A Latency and Latency Jitter Simulation Framework with OSVR

Jan-Philipp Stauffert* University of Würzburg
Florian Niebling† University of Würzburg
Marc Erich Latoschik‡ University of Würzburg

ABSTRACT

Latency is a pressing problem in Virtual Reality (VR) applications. Low latencies are required for VR to reduce perceptual artifacts and cyber sickness. Latency jitter, i.e. variance in the pattern of latency, prevent coping mechanisms as users can't adapt.

Low latency is a fundamental timeliness requirement to reduce the potential risks of cyber sickness and to increase effectiveness, efficiency, and user experience of Virtual Reality Systems. The effects of uniform latency degradation based on mean or worst-case values are well researched. In contrast, the effects of latency jitter, the distribution pattern of latency changes over time has largely been ignored so far, although today's consumer VR systems are extremely vulnerable in this respect.

In this paper, we propose to create a model of latency and latency jitter with empirical distributions as well as a method of using those models to inject latency. The process of creating a latency model is demonstrated with an example of gathering and converting latency samples from an example application. We show how to simulate latency and motivate to use it in middleware to allow for less intrusive latency effect evaluations.

Index Terms: D.1.3 [Programming Techniques]: Concurrent Programming—Parallel programming; D.4.8 [Operating Systems]: Performance—Measurements; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

1 INTRODUCTION

Virtual Reality applications often consist of multiple components to handle input processing, simulations, artificial intelligence, or rendering etc. Non-functional software quality requirements like modularity, maintainability, and reusability can have an unforeseeable impact on the temporal behavior of software, especially for a Real-Time Interactive System (RIS), i.e., in Virtual, Augmented, and Mixed Reality (VR, AR, and MR) and computer games. Due to the complexity of many RIS applications, they are often split into different parts to foster cohesion and decoupling. To exploit today's multi-core and multi-CPU architectures and to avoid unnecessary blocking, these parts often will be executed concurrently or they will be completely distributed [2, 12].

Each computation in each application part takes time with the orchestration and synchronisation adding additional overhead. The computations cause a latency between input data entering the system and output data based on them exiting the system.

In theory, the created latency can be determined deterministically by inspecting the system and the control paths taken. In practice, today's hard- and software is too complex to determine how its latency behaves. It is agreed upon that more latency, i.e. a bigger time discrepancy between input and the resulting output, lead to a decreased performance operating a system and especially in VR to decreased immersion and acceptance due to an increase in cyber sickness [10, 18].

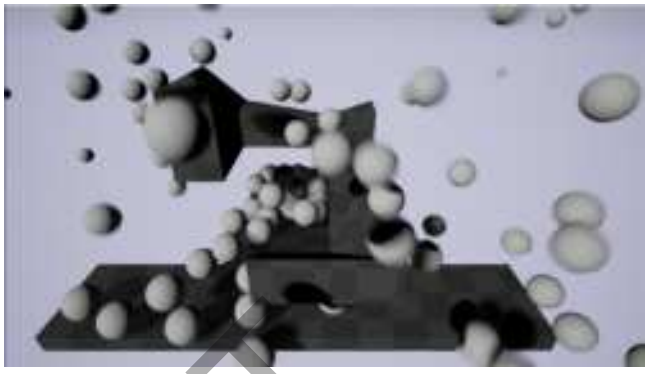

Figure 1: Example scene to create latency measurements. Spheres are spawned every second and collide with each other as well as with the environment.

While time invariant latency is well researched, this paper focuses to create a model for latency jitter. Latency jitter describes latency that changes over time, here with the focus on latency spikes. Latency jitter leaves the user unable to adapt as it is constantly changing.

To provide a tool for further research of latency spikes, this paper proposes to use the here introduced latency models as a basis to simulate latency. This simulated latency can be inserted into selected parts of an application to enable detailed observations.

The contribution of the work presented here are as follow:

- Description of how to create a model for latencies from measured data
- Description how to use a latency model to simulate latency

This paper is structured into first discussing related research, followed by the description of our method to model and simulate latency and latency jitter. We will then introduce a latency injection system that allows for the introduction of latency and latency jitter based upon our model without changes to the VR application. In the end there is a discussion of the findings with a conclusion and ideas for future work extending the research presented here.

