VRSLOG: An Approach to Log Immersive Experiences in Virtual Reality Systems

Divij D.[a], Y. Raghu Reddy[b], Radha Krishna B.[c] and Sai Anirudh Karre[d]
Software Engineering Research Center, International Institute of Information Technology, Hyderabad, India

Keywords: Software Logging, Virtual Reality, Log Analysis.

Abstract: Software developers commonly use logging mechanisms to gather runtime data. Over the years, this information has been used for various purposes like debugging, behavioral analysis, system comprehension, etc. Frameworks such as log4j and Logback have helped standardize logging practices by incorporating Software Engineering principles. Logging interactions in Virtual Reality (VR) applications can help understand user behavior and assist with automated conformance checks. In this paper, we introduce VRSLOG, a generalized logging framework that can capture interactions across diverse VR scenes without any changes to the framework. We implement a prototype of the VRSLOG framework and demonstrate the generation of log files. Further, the log files are used to conduct conformance checks against a predefined expected sequence of events within the VR scene.

1 INTRODUCTION

Emerging technologies like Virtual Reality (VR) have been primarily focused on the consumer segment in the past and most the developers of the VR applications had their origins from the gaming community. In recent years the adoption of VR applications has increased in enterprise application domains like healthcare, education, manufacturing, etc. Karre et al. have surveyed VR practitioner community and argued that the development process for building VR enterprise applications needs to be more rigorous and some of the practices from traditional enterprise software development process need to be adopted by VR community (Karre et al., 2019).

One of the aspects of application development that has lot of support in traditional software development but is still not well researched/implemented in VR application development is debugging and error tracking. Debugging VR applications can be challenging due to their complex nature, involving 3D rendering, input devices, and immersive environments. Comprehensive logging helps in capturing and tracing errors, crashes, or unexpected behavior. Additionally, appropriate checks can be put in to ensure sequence of interactions.

Logging is a popular technique used by developers to debug their applications. Logging, in simple terms, is keeping a record of events in a system. As software applications scale to enterprise-grade levels, the limitations of ad-hoc logging become increasingly apparent, underscoring the necessity for structured logging. Logging frameworks such as Log4j and Logback facilitate structured logging and provide the user with ready-made logging features such as log levels, filters, formatters, etc., and help in standardizing the logs. A logging framework, at minimum, needs to accomplish the following[1]:

- Easily allow the creation of log messages
- Send log messages to the output stream
- Be transparent to the running application i.e., the application should behave the same independent of whether the logging feature is enabled or disabled

Logging user interactions in VR can provide valuable insights into how users navigate, interact with environments, and respond to stimuli. This data can be analyzed to improve the design of VR experiences, enhance usability, and create more personalized or adaptive environments. Capturing the sequence and

[a] https://orcid.org/0009-0007-1964-5932
[b] https://orcid.org/0000-0003-2280-5400
[c] https://orcid.org/0009-0000-9363-494X
[d] https://orcid.org/0000-0001-7751-6070

[1] Apache Logging Services: What is logging?

D., D., Reddy, Y. R., B., R. K. and Karre, S. A.
VRSLOG: An Approach to Log Immersive Experiences in Virtual Reality Systems.
DOI: 10.5220/0013434900003928
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 229-239
ISBN: 978-989-758-742-9; ISSN: 2184-4895

correctness of the execution of the task(s) within the immersive environment can help provide evidence of compliance, aid in audits, and demonstrate that users are performing the tasks according to specified guidelines. For example, in a VR application designed to train users in laboratory procedures and the safe handling of potentially hazardous chemicals, a key aspect of the learning process is understanding the correct protocols for carrying out the experiments. In practice, however, a user may begin heating the chemicals without closely monitoring the experiment. In the real world, a supervisor has to monitor and correct the user's actions. Even in a VR application, current systems predominantly require a human evaluator to supervise the user interactions in the VR session or record the videos to provide feedback at a later point. While this method can be effective, scaling such an approach vis-a-vis trainers and learners is difficult. This scalability limitation can be a bottleneck to adopting VR applications, especially in training.

In this paper, we propose automating the capture of user interactions via logging mechanisms. Our goal is to circumvent the need for a human supervisor, thus making the system more scalable. A common characteristic in training is the expectation from the user to perform actions within the VR environment according to some predetermined sequences. By capturing user behavior and interactions through logging, we can evaluate a user's performance based on a general pattern of expected interactions through conformance checks on the generated logs. At first glance, it may seem like these logs can be captured easily using existing frameworks such as log4j. However, the absence of such a solution in the real world indicates that logging in VR applications remains a non-trivial task due to its unique nature. While Unity and Unreal provide some built-in logging mechanisms, these are not directed to providing developers with the ability to conduct conformance checks. Further, they are also unable to help the developer solve issues particular to VR which is explained in the following text. In fact, these built-in logging mechanisms can be used to implement our logging framework in the respective engines.

