugs, I had enough time to at least try to implement the “Selecting between pre-sets” method before having to start performing actual tests.

Due to being able to reuse a lot of the previous code and “Selecting between pre-sets” method being a lot simpler to implement I was able to finish it before the deadline.

When doing more proper tests, I discovered, that the automatic movement could get stuck if the frame rate is too low. While I initially planned to fix it, I realized that similar issues could potentially happen to other systems, so a more system generic fix should be used. The fix in question was for the test to end if it’s below target frame rate time reached above a certain threshold, which would be treated as an indicator of bad performance.

4.4 Scripts

PerformanceOptimizationHandler

PerformanceOptimizationHandler script is the main handler of the automatic setting optimization system and is responsible for either handling the required logic itself or to tell other scripts to do so.

To work the script requires access the PerformanceDataTracker script, GraphicsSettings script and a AutomaticMovementHandler scripts. It uses the PerformanceDataTracker script to tell if the program is performing badly during a test. The AutomaticMovementHandler script needs to be set to handle the user (or more precisely the main camera) movement as a part of a test scene and is used to ensure consistency between multiple tests. Lastly the GraphicsSettings script is used by the handler to change the settings. When the handler is given access to the other scripts, only the handler targetFPS and delayTest values need to be set.

The script allows the start of the test be delayed which will delay both the start of the performance tracking and when the AutomaticMovementHandler script starts to move the player. The purpose of the delay is to stop the loading of the test scene from impacting the test.

The script allows the developer to select between the “Selecting between presets” and “Selecting the settings based on priority” methods. When using the “Selecting between presets” method, the developer can either use the graphics setting preset that are built in or create their own in the inspector. It should be noted that the automatic optimization system will use and test the presets in order from last to first.

When using the “Selecting the settings based on priority” method, the user either use the build in settings list or create their own. The list will then be sorted based on its impact values. In case of ties the system will attempt to solve it, based on whether the variable favorGraphics is true (prioritize the setting with

the lower graphics impact) or false (prioritize the setting with the higher performance impact).

The script also contains the static class APOTData, which is used to store and pass information between scenes or when reloading a scene.

PerformanceDataTracker

PerformanceDataTracker script is used to track the programs frame rate and to write the results into a text file for laser analysis. The script tracks both the current frame rate, the average frame rate (over a pre-set time frame), the highest and lowest frame rates during the tracking period and the time the frame rate spends above and/or under the target frame rate.

While the script itself doesn't determine whether the program is running badly or not, it does collect the data that is integral for those. It should be noted that the script does collect information, which are for the most part unnecessary (as later development of the system has proven) for the optimization systems. In practice only the testTime, timeUnderTargetFPS, timeAboveTargetFPS, averageUnderTargetFPS and averageAboveTargetFPS data is used in the actual optimization processes.

It should also be noted that while the script does contain functions to write the test results to a text file, the files themselves are only otherwise accessed by the program and as such not used by optimization processes.

The script file also contains the class PerformanceDataContainer, which is used to hold the collected data and pass it to other scripts.

PerformanceDataDisplay

PerformanceDataDisplay script allows the information, gathered by the PerformanceDataTracker script, to be displayed during gameplay. While helpful

for creating and testing the code, it isn't actually required for the automated performance optimization systems.

GraphicsSettings

GraphicsSettings script is used to hold and modify the games graphics settings. Besides invidual graphics settings, it also handles the graphic settings presets. The script file contains two other classes: SValue and SettingValues.

Svalue holds the data for a single setting, which mainly consists of the various values needed by the optimization systems to properly handle the individual setting. Additionally use the class allows settings with different data types (float, int, bool) to be placed into a single standard list, which is needed by the PerfromanceOptimizatuinHandler script for sorting purposes. It should be noted that the class is not needed, if only using the "Selecting between presets" method and a better way to allow all the settings to be placed into a single list would likely be with inheritance.

SettingValues contains values for each separate setting, that can be modified in game. It is used both by the preset system and the "Selecting between presets" method.