2 RELATED WORK

Simulator sickness is a problem of VR applications where users are experiencing symptoms such as nausea [11]. While some users are more sensible, there are certain factors that make simulator sickness worse for most users. Visual delay was found as a major contributing factor already in early simulators [8]. Latency also influences the performance of test subjects both if time variant latency is added [10] or for latency spikes [18]. The assumption is consequently that latency spikes influence simulator sickness with a similar impact as the better researched time invariant latency.

The performance of VR applications is usually assessed by measuring motion-to-photon latency which tracks the time between an input on a certain input channel and the time it takes to show its

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:marc.latoschik@uni-wuerzburg.de

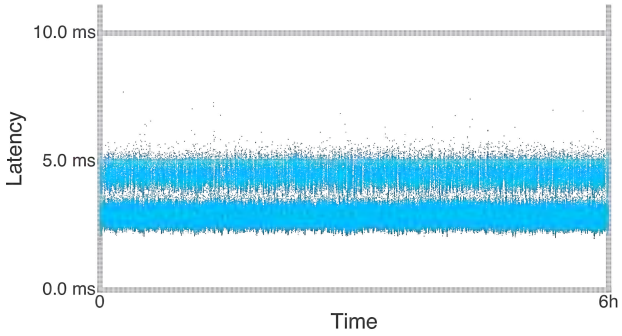

Figure 2: Latency measurements of the physics simulation in our test scene, scaled to show the mean latency with most of the samples and above a region with sparser sample density.

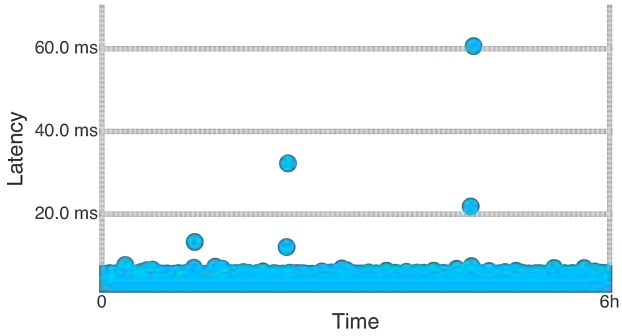

Figure 3: Latency measurements of the physics simulation in our test scene, scaled to show all values with bigger point size to make the outliers visible.

effect on a display. Approaches to measure this latency are sine fitting [17], light sensing [5] and automated frame counting [9]. In this paper, the focus is on latency that is contributed by the VR framework and its internal software processes with their interactions, a subset of motion-to-photon latency. While there are many optimization techniques for the rendering stage like frameless rendering [3], latency at the application stage is yet less researched.

Latency has been injected into virtual environments to evaluate its effect on task performance, presence, and other factors. Most of these experiments delay tracker input data by a controllable amount of time units — frames or multiples of the tracker sampling rate — by employing a ring buffer or other FIFO data structures either inside the tracker itself, its software driver, or the VR application. Experiments are then performed with different, yet constant per experiment, amounts of latency artificially injected into the system.

Ellis et. al. tested distinguishability of changes in latency for hand [7] as well as for head [6] movements. They employ custom tracker drivers to ensure a low base latency and to provide the ability to add custom latency to their input devices. Building on this work, Mania et. al. test sensivity to head tracking latency in virtual environments [13]. Meehan et. al. studied the effects of latency on presence in stressful virtual environments [14]. To do user studies with different latency settings, they adapted their VRPN client implementation to delay tracker input data by a fixed amount of time to add constant end-to-end latency to their system, enabling controlled experiments with 50ms and 90ms of latency respectively. Other studies that control latency, e. g. performed by Allison et. al. [1], or more recent work on latency control by Papadakis et. al. [15] as well as by Waltemate et. al. [19], also only allow for the insertion of constant latency by delaying tracker input data using ring buffers.

Time invariant latency, however, ignores that latency in applications changes over time. The effects of latency jitter are far less researched as discussed above. To describe the effects that can be observed, a model of time variant latency is needed. Additionally, there needs to be a way to introduce time invariant latency to then allow more research of the effects on the systems and for the users.