Typically, VR applications are not developed from scratch. VR developers use game engines such as Unity[2], Unreal[3], etc., which have its own physics and logic to determine object behavior. For instance, Figure 1 shows a partial scene of a VR application designed for training a laboratory user to follow the correct protocol while conducting experiments. In such an application, a user may accidentally drop the beaker on the ground while performing the experiment. The notion of gravity is typically handled by the in-built physics engine in Unity as well as Unreal. So, it becomes difficult for the developer to log such interactions even though they are relevant. As the scenarios get more complex, the role of the underlying game engine increases. Given such limitations in logging certain user interactions in VR applications, the direct utilization of standard logging frameworks like log4j in the VR domain is near impossible. Hence, there exists a need for a logging framework specifically tailored to the VR domain.

In this paper, we introduce a logging framework called VRSLOG that aims to automate the tracking of object properties along with developer-defined logs. VRSLOG is a plug-and-play logging framework for VR scenes which focuses on the bare minimum attributes to capture user behavior as suggested by the meta-model for VR systems (Karre et al., 2023). We

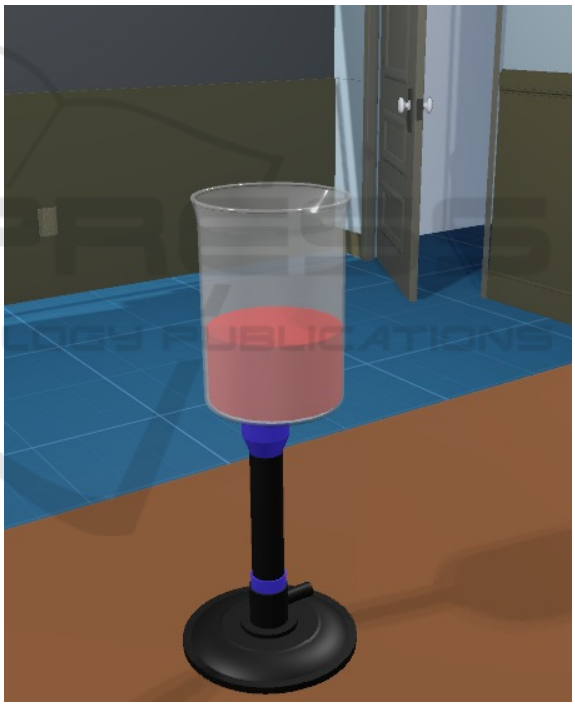

Figure 1: Chemical beaker on a Bunsen burner.

identify 2 key properties that VRSLOG must have for addressing the problem and facilitating the evaluation of interactions with only log files:

1. **Generalizability:** It must be fit for use across diverse VR scenes rather than a specific scenario. Further, it should not require framework-level modifications to ensure it works across multiple scenes.
2. **Conformance:** It should facilitate conformance

[2]VR Solutions with Unity

[3]Unreal Engine for Extended Reality (XR)

checks on the generated logs, comparing them against a predefined sequence of events.

VRSLOG is built with the above two key properties as the main focus. Even though, we demonstrate the logging framework using Unity as the underlying game engine in this paper, it can easily be applied to other scenarios with different game engines, editors, and languages. It's possible to extend the use case of the framework to that of general 3D game development which has slightly similar requirements to that of VR. However, we choose to focus solely on VR scenarios since both differ in the specifics such as point-of-view offered, hardware capabilities, immersiveness, etc.
Overall, in this paper, we introduce a novel logging framework with the following key contributions:

- **Generalizability and Ease of Use:** The proposed framework can be used with a diverse set of VR scenes without requiring changes to the scene itself.
- **Portability Across Technologies:** It is compatible with multiple VR editors.
- **Conformance Checking:** The framework supports functionality for verifying conformance to predefined sequence of events.

The rest of the paper is structured as follows. Section 2 presents the related work, and Section 3 presents our proposed logging framework VRSLOG. Section 4 describes the experiment scene along with a basic implementation. Section 5 discusses the results obtained from our experiment, and Section 6 highlights the current study's limitations and possible directions for future work. Section 7 concludes the paper.

2 RELATED WORK

Multiple studies have been done on improving and analyzing logging methods in the literature. Some studies have explored the application of rigorous Software Engineering practices to logging within typical software systems (Jia et al., 2018; King et al., 2017; Veeraraghavan et al., 2011; Zhang et al., 2011; Chen, 2019). The findings from these studies demonstrate that the application of Software Engineering practices to logging has led to improvements in logging methodologies. In a general logging study, Li et al. analyzed the benefits and costs of logging along with the strategies to balance them by surveying developers directly. These strategies included identifying different log levels, identifying critical and unexpected parts/outcomes of the system, and reducing repeated logging (Li et al., 2021). Rong et al. studied the trade-off between extensive logs and the overhead cost. Additionally, their work showed that although logging is an important practice, it's still largely done in an ad-hoc manner based on the individual experience of developers (Rong et al., 2017). Gu et al. reinforced the idea of having a standardized logging format in their work. They observed an increase in the effectiveness of log analysis when a common format was followed. Their work also identified performance overhead as one of the major issue while logging, and introduced ways such as dynamic logging as a potential solution (Gu et al., 2023). Even though the logging framework discussed in our paper focuses on the VR domain, many of the benefits, challenges, and potential solutions identified in the above-related works are applicable. For instance, dynamic logging can be applied to the VR domain to control the granularity of logging during a particular session.

