Towards the Adoption of Attribute Scaling in Cloud-based Game Streaming

A Major Qualifying Project Report

By:

Marek Garbaczonek
Jonathan Hsu
Mark Renzi

Project Advisor:	Graduate Mentor:
Mark Claypool	Xiaokun Xu

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on the web without editorial or peer review.

Abstract

Cloud-based game streaming allows end users to play games that they would otherwise not be able to run due to hardware limitations. This technology relies heavily on network speed and adds unwanted latency between the client and the streaming host, which detracts from the gameplay experience and adds unintended difficulty. Attribute scaling is a latency compensation technique that can aid in mitigating the added difficulty that players face as a result of their network limitations and the decrease in the quality of their gameplay experience. This project explores how attribute scaling can be implemented in a plugin for a commonly used game engine for developers to utilize, as well as the impact of attribute scaling on player performance in a First-Person Shooter (FPS) game environment. We conducted user studies to understand both of these areas, with users tasked with playing an aim-training FPS game under a variety of network conditions and game scenarios, and developers tasked with utilizing the plugin in a provided game development project. Through these studies, we show that there is a positive impact on player accuracy and gameplay experience in-game and that it is possible for game developers to use this latency compensation technique without prior knowledge in the field.

Table of Contents

Table of Figures

1) Introduction

In cloud-based game streaming, games are streamed from a game streaming host to a player's thin device. In return, game inputs are sent from the client back to the server to allow the player to control game elements. The architecture as a whole has the disadvantage of added latency between the thin device (client) and the game host (server), detracting from players' quality of experience (QoE) when playing games.

Latency can result in additional difficulties for the player, such as the inability to aim well in a first-person shooter (FPS) game [1]. Attribute scaling seeks to alleviate the additional difficulty introduced into cloud-streamed games that come from having to act upon the game state when the player is viewing a delayed video stream of a previous game state. Attribute scaling can scale any part of a game, including but not limited to, physical object scales and value scales. Changing the scale of values allows aspects of gameplay to be more lenient, such as time-based and accuracy-based gameplay tasks.

Little large-scale research has been done into the effects of attribute scaling, but it has thus far demonstrated a positive impact on the player QoE overall [2]. The effects of attribute scaling must be moderated to maintain the original difficulty intended by the game designer [3], but there are still many ways to apply the technique, such as a linear relationship between attributes and latency, or applying the scaling to temporal or spatial components of the game. This study seeks to explore the relationship between attribute scaling and the overall quality of experience during aim-heavy gameplay, make attribute scaling more accessible to game developers, and explore the novel applications of these techniques with a game still in development.

To explore the effects of attribute scaling on player performance and determine whether this latency compensation technique can be made more generally accessible for game developers to incorporate into their projects, we focused on two research questions. The first involves a user study assessing if attribute scaling can help mitigate the negative effects of latency in cloud-based game streaming for FPS games, while the second involves a determines whether this latency compensation method can be made generally usable by developers. Our method for the first question is to build an FPS aim-training game and conduct a user study that gathers data such as player accuracy and their perceived quality of experience. This allows us to determine the effect of attribute scaling in an FPS game environment. Our method for the second question is to build an attribute scaling plugin for Unreal Engine, along with a parkour game project in the same engine. This allows us to gather user feedback from game designers on how they utilize the plugin and determine whether attribute scaling can be made available for general use by developers through the plugin.

Through these studies, we found that attribute scaling is a viable method to mitigate the negative impacts of latency on gameplay difficulty and experience when players use cloud-based game streaming to play games with FPS aiming elements. In addition, we found that attribute scaling can be effectively packaged as a game engine plugin for game developers to utilize in their projects, without the need for the game designer to understand and implement this latency compensation technique independently.

Chapter 2 provides background information on cloud-based game streaming and the technologies we used to develop our plugin and self-developed games. Chapter 3 describes the research questions we explore through our project and a overview of the methodology and design of the attribute scaling plugin, and the user studies conducted. Chapter 4 details the user study conducted to determine whether attribute scaling positively impacts player accuracy and quality of experience when implemented in an FPS shooter game. Chapter 5 details the implementation of the attribute scaling plugin and the parkour game that was presented to game developers for them to test the plugin in a game development engine. Chapter 6 summarizes the key findings of the studies conducted. Chapter 7 discusses future work that could be explored regarding attribute scaling as a latency compensation technique.

2) Background

This chapter provides an overview of the cloud-based software tools used throughout this study and the latency compensation technique of attribute scaling. This chapter describes the Sunshine streaming host and Moonlight GameStream client, the attribute scaling latency compensation technique, and the implementation of attribute scaling in game development.

2.1 Cloud-Based Game Streaming

Network latency is introduced when data is sent from a game server to a client, and vice versa. This latency impacts players by introducing a delay between inputs and feedback from the game and other artifacts such as the desynchronization of game variables that must be accounted for. All of these factors can detract from a player's QoE.

In cloud-based game streaming, this latency presents itself as a delay between player actions and the corresponding visual response from the game server, as player inputs are not processed on the client machine, but are instead sent back to the server. There is a delay between each action the player takes and its corresponding feedback in the video streamed from the server to the client machine. This can create a sense of "lag" for the player, where the gameplay feels disconnected from the player's actions, as they are not being reflected in the game state until a later point in time. In addition, the gameplay difficulty can seem artificially high to players when there is high latency between the server and the client. As the player is acting upon the video stream as it comes in, they are executing actions based on the visual feedback that is available to them. This visual feedback is delayed by the data transfer time between the server and client, as the game state on the server has progressed further as the video stream is sent and decoded by the client. This delayed visual feedback unintentionally causes the player to act upon feedback from older game states, as the video feed they are viewing is delayed from the actual game state on the server. This can cause over-correction in spatial-related attributes of the game, such as aiming or moving.

Prior research into latency compensation techniques has primarily been focused on networked multiplayer games, where the developers have approached the project knowing that network latency would impact gameplay. In a taxonomy of latency compensation techniques, few have focused on addressing the latency that is introduced when players must play games through a cloud-streamed video feed. Out of "85 peer-reviewed publications that [were] surveyed and taxonomized", fewer than 10 mention attribute scaling as a means to compensate for latency, and even fewer explore the subject in the context of cloud-based game streaming [2].

2.2 Sunshine

Sunshine is a cloud-gaming stream host primarily used in conjunction with Moonlight and is an open-source derivative of the NVIDIA GameStream protocol. It allows users to stream desktop applications to a remote client, from which user inputs are sent back to the Sunshine host. This is achieved by capturing and encoding video output from the host and streaming it to a receiving game-streaming client. Figure 1 shows the interface used to select an application to stream.

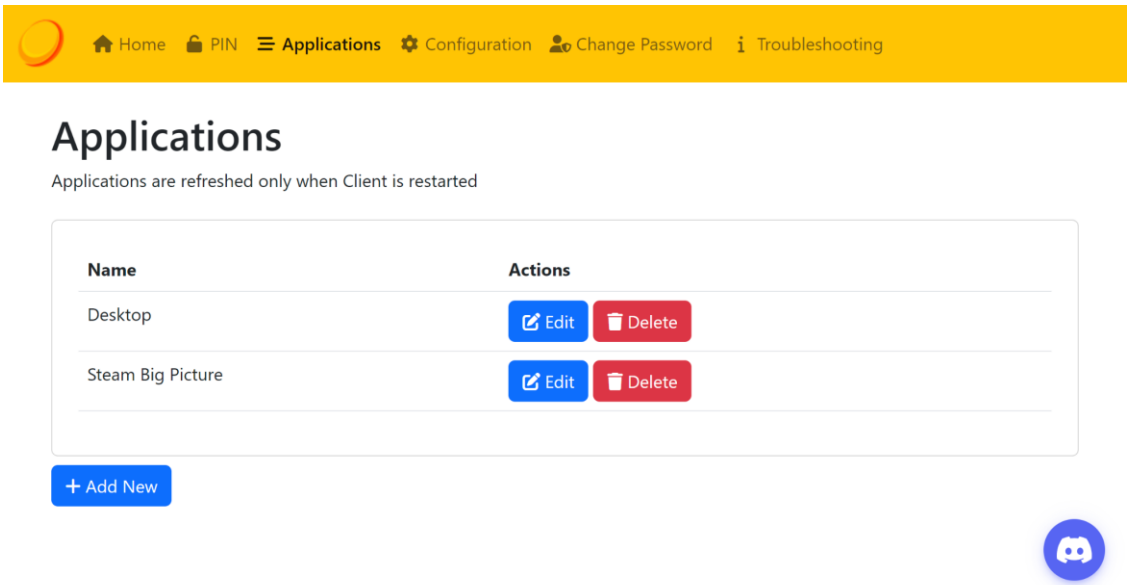

Figure 1: Sunshine Web User Interface

In this study, we used and modified Sunshine to host a stream of our attribute scaling implementation in a video game, as well as using it to expose latency data such as round-trip time (RTT) to be used in the aforementioned attribute scaling plugin. There are no pre-existing latency compensation techniques incorporated into Sunshine, providing us with a control variable for the effectiveness of our implementation of attribute scaling.

2.3 Moonlight

Moonlight is an open-source, standalone client that implements NVIDIA's GameStream protocol. It allows users to connect to a GameStream-compatible host, and to view and play games remotely. User inputs are sent back to the host, allowing remote control of desktop applications and video games. Figure 2 shows the Moonlight interface to control the connection to Sunshine.

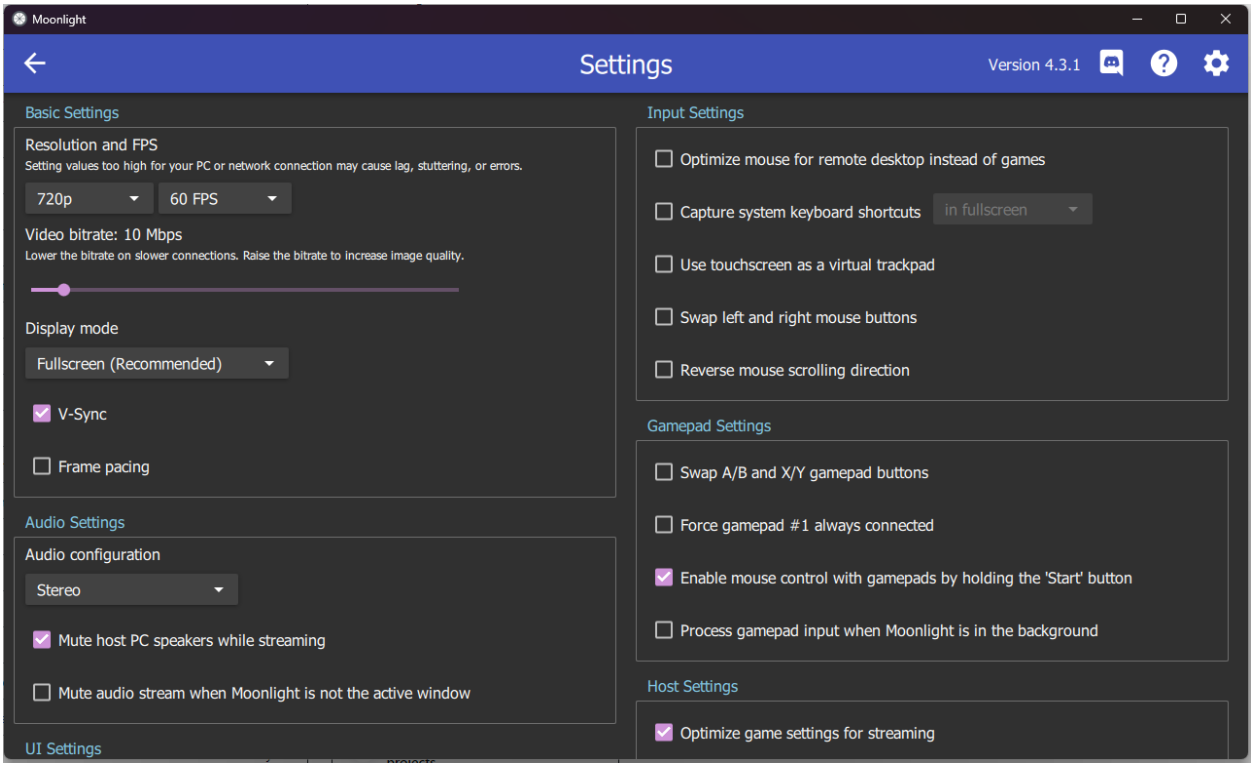

Figure 2: Moonlight Client Interface

2.4 Attribute Scaling

In the study, we utilize attribute scaling as a latency compensation technique for cloud-based game streaming. Attributes such as hitbox dimensions, player and enemy health, and movement speed can be scaled up or down to accommodate variances in network conditions. In this study, we utilize a plugin-based approach in Unreal Engine to allow game developers to apply attribute scaling to game elements with cloud-streaming in mind.

Using latency measurements, the plugin can scale the physical size of objects in the game or the value of constants defined in any code present in the game according to scaling parameters set by the game developer. These parameters include the function that determines the scale factor which attributes will change according to and the conditional smoothing of latency values.