3 Method

Distinguishing the work presented here from past research outlined in the previous section, we are presenting a method to inject latency jitter (instead of constant latency) into VR applications, based upon a model created from measurements of existing VR systems, without the need to change the VR application itself.

Our approach is split into two parts:

- Modelling latency by deriving an empirical distribution from measurements
- Using a latency model and a latency injector, based upon established VR middleware, to simulate latency

We discuss latency modelling at the example of timing the physics calculation of a test scene created with the Unreal Engine 4.

Afterwards, latency simulation based on a latency model is discussed at the example of creating an OSVR plugin to delay tracking data without needing to alter the affected application.

3.1 Latency Model creation

The latency model described here is based on latency measurements at the application stage. We explain our approach with example measurements taken with a test scene made with the Unreal Engine 4. This offers measurements relateable to real world VR scenarios.

The measurements are divided into outlier groups to separate expected latencies close to the mean value from the less often occuring latency spikes. Latency spikes are further categorised by recursively applying the outlier detection algorithm as described by Stauffert et al. [16]. For each category of detected outliers, two empirical distributions are derived. One describes the duration of the latency spike. The other describes how much time passes between latency spikes.

Figure 4 shows the process of deriving a model with graphs demonstrating how example measurements are transformed at every step. The following sections will further explain the steps undertaken.

3.1.1 Measurements

The basis for the latency model are latency measurements. Here, we demonstrate how to measure them, and what appearance latency measurements have.

We created a scene with the Unreal Engine 4, where a physics enabled sphere is spawned every frame. The balls collide with the sparse environment and with themself, and get despawned once they fall through holes in the environment. An overview of the scene is given in figure 1.

The engine allows to create objects that are updated at different times during the game loop. Utilising this, we register two functions of an object. One gets invoked before the physics calculation, the other after the physics system has updated. With the two invocations, the elapsed time in between is measured which is taken to create a latency profile for the engine execution.

With this setting, a big part of the computational time of the application to survey is measured. Measurement, however, can be conducted for far smaller parts such as the duration of AI computation for only one object in a scene or one small algorithm.