Louto presented an extensive study in the systematic literature review on user logging in VR. The work argued that research on logging development in VR is rare and even when logging is done for research purposes, the details of the method are not discussed extensively. Further, the work suggests that majority benefits of logging are due to log analysis (Luoto, 2018). This helps us establish that while capturing data is necessary for logging, the major focus should be kept on the analysis of these logs.

Some studies on VR Logging have attempted to provide a solution to log user behavior while limiting their scope to a very specific use case for VR. Ritchie et al. investigated capturing user rationale and intention in a non-intrusive manner through user logging. However, their proposed solution was limited to the context of the example that they used, cable organization. This is a major drawback since their framework must be changed and adapted manually to each specific use case (Ritchie et al., 2008). Similarly, Kloiber et al. presented a solution closely tied with their scenario of assembly simulation. While their approach allows users to manually simulate an assembly process while tracking their performance, it creates high coupling between the VR scenario and the logging system (Kloiber et al., 2021). VRSLOG alleviates this issue by focusing on generalizability to ensure that the solution can be integrated into multiple use cases with minimal extra effort on the developers' part.

Belfore et al. utilized Virtual Reality Modeling Language (VRML) and Java servlets to capture the state information of all the objects in the a scene (Belfore and Chitithoti, 2000; Carey, 1998). However, capturing the general state and properties of all the objects present in a scene is not an effective

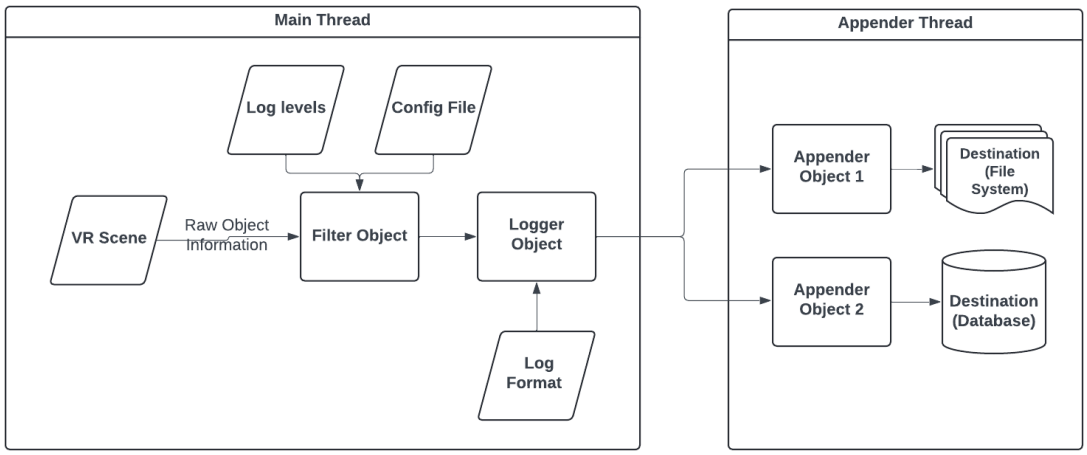

Figure 2: Workflow of VRSLOG.

solution as it can result in performance degradation of the VR scene due to the computation involved.

Pathmanathan et al. presented a system that utilizes 3D eye tracking data from real-world environments (Pathmanathan et al., 2023). Their proposed solution works well when in-depth analysis of the VR scene is required since they capture relatively complex and large amount of data. However, this makes data processing difficult and may necessitate the presence of an expert for effective understanding. VRSLOG can compliment their work by facilitating filtering, using standardized log format, log levels, multiple appenders, etc.

A few other unique approaches for log analysis have also been proposed. Flotynski et al. explored the challenge of logging interactions in immersive systems by employing temporal relations, known as 4D Fluents, to represent information. They proposed a method of creating and assigning time slices to objects as their properties change over time (Flotyński and Sobociński, 2018). Their work can serve as a foundation for others to improve upon. In contrast, the work presented by Hubenschmid et al. provides a comprehensive solution which can be used for detailed analysis of the VR scene from both an internal and external perspective thus providing the benefits of both (Hubenschmid et al., 2022). Their work focuses on advanced replay mechanisms and in-depth analysis of the data. Similar to the work done by Pathmanathan et al., VRSLOG can compliment their framework by retaining the benefits of both approaches. In the current work, however, we have avoided capturing complex data as our goal is to build a framework that be used to conduct simple conformance checks. Capturing more complex data can make data post-processing as well as analysis difficult in this scenario.