UserHandler

UserHandler script handles the functions affecting the player object or more precisely the game object, which has the main camera of the scene either as a component or a child object. The script uses Unity's CharacterController script to provide a collision detection and gravity systems to the player object.

The handler uses Unity's Input Manager to set a series of inputs, which allow the control of the player object position and camera rotation, jumping, changing

the movement speed and pausing the game. The developer can alter various values related to these actions like move speed and jump height.

When working together with the AutomaticMovementHandler script the handler can also allow the automatic movement system to use both the gravity and collision systems. To use the automatic movement, the handler needs access to the AutomaticMovementHandler script set to the player object and have its moveAutomatically variable set to true.

AutomaticMovementHandler

AutomaticMovementHandler script allows game objects to move automatically toward another game object, that contains an AutomaticMovementPoint script. The game object set at the currentMovePoint variable, determines the direction that the handler moves towards. For the handler to work properly, it needs the first object on its path to set in the Unity inspector.

Besides movement the handler also has the option of changing the rotation of the object towards a pre-set point. The view direction can either be the current direction it is moving towards or a separate one. Disabling this option allows other script to alter the object rotation and could for example allow the use of free camera when using the handler together with the UserHandler script.

The default movement provided by the handler is rather simple and ignores collision and gravity. If a more complex movement system is needed, the handler can still be used to provide the move and view directions, while another script handles the actual movement (as it is used with the UserHandler script).

AutomaticMovementPoint

AutomaticMovementPoint script is used to provide move and view directions to the AutomaticMovementHandler script. The script can either point the handler towards another game object or stop its movement. The script also provides

rotation directions for the handler (leaving the direction unassigned causes the handler to rotate towards the object the script is attached to).

MenuHandler

MenuHandler script is a collection of various functions used by the menu system. It is used to provide functionality to the various menu elements like the buttons and dropdown lists. While the handler is used to customize settings and start the various optimization related tests during runtime, it does not deal with the actual logic beyond calling the relevant functions from other scripts.

SceneModeHandler

SceneModelHandler script is used to change the current mode of the scene it is placed at. Changing a scene mode essentially means enabling certain game object and disabling others.

4.5 Prefabs

AutomaticMovePoint prefab

The AutomaticMovePoint prefab consists of the AutomaticMovementPoint script and a simple colored 3d sphere shape. The object is used by the AutomaticMovementHandler script to select the movement direction.

SceneModelHandler prefab

A simple prefab that uses the SceneModeHandler script to control the modes used by the TerrainDemoSceneABOTDev scene.

User prefab

User prefabs consist of a large number of game objects and scripts and is intended as an easy way to add the test and movement systems to a scene. Essentially any custom script not found in other prefabs is part of the user prefab and the prefab besides simplifying the implementation of the testing system to new scenes, serves as a guide on how to add the scripts on other gameobjects.

4.6 Scenes

MainMenu scene

MainMenu.unity scene contains the main menu of the project and is the scene, where the project first opens to, when started. The menu scene includes three tabs, that take the user to the various submenus included in the scene. The tabs are as follows: Load Scene, Performance Optimization and Graphical Settings.

Load Scene menu allows the loading of the TerrainDemoSceneABOTDev with the testing mode disabled. It is its own submenu, because my original plan was to have multiple test scenes, which failed due to lack of time and suitable free 3rd party content.

The Performance Optimization menu is used to start the performance optimization process and consist of a button to start the optimization, a input field, that is used to set the target frame rate and dropdown list used to select whether the optimization uses the “Selecting the settings based on priority” or “the Selecting between pre-sets” method.

TerrainDemoSceneABOTDev scene

TerrainDemoSceneABOTDev.unity is the essentially the main scene used for testing. The scene is essentially a copy of a scene included in the Unity Terrain – URP Demo Scene package, that has been modified for testing purposes. The original scene consists of an automatically moving camera on a environment made mostly of water, trees, grass and other plants.