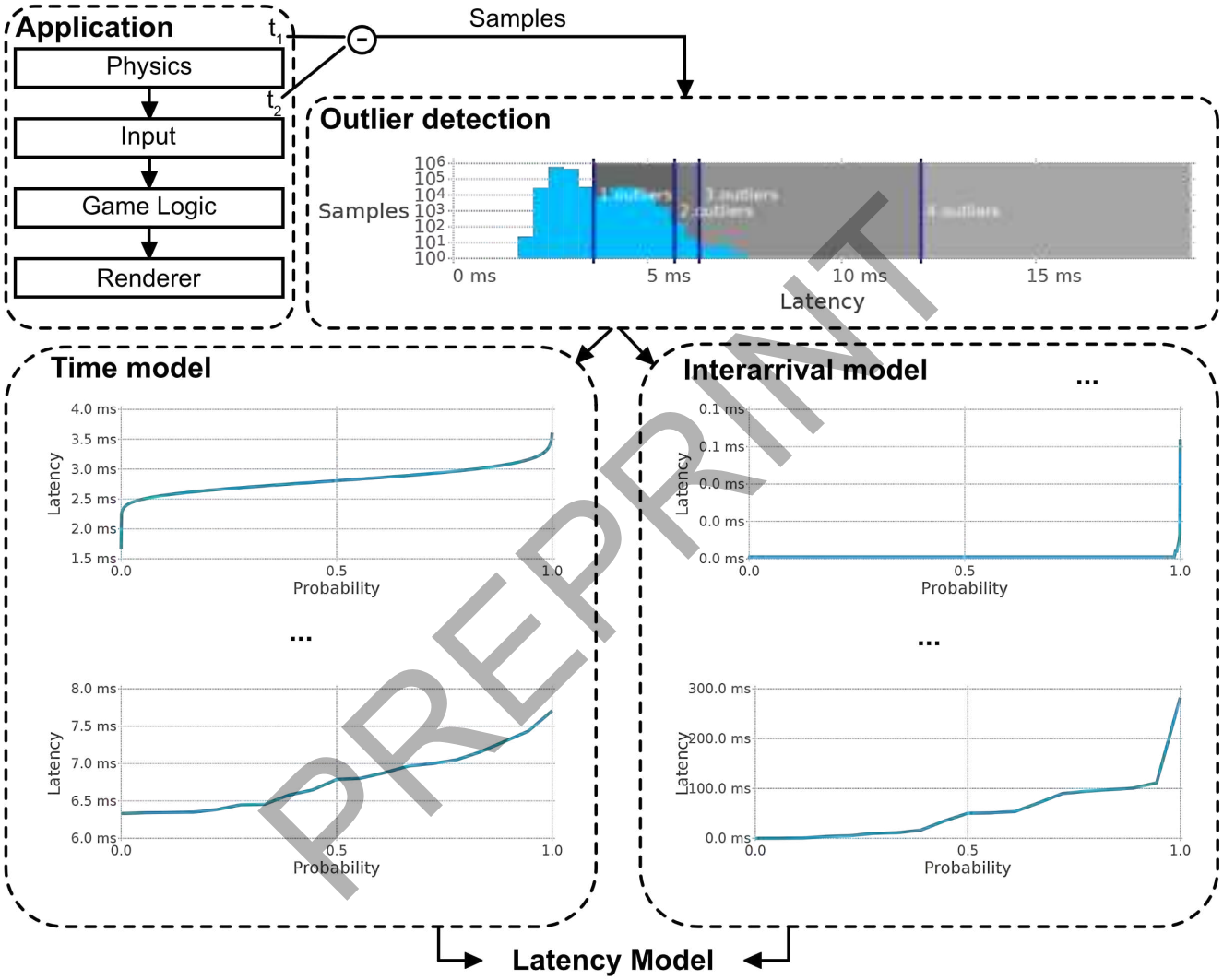

Figure 4: Steps to create a latency model: Latency samples are gathered in the application. Outliers are extracted and sorted into different categories. The distributions for the duration and interarrival times or latencies are expressed by their inverted cdf.

Figure 2 shows a scatter plot of the latencies measured for each frame. Note that most samples gather around a mean with less samples above the mean latency. Figure 3 shows all samples including the extreme outliers. The mean is 2.93ms, variance 0.2ms^2, min 1.6ms, max 60.6ms.

3.1.2 Outlier detection

The measurements are separated into outlier categories. For this, we recursively use the modified z score test to determine outliers as proposed for latency data by Stauffert et al. [16]. They calculate the z-score for each sample to determine if it is an outlier. Once, all outliers are found in the dataset, the algorithm is applied to separate the outliers from the outliers of the outliers. This is repeated recursively until no outliers can be found anymore.

The z-score over all samples x_i is calculated with

$$Z_i = \frac{0.6745(x_i - \tilde{x})}{\text{MAD}} \qquad \text{MAD} = \text{median}(|x_i - \tilde{x}|) \tag{1}$$

Samples with a Z-score larger than 3.5 are detected as outliers. For the example data, after the fourth application of the algorithm, no more outliers can be found.

3.1.3 Distribution derivation

An empirical distribution is derived for each outlier category and the samples around the mean. The distributions are described by their cumulative density function (cdf). For convenient generation of samples from the distribution, the cdf needs to get inverted. The quantile function represents the inverse cumulative distribution function and is faster to compute than first generating the cdf and then inverting it. The quantile function is sampled to avoid the need to save all latency samples but be able to represent the distribution with a small amount of values. Using more samples when sampling the quantile function leads to a more precice representation of the observed distribution.

The result from sampling the quantile function is an array of latency values. To draw a random value that is distributed by the empirical distribution, a random uniformly distributed number is drawn. The random number needs to be between 0 and the number of samples used for sampling the quantile function. This random number is used to index into the array.

The latency model proposed here consists of empirical distributions for each outlier category for both the latency duration and the time in between the observed latency spikes.