Buche et al. also introduced a novel approach for building Intelligent Tutoring Systems in virtual environments. They proposed MASCARET, a model capable of organizing interactions between agents and providing these agents with reactive, cognitive as well as social abilities (Buche et al., 2003). This generic organizational model can be applied upon scenarios to help users identify the important agents and properties of the scenario. In this manner, MASCARET can complement our logging framework VRSLOG by allowing users to describe the scenario details and other relevant information with ease.

3 LOGGING FRAMEWORK

Typically, VR developers define the themes, expected actions, and responses of VR scenes. However, the potential action-responses of a VR scene participant can be unpredictable and thus, it may not be feasible to create an exhaustive set. They may not be limited to the behaviors explicitly defined by the VR developer (as discussed previously in the chemical laboratory example). Logging such undefined system-level interactions makes VR development difficult for developers and may shift their focus from creating quality VR scenes to data logging. In this work, we strive to bridge this gap by introducing a novel logging framework, VRSLOG.

VRSLOG's is based on the bare minimum model template proposed by Sai et al. (Karre et al., 2023). Their work provides a template for the minimalistic set of elements and relationships that must exist in a VR application. The meta-model template proposed by them facilitates specification and development of VR applications. They identify different model el-

ements such as *Article* (any 3D object with specific dimensions and physical properties), *Action*, *Behavior*, and other model elements as the basic building blocks of any VR software system. The model template can be extended to various domain-specific or application-specific elements.

VRSLOG can work with different VR editors and languages. The framework can also be implemented easily as packages across different editors since the requirements are kept minimal to ensure generalizability. This shows the portability of VRSLOG across various VR editor technologies. In its current form, VRSLOG supports the following methods for logging:

1. **User-Defined Logging at Multiple Levels:** This is the same as traditional logging frameworks. VRSLOG allows the VR developer to insert their own log statements and control the level of logging thereby providing flexibility and control over the logs that are generated.
2. **Auto-Logging Through Configuration File:** This allows the user to define a configuration file that outlines various objects and properties that are of interest to the user. Properties such as position, and rotation, which are essential for defining the state of an object, are captured automatically at definite intervals or after the occurrence of specified events. While this method does not allow for as much control over the logs as user-defined logging, it offers a way for the user to define directives which the system uses to automatically log information and track object behaviors.

Furthermore, to facilitate a streamlined and simple integration of VRSLOG within VR scenes, minimal changes (if any) are required to the existing code for logging purposes. Unlike other logging frameworks, the developer is not required to author the log statements manually as auto logging is facilitated through our configuration file, though it is possible to do so if required. Figure 2 presents the overall workflow of our system further described in this section.

The configuration file illustrates the auto-logging behavior of VRSLOG offering developers significant control on customizing logging. Developers are required to specify the names or references of the objects that are to be tracked along with the scope and frequency of tracking them individually. Following are the detailed properties that are required as part of the configuration file for a given object:

- **Position:** This property captures the global coordinates of the object at a particular instant along with the object name and time of capturing the log. We additionally allow the user to define the minimum time/displacement after which a log should be registered for the object. For example, if a log was registered when an object was present at (0,0,0) and we have defined the minimum displacement for logging position to be 1 unit, then the object will need to move out of the unit sphere centered at the origin to register another log message. If neither the minimum time nor minimum displacement is provided by the developer in the configuration file, the filter object assumes a minimum time of 1 second for this property by default.
- **Rotation:** This tracks the 3D vector representing the rotation of the object around its own axes. The additional options provided to the developer are similar to those described for position previously.
- **Seen:** This property tracks whether an object is currently visible to the VR user and registers a log whenever an object enters or leaves the field of view of the user.
- **Interaction:** This property tracks whether any interaction/input has been registered by the user on an object. This interaction can be made through controllers or hands directly depending on the VR scene.

Although enabling logging of more specific object properties might ease the developer's task, we refrain from doing so to maintain the generalizability across various VR scenes and editors. These properties enable us to record essential aspects of the objects within the virtual environment. We track position, and rotation as they are mandatory for defining the absolute location and orientation of an object in the VR scene. In contrast, the "seen" property captures an object's location relative to the user, distinguishing between objects that are visible to the user and those that are not. This distinction is important because we anticipate that the user will primarily respond to objects within their current field of view.
All of the logs generated by automatic logging are set to the "Auto" log level in order to suppress the generation of these logs, developers must explicitly define it within the configuration file.

VRSLOG is divided into 3 major modules: (a) Filter Object, (b) Logger Object, and (c) Appenders.

3.1 Filter Object

This component is primarily tasked with distinguishing between different logging levels. We define the following levels for our custom logs:

1. **Info:** This log level is used to provide information about what is part of an application's regular operation, such as a startup/shutdown.