There exist two types of attribute scaling: temporal and spatial. Temporal attribute scaling changes the required time frame surrounding a certain action or status. For example, it could adjust the length of the window during which an action can be made. This could be the time window in which a button can be pressed in a rhythm game, or how much time a player has to jump from a moving platform. Spatial attribute scaling adjusts physical bounds such as hitbox sizes and physical parameters such as health. In a game, this could include the hitbox of an enemy target or the number of hits it takes to destroy a wall.

Past work in the utilization of attribute scaling has included research showing the possible benefits of attribute scaling. In a study conducted in 2022, there was an implementation of attribute scaling developed on the Google Stadia game streaming framework [4]. This project developed two games, a first-person capture-the-flag shooter game and a target-selection rhythm game. In the shooter game, the acceptable hitboxes of enemies were increased proportionally to latency. In the rhythm game, the timing window the player was given to act was increased proportionally to latency as well. There were user studies conducted to analyze the performance of users with and without attribute scaling enabled to verify the effectiveness of this as a latency compensation technique. The QoE of the end user trended upward when this technique was applied in the presence of latency.

2.5 Game Development

Plugins can accelerate game development time and make implementation of complex tasks simple. Most plugins on the market for developers are plug-and-play: once added, the plugin automatically makes desired changes or gives the developer simple options that they can use to modify the plugin's changes in their project. This allows the developers to focus on other aspects of their game, and design the game with the tools they are accustomed to.

Currently, there are no publicly available tools that help specifically with latency compensation for cloud game streaming. Modern cloud game-streaming typically involves already-developed games being hosted on a server, with no modifications made to account for the additional latency introduced by game-streaming. This results in a negative impact on QoE for end-users with a poor internet connection for any game on the platform. The current alternative to our project involves game developers creating custom solutions from scratch, or developing a tool similar to our implementation. This option is impractical for small development teams as such a feature would require intensive development and testing. The undertaking would require background knowledge of or research into latency compensation, which would consume considerable resources and development time.

Game developers would be able to use our plugin seamlessly with their new or existing games by simply adding it to the project. We will make the interface as easy to use as possible, allowing developers to treat it as a black box that returns the modifications they desire. In addition, they will also have the option to alter parameters that affect how each attribute is scaled if they so choose. Using the plugin combats the negative effects and increased difficulty from input latency allowing players on any platform to get the experience intended from the start, without having large amounts of knowledge about the cloud gaming architecture.

3) Methodology

This chapter details the research questions and the design of the tools and experiments used. This includes a brief explanation of the attribute scaling plugin and the two user studies that were designed.

3.1 Research Questions

In compensating for latency by using attribute scaling for cloud-based game streaming, we have two research questions:

> **R1.** Is attribute scaling effective in mitigating the negative effects of latency in cloud-based game streaming for FPS shooter games?
>
> **R2.** Can attribute scaling be made available for general use by developers in a commonly used game development engine?

The first research question was selected as FPS shooter games are latency-sensitive, with players tasked with using fast mouse movements to aim at targets on-screen. This creates high precision and tight control requirements, which are impeded by any additional latency between a player's input and visual feedback. To determine the effectiveness of attribute scaling in mitigating the negative effects of latency in cloud-based game streaming, we developed an FPS aim-training game. The user study conducted using this game as a test platform allowed us to gather quantitative data such as player accuracy and their perception of the quality of experience in-game.

The second research question was selected with the general-purpose use of attribute scaling in mind. Because many games are developed in just a few popular game engines, a plugin designed to be used in one of these engines would allow this scaling technique to be applied to more games. A second game and attribute scaling plugin allowed us to explore whether this latency compensation technique could be packaged for use by developers in the commonly used engine Unreal Engine. A user study was conducted with game developers tasked with using the attribute scaling plugin to modify a parkour game.

3.2 Plugin Design

To implement the attribute scaling technique in a video game, a developer would need a game-streaming host software with an inter-process communication (IPC) method designed to measure and make available the latency between the server and player to the game itself. The developer would also need to deserialize this data from the IPC, process it so that it can be accessed from all objects they wish to scale, and then modify all the variables in real time for

each game update. The variables may need to be modified using different functions. If the data is too infrequent, a smoothing function would be necessary to clean up the sudden and drastic changes.

To avoid implementing this for each game developed, we designed and implemented a plugin for Unreal Engine which provides a GUI to facilitate incorporating attribute scaling into a game without the heavy background knowledge and code necessary to develop the functionality.

3.3 Experimental Design

The first game we developed was an FPS aim-trainer to measure player performance in metrics such as accuracy while running on a server-client setup that allows us to replicate latency conditions for each match. Players experience the same latency conditions and complete matches with attribute scaling turned on or off. The experimental setup allows for variables such as latency, target movement, and attribute scaling to be adjusted while collecting performance data from each round.

To see how developers interact with and utilize attribute scaling in a video game, we developed a parkour game compatible with the plugin in Unreal Engine. This allowed us to examine which attributes game developers chose to scale, and whether the latency compensation technique could be made available for use by developers without prior experience with implementing the technique on their own.

4) Impact of Attribute Scaling on Aiming

For this user study, we sought to determine the impact that attribute scaling has on player aiming performance, a common gaming action, and their quality of experience. To achieve this, the aim-trainer experiment was developed with certain independent variables in mind, allowing us to change these parameters for each round.

Independent Variables	Dependent Variables
Use of Attribute Scaling	Player Accuracy/Performance
Latency	Quality of Experience
Target Movement	

Table 1: Experimental Variables

4.1 FPS Aim-Trainer Game

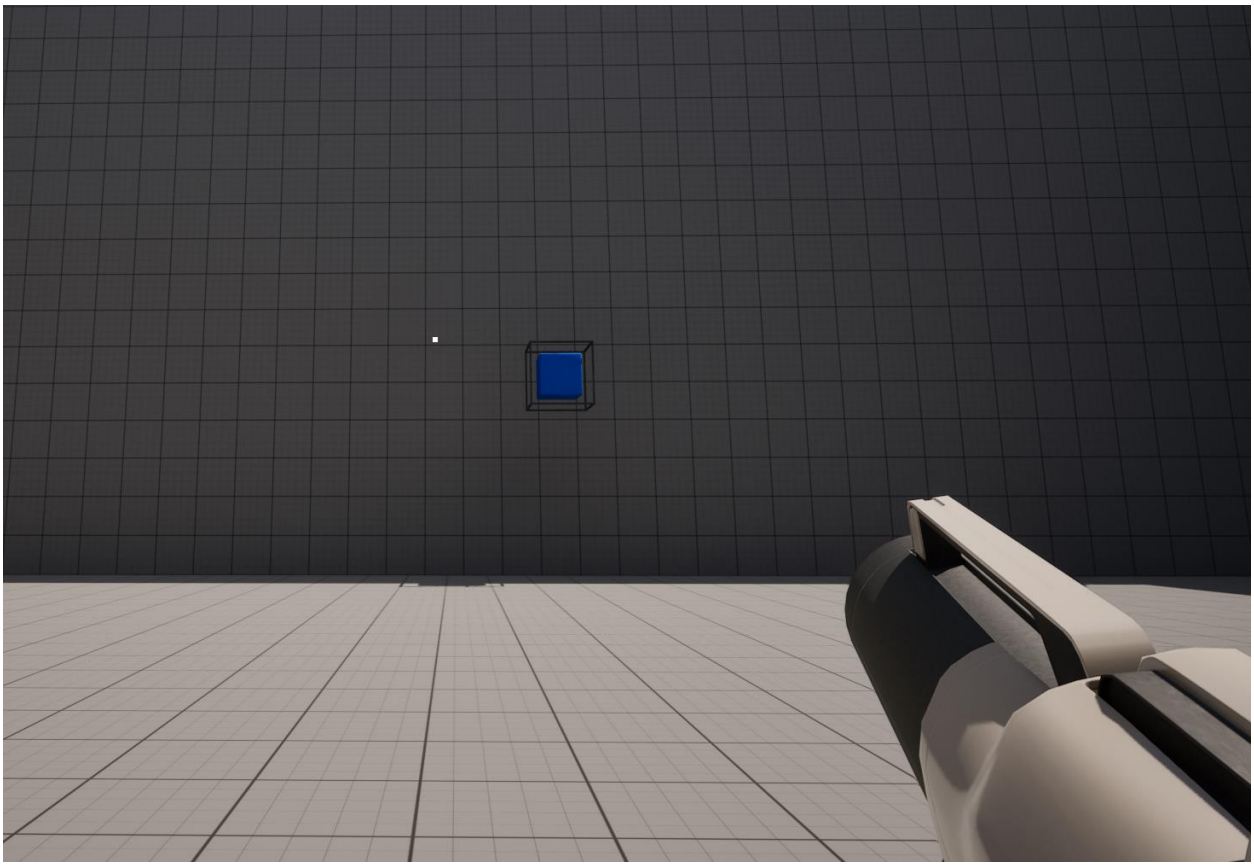

Figure 3: The player's point-of-view (POV) when aiming at the provided target. In this image, a visual representation of the hitbox can be seen as a gray frame around the target. The player's point of aim (crosshair) is visible as a white dot.

For each 30-second round, players are provided with a hitscan weapon and a single target to shoot at. Once the target is hit, it respawns immediately along the wall the player is aiming at. The objective of each round is to hit the target as many times as possible in the allotted time. The weapon is recoilless, and there are no restrictions on fire rate and ammunition.

A countdown timer is shown at the top of the player's screen, displaying the remaining time during each 20-second round. As shown in figure 3, displayed on-screen is a white dot representing the player's point of aim. The shot fired by the player is a line trace that begins at the center of the player's field of view detailed by the dot, and matches the player's direction vector.

The map consists of an open, flat plane. Player movement is restricted upon the game starting, with only their cursor free to pan around. The shooting range is walled on all 4 sides, with the player facing the north wall at the start.

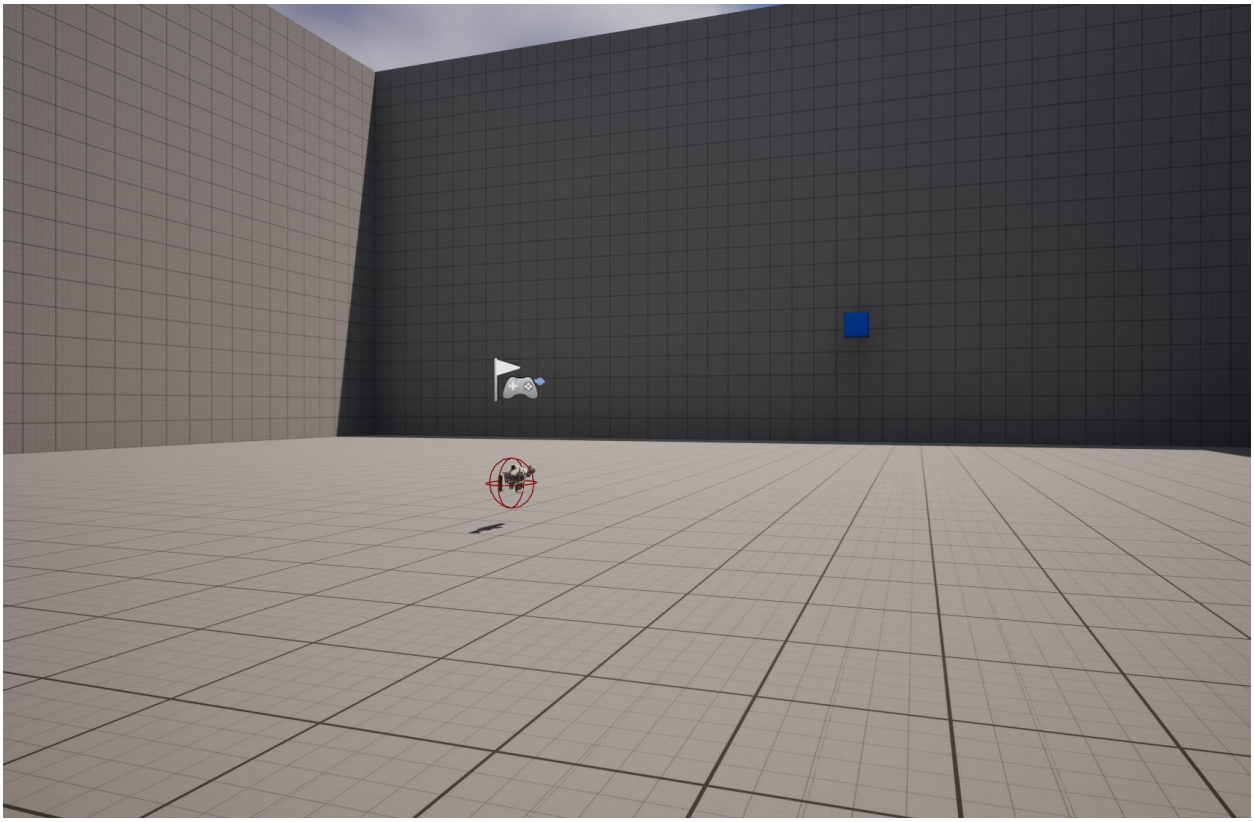

Figure 4: An overview of the game environment. The gun model represents the player's location. The walls surrounding the field can be seen.