The modifications to the scene include disabling the in-build camera and movement systems and adding the scripts and prefabs needed for performance testing and optimization. Furthermore, while all of the original graphical objects are left in place, some of the scene objects, like the trees, have been duplicated to increase the amount of system resources it requires.

It essentially has two modes, that switch the scene between normal gameplay mode (which allows the user to move around the scene freely in first person view) and a testing mode (which moves the user view automatically on a pre-set track). The mode the scene starts, is depends on the sceneMode value that the SceneModelHandler has, when starting the scene. Which can be set either manually on the inspector or by the MainMenu button used to enter the scene.

TestScene

TestScene.unity is currently not used for anything but was originally used for testing the movement scripts.

4.7 Other assets

URPAssets

URPAssets are a group of assets consisting of various setting values and allow shadow quality setting to be changed through code.

AutomaticMovementPointMaterial

The material included in the prefabs folder is used, by the AutomaticMovementPoint prefab, to help make it more noticeable in a graphical scene.

4.8 How does the optimization system work?

APOT (aka. Automatic Performance Optimization Test) is the name of the test project. This section seeks to briefly explain the logic of how APOT tries to optimize the settings.

Detecting the bad performance

To detect bad performance, the system runs a performance test, while keeping track of the frame rate. In the test, the player character is moved through a pre-set path and during it, the frame rate data is evaluated against the target frame rate value. During the player movement, the system keeps track of the time the frame rate stays either above or below the target value. After finishing the movement, the system compares whether the time spend below the target frame rate is above the allowed threshold percentage (which can be set by the developer to be any value between zero and a hundred), which would indicate bad performance.

The system tracks the average frame rate value at a one second interval, as the current frame rate can have brief spikes, that aren’t necessary an indicator of bad performance. Furthermore, loading a scene might cause some temporary frame rate drops at the start of the test. To avoid those drops from interfering with the test results, the movement and data tracking can be delayed.

Additionally, the system can end the test, if the time under target frame rate reaches above a pre-set threshold. If the test times out, it is considered an indicator of bad performance. This system is used due to the possibility of the

test getting stuck in especially low frame rates. By default, the time out value is set above the maximum runtime of the test.

Selecting between pre-sets method

When using the “selecting between pre-sets” method, the system starts by running a test using the currently selected setting pre-set. If bad performance is detected with the currently selected pre-set, the system starts the optimization process.

The setting pre-sets are stored in a list. The system assumes, that the pre-set list is sorted from least demanding to most demanding. The developer needs to ensure the correct pre-set order, as the system does not have the ability to automatically sort the pre-set list.

In the optimization process, the system selects a pre-set from the list, that is immediately under the current one and then runs a new performance test. If the test results in desired performance, the process ends, otherwise it continues until either the desired performance is reached, or the last available pre-set is tested.

Selecting the settings based on priority method

When using the “selecting the settings based on priority” method, the system starts by running a test using the current settings. It should be noted, that inside the code, the method is referred to as the “setting method”, for code readability reasons. If bad performance is detected, the optimization process is started.

Each setting stores optimization specific information, that is assigned by the developer. The stored information consists of values representing the impact the setting has on system performance and graphical quality, a penalty that is assigned to impact calculations and the amount it should be lowered at once.

The system creates a list from the available settings and then sorts it based on the setting specific impact values. Afterwards, the system selects the first setting in the list and lowers its value and runs a new performance test. The system will continue to select settings, until the target performance is reached, or all the settings are set to their lowest values.

Each setting has two impact values, with one representing its performance impact and another the graphical impact. When evaluating the total impact of a setting, the graphical impact value is subtracted from performance impact value. The goal with the two impact values is to ensure, that the settings with most impact to performance and least to graphics are lowered first. To ensure that the changes are spread over multiple settings, the system can assign a setting specific penalty to the settings impact value and resort the setting list. The penalty is multiplied by the number of times the setting has been previously lowered during the optimization process.