2. **Debug:** This log level is used for diagnostic information in the development process that is not necessarily harmful to the software.
3. **Warn:** This log level can be a precursor to an error and indicates that an unusual situation might have been detected.
4. **Error:** This log level is used to track when the VR scene has entered into an undesired state that should not be possible if the software was working as intended.
5. **Off:** This log level denotes that no manual logs will be registered by the logger and is the highest possible rank.

Additionally, we introduce an additional log level **"Auto"**, which contains the log messages generated through our auto-logging system based on the configuration file. Apart from distinguishing between log levels, the filter object is also tasked with filtering object information for automatic logging based on the configuration file provided by the user. The configuration file contains the names/references of the objects the developer wants to track, the desired properties to log, and other such necessary details. The filter object assumes default values for any missing entries.

3.2 Logger Object

The logger object is the central component of VRSLOG, which receives filtered log information from the filter object. It serves as the connection between raw information and the appenders and is responsible for formatting the logs to ensure format consistency between logs. The logger object can be associated with multiple appenders to allow logging to multiple locations at the same time through multi-threading. This significantly decreases the performance overhead faced by the system.

3.3 Appenders

Appenders are responsible for printing logging messages to multiple locations, such as consoles and files, and even for sending these messages over the network. Appenders are independent of each other's functioning and the only relation between them is that their source of data is the same. Furthermore, since appenders perform the write operation, which can be costly, we utilize threading to enhance the performance. The following types of appenders are supported by the framework:

1. **Remote Appender:** It uses REST to send log messages to a remote server.
2. **File Appender:** It appends the log events to a file present locally on the device.
3. **Console Appender:** Console Appender appends the log events to the console that the program currently has access to.

These appenders find their use in different scenarios. For example, a console appender may be used for printing custom log messages by the developer when running the scene on an editor instead of the Head Mounted Display during its development phase whereas a remote appender may be used during the execution of a scene for collecting all the logs at a central location. REST (Fielding and Taylor, 2000) protocol is used here to maintain generalizability as it is widely supported. Other mechanisms (for instance, wired) may be faster but they may also limit the scope of the framework. In cases when high-frequency tracking information is required, File Appender can be used to reduce the network overhead created by a remote appender. Considering the potential performance impact of intensive logging operations, especially when using a remote appender, we optimize by making use of multi-threading and separating expensive POST requests from the rest of the computation.

We integrated multi-threading by maintaining a thread pool used for queuing operations. The thread pool is configured with a limited number of threads in order to ensure an upper bound on thread creation by the system. This helps avoid scenarios where an unbounded number of threads can lead to excessive memory consumption. Further, this approach also addresses the overhead associated with creating a new thread for each operation. However, parallelization may lead to the logs being de-synced as events may be written in an arbitrary order by different threads. In order to address this, we assign a serial number to each log message and maintain a priority queue on the server-side which receives the logs from the system and consumes them serially. In this manner, we can queue the expensive POST requests on other threads allowing parallelism in logging. This helps in separating scene execution and the most computationally expensive parts of our logging approach.

The three modules along with the configuration file greatly enhance the flexibility of logging operations within the system. As various aspects such as filtering, text formatting, and log destination are managed by distinct components within VRSLOG, it becomes effortless to change any one of them independently without affecting the others. Moreover, developers need only declare and define the relevant objects (filter object, configuration file, logger object, appender) to integrate this framework into their VR

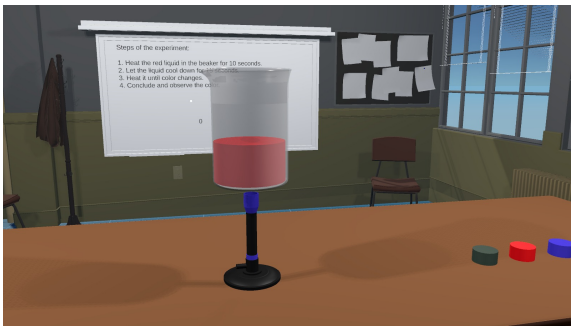

Figure 3: Anteroom with gloves and wash basin.

scene, thereby eliminating the necessity for modifications to the existing software/code base.

4 AN EXAMPLE IMPLEMENTATION

To demonstrate the efficacy of VRSLOG in capturing user behavior and enabling evaluation, we implemented a prototype within the Unity engine, which uses C# programming language. The source code is made available as part of our resources [4]. We define a VR scene aligning with the following conditions:

- The scene requires the user to follow a sequence of steps that reflect multiple state changes rather than simply checking the end state as a sufficiency criteria for evaluation.
- The scene must have a need for user evaluation which would otherwise require oversight from a human supervisor.