The player is asked to shoot a square target as many times as possible during the match. This target is mobile for the entire round for half of the rounds played, translating back and forth on the x-axis at a constant speed. When attribute scaling is enabled, the target's hitbox scales linearly with the latency experienced by the player. The hitbox dimensions of the target were increased by 50% at 60 milliseconds of added latency and 100% at 120 milliseconds of added latency. Shown in figure 5 is a visual representation of the target's hitbox at 120 milliseconds of added latency. The hitbox boundary is denoted by a gray border.

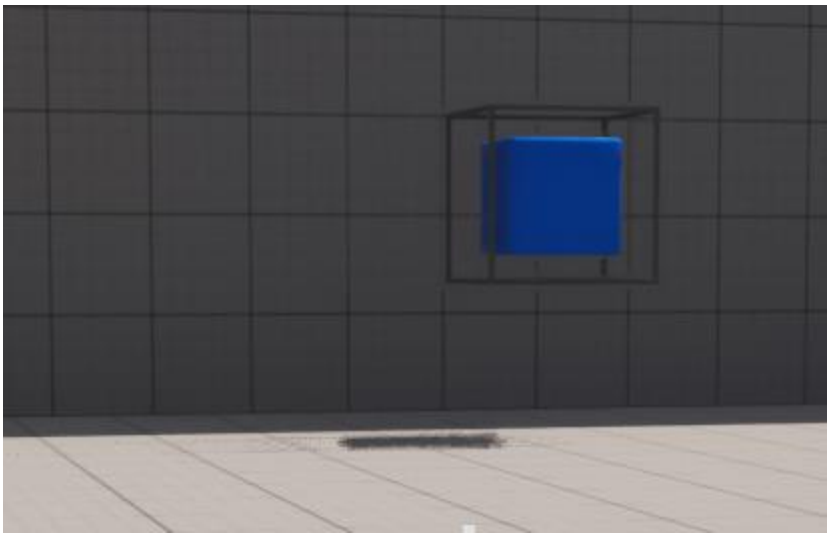

Figure 5: A view of the target with attribute scaling enabled. The hitbox is made visible to demonstrate the size increase at 120 milliseconds of added delay.

The target translates on the X-axis for half of the rounds played, moving back and forth at a constant velocity of 5 meters per second and switching directions every 2 seconds. The player stands 25 meters away from the wall, which measures 100 meters in height and 150 meters in width.

4.2 Player User Study Design

In total, 14 rounds are played by the participant, consisting of all permutations of the following parameters presented in Table 2, and two practice rounds:

Latency	Attribute Scaling	Target Movement
0 / 60 / 120 ms	On/Off	On/Off

Table 2: Rounds parameters

The practice rounds were conducted without additional latency, target movement, or attribute scaling. The participants were allowed to change the mouse sensitivity to their liking during the practice rounds. This sensitivity was kept constant throughout the remainder of the rounds.

Study participants were recruited voluntarily from the WPI student body. The details provided to them about the experiment included a brief explanation of the structure of the study, such as the number of rounds and the mechanical features of the game they were to play. Further

details such as an explanation of attribute scaling, what aspect of the game would be scaled, and what latencies the players would be experiencing in-game were withheld.

The hardware used during user studies allows us to run the game instance and stream it at a monitor refresh rate of 60 Hertz. The games were run on the server while streamed through Sunshine to Moonlight on the client computer. Test participants were not informed of the nature of the latency compensation technique, and were generally unaware about how attribute scaling was implemented. The client and server computers possessed the same specifications: a Windows 11 installation alongside an Intel Core i7-8700K Processor, NVIDIA GTX 1080 Graphics Processing Unit (GPU), and 64 gigabytes of Random Access Memory (RAM). In addition to this, the participants used a Logitech G502 mouse.

4.3 Experimental Data and Observations

In each round of gameplay, data was collected which measured player accuracy, as well as their opinion on the gameplay experience. All gameplay and survey data were saved to a database.

The quantitative data collected includes the location of the target each time the player fires the gun, as well as the location of where the hitscan weapon impacted either the target or the back wall of the shooting range. From these numbers, a delta can be computed to determine the “error” between the center of the target and where the player was aiming at the time of the shot.

4.3.1 Player Demographics

In total, 16 participants completed valid user study test sessions. Before each user study session, participants filled out a demographic survey. We asked players to rate their FPS game experience on a scale of 1-5 with 5 representing the greatest amount of experience. Participants were also asked to list their most frequently played FPS games.

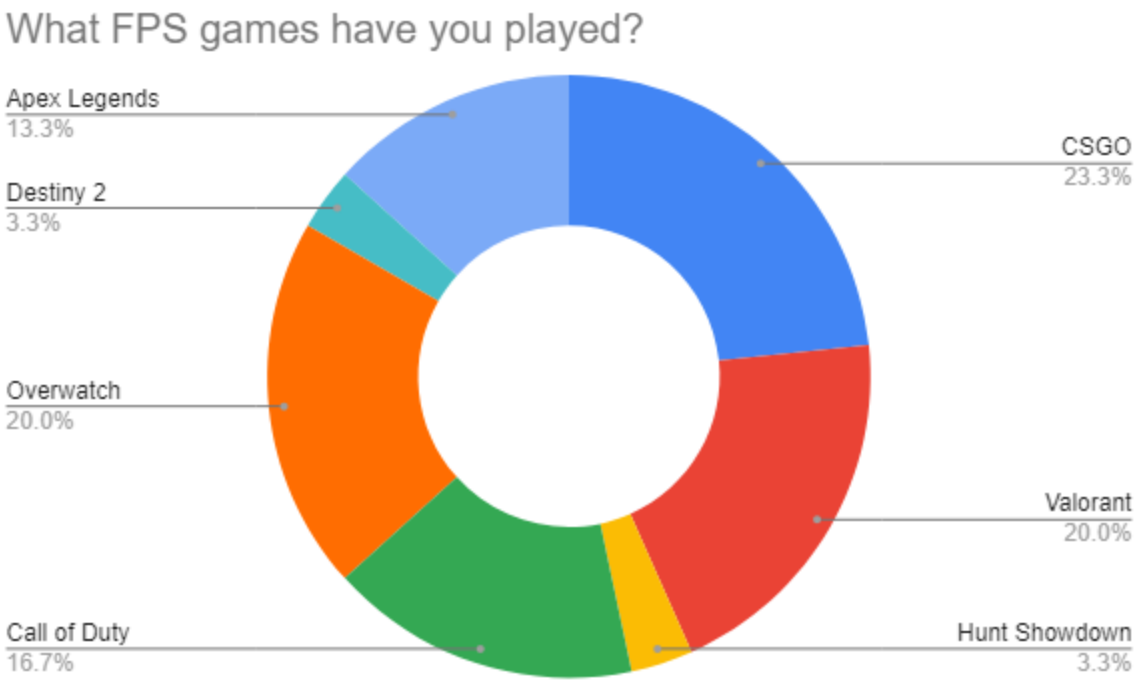

Figure 6: Survey results detailing which FPS games are most commonly played by users participating in the FPS Aim-Trainer study.

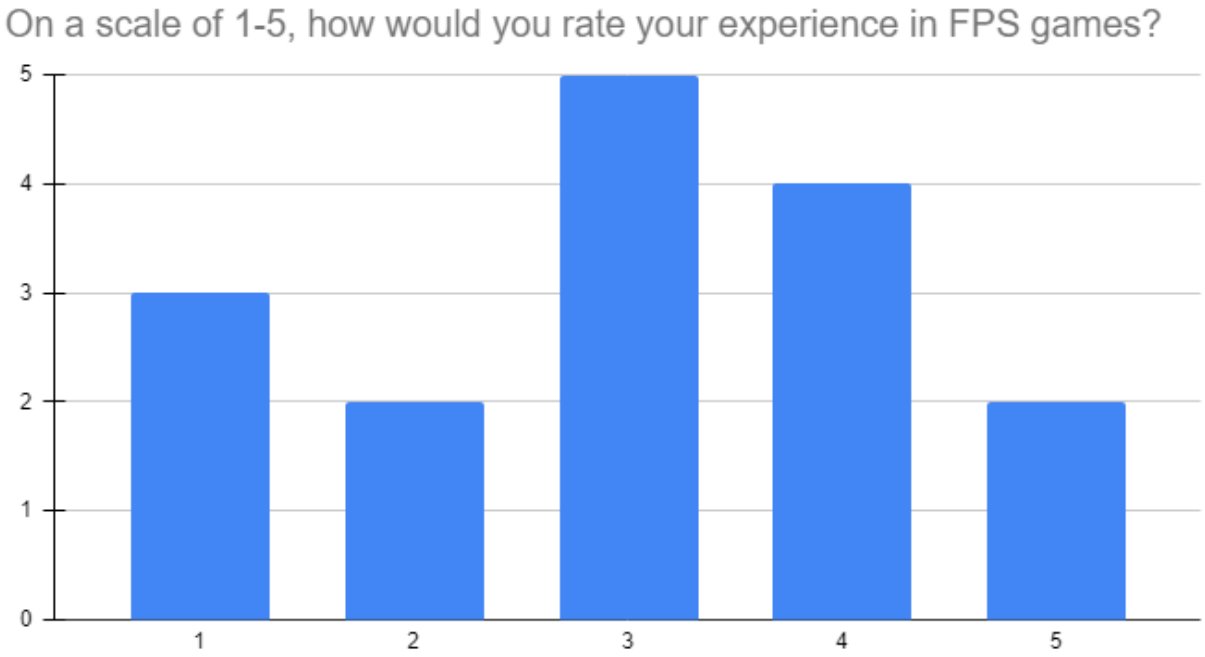

Figure 7: Survey results detailing players' self-attributed experience levels in FPS games.

From the collected data above, participants in the aim-trainer study varied considerably in experience levels in FPS games. Per figure 7, nearly a third of the users involved in the study stated that they had relatively little experience in FPS video games. Anecdotally, this resulted in data collection from some players who were unfamiliar with the nature of latency in networked video games and what effects they may cause, as they may not be familiar with experiencing these conditions in FPS games.

As seen in figure 6, within the variety of games played, nearly half of the participants indicated they have had some experience with games such as CS:GO and Valorant. Player performance in these games relies heavily on fast mouse movement and target acquisition, with many players utilizing aim-training software/games to seek a competitive advantage by practicing their aiming skills.

4.3.2 Analyzing the Impact of Attribute Scaling on Player Performance

The quantitative performance of the players participating in the aim-trainer user study was measured based on key metrics that include the number of shots they landed on the target, and where on the target those shots landed. For each round, these key pieces of data were collected, allowing us to compare the average performance of players between rounds.

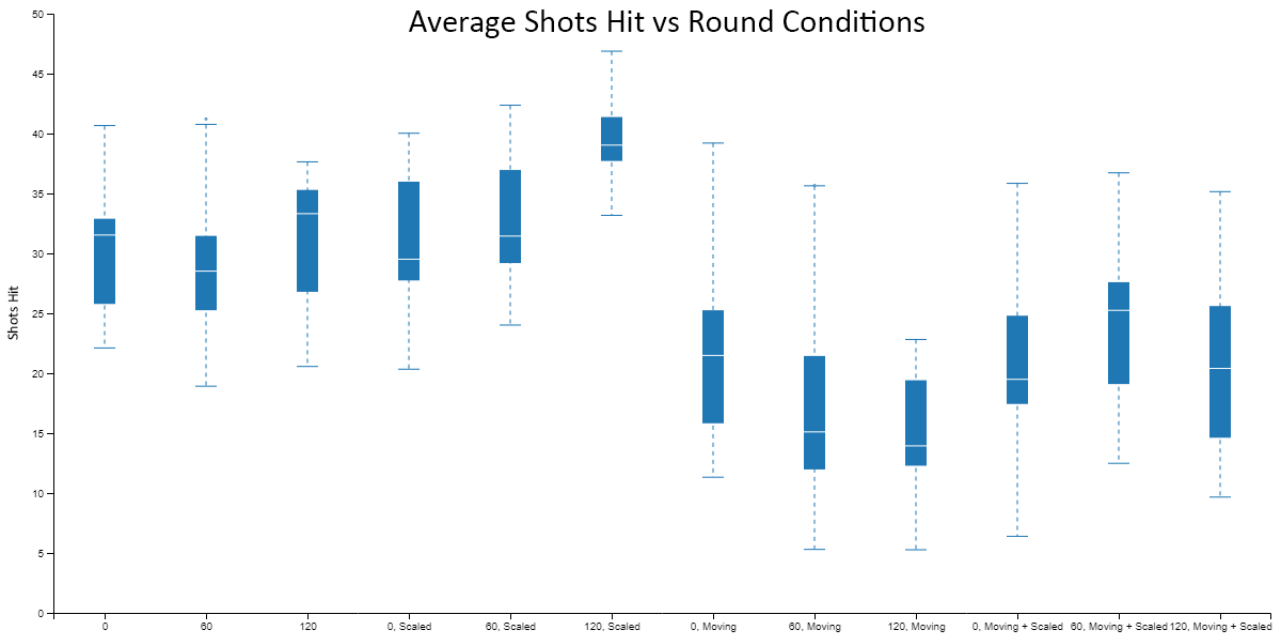

Figure 8: Box-plot detailing the number of shots landed on target at varying round conditions