3.2 Latency Injection

This section describes how to use a latency model as described above to simulate latency. The discussion is lead with an example of an OSVR plugin where additional latency is simulated for processing tracking data. While the example helps to describe the process, the approach isn't limited to an implementation in a middleware, but can be used to simulate latency in parts of the application itself. We will describe the benefits there are in simulating latency in a middleware like OSVR, before presenting our latency simulation approach.

3.2.1 Placement in middleware

OSVR provides a middleware supporting many devices, presenting a uniform API to applications abstracting away differences. Introducing latency simulation in this layer allows for little intrusion of the application code. This enables the evaluation of different amounts of latency jitter in existing VR systems. The existing software does not need to get modified in the process.

Many VR applications couple physics, logic and the renderer very tightly to guarantee that for each frame, all systems have updated their state. Introducing latency there is more prone to influence the whole application performance. Middleware like OSVR runs parallel to and decoupled from the application. This allows to introduce latency into selected tracking devices without affecting others. Having only one device input modified promises to analyse effects of latency in different modalities with more detail.

Compare figure 5 for an overview where a latency simulating plugin is located within OSVR. OSVR contains plugins that handle the connection to various tracking devices. The data is either directly forwarded to an application or modified by another plugin, called analysis plugin. Introducing latency in an analysis plugin allows for targeted modification of tracking data by only delaying data of one selected device. OSVR works like a dataflow environment, where data is gathered at the input devices, then processed by plugins in a configurable order until it eventually gets delivered to an application. The application doesn't need to be changed as it can't distinguish whether its data stems directly from the hardware plugin or was tampered with on the way.

3.2.2 Algorithm

We implemented a configured device plugin. This is a plugin that can be used multiple times with different configurations to allow binding to different other devices with individual latency behaviour.

The latency plugin simulates a component in the system that receives data, works with it and then forwards it. The delay between receiving and forwarding a dataset d_l consists of the expected delay $\bar{d}_l$ and a time d_s indicating a spike in latency. If the simulator shall only simulate latency spikes, the expected mean latency to simulate $\bar{d}_l$ is set to zero.

The simulator itself introduces latency by simply executing code. Care has to be taken that the simulated latencies are significantly larger than the values that are introduced by the execution of the simulator. We will discuss this and similar problems at the end.

A sample code describing the execution is shown in listing 1: Initially, a time for the first latency spike is determined. The simulator then receives a dataset and decides d_l.

First, it is initialised with $\bar{d}_l$. If the current time is larger than the time, a latency spike was scheduled, a sample for the latency spike length is drawn from the respective distribution. This length is added to the delay time. Afterwards, the time of the next latency spike is computed by adding a randomly drawn sample of the spike interarrival distribution to the current time.

The simulator then delays further processing of the data by other components by sleeping for the calculated time. Note that a sleep only guarantees that the execution is not resumed before the given amount of time has passed. It might take longer than requested until execution is continued.

This method assumes that there is a buffer in place that stores incoming tracking data until they get processed. This buffer might hold only the most recent data and discard older data elements if new ones arrive, or be implemented with a queue. If the simulated working time exceeds the arrival time, the queue might fill up and overflow.

3.3 Evaluation

For first evaluations, we created an OSVR hardware plugin that continuously sends timestamps. These timestamps are received in an OSVR client application and the difference of the received timestamp to the time it is received is saved.

We gather the latency samples collected with the described method with and without an additional latency injector plugin in the pipeline. In total, we compare three different scenarios:

- Without latency injector

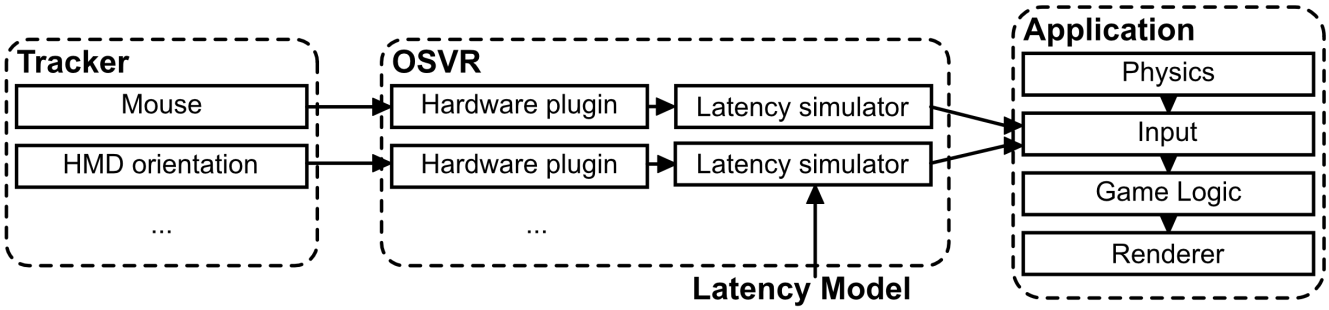

Figure 5: Latency can be injected in the middleware, that administrates the hardware, without needing to change the application.

```
next_spike = now() + interarrival_distribution.draw_value();
while data = receive_data() {
  worktime = mean_work_time;
  if now() + worktime > next_spike {
    worktime += spike_distribution.draw_value();
    next_spike = now() + interarrival_distribution.draw_value();
  }
  sleep(worktime);
  forward_data(data);
}
```

Listing 1: Simple latency simulator

- With latency injector based on a latency distribution that never causes spikes, i.e. reducing the latency injector to be a pass through stage
- With latency injector based on a latency distribution simulating only spikes

We expect to find higher latency outliers with the latency injector in place with more latency outliers when using a latency distribution.

For comparing the two measurements, we use QQ-plots. QQ-plots are used to compare two distributions with each other. Usually, this is done to see how well one distribution fits another. As we already use the quantile function to model latency jitter, the plot shows the quantiles of the measurements without latency on one axis against the quantiles of the measurements with latency on the other axis. A line above the bisector signals more latency spikes with latency injected. See figure 6 for an example of one of these plots.

We find that the mean latency is the same for all cases.

With the latency injector only working as a pass through for messages, there are more extreme spikes with the lower outlier categories exhibiting similar distributions to the run without the added plugin.

Having the latency injector simulating only spikes, there are more spikes visible in the QQ-plots for the outlier categories than without the plugin. The means match and the first outlier category distribution is similar.

This means, our latency simulator is able to simulate latency spikes with little overhead, leaving the latencies around the mean mostly unchanged.

4 Discussion

We proposed a method to model latency and latency jitter with an empirical distribution.

In the introduction, we stated that systems elicit non deterministic latency behaviour due to the complexity of today's computers and applications. Likewise, introducing additional latency into a system can lead to different behaviour. For this discussion, we assume that VR systems consist of different parts that run partly in parallel and exchange state at synchronisation points. The timing of these synchonisation points plays a crucial role of how latency injection into an application is visible to the user.

Latency injected into an application could not change the application at all. If it affects a part of an application that has sufficient buffer time between the time its own execution has finished and the time point it needs to synchronise, this free buffer time can swallow the latency. Similarly, if a part with additional latency injected urges another part to wait but this dependent part has sufficient buffer time, the effect might vanish.

If the effect of added latency does not vanish in the interplay of parts of the application, the effects can be visible in different ways. They can be restricted to simple aspects of the application. An example would be that only the controller inputs lag behind the rest of the application. There could also be a ripple effect where delay in one part of the application causes other parts to miss their assigned time window leading the whole application performance to suffer.

With time invariant latency, the effects are better observable as there will be one pattern emerging as a result. The time variant latency, as observed e.g. in the described example application and modelled here, will exhibit complex effects and will create interference patterns when interacting in a sophisticated system.

This means that after introducing latency, especially time invariant latency, it is mandatory to measure again how the actual effects are.

Using the proposed tools to research latency jitter will allow to better understand the relationship of latency jitter to constant latency. A dejitter buffer [4] as used for packet-switched networks can reduce the jitter at the cost of added constant latency. If latency jitter proves to be worse, this is a tool to alleviate it.