5 Summary of results

This chapter presents the conclusions on the feasibility of the automatic optimization system. The conclusions are based on the observations made regarding the test project APOT.

5.1 Testing the system

In conclusion on the feasibility of the automatic optimization system, two methods were tested: (a) a preset method; and (b) a priority method. Each method was tested separately. Each method was tested against 30 fps and 60 fps target frame rates. Each target frame rate was tested tree times using the normal trees, dense trees and extra dense trees scene settings. This resulted in a total of twelve different tests with six tests for each method.

The two separate frame rates were used ensure that the system can target multiple different performance levels. The multiple tree amounts were used to ensure that the test behaved consistently at different system loads.

To ensure similar starting points, before each test the graphical settings was set to the very high preset. After each test the application was closed to reset any potential values in it. Additionally, after each test the test results data file was copied to a different location to ensure that it was not overwritten. After finishing the tests, the resulting data files were transferred to an excel table for easier analysis. To avoid the strain from loading a test scene from affecting the results a ten second delay was used between the initial scene loading and start of the test. Lastly, to keep the test from getting stuck, they would time out after staying over ten seconds under the target frame rate. If the test did not time out, it would last approximately seven seconds.

The order of steps for each test

The following steps were performed with each test, with the only difference being the tree density (normal, dense or very dense), target frame rate (30 or 60) and optimization method (preset or setting).

1. Start the application.
2. Select the tree amount.
3. Select the Graphics menu.
4. Change the setting to the Very High preset.
5. Press the Apply Settings button.
6. Select the ABOT menu.
7. Set the target frame rate.
8. Select the optimization method.
9. Start the test.
10. Wait for the test to end.
11. Close the application.
12. Copy the "ABOT Performance Data.text" to another data location.

5.2 Test results

Preset based method, 30 fps and normal trees:

Baseline test: Preset: Very High, Test time 10.00442 seconds, Lowest frame rate: 11.98598, Highest frame rate: 20.02804.

Optimization attempt 1: Preset: High, Test time: 10.00447 seconds, Lowest frame rate: 19.89709, Highest frame rate: 20.09291.

Optimization attempt 2: Preset: Medium, Test time: 7.336454 seconds, Lowest frame rate: 29.81763, Highest frame rate: 30.18485.

Preset based method, 30 fps and dense trees:

Baseline test: Preset: Very High, Test time: 10.00461 seconds, Lowest frame rate: 11.98111, Highest frame rate: 20.0488.

Optimization attempt 1: preset: High, Test time: 10.00444 seconds, lowest frame rate: 19.89294, Highest frame rate: 20.08355.

Optimization attempt 2: Preset: Medium, Test time: 7.403235 seconds, Lowest frame rate: 29.75898, Highest frame rate: 30.14981.

Preset based method, 30 fps and extra dense trees:

Baseline test: Preset: Very High, Test time 10.00426 seconds, Lowest frame rate: 11.99197, Highest frame rate: 20.05881.

Optimization attempt 1: Preset: High, Test time: 10.00459 seconds, Lowest frame rate: 19.88767, Highest frame rate: 20.07746.

Optimization attempt 2: Preset: Medium, Test time: 7.403329 seconds, Lowest frame rate: 29.76067, Highest frame rate: 30.21376.

Preset based method, 60 fps and normal trees:

Baseline test: Preset: Very High, Test time: 10.00441 seconds, Lowest frame rate: 14.95269, Highest frame rate: 15.04354.

Optimization attempt 1: Preset: High, Test time: 10.00445 seconds, Lowest frame rate: 19.88843, Highest frame rate: 20.09953.

Optimization attempt 2: Preset: Medium, Test time: 7.369946 seconds, Lowest frame rate: 29.81755, Highest frame rate: 30.16791.

Optimization attempt 3: Preset: Low, Test time: 6.969691 seconds, Lowest frame rate: 59.54325, Highest frame rate: 60.03303.