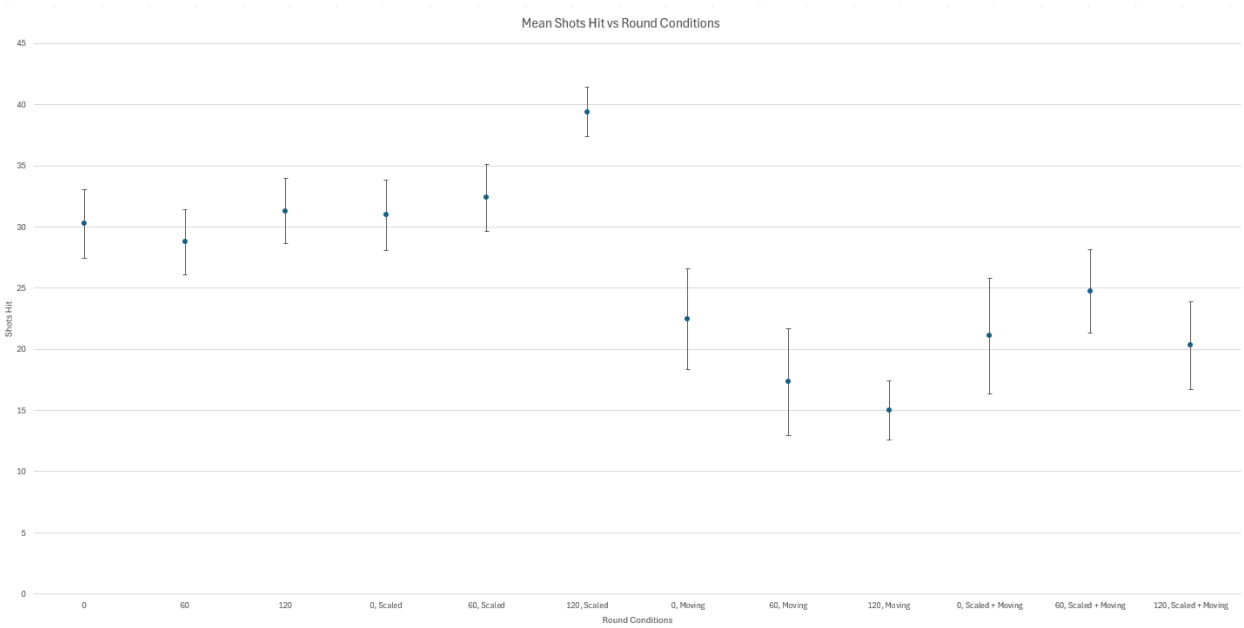

Figure 9: Dot plot detailing mean number of shots landed on target at varying round conditions

From the data gathered on the number of shots hit for each round, we see a steep drop in the average number of shots landed once the target begins to move in figure 9. This demonstrates that the moving target added a layer of difficulty to the task of shooting at the target. With attribute scaling enabled, the average number of shots landed on moving targets between all latencies tested rose by approximately 3.79, or 20.4%. In addition, the average number of shots hit more closely matches the average number of hits at 0 ms of added latency. This shows that attribute scaling is an effective solution to mitigate the difficulties encountered by players when hitting a target moving target when there is added network latency, helping to match the performance they had when no latency was added.

Additionally, we see an increase in shots hit when attribute scaling is enabled and the target is stationary. This increase demonstrates that the difficulty of the task of aiming is mitigated by attribute scaling, however, the scale factor may be too aggressive. This could cause the gameplay challenge to become too easy, even at high latencies. The difference in the number of shots landed on the stationary targets and the number landed on the moving targets indicates that the movement speed of the target should be taken into account when determining the scaling factor.

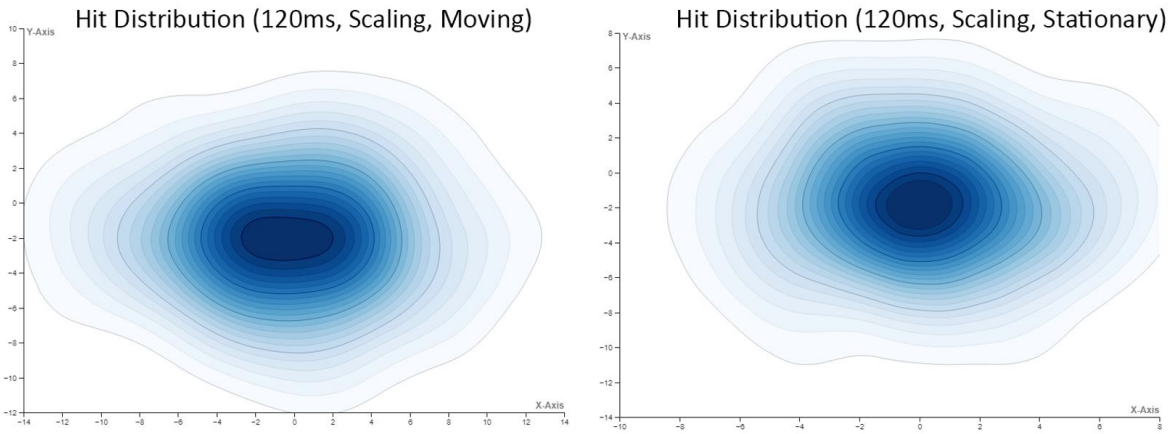

Figure 10: Topographical heatmaps representing the X-Y coordinate distribution of shots hit at 120ms of added latency, with attribute scaling enabled.

Based on topographical heatmaps of the aggregated shots landed on the target, we can see that there is a wider variance of shot placement when attribute scaling is enabled. In figure 10, the heatmap representing the location of shots hit on the target shows that the shots hit include many that landed outside of the visual representation of the cube, which spans from -10 to 10 units on the X and Y-axes. This is expected, as the larger, scaled hitbox allows players to hit the target even when the shot does not land on the physical blue mesh that represents the target. One ramification made apparent here is the difference in objective difficulty that attribute scaling creates. Players can score points in this aim-trainer game despite not landing their shots on the visual representation of the target. In a competitive multiplayer environment, this could result in claims of unfairness for players who experience higher latencies benefitting from attribute scaling.

4.3.3 Analyzing the Impact of Attribute Scaling on Quality of Experience

After each round of gameplay, players were asked two questions to gauge their perception of their quality of experience for the round. These questions were:

1. For a networked game, was the level of lag acceptable?
    a. Yes
    b. No
2. On a scale from 1-5, rate the gameplay experience for the round, with 1 being least enjoyable, and 5 being most enjoyable.

The first question asked in the after-round survey allowed us to gather the players' binary perception of the quality of their gameplay. This provides us with feedback about how players feel about the performance of the game environment compared to their previous experiences

with network video games. The second provides more granular feedback on how the rounds compared against each other in terms of the player's quality of experience.

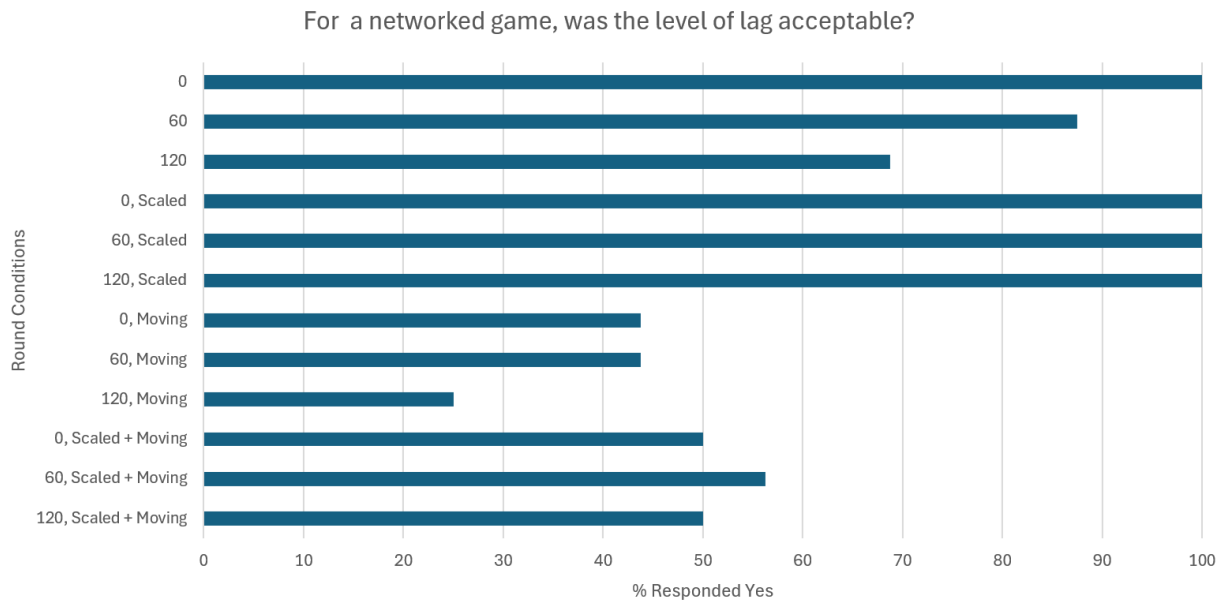

Figure 11: Bar chart detailing player responses to the question: "For a networked game, was the level of lag acceptable?"

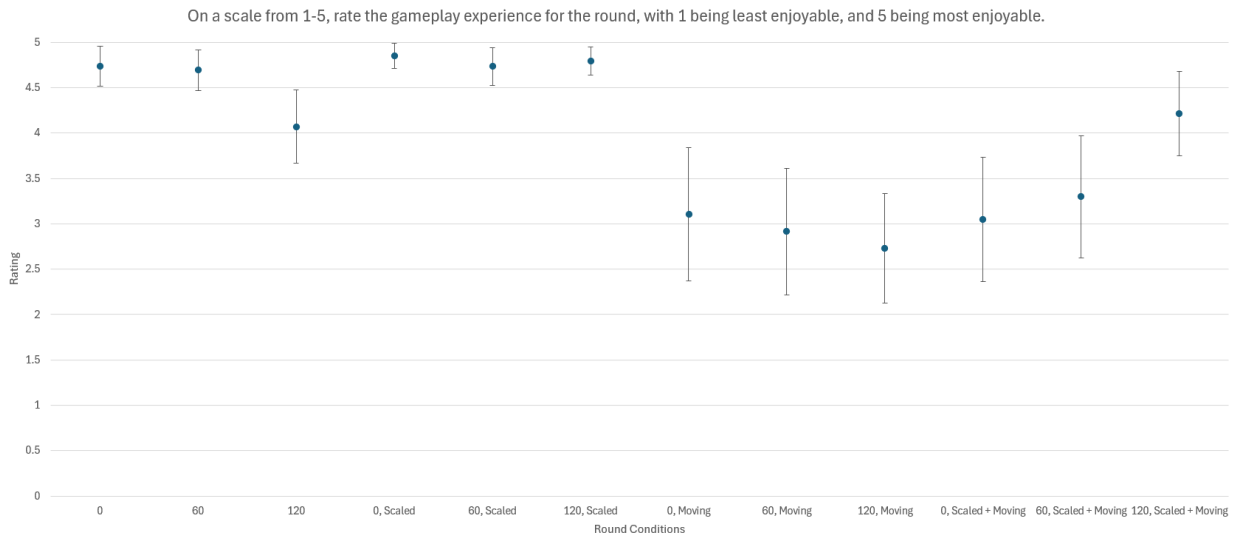

Figure 12: Dot plot detailing player responses to the question: "On a scale from 1-5, rate the gameplay experience for the round, with 1 being least enjoyable, and 5 being most enjoyable."

From the data gathered, we see a sharp rise in players finding the perceived latency unacceptable when the target was translating back and forth. Shown in figure 11, close to 50% of players reported that the perceived latency was unacceptable in rounds where the target was

moving, whereas the majority of players reported that the latency was acceptable in rounds with a stationary target. This is likely due to the added difficulty of tracking and hitting a moving target. Even at no additional latency, the moving target added sufficient difficulty to the player's task that many were confident that there was an unacceptable amount of latency added.

The added movement also impacted the player's answers to the second survey question, with the mean rating falling during all rounds with the mobile target. Figure 12 details the players' responses to a survey question asking how they would rate their quality of experience for each round. Between the rounds involving a stationary target and those involving a moving target, with no attribute scaling, the average quality score for the round dropped from approximately 4.50 to 2.91. From the results during rounds with the attribute scaling, there was a marked improvement during rounds in which the target was moving. With scaling enabled, during rounds with an added 120ms of latency and a moving target, the average quality score rose from approximately 2.73 to 4.22. This can be attributed to the increase in hitbox size alleviating some of the additional difficulty of aiming at a target at higher latencies when the target is moving.

The rounds involving a stationary target did not see such a large difference in players' perception of the round quality when attribute scaling was enabled. While the rounds played with 120 ms of additional latency saw an increase of 0.72 in average rating with a stationary target, the overall average increase in rating between all rounds played with a stationary target was only 0.29 points. This could be due to how players shot at the target, only clicking to fire once the visual feedback of the on-screen crosshair overlapped the cube.