In light of the above constraints, we have chosen to use a scene in a chemical laboratory setting. In this scene, the user is required to follow the prescribed protocol to enter the laboratory and perform an experiment (figure 3). The user is first required to enter the anteroom, wash their hands at the basin, and then put on gloves (figure 4) before proceeding into the main laboratory area. Additionally, there are two separate gates to the laboratory: one designated for entry and the other for exit (figure 5). After entering the laboratory, the user is expected to first read the given instructions, and then perform the experiment accordingly while making use of the stopwatch provided. This stopwatch can be operated using the three buttons visible on the table. After completing the experiment, the user is required to exit through the designated exit gate, discard their gloves in the dustbin, and wash their hands before leaving. We incorporate VRSLOG into the environment only after initially developing the virtual laboratory scene. This sequence was intentionally followed to highlight the minimalistic nature of changes needed to the preexisting code base and objects in order to enable logging capabilities.

In order to plug our logging framework in their scene, a developer would need to create a ”logger” script that constructs the different components of the framework (FilterObject, LoggerObject, AppenderObject(s)) and connect them via class attributes. They would also be required to pass relevant parameters such as the config file path and log level when constructing FilterObject and then call the *FilterAndPass* method of the FilterObject in the Update function provided by Unity. Similarly, if VRSLOG is to be applied on applications written in Unreal Engine, they can follow a similar approach and call the *FilterAndPass* method in the *Tick(float DeltaSeconds)* function provided by Unreal. Although, the application is implemented using C++ in Unreal (instead of C# for Unity), the framework itself is not dependent on any programming language. This makes it possible to implement it in any language easily.

Table 1 shows the properties of the different objects present within the scene. A brief reason for selecting the component and its respective properties is also provided in the table. Further, we set the log level to ”Info” and attach a File Appender and a Remote Appender to the Logger Object. Finally, we conduct a runtime session with a user and generate the logs which can be used for analysis.

5 DISCUSSION

The log file obtained from VRSLOG can be used for various purposes. Currently, the analysis is a manual process where the logs are inspected to understand what transpired in the scene. This method utilizes logs to circumvent the requirement of a human supervisor. Due to the large nature of the log file, we dis-

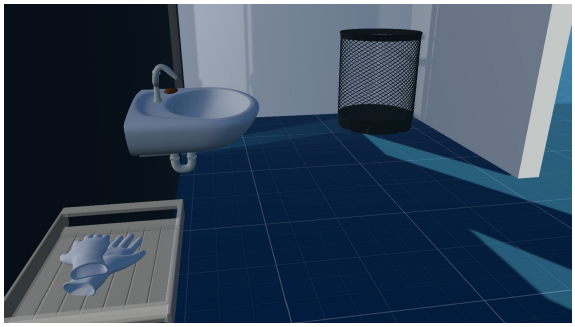

Figure 4: Instructions and the experiment setup.

[4]https://doi.org/10.5281/zenodo.13789213

Table 1: Logging Properties of Experiment Components.

Component	Property	Reason
Faucet	Interaction	Used for washing hands before and after the experiment
Lab Gloves	Interaction	Worn before the start of the experiment
EntryLabel	Seen	Labels the entry gate
ExitLabel	Seen	Labels the exit gate
Stopwatch Button	Interaction	Used for turning stopwatch on and off
Reset Button	Interaction	Used for resetting stopwatch
User Capsule	Position	Used for tracking user's position
Blackboard	Seen	Instructions and timer readings are present on it
Bunsen burner	Interaction	User for heating the beaker
Beaker	Seen	Contains the solution for experimenting
Dustbin	Interaction	User for discarding the gloves

cuss a small excerpt from the log. We have also presented a part of the log file along with the discussions for easier understanding of the workings of our implementation. The complete log file generated as part of this example is available as part of our resources.[5].

The log data for the initial part of the scene execution reveals that the user first interacts with the faucet and then with the gloves. The gate labels then enter the field of view of the user in order for them to find the entry gate after which they enter the main room. The data then reveals that the blackboard with instructions remains in the field of view of the user for a significant period. We can infer that the user spent this time looking at the blackboard as this behavior aligns with the expected workflow, wherein users would need to read the steps of the experiments at the start.

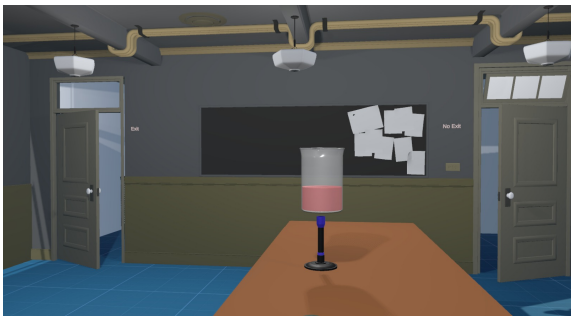

Figure 5: Entry and exit gates.