Preset based method, 60 fps and dense trees:

Baseline test: Preset: Very High, Test time: 10.00425 seconds, Lowest frame rate: 14.93701, Highest frame rate: 15.0401.

Optimization attempt 1: Preset: High, Test time: 10.00445 seconds, Lowest frame rate: 19.92076, Highest frame rate: 20.0605.

Optimization attempt 2: Preset: Medium, Test time: 7.336515 seconds, Lowest frame rate: 29.83285, Highest frame rate: 30.162.

Optimization attempt 3: Preset: Low, Test time: 6.953128 seconds, Lowest frame rate: 59.4629, Highest frame rate: 60.03303.

Preset based method, 60 fps and extra dense trees:

Baseline test: Preset: Very High, Test time: 10.00438 seconds, Lowest frame rate: 12.00588, Highest frame rate: 19.94965.

Optimization attempt 1: Preset: High, Test time: 10.00445 seconds, Lowest frame rate: 19.8927, Highest frame rate: 20.05491.

Optimization attempt 2: Preset: Medium, Test time: 7.303286 seconds, Lowest frame rate: 29.80297, Highest frame rate: 30.18504.

Optimization attempt 3: Preset: Low, Test time: 6.936389 seconds, Lowest frame rate: 59.49076, Highest frame rate: 60.03303.

Priority based method, 30 fps and normal trees:

Baseline test: Test time: 10.00436 seconds, Lowest frame rate: 14.95618, Highest frame rate: 15.04216. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 500.

Optimization attempt 1: Test time: 10.00447 seconds, Lowest frame rate: 14.94869, Highest frame rate: 15.04644. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 2: Test time: 10.00448 seconds, Lowest frame rate: 14.95416, Highest frame rate: 15.0416. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 3: Test time: 10.0379 seconds, Lowest frame rate: 14.93527, Highest frame rate: 29.92784. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 4: Test time: 10.00441 seconds, Lowest frame rate: 14.95613, Highest frame rate: 15.03649. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100,

Priority based method, 30 fps and dense trees:

Baseline test: Test time: 10.00435 seconds, Lowest frame rate: 12.0007, Highest frame rate: 20.01052. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 500.

Optimization attempt 1: Test time: 10.00451 seconds, Lowest frame rate: 14.9564, Highest frame rate: 15.03784. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 2: Test time: 10.00459 seconds, Lowest frame rate: 14.95182, Highest frame rate: 15.03316. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 3: Test time: 10.00456 seconds, Lowest frame rate: 14.95258, Highest frame rate: 15.03777. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 4: Test time: 10.00447 seconds, Lowest frame rate: 14.952, Highest frame rate: 15.03043. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100

Priority based method, 30 fps and extra dense trees:

Baseline test: Test time: 10.0046 seconds, Lowest frame rate: 11.9716, Highest frame rate: 20.00492. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 500.

Optimization attempt 1: Test time: 10.00444 seconds, Lowest frame rate: 14.95969, Highest frame rate: 15.02858. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 2: Test time: 10.00455 seconds, Lowest frame rate: 14.94581, Highest frame rate: 15.03405. Settings: Fullscreen: False, V-Synch:

4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 3: Test time: 10.00448 seconds, Lowest frame rate: 14.95278, Highest frame rate: 15.03669. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 4: Test time: 10.00438 seconds, Lowest frame rate: 14.9444, Highest frame rate: 15.03671. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Priority based method, 60 fps and normal trees:

Baseline test: Test time: 10.02135 seconds, Lowest frame rate: 11.98746, Highest frame rate: 15.04864. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 500.

Optimization attempt 1: Test time: 10.00461 seconds, Lowest frame rate: 14.93857, Highest frame rate: 15.04422. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 2: Test time: 10.00448 seconds, Lowest frame rate: 14.9332, Highest frame rate: 15.05129. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 3: Test time: 10.00447 seconds, Lowest frame rate: 14.95294, Highest frame rate: 15.03741. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 4: Test time: 10.00459 seconds, Lowest frame rate: 14.94491, Highest frame rate: 15.03312. Settings: Fullscreen: False, V-Synch: 1, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 5: Test time: 7.03648 seconds, Lowest frame rate: 59.56589, Highest frame rate: 60.03303. Settings: Fullscreen: False, V-Synch: 1, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Priority based method, 60 fps and dense trees:

Baseline test: Test time: 10.02114 seconds, Lowest frame rate: 11.99023, Highest frame rate: 15.04375. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 500.