5) Improving the Accessibility and Implementation of Attribute Scaling

As previously described, the work necessary to implement attribute scaling in a video game is non-trivial, and developers can either create a single-game implementation or a tool that will allow them to implement attribute scaling in more of their games. The benefits of improving the experience of a small portion of their players do not outweigh the work and time necessary for either of these options. To make this technique a possible solution for game developers, we created the tool as a plugin for the Unreal Engine. This allows the tool, and subsequently the solution, to be integrated seamlessly into their game design projects.

5.1 Plugin Implementation

The theory and implementation behind our plugin are outlined here, the following solution could transfer to any game engine.

5.1.1 Sunshine Modifications

The modifications made to Sunshine allow data to be transmitted to the Unreal Engine game client itself. Sunshine relies on enet, an open-source UDP networking library for C, to handle online communication [5]. This involved making a thread-safe atomic struct that would shuttle the latency data generated by the enet C project to the client connection flow running in C++. Next, Sunshine was further modified to create a new thread when it detects that a client connects. This thread opens a socket using Boost, a collection of libraries for C++, for broadcasting this data. This transmits an unsigned integer of the latency in milliseconds whenever the round-trip latency from enet changes, repeating this action up to 60 times per second. The client that receives data from this socket reads up to 60 times a second, only updating locally when the read value changes.

5.1.2 Attribute Scaling Listener

The attribute scaling listener is a blueprint in Unreal Engine generated from C++ code that is packaged in any relevant levels of the game that require scaling of objects[1]. This stores a list of lists that each contain Unreal Engine references to objects along with any configuration information relevant to each of the objects and each of the lists. Each list represents a pattern in which object scales or attribute values can react to changes in latency. Each element within these lists references the specific object that responds to latency as well as what parts of it will change. The listener also serializes this information to be mirrored to a config file in case the listener is deleted from the scene.

[1] In reference to the plugin, objects can be a blueprint class or an instantiated actor.

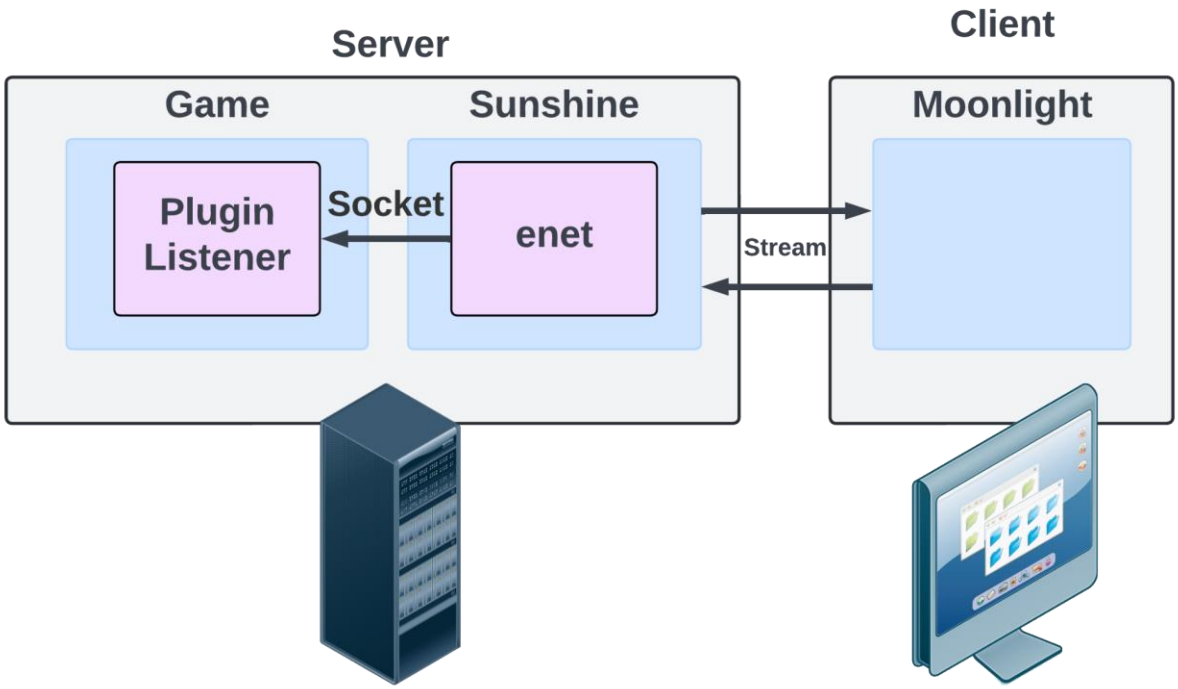

Figure 13: Networking flow between enet and the plugin listener.

The listener also receives the latency info from the modified Sunshine program and iterates through these lists, applying scaling to all of the contained objects and attributes listed by the developer. As demonstrated in figure 13, the socket communication implemented in Sunshine enables communication to the listener, which is packaged as part of the final game.

5.1.3 Plugin GUI

Although the attribute scaling listener facilitates all of the storage and scaling of the objects in the scene, the GUI inside of the Unreal Engine Editor shown in figure 14 serves as a visual way to interact with the listener and its subsequent configuration file, serving as a better point of reference for explaining the implementation of the plugin. In addition, the GUI of the plugin always communicates its latest state to the listener that gets packaged with the final game so the state of the plugin should always be in sync with the listener. The parity between the GUI of the plugin allows further references to the ‘plugin’ to refer to the package of both as a whole.

The GUI main page consists of a left panel to select actors or blueprints using standard selection modifier interactions, and a right panel that contains lists of objects that all follow one scaling amount. The right panel with a list included is shown in figure 15.

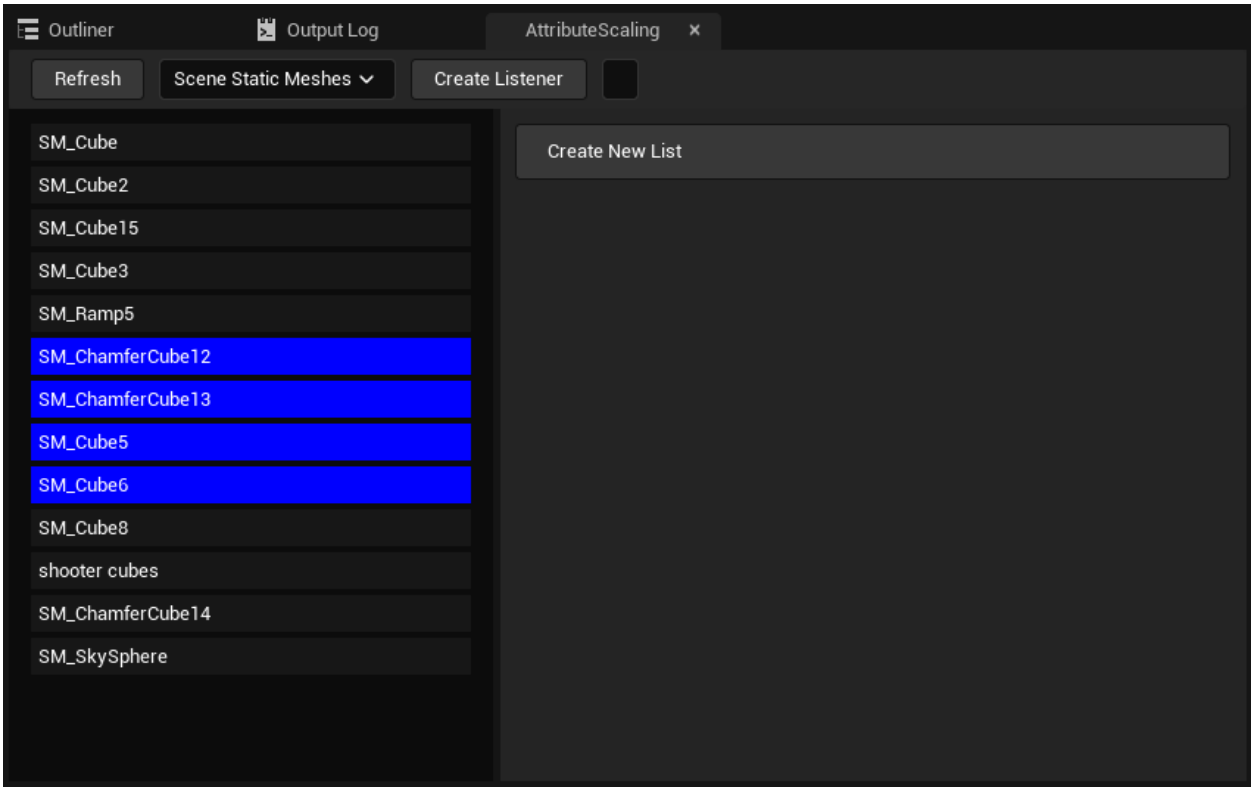

Figure 14: Plugin main page with selected objects but no list

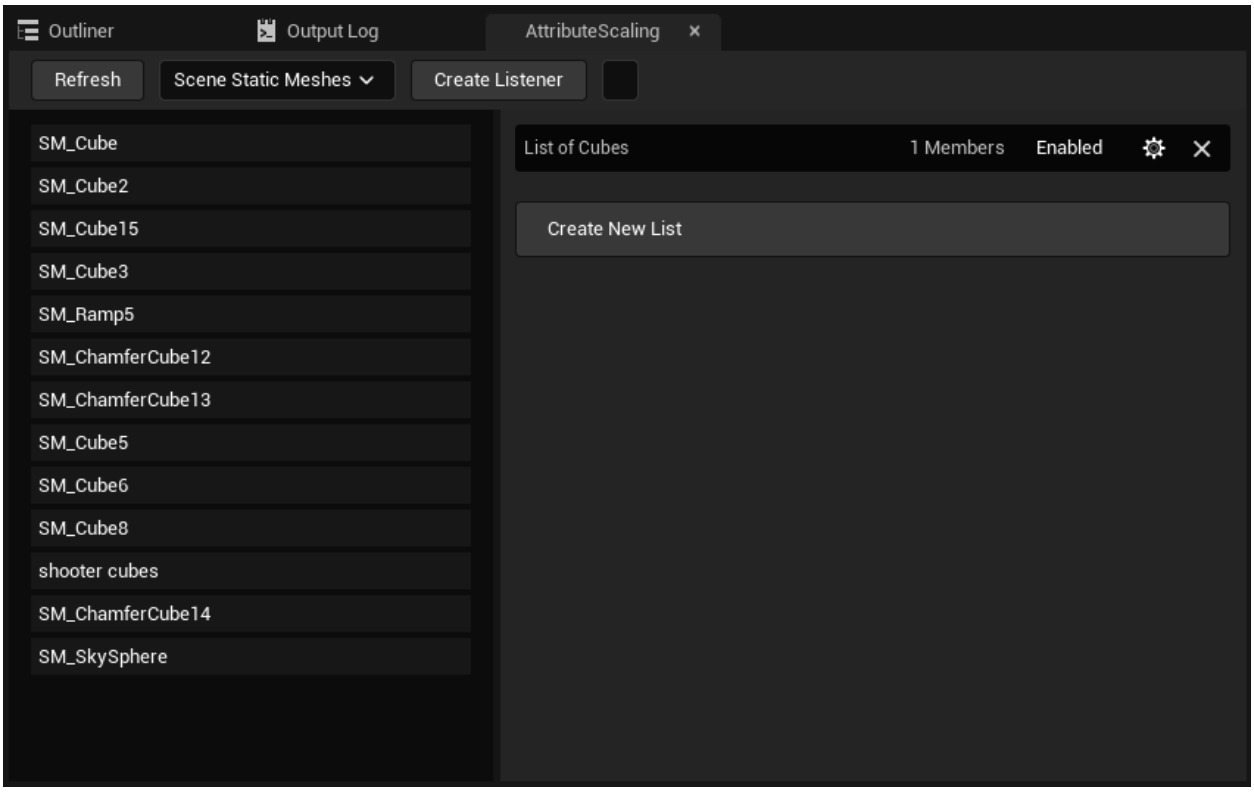

Figure 15: Plugin main page with an object added to a list

The top navigation bar includes a button to refresh the listing of current objects from the outliner, a button to forcefully create or update the listener should it need to be updated without making changes to any settings, and a dropdown menu allowing the user to filter between the left pane's objects. This dropdown can list subsets of actors in the outliner or list all of the blueprints or classes accessible from this Unreal Engine project's asset registry. Last on the navigation bar is a small textbox that allows the developer to input a sample latency value during play mode to temporarily affect objects in the scene without directly receiving latency data from a cloud-streamed connection. Hovering over each of the components of the GUI offers tooltips to help developers understand the specifics of the intended functionality.

After clicking on the settings of the list to view its details, the developer is presented with the ability to change the function that determines the scaling factor of the list based on the current latency between the server and the client as demonstrated in figure 16. Beneath this, the alpha and beta number input boxes can be used to tweak the behavior of this function at different latency values. Next, there is a table to help the developer visualize the effect some sample latency values will have on the items in the list. Last, there is an option dropdown to smooth the gaps between latency values. This is helpful for drastic changes that need to be less abrupt or anything that will change visually to make it less detectable as the player. The left panel shows each object contained in the list.