[5]https://doi.org/10.5281/zenodo.13789213


```
{
    {
        "timestamp": 7.871,
        "name": "Capsule",
        "property": "position",
        "value": [4.98,1.09,-5.17]
    }
    {
        "timestamp": 14.256,
        "name": "Faucet_1",
        "property": "interaction",
        "value": [2.74,1.1,-3.85]
    }
    {
        "timestamp": 19.607,
        "name": "Glove_1",
        "property": "interaction",
        "value": [2.66,1.03,-4.96]
    }
    {
        "timestamp": 22.042,
        "name": "ExitLabel",
        "property": "seen",
        "value": true
    }
}
```


Afterwards, the log data shows that the user turns on the burner for the first step of the experiment and starts the stopwatch right after to ensure minimal error. It is clear from the logs that the beaker remains in the field of view of the user for the period when it's being heated. This also aligns with our expected behavior from the user, since they are expected to monitor the experiment closely at all points of time.


```
{
    {
        "timestamp": 55.664,
        "name": "Burner_1",
        "property": "interaction",
        "value": [-0.98,0.95,-4.64]
    }
    {
        "timestamp": 55.98,
        "name": "StopwatchButton",
        "property": "interaction",
        "value": [-1.48,0.86,-4.57]
    }
    {
        "timestamp": 65.58601,
        "name": "StopwatchButton",
        "property": "interaction",
        "value": [-1.62,0.86,-4.51]
    }
```

```
    {
        "timestamp": 66.304,
        "name": "Burner_1",
        "property": "interaction",
        "value": [-0.76,0.95,-4.65]
    }
}
```


When the user turns off the heating, however, the log data indicates that the stopwatch was started a few seconds late hence introducing a significant time delay between the two events. This extra time delay causes the solution to cool down for longer than the prescribed time which can be hazardous in certain cases. This indicates a clear error in the workflow employed by the user.


```
{
    {
        "timestamp": 66.304,
        "name": "Burner_1",
        "property": "interaction",
        "value": [-0.76,0.95,-4.65]
    }
    {
        "timestamp": 66.68101,
        "name": "ResetButton",
        "property": "interaction",
        "value": [-1.91,0.86,-4.4]
    }
    {
        "timestamp": 69.98,
        "name": "StopwatchButton",
        "property": "interaction",
        "value": [-1.59,0.86,-4.53]
    }
}
```


Finally, the user starts heating the beaker again for the third step. However, the log data indicates that the beaker remains non-visible for a significant period. This means that the user had not been monitoring the experiment while it was being heated. This indicated another error in the workflow of the user which may prove to be a hazard since certain chemicals must always be monitored closely.


```
{
    {
        "timestamp": 84.052,
        "name": "Burner_1",
        "property": "interaction",
        "value": [-0.82,0.95,-4.56]
    }
    {
        "timestamp": 84.68101,
        "name": "ResetButton",
        "property": "interaction",
        "value": [-1.94,0.86,-4.32]
    }
    {
        "timestamp": 87.708,
        "name": "Beaker_1",
        "property": "seen",
        "value": false
    }
}
```


The analysis presented above demonstrates the ability of our logs to capture user behavior. We can successfully record both the expected and unexpected behavior of the user and distinguish between the two. This showcases VRSLOG's ability to not only capture user behavior but for evaluating it as well.

6 LIMITATIONS AND FUTURE WORK

The primary limitation of our work remains that we demonstrate a manual evaluation of our logs to determine user behavior. While this method circumvents the requirement of a human supervisor to oversee the VR session through logs, it still involves a human in the loop presenting a potential challenge. As part of our future work, we are planning to extract the relevant data from the logs without involving a human, thus automating the evaluation process and generating relevant insights. In continuation, we are also exploring approaches to increase the scalability of logging-based evaluation.

We are also looking at proposing standards similar to SCORM[6] and xAPI[7] (interaction standards for web and mobile e-learning content) for VR software using VRSLOG. It facilitates the standardization of VR content consumed through e-learning or simulations. This will significantly impact the phase shift of conventional e-learning content towards the VR domain.

There are several advantages to building a logging framework that can log VR applications. In addition to teaching and learning, we plan to build an automated conformation conformance checking of user behavior in future. Further, logging in VR applications can provide extensive datasets that can be used in training models to better understand user behavior, predict actions, or enhance immersive experiences by

[6]SCORM: Technical standard for e-learning
[7]xAPI: Experience API

making the environments more responsive and adaptive.

7 CONCLUSION

Despite the large potential of VR in various domains, the scale of its application in the market remains relatively small even today. A possible reason for this could be the ad-hoc development landscape in the VR domain which lacks any standardized development methodologies. This makes the development and maintenance of VR applications very difficult and costly. The application of rigorous Software Engineering practices in the VR domain is necessary to mitigate these issues. This is especially true with the rise of enterprise VR and the increase in the scale of VR software systems in recent years. We highlight the advantage that VR holds over traditional 2D screen interfaces for capturing user behavior in our work and propose VRSLOG, a logging framework that can aid developers in the process.