Optimization attempt 1: Test time: 10.00448 seconds, Lowest frame rate: 14.95253, Highest frame rate: 15.0345. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 2: Test time: 10.0046 seconds, Lowest frame rate: 14.94567, Highest frame rate: 15.0302. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 3: Test time: 10.00438 seconds, Lowest frame rate: 14.9551, Highest frame rate: 15.03646. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 4: Test time: 10.00442 seconds, Lowest frame rate: 14.93717, Highest frame rate: 15.04551. Settings: Fullscreen: False, V-Synch:

4, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 5: Test time: 7.019815 seconds, Lowest frame rate: 59.44278, Highest frame rate: 60.03303. Settings: Fullscreen: False, V-Synch: 1, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Priority based method, 60 fps and extra dense trees:

Baseline test: Test time: 10.02114 seconds, Lowest frame rate: 12.01187, Highest frame rate: 15.03325. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 500.

Optimization attempt 1: Test time: 10.00432 seconds, Lowest frame rate: 14.94268, Highest frame rate: 15.03077. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 2: Test time: 10.05439 seconds, Lowest frame rate: 14.94672, Highest frame rate: 20.00216. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 3, Shadow distance 100.

Optimization attempt 3: Test time: 10.00442 seconds, Lowest frame rate: 14.95797, Highest frame rate: 15.04431. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 2, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 4: Test time: 10.00439 seconds, Lowest frame rate: 14.95226, Highest frame rate: 15.03664. Settings: Fullscreen: False, V-Synch: 4, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

Optimization attempt 5: Test time: 7.003181 seconds, Lowest frame rate: 59.56883, Highest frame rate: 60.03303. Settings: Fullscreen: False, V-Synch: 1, Resolution: 4, Texture quality: 3, Antialiazing method: 0, Antialiazing quality: 2, Shadow quality: 0, Shadow distance 100.

5.3 Analyzing the test results

It should be noted that any test lasting more than ten seconds means that it was getting stuck because of low frame rate and was timed out.

The preset based method results

The preset based method tests gave the following data:

The results for the preset based method test show, that it behaves as intended, at least for the most part. In each test, it moves through the preset in correct order and stops, when getting to the target frame rate. At no point, during the six different tests, was the lowest preset (which has the name of “Very Low”) reached. Overall, the method seems to work consistently regardless of the settings it is used with.

The priority-based method results

The test results for the priority-based method on the other hand are a mess. Like the previous method, each test ran through the same changes, in the same order. It even managed to reach the desired performance when targeting 60 frames per second.

Unfortunately, the previously mentioned parts are almost the only things that went as intended. When targeting 30 frames per second, at no point did the tests reach the target frame rate, yet they also did not reach the lowest possible setting values either. After the baseline test, there seemed to be two optimization attempts with identical test values. The system should change

settings only at one step at a time (or 50 points in the chase of Shadow Distance setting), yet many of the settings have been reduced by multiple steps.

While the system does have it problems, it did change the settings in their intended order. While there are two identical optimization attempts at the start, both seem to affect the shadow distance setting and based on the impact values, that setting indeed should be picked twice. The rest of the settings are also selected in the correct order, based on their impact values.

5.4 Ease of implementation

How easy an automatic optimization system is to implement, seems to depend most heavily on the chosen optimization method.

Of the two chosen methods the “Selecting the settings based on priority” method proved to be the most complicated to implement. Things adding to the complexity of implementing the system included the need to link additional data to each setting (priority values, change amount, etch.), the need to collect all the setting (with different data types) to a single list, sorting the list based on priority values system, determining the priority values and so on. While none of the individual system parts are particularly complex separately, adding them all together resulted in a system that was a lot more complicated and time consuming to develop than what I expected.