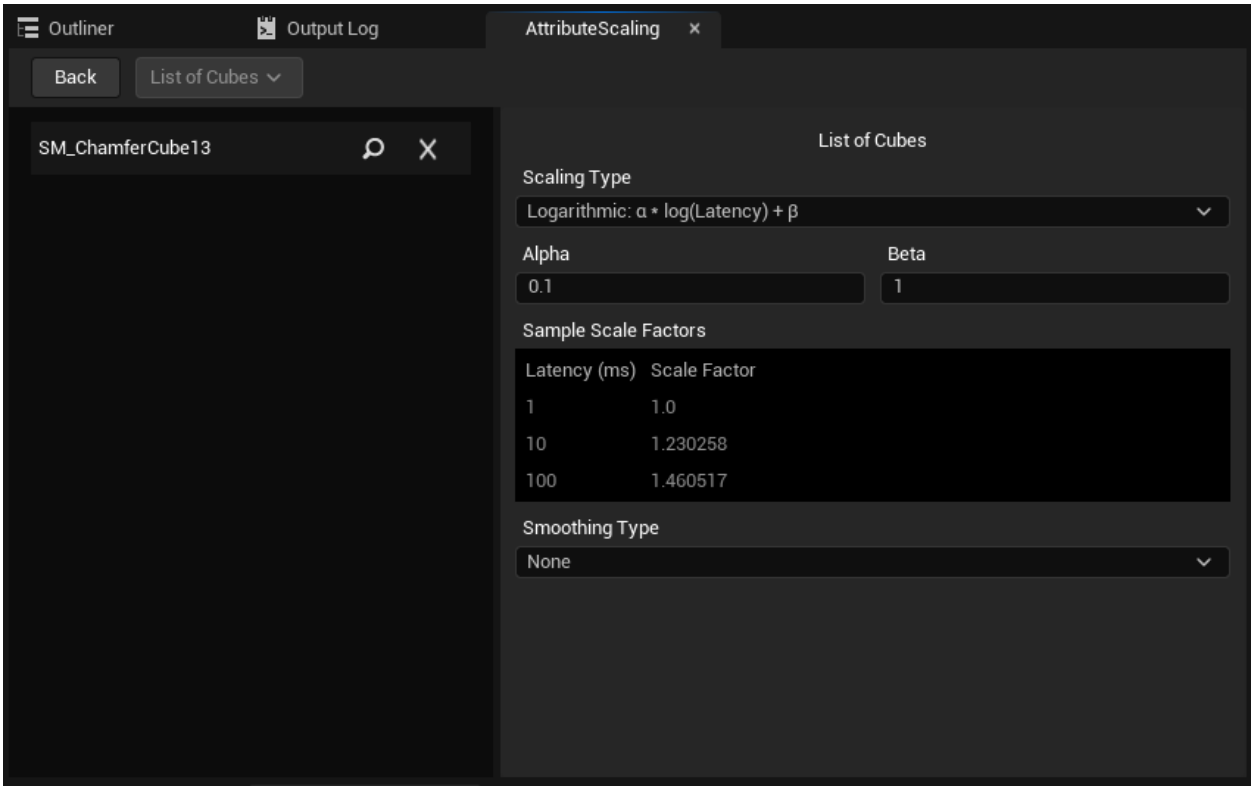

Figure 16: List settings with one object in the list

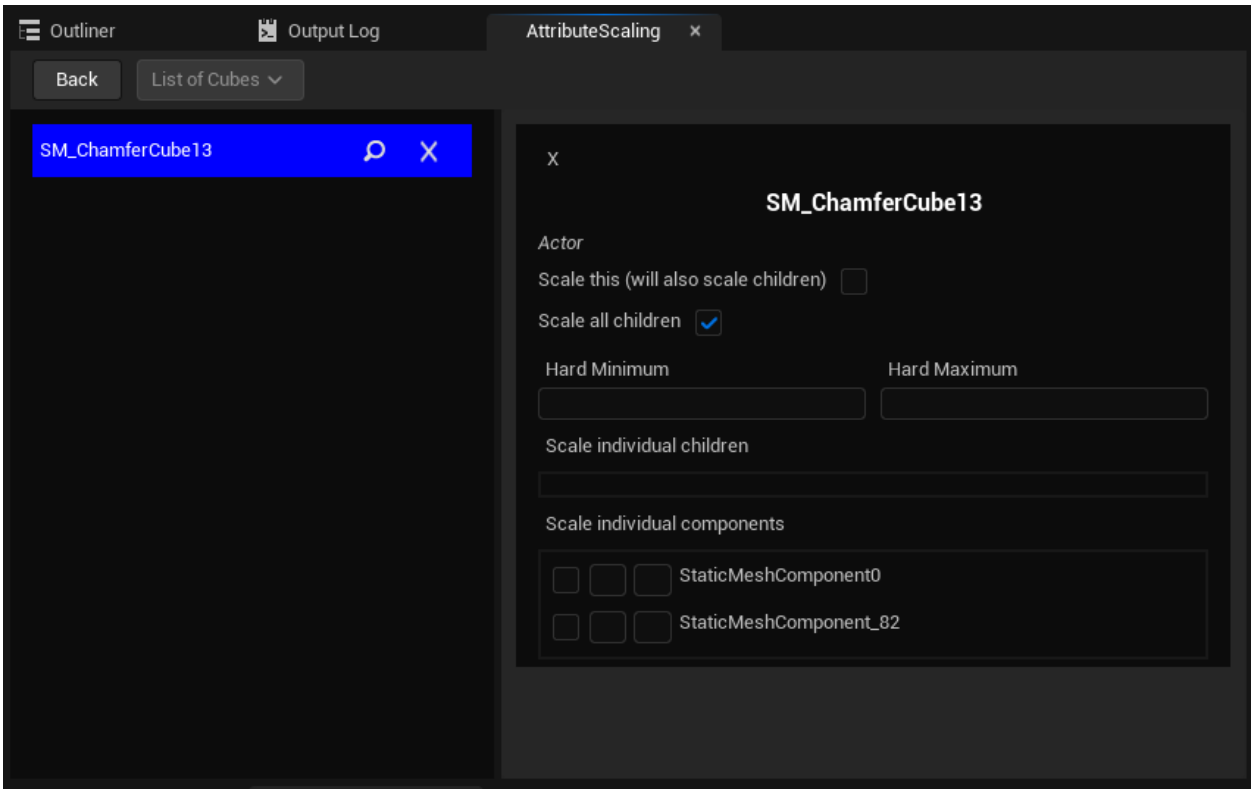

Figure 17: Object settings from within a list

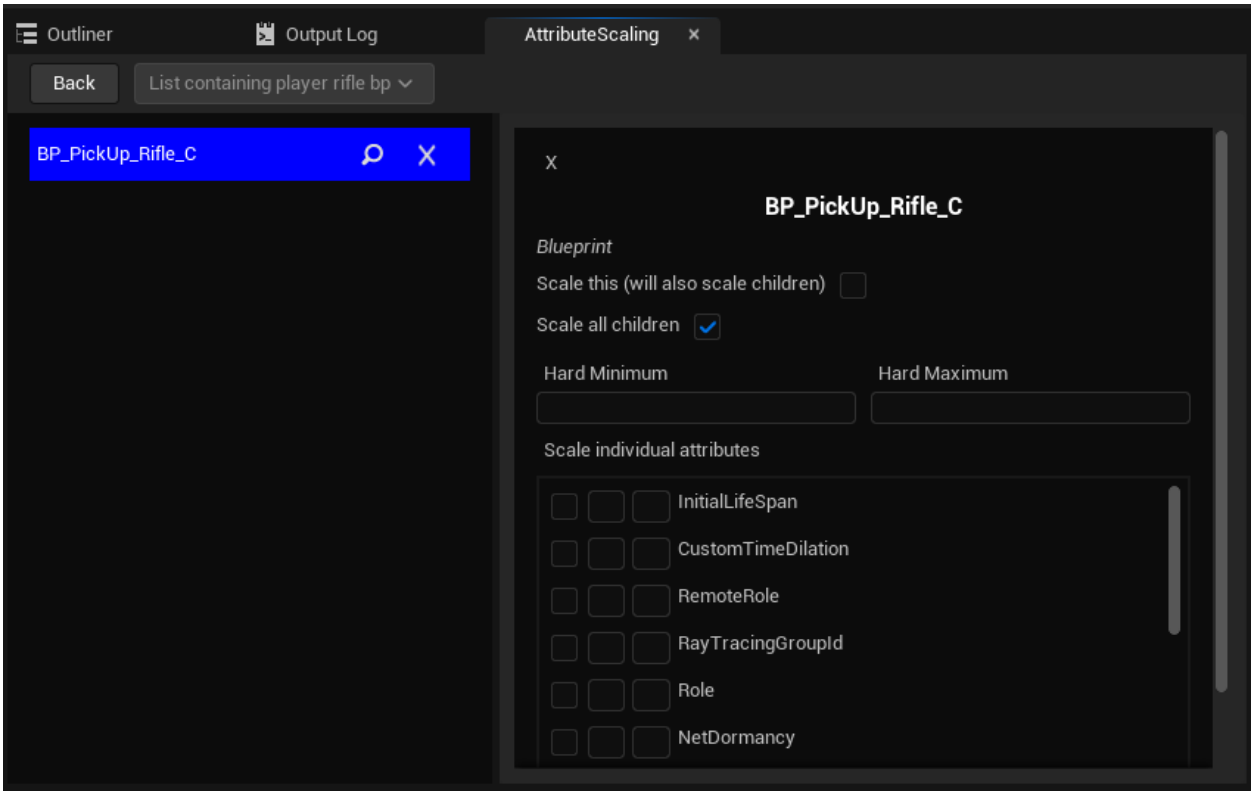

Figure 18: Blueprint settings from within a list

As seen in figure 17 and 19, clicking on an object to view details about it reveals a new right pane that is different depending on whether the developer clicked on an instance of an actor or an entire blueprint class. In the first case, where an instance of an actor is selected, the right pane contains an option to scale this specific object, all children[2] of the object, options to restrict the scaling factor for anything from the actor, a list of children to scale along with their own individual restriction options, and a list of components to scale along with their own individual restriction options. In the second case, where a blueprint class is selected, the right pane contains an option to scale all instances of this class across the scene, all children of instances of this class across the scene, options to restrict the scaling factor for anything from the actor, and a list of publicly accessible numeric values exposed from within the class along with their own individual restriction options, demonstrated in figure 18.

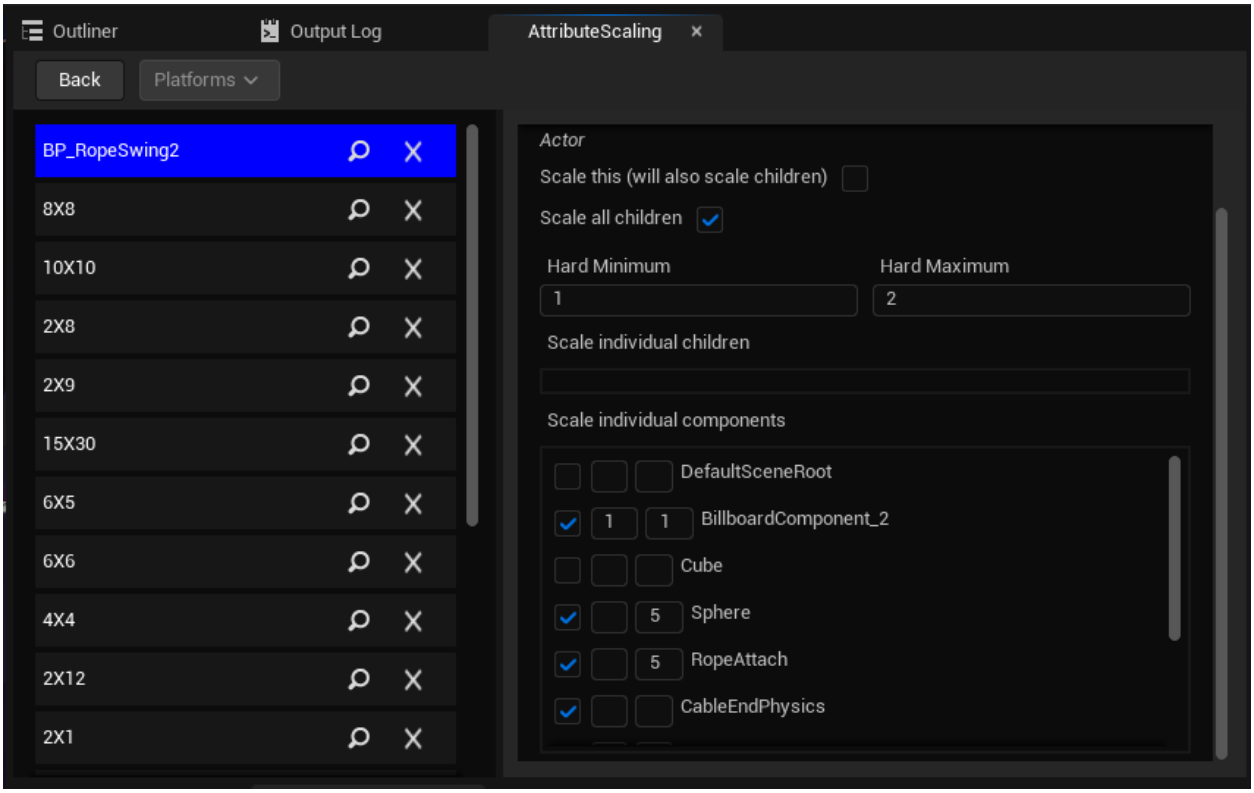

Figure 19: Actor settings (instance of blueprint) with components selected

5.1.4 Plugin Lifecycle Behavior

Given that the plugin needs to maintain a consistent and manageable loop in order to keep the game functioning as expected, the plugin listener that gets packaged with the game follows a strict pattern.