In the future, it will be desirable to measure application stage latency without needing to alter the application but placing the complete benchmarking code into a middleware. For this, it is necessary to receive information from the application. Upon writing this

paper, OSVR doesn't support plugins that take input from the application. There are only plugins that sit between tracking devices and the application. There are no plugins for messages from the application to a device like a force feedback device or an audio device yet.

Once this is implemented, it will be possible to create plugins that detect which message sent by the application belongs to which tracking value. Then, the simulation latency can get determined by the middleware, allowing application stage latency measurements without the need to influence an application at all.

We proposed to use the quantile function for the inverse cumulative density function, which can be directly derived from the samples. This, however, limits the simulation to only produce random numbers that were observed before. Using a kernel density function over the data provides a smoothed probability density function that can be integrated to yield the cumulative density function, which in turn can be inverted. This way around leads to a smoothing of the observed distribution and can also generate values that are close to the observed.

The example data used for the model creation was collected from a specially designed application. As all other applications, it exhibits its own latency pattern. With repeated spawning and despawning of balls, the number of balls in the scene is constantly changing. The available room is restricted to provoke many collisions. This leads to the physics simulation working with a variable number of entities that have different amounts of collisions each frame. The measurements show a latency behaviour with repeated spikes. We did, however, not analyse if the spikes are results of the changing conditions like ball count and collision count from frame to frame or if they have another source. The many objects in our scene could also have lead to lead to the changing numbers being too small in comparison to the overall number of collisions to not be responsible for the variations.

While we proposed a latency injector that adds latency according to a distribution, its own execution adds additional latency. The described sleep to simulate a time of working yields the processor to let other processes work in the mean time, which might or might not be desirable for the application to test. Waking from sleep involves a second context switch which is costly timewise and adds to the latency. A busy loop can be used as an alternative. OSVR needs to gather the measurement from another plugin to pipe it to the latency inducing plugin to afterwards pipe it to either the application or other plugins. This causes additional overhead and therefore latency as described in the evaluation section.

The discussions so far demonstrate that it is possible to measure and induce latency into a system. It is, however, difficult to argue what the source of latency jitter in a measured dataset is as there are too many influences. On the other side, it is difficult to forsee what effects introducing latency has on the affected system and in turn on the user.

5 Conclusion

VR applications get optimised for mean and worst case behaviour, which we argue is not enough to capture latency behaviour as it does not account for different patterns of latency outliers.

This paper proposes to create latency models using empirical distributions based on measured latencies. Such latency models can then in turn be used to simulate latency behaviour at the application stage, either in the application itself or in a middleware. Usage of a latency simulator will allow to better study effects of latency and latency jitter on the user.

6 Further work

The example measurements illustrate the process of generating a latency model. The measurements shown describe the time, the physics engine runs for a test scene. The measurements seem to be similar to the measurements taken in Stauffert et al. [16], who only measured the time for single message passing. It is yet to be shown how similar latency patterns of a small algorithm are compared to a more complex system.

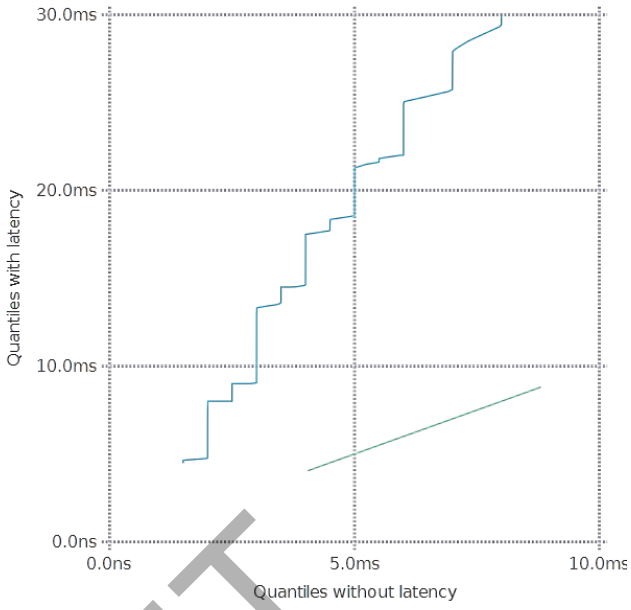

Figure 6: QQ-Plot of the spike duration of the fourth outlier category of the test setting without and with latency simulated. With latency simulated, there are more spikes visible.