The “Selecting between presets” method seemed to be a lot simpler to implement. While some of the apparent simplicity came from the ability to make use of the systems implemented for the previous method, there were several factors unrelated to them. First, there was no need to link priority values for individual settings. Second, the system for selecting the presets manually only required light modifications to work with the automated system. Thirdly, as each preset was effectively its own list of changes, the system could simply move through the list in order, without needing make its own changes. Lastly, the system did not use priorities to sort the preset list and instead relied on that it was built in the correct order. All together, these changes meant that the

“Selecting between presets” method was a lot simpler to implement compared to the previous method.

Further difficulty, that affected the implementation of both methods, was the fact how unity handles settings. It was surprisingly difficult to find a way to change certain settings through code. Making the testing more complicated was the fact that changes to certain settings did not take effect, when running the program in editor.

It should be noted that the optimization system was implemented (at least partially) simultaneously with the test scenarios. This, in turn might skew with the observations on how easy or difficult it is to implement the system.

5.5 Feasibility of the system

Of the two tested systems the most feasible seems to be the preset based method. The method gave the intended test results and was relatively simple to implement. Something of note is, that the ability to select a preset manually seems to be quite a common feature in modern games. Such a system should be quite easy to adapt to be usable with an automated system. The only thing it would really need to have added, is a way to test for preset performance. With proper effort it could be possible to implement presets that works with most systems.

While the priority-based system had a lot of issues when it came to test results, it could still be feasible. After all it did change the correct settings and the issues it does have, could be explained as programming errors. Ultimately, the biggest issue with the method is, how complicated it is, to implement. This complexity makes it possible for a lot of errors to slip by and in general makes it harder to implement the system. Still, with proper implementation, the method could be used to obtain more optimal settings compared to the preset based one. Afterall, there does come a point, were trying to add a preset for every possible setting combination, stop’s being practical.

Lastly while the test environment isn’t necessarily indicative of a real use case, the system should allow the average user to obtain a functional frame rate without ever changing a single setting.

5.6 Further development

The system, as implemented in the test project, should be considered a prototype of an automated setting optimization system. There are several areas, where the system could be improved.

First, the system could use some simplification, a lot of the settings used in the system were added either for testing purposes or because I lacked the knowledge to recognize them as unnecessary ahead of time. As the project got further in development and I got a clearer picture of what works, a lot of the previously implemented settings proved to be unnecessary. The use of multiple setting optimization methods, for example, is likely to be unnecessary. The two separate impact values could also be replaced with a single priority value.

Secondly, getting rid of the dedicated tests could make the system simpler to use from the user perspective. Instead of having to run performance tests from a separate menu, the system could instead monitor the performance during gameplay. When detecting poor performance, it could then ask the user, if he wanted to try to automatically optimize the settings. When optimizing the settings, the system should do it during normal gameplay if possible. Doing the optimization during actual gameplay, could possibly even produce more accurate results, compared to using dedicated tests scenarios. The system should only ask the user to optimize the settings once to avoid annoying them. It should however also give them the option to start it manually later.

Thirdly, it could be helpful to make the optimization system more generic and easier to add to other project without major changes. While a fully system generic version of the system might not be possible, a system usable in all the projects, made by a single developer or company, could be a possibility. Even if

project specific changes to the code are needed, the use of things like C# virtual methods could make implementation easier.

Lastly there is a possible modification to the “selecting the settings based on priority” method, that could be used to lower the settings, which I did actually try to add to the test project but ran out of time. In this method, a list created by the developer, is used instead of the one made by the system. The list needs to be filled with multiple iterations of each setting, which represent the different values they could have (etch. the shadow quality setting needs to have iterations representing its none, low, medium, and high values). When running the optimization process, the system will sort the list once and move through it in order. When changing a setting, it will assign the values stored in the currently selected list index and continue to do so, until it either reaches the target frame rate or the end of the list.