[2] Children of an actor are actors in the hierarchical outliner that are attached to and lower than their "parent" actor.

Each game update, the plugin listener checks to see if it has ever received latency data from Sunshine. If it has not, nothing in the game is affected. If the game has received latency data, but not in the last three seconds, all scaling info is deleted, and the game acts as if it has never received latency data. If the plugin receives latency data for the first time, it stores all of the initial scales of objects to be scaled and all of the initial values of attributes to be scaled by iterating through the lists, using their reference to find them in the world. For reference, the plugin uses FName, which should be consistent across LODs, but is not guaranteed to be unique across levels. This tradeoff accepts collision chances for performance over the guaranteed unique ActorName string comparisons.

Once the initial sizes have been stored, the plugin iterates through all of the lists each update, scaling each item in each scaling list according to its list's settings and its object settings. For consistency, if physically scaling an object is enabled, the following scale all children setting is ignored, and the individual child scaling settings are ignored. If scaling all children is enabled, then individual child scaling settings are ignored. If at any point the plugin encounters an object without an initial value, the initial value is saved from when it is first seen.

As the Unreal Engine may Tick() on a differently addressed instance of the listener that is different than is accessible from the outliner or search through the world, any game blueprints designed to modify the behavior of the listener during the play mode lifecycle will lead to unpredictable behavior. Thus, the settings of the scaling lists cannot be modified while the game is running or in play mode.

Using the repository containing our game and Unreal Engine plugin, the top-level Plugins folder contains source code to the plugin UI and the plugin listener. Place these next to the .uproject file of the project to use the plugin in. These would be built using the Visual Studio editor or Rider editor if they are not automatically built by Unreal Engine. The plugin is now accessible from the Unreal Engine editor plugins menu.

5.2 Parkour Game

5.2.1 Game Implementation

Parkour Escape was created from the ground up with the objective of being a game that was not purpose-built for attribute scaling, but could still be effective for evaluating the attribute scaling plugin. If the plugin could work in a game still in development, it means that the idea could work in a development environment. The theme of parkour was chosen because it involves many complex actions, with a combination of time-based input requirements and precision-based movement decisions. The wide variety and large number of moving parts and mechanics of the game provide ample opportunities for attribute scaling. The goal for this game was to create an intuitive playground for game developers to freely use the plugin. At the end of development,

many attributes were made public, each of which, when scaled, could visibly or functionally change the gameplay. It also served as a good example of a graphically intensive game where users would use a cloud-gaming service to play this on a thin device.

5.2.2 Game Objective

The objective of the parkour game shown in figure 20 is to escape out of a crystal mine shaft by navigating upward through platforming. Players utilize map elements such as wall-rides, jump pads, and rope swings to traverse gaps that would be otherwise inaccessible with a regular jump between simple platforms.

Figure 20: Section of the game map with map elements visible

5.2.3 Gameplay Loop

The player starts at the login screen in figure 21, where they enter a username and password to keep track of their scores. After logging in, they are directed to the main menu, where they can select a specific segment of the map to play and keep track of their completion times for. Additionally, the menu provides additional buttons to access the controls menu for new players to familiarize themselves with the game's mechanics, exit the game, or log out of their account.

Figure 21: Parkour Escape main menu

When the player quits the game the login state and username are stored in a save game object to ensure that reopening the game returns players directly to the main menu screen without the need to log in again. All user data, including login information and scores for each segment, are stored in a MongoDB cloud database for security and the ability to have live leaderboards for each segment in game.

When clicking on a segment of the map from the menu, the player starts their parkour run after a three-second countdown timer as shown in figure 22. The objective is for the player to reach the next yellow checkpoint as quickly as possible using any means permitted by the game. Usually, this involves following the platforms with lights next to them, but players always have the option to create their own, possibly unintended, path.

During the run, the player sees two elements in their in-game heads-up display (HUD): the leaderboard for their segment, which displays the top 5 times in the world with their current best time at the bottom, and a timer indicating their performance in the current run.

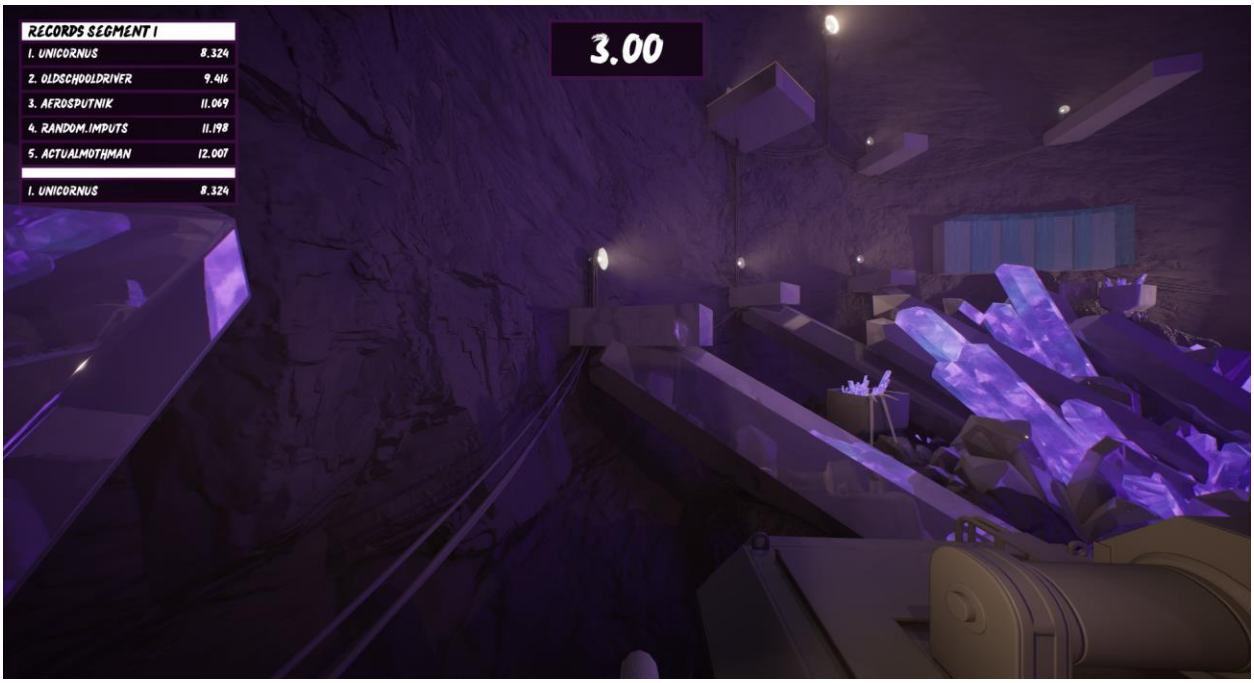

Figure 22: Start of first segment

Once the player finishes the segment by getting to the next yellow checkpoint, they are presented with their personal best time along with the time difference from the time they just achieved, as demonstrated in figure 23. They also have the option to restart the segment, proceed to the next segment, or return to the level selection menu. Additionally, at any point during the run, the player can press “R” to restart the run or “M” to return to the main menu.

Figure 23: End of first segment run display

5.2.4 Attribute Scaling Applied to Parkour Game

Every map element (wall-ride, jump pad, and rope swing), as well as the player and the platforms, were implemented with attribute scaling in mind. This involved exposing constants that could be scaled or components that could be adjusted in size. To expose these constants for scaling, they needed to be numeric values set to public. Figure 24 and 25 show a few examples of different attributes that developers could scale using the plugin.

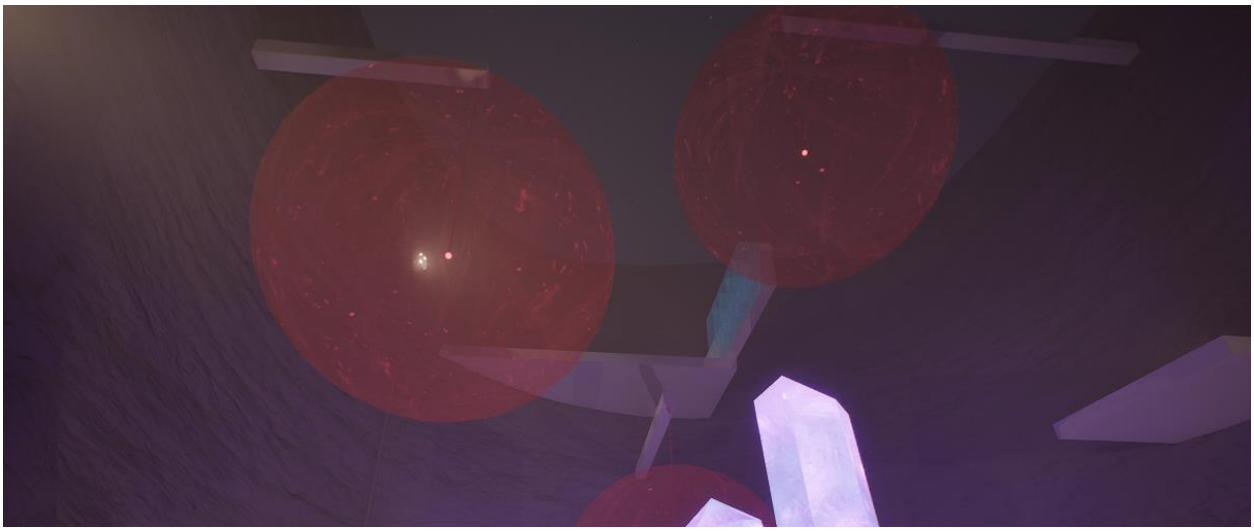

Figure 24: Rope swing colliders scaled with the plugin

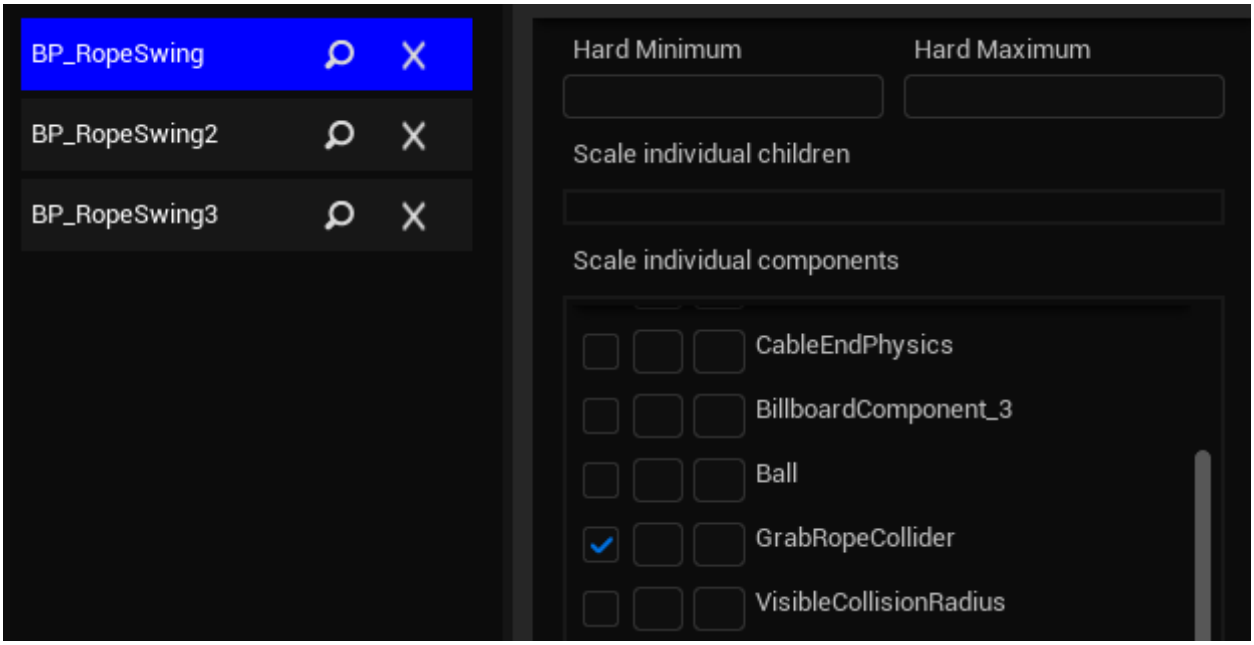

Figure 25: The plugin is used to increase the distance from which players can interact with rope swings

Developers can scale the grab rope colliders on rope swings to help players deal with added latency. This can be achieved by adding the appropriate instances of blueprint actors to a list in the plugin. Then, under the "Scale Individual Components" section, developers can check the box next to "GrabRopeCollider" to enable it to be scaled using the list's scaling function.