Using the shown test scene, a variation of the scene in terms of available space and amount of objects will create different load on the physics system and a changed latency behaviour. Comparing applications or application parts under different load in respect to latency will uncover new insights for the system performance and how latency changes under different conditions.

The artificially injected latency has to get further researched to see if it elicits the desired effects. In any case, the evoked effects need more research.

Every machine will elicit different latency behaviour especially, different spike behaviour. It is up to future work to analyse how comparable latency models based on latency measurements of different machines are.

With a latency inducing plugin in place, effects of latency for different devices can be measured. Next steps include researching of simulator sickness when latency is injected into a head mounted display. Another is to measure performance and task load with motion controller drag and drop tasks.

References

[1] R. S. Allison, L. R. Harris, M. Jenkin, U. Jasiobedzka, and J. E. Zacher. Tolerance of temporal delay in virtual environments. In *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, VR '01, pages 247–, Washington, DC, USA, 2001. IEEE Computer Society.

[2] T. Arcila, J. Allard, C. Ménier, E. Boyer, and B. Raffin. FlowVR: A framework for distributed virtual reality applications. *Journees de lAFRV*, 2006.

[3] G. Bishop, H. Fuchs, L. McMillan, and E. J. S. Zagier. Frameless rendering: Double buffering considered harmful. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 175–176. ACM, 1994.

[4] R. G. Cole and J. H. Rosenbluth. Voice over IP performance monitoring. *ACM SIGCOMM Computer Communication Review*, 31(2):9–24, 2001.

[5] M. Di Luca. New method to measure end-to-end delay of virtual reality. *Presence*, 19(6):569–584, 2010.

[6] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes in latency during head movement. In *Proceedings of the HCI International '99 (the 8th International Conference on Human-Computer Interaction) on Human-Computer Interaction: Communication, Cooperation, and Application Design-Volume 2 - Volume 2*, pages 1129–1133, Hillsdale, NJ, USA, 1999. L. Erlbaum Associates Inc.

[7] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes of latency during voluntary hand movement of virtual objects. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 43, pages 1182–1186. SAGE Publications Sage CA: Los Angeles, CA, 1999.

[8] L. H. Frank, J. G. Casali, and W. W. Wierwille. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 30(2):201–217, 1988.

[9] S. Friston and A. Steed. Measuring latency in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on*, 20(4):616–625, 2014.

[10] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe. Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3d Shooter Games. pages 135–144. ACM Press, 2015.

[11] R. S. Kennedy, N. E. Lane, K. S. Berbaum, and M. G. Lilienthal. Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness. *The international journal of aviation psychology*, 3(3):203–220, 1993.

[12] M. E. Latoschik and H. Tramberend. A scala-based actor-entity architecture for intelligent interactive simulations. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop on*, pages 9–17. IEEE, 2012.

[13] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*, APGV '04, pages 39–47, New York, NY, USA, 2004. ACM.

[14] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. Brooks. Effect of latency on presence in stressful virtual environments. In *IEEE Virtual Reality, 2003. Proceedings.*, pages 141–148, March 2003.

[15] G. Papadakis, K. Mania, and E. Koutroulis. A system to measure, control and minimize end-to-end head tracking latency in immersive simulations. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pages 581–584. ACM, 2011.

[16] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Towards comparable evaluation methods and measures for timing behavior of virtual reality systems. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pages 47–50. ACM, 2016.

[17] A. Steed. A Simple Method for Estimating the Latency of Interactive, Real-time Graphics Simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, VRST '08, pages 123–129, New York, NY, USA, 2008. ACM.

[18] R. J. Teather, A. Pavlovych, W. Stuerzlinger, and S. I. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pages 43–50. IEEE, 2009.

[19] T. Waltemate, I. Senna, F. Hülsmann, M. Rohde, S. Kopp, M. Ernst, and M. Botsch. The impact of latency on perceptual judgments and motor performance in closed-loop interaction in virtual reality. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pages 27–35. ACM, 2016.