The first advantage of this modification is, that it gives the developer more control of the settings and their values. Second advantage is, that each value of given setting can have different impact values. Finally, the method removes the need for the penalty and change values in addition to the requirement to sort the setting list after each change. The downside of the modification is, that it increases the workload of the developer.

6 Conclusion

So, is it possible to create a system to automatically optimize the settings of a video game for performance? As a second goal would such a system be easy to use for the average user?

During this thesis several different methods were theorized that could have the capability to handle the tasks in theory and then a program was built to test them in practice. While the system had its issues, it was able to not only recognize when it was running badly, but also able to change the settings in a predictable and intended manner. It even fulfills the easy-to-use goal, by essentially only requiring the user to press the button to start the optimization process and the handling everything else automatically.

So, in conclusion, based on results obtained from the test system, the answer to both questions is yes.

References

Anderson, J. B. 2022. Quora: keyword: What settings affect FPS the most?. Referenced on 5.2. 2023. https://www.quora.com/What-settings-affect-FPS-the-most

Archer, J. 2022. Elden Ring PC performance and best settings guide. Referenced on 11.11.2022. https://www.rockpapershotgun.com/elden-ring-pc-performance-and-best-settings-guide

Brookes, T. 2021. What Is Dynamic Resolution Scaling (DRS)?. Referenced on 3.3.2023. https://www.howtogeek.com/764408/what-is-dynamic-resolution-scaling-drs/

Jones, C. P. 2019. Subjectivity and Objectivity in Art. Referenced on 17.3.2023. https://christopherpjones.medium.com/subjectivity-and-objectivity-in-art-cc41d55c76a5

Kalbere, A. 2022. Understanding graphics settings for better frame rates. Referenced on 9.2.2023. https://www.reliancedigital.in/solutionbox/understanding-graphics-settings-for-better-frame-rates/

Kozlowski, S. 2022. Gaming Graphics & Optimization | What Settings Affect Performance The Most?. Referenced on 7.2. 2023. https://www.wepc.com/tips/what-game-graphics-affect-performance/

Micsosoft, n.d. GitHub Copilot and Visual Studio 2022. Referenced on 19.5.2023. https://visualstudio.microsoft.com/

Microsoft, 2023. A tour of the C# language. Referenced on 19.5.2023. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Roach, J. 2022. The best settings for Elden Ring: PC performance guide. Referenced on 11.11.2022. https://www.digitaltrends.com/computing/best-settings-for-elden-ring-pc-benchmarks-performance/

Roach, J. 2022. 5 graphics settings you need to change in every PC game. Referenced on 10.2.2023. https://www.digitaltrends.com/computing/graphics-settings-change-every-pc-game-performance-frame-rate/

Schardon, L. 2023. What is Unity? – A Guide for One of the Top Game Engines. Referenced on 15.5.2023. https://gamedevacademy.org/what-is-unity/

Syed, A. PC Graphics Settings Explained: MSAA vs FXAA vs SMAA vs TAA. Referenced on 20.2.2023. https://www.hardwaretimes.com/pc-graphics-settings-explained-msaa-vs-fxaa-vs-smaa-vs-taa/

Unity Technologies, n.d. Coding in C# in Unity for beginners. Referenced on 19.5.2023. https://unity.com/how-to/learning-c-sharp-unity-beginners

Unity Technologies, n.d. Unity Terrain - HDRP Demo Scene. Referenced on 19.5.2023. https://assetstore.unity.com/packages/3d/environments/unity-terrain-hdrp-demo-scene-213198

Vries, J. n.d. HDR. Referenced on 1.3.2023. https://learnopengl.com/Advanced-Lighting/HDR

Walton, J. 2022. How to Make a Good Built-in Game Benchmark. Referenced on 10.3.2023. https://www.tomshardware.com/news/what-makes-a-good-game-benchmark