Developers can then use the same method to scale the sizes of other in-game objects, such as adjusting the size of a single platform that may become difficult to land on due to latency. Figure 26 shows a singular platform being scaled to become larger than the others.

Figure 26: Platform scaled with attribute scaling

Developers can scale constants that change temporal aspects of the game. One example of this is coyote time. Coyote time is the amount of time allotted to the player to press the jump button after losing contact with the ground. This allows leniency in the timing of inputs. Scaling temporal constants can make time-sensitive inputs like pressing jump more forgiving. Figure 27 shows coyote time exposed in the blueprint and subsequently scaled in the plugin.

5.3 Developer User Study Design

Given the novelty of the attribute scaling technique and the unexplored capabilities of the plugin, our second user study was a qualitative analysis of developer usage of the plugin in order to determine:

What does the developer of a game change in order to improve the quality of the experience when playing the game on cloud-based game streaming?

How do developers apply the technique when given the tools necessary for implementation?

What are some future areas of work for the plugin?

We invited game developers uninvolved with the project to participate in our study. Given the small number of participants, observations and analyses require further testing and serve to facilitate speculative analysis towards further research and development.

5.3.1 Developer Study Environment

The developers are given access to two computers. The client and server computers possessed the same specifications as our previous study: a Windows 11 installation on an Intel Core i7-8700K Processor, NVIDIA GTX 1080 GPU, and 64GB of RAM. The participants used a Logitech G502 mouse. The server computer has the Unreal Engine editor with our parkour game and the plugin. The client computer receives a game stream of the host computer with a latency of about 1 ms. 60 milliseconds of latency is then added through a utility called clumsy [6], used for adding poor network conditions to an existing connection. This setup effectively allows the developer to compare the experience of playing the game natively with the simulated experience of playing the game through a cloud game streaming service. Throughout the parkour game, we prefixed every single constant in every blueprint of the project with a "C_" with examples shown in figure 27. This impartially allowed the developers to be creative in changing anything they wanted without being burdened with learning which values were constants.

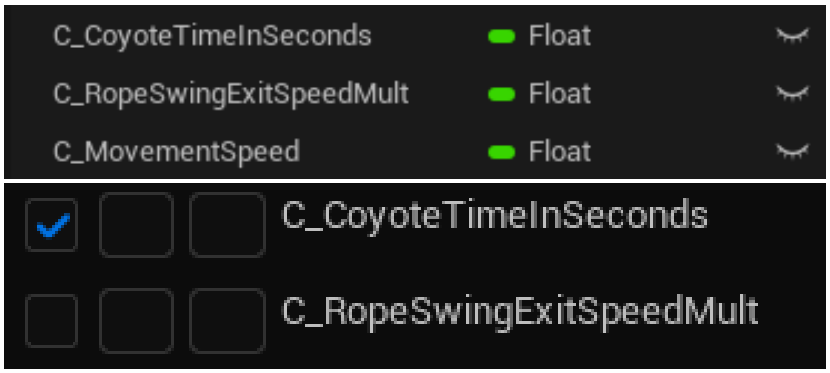

Figure 27: The constants in blueprints and our plugin UI

The goal of the developer is to balance the difficulty of the game so that it presents the same challenge on the computer with latency as it does when played on the computer without latency by applying attribute scaling using our plugin. They are given a video introduction on the capabilities of our plugin, shown examples of use, and asked to follow along with the basic tutorial on scaling one of each type of scaling made possible by the plugin. They are then given time to familiarize themselves with the plugin. After the video is complete, they are then allowed to change anything in the game, asked to explain some of their motivations behind these decisions, and whether or not they think they were effective in balancing the cloud-streamed difficulty of the game with the native version.

5.3.2 Developer Data and Observations

Our participants were two game developers with experience releasing games on their own. They are aged 21 and 22, and one has some professional experience.

Both of our developer participants tried many combinations of settings when using the plugin to apply attribute scaling, but they mentioned specific attributes/objects that helped the most. These were: scaling the amount of air control, scaling the size of platforms for landing, and increasing the size of the rope swing grab radius.

Amongst the positive observations offered, both developers mentioned that the plugin was very easy to use, and one mentioned that it gives a lot of freedom in how you want to modify the game, also explaining that it is great to be able to modify overcorrecting-friendly attributes as over-correcting is a common strategy when playing games with latency.

Amongst negative observations, both of our testers mentioned that it would be preferable to be able to scale individual axes of objects as well as their overall size. One tester suggested that it would be ideal to have a custom bezier curve or piecewise function in addition to the included functions for more control over certain lists. One of our developers explained that it was hard for them to distinguish when the game was difficult naturally as opposed to when it was difficult with latency. This caused their scaling to be stronger than they envisioned despite having a direct comparison. They suggested isolated playtesting as a means of gaining feedback on the optimal correction strengths.

Both of the developers found it common to make the game easier than initially intended, overshooting the intended compensation amounts. One of our developers offered multiple solutions to this. They suggested that during regular playtesting of the game, there should be groups of players that play with latency and scaling, with latency and no scaling, and without latency. Comparing the performance between these groups would offer helpful insights into ensuring the technique is not applied too aggressively. The developer also suggested that to decrease the amount of unpredictability in player experience, the game could change by a flat scaling factor instead of functions. Finally, both expressed concerns of players potentially using this system to cheat, a scenario involving players artificially increasing their latency to make the game easier. Avoiding this requires careful minimum and maximum value choices.

6) Conclusion

When playing video games through cloud-based streaming, latency can negatively affect players' gameplay experience. To mitigate this, a latency compensation technique such as attribute scaling can be used to change in-game values to decrease the difficulty of certain in-game tasks, thus rebalancing the difficulty of these tasks in real time.

In this research project, we implemented attribute scaling through a plugin that can be utilized in games developed in Unreal Engine. By allowing developers to interact with the plugin to attempt the task of preserving the intended difficulty of a video game, we learned that they find it challenging to avoid making the game much easier than initially designed. The plugin offers an otherwise intuitive and invaluable tool for implementing the technique as a game developer.

A positive relationship between the use of attribute scaling and the players' quality of experience in FPS games was observed, with players reporting a more enjoyable gameplay experience when faced with high network latency once attribute scaling was enabled for target hitboxes. Additionally, an increase in the average number of targets hit per round was observed when attribute scaling was enabled at higher latencies. This demonstrates that there was an overall positive impact on the gameplay experience when network latency from cloud-based game streaming would otherwise degrade the quality of experience.

7) Future Work

Our aim trainer user study reveals insightful data about the positive impact of this technique on mouse-controlled aiming in cloud-based streaming of first-person shooters. An important path forward would be to assess the impact of this technique in more genres and scenarios.

In terms of future plugin development, an opportunity for progress is to integrate support for different sources of latency from cloud-based streaming programs to provide alternatives to our modified Sunshine streaming host, increasing accessibility. Another opportunity for future work is to rework the plugin implementation to support a globally active latency listener instance, allowing changes made in runtime to roll back, better cross-level support, and enabling consistency with the API that allows other blueprints to dynamically change scaling settings in-game. There should be work put into implementing custom scaling functions and the ability to scale individual axes of objects instead of uniform scaling.

Another area to explore in future studies would be a function to scale attributes based not only on network latency, but also based on in-game variables. This would follow naturally from the decrease in average shots hit in the aim-trainer study only when the targets begin moving. Game attributes like the speed of a target could affect the parameters of a scaling function. In a related paper, the authors attempt to derive a model that chooses a scale factor based on the player accuracy the game designer intends [1]. This could be implemented as another control mechanism for the strength of scaling functions in the plugin.

There should be more developer testing done to discover novel applications of the attribute scaling technique and future testing done to evaluate the effectiveness of these applications and strategies for implementation.

References

[1] Edward Carlson, Tian Fan, Zijian Guan, Xiaokun Xu and Mark Claypool. Towards Usable Attribute Scaling for Latency Compensation in Cloudbased Games. In *Workshop on Game Systems, September 28-October 1, 2021, Istanbul, Turkey*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3458335.3460964.

[2] Robert Salay and Mark Claypool. A Comparison of Automatic versus Manual World Alteration for Network Game Latency Compensation. In *2020 Annual Symposium on Computer-Human Interaction in Play (CHI-PLAY '20 EA), November 2–4, 2020, Virtual Event, Canada*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3383668.3419910.

[3] Xiaokun Xu et al., "Compensating for Latency in Cloud-based Game Streaming using Attribute Scaling," *2022 14th International Conference on Quality of Multimedia Experience (QoMEX)*, Lippstadt, Germany, 2022, pp. 1-4, doi: 10.1109/QoMEX55416.2022.9900875.

[4] Liu, Shengmei, Xiaokun Xu, and Mark Claypool. "A Survey and Taxonomy of Latency Compensation Techniques for Network Computer Games." *ACM Computing Surveys* 54, no. 11s (September 9, 2022): 1–34. https://doi.org/10.1145/3519023.

[5] "Enet: Reliable UDP Networking Library." ENet, enet.bespin.org/.

[6] "Clumsy 0.3." Clumsy, a Utility for Simulating Broken Network, jagt.github.io/clumsy/.

[7] Epic Games. Slate Editor Window Quickstart Guide for Unreal Engine. Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/slate-editor-window-quickstart-guide-for-unreal-engine?application_version=5.0

[8] Epic Games. Slate UI Widget Examples for Unreal Engine. Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/slate-ui-widget-examples-for-unreal-engine?application_version=5.0

[9] Epic Games. Slate Overview for Unreal Engine. Epic Developer Community. https://dev.epicgames.com/documentation/en-us/unreal-engine/slate-overview-for-unreal-engine?application_version=5.0

Appendix A

IQP Demographic Survey

1) On a scale from 1-5, how much experience do you have with First Person Shooter video games, with 1 being completely inexperienced, and 5 being proficient?

1 2 3 4 5

---

2) Which of these FPS games have you played?

Appendix B

IQP Questions After Each Round

1) For a networked game, was the level of lag acceptable?

Yes No

---

2) On a scale from 1-5, rate the gameplay experience for the round, with 1 being least enjoyable, and 5 being most enjoyable.

1 2 3 4 5

Appendix C

IRB Approval Letter 1

WORCESTER POLYTECHNIC INSTITUTE

100 INSTITUTE ROAD, WORCESTER MA 01609 USA

Institutional Review Board
FWA #00030698 - HHS #00007374

Notification of IRB Approval

Date:	02-Apr-2024
PI:	Mark L Claypool
Protocol Number:	IRB-24-0652
Protocol Title:	Analyzing the Effects of Attribute Scaling in Cloud-Based Game Streaming
Approved Study Personnel:	Hsu, Jonathan~Garbaczonek, Marek S~Renzi, Mark A~Claypool, Mark L~
Effective Date:	02-Apr-2024
Exemption Category:	3
Sponsor*:	

The WPI Institutional Review Board (IRB) has reviewed the materials submitted with regard to the above-mentioned protocol.We have determined that this research is exempt from further IRB review under 45 CFR § 46.104 (d). For a detailed description of the categories of exempt research, please refer to the IRB website.

The study is approved indefinitely unless terminated sooner (in writing) by yourself or the WPI IRB. Amendments or changes to the research that might alter this specific approval must be submitted to the WPI IRB for review and may require a full IRB application in order for the research to continue. You are also required to report any adverse events with regard to your study subjects or their data.

Changes to the research which might affect its exempt status must be submitted to the WPI IRB for review and approval before such changes are put into practice. A full IRB application may be required in order for the research to continue.

Please contact the IRB at irb@wpi.edu if you have any questions.

*if blank, the IRB has not reviewed any funding proposal for this protocol

Appendix D

IRB Approval Letter 2

WORCESTER POLYTECHNIC INSTITUTE

100 INSTITUTE ROAD, WORCESTER MA 01609 USA

Institutional Review Board
FWA #00030698 - HHS #00007374

Notification of IRB Approval

Date:	25-Mar-2024
PI:	Mark L Claypool
Protocol Number:	IRB-24-0531
Protocol Title:	CS/IMGD MQP - Latency and Jitter Compensation for Cloud-based Game Streaming
Approved Study Personnel:	Hsu, Jonathan~Claypool, Mark L~Renzi, Mark A~Garbaczonek, Marek S~
Effective Date:	25-Mar-2024
Exemption Category:	3
Sponsor*:	

The WPI Institutional Review Board (IRB) has reviewed the materials submitted with regard to the above-mentioned protocol.We have determined that this research is exempt from further IRB review under 45 CFR § 46.104 (d). For a detailed description of the categories of exempt research, please refer to the IRB website.

The study is approved indefinitely unless terminated sooner (in writing) by yourself or the WPI IRB. Amendments or changes to the research that might alter this specific approval must be submitted to the WPI IRB for review and may require a full IRB application in order for the research to continue. You are also required to report any adverse events with regard to your study subjects or their data.

Changes to the research which might affect its exempt status must be submitted to the WPI IRB for review and approval before such changes are put into practice. A full IRB application may be required in order for the research to continue.

Please contact the IRB at irb@wpi.edu if you have any questions.

*if blank, the IRB has not reviewed any funding proposal for this protocol