We reason that logging in VR is a non-trivial task due to the dynamic nature of the objects, and the existence of interactions and events that are not predefined by the developers. We take inspiration from existing frameworks like log4j, which work for traditional software systems, to build a solution for VR logging. We focus on two major properties while developing the framework, the first of which is generalizability. The second property is the ability to conduct conformance checks on the generated log files. To enforce generalizability, we build VRSLOG while focusing on its compatibility with a minimal template of VR software (Karre et al., 2023).

Finally, we implement a prototype of the suggested framework in Unity and generate logs from a VR scene. The VR scene used for this purpose is a training application for laboratory technicians. We analyze the generated logs to demonstrate how they can be used for capturing user behavior circumventing the need for a human supervisor.

REFERENCES

Belfore, L. and Chitithoti, S. (2000). An interactive land use vrml application (iluva) with servlet assist. In *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*, volume 2, pages 1823–1830 vol.2.

Buche, C., Querrec, R., De Loor, P., and Chevaillier, P. (2003). Mascaret: pedagogical multi-agents systems for virtual environment for training. In *Proceedings. 2003 International Conference on Cyberworlds*, pages 423–430.

Carey, R. (1998). The virtual reality modeling language explained. *IEEE MultiMedia*, 5(3):84–93.

Chen, B. (2019). Improving the software logging practices in devops. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 194–197.

Fielding, R. T. and Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis. AAI9980887.

Flotyński, J. and Sobociński, P. (2018). Logging interactions in explorable immersive vr/ar applications. In *2018 International Conference on 3D Immersion (IC3D)*, pages 1–8.

Gu, S., Rong, G., Zhang, H., and Shen, H. (2023). Logging practices in software engineering: A systematic mapping study. *IEEE Transactions on Software Engineering*, 49(2):902–923.

Hubenschmid, S., Wieland, J., Fink, D. I., Batch, A., Zagermann, J., Elmqvist, N., and Reiterer, H. (2022). Relive: Bridging in-situ and ex-situ visual analytics for analyzing mixed reality user studies. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA. Association for Computing Machinery.

Jia, T., Li, Y., Zhang, C., Xia, W., Jiang, J., and Liu, Y. (2018). Machine deserves better logging: A log enhancement approach for automatic fault diagnosis. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 106–111.

Karre, S. A., Mathur, N., and Reddy, Y. R. (2019). Is virtual reality product development different? an empirical study on vr product development practices. In *Proceedings of the 12th Innovations in Software Engineering Conference (Formerly Known as India Software Engineering Conference)*, ISEC '19, New York, NY, USA. Association for Computing Machinery.

Karre, S. A., Pareek, V., Mittal, R., and Reddy, R. (2023). A role based model template for specifying virtual reality software. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA. Association for Computing Machinery.

King, J., Stallings, J., Riaz, M., and Williams, L. (2017). To log, or not to log: using heuristics to identify mandatory log events–a controlled experiment. *Empirical Software Engineering*, 22:2684–2717.

Kloiber, S., Settgast, V., Schinko, C., Weinzerl, M., Schreck, T., and Preiner, R. (2021). A system for collaborative assembly simulation and user performance analysis. In *2021 International Conference on Cyberworlds (CW)*, pages 93–100.

Li, H., Shang, W., Adams, B., Sayagh, M., and Hassan, A. E. (2021). A qualitative study of the benefits and costs of logging from developers' perspectives. *IEEE Transactions on Software Engineering*, 47(12):2858–2873.

Luoto, A. (2018). Systematic literature review on user logging in virtual reality. In *Proceedings of the 22nd International Academic Mindtrek Conference*, Mindtrek '18, page 110–117, New York, NY, USA. Association for Computing Machinery.

Pathmanathan, N., Öney, S., Becher, M., Sedlmair, M., Weiskopf, D., and Kurzhals, K. (2023). Been there, seen that: Visualization of movement and 3d eye tracking data from real-world environments. *Computer Graphics Forum*, 42(3):385–396.

Ritchie, J. M., Sung, R. C., Rea, H., Lim, T., Corney, J. R., and Howley, I. (2008). The use of non-intrusive user logging to capture engineering rationale, knowledge and intent during the product life cycle. In *PICMET '08 - 2008 Portland International Conference on Management of Engineering & Technology*, pages 981–989.

Rong, G., Zhang, Q., Liu, X., and Gu, S. (2017). A systematic review of logging practice in software engineering. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 534–539.

Veeraraghavan, K., Lee, D., Wester, B., Ouyang, J., Chen, P. M., Flinn, J., and Narayanasamy, S. (2011). Doubleplay: parallelizing sequential logging and replay. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, page 15–26, New York, NY, USA. Association for Computing Machinery.

Zhang, C., Guo, Z., Wu, M., Lu, L., Fan, Y., Zhao, J., and Zhang, Z. (2011). Autolog: facing log redundancy and insufficiency. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11, New York, NY, USA. Association for Computing Machinery.