EPiC Series in Computing
Volume 88, 2022, Pages 82–90
Proceedings of 31st International Conference on Software Engineering and Data Engineering

Determining Optimal Control Frequency for Large Numbers of Virtual Agents within an Augmented Reality Application

Dr. Bradford A. Towle Jr.
Florida Polytechnic University
btowle@floridapoly.edu

Abstract

Augmented reality (AR) devices are becoming more prevalent and powerful enough to allow more virtual agents to be run simultaneously. This paper explores the ideal frequency to update the control logic for each virtual agent using the framerate as a measurement. This paper details the averaged result of a stair-step confidence test that was run five times for each frequency. This experiment was run on the Microsoft HoloLens 2.

1 Introduction

Augmented reality (AR) is growing more popular and approaching widespread availability to the public [1]. Newer devices with more capabilities are being developed along with more accessible libraries. AR projects virtual objects onto the real world providing the ability to add information to the visual perception of the user [2]. This augmentation can allow virtual characters, or agents, to appear as if they are present in the real-world providing information to the user. Because of this nature, it is practical to have the AR device mobile and untethered to a stationary computer.

Therefore, many of the modern AR devices are not personal computers, but rather smaller, lighter mobile devices that can be carried or worn. Some examples of AR devices include smart phones, Microsoft HoloLens, and Magic Leap 1 [3]–[5] . While these devices are continuously improving their capabilities most cannot match the raw processing power of a modern-day personal computer configured for research.

Along with the limitation of processing power, there are no libraries or APIs for developing on these devices. In general, a developer can choose to use the manufacturer's drivers directly, Unity3D Game Engine, or the Unreal Game Engine. Using the game engines will help abstract some of the more tedious problems that come with the device allowing the developer to focus more on the application. However, these development platforms are designed specifically to create games, meaning any virtual agent that requires a periodic update will most likely be programmed from a typical game design approach.

Considering both the limited computation of the mobile device and the influence of game design on the augmented reality development this paper determines the most efficient frequency to update control scripts on the virtual agent while still maintaining smooth performance. Please note, this paper is not addressing rendering or physics engine optimization but test for the optimal frequency of invoking control scripts for virtual agents. The control scripts used in the test have been written to mimic a typical game programmer profile; the code will employ best practice but not optimized to the level used in algorithm analysis research.

The experiment detailed below employs an unbounded stair-step test to compare the number of active virtual agents against the framerate of the AR application. Framerate is important to AR because it may cause the user to experience motion sickness if it falls too low. The unbounded test will consistently create new virtual agents for five seconds, wait for five seconds, and then repeat. This provides an opportunity to determine if the device can handle the computation while more load is added and when it is stable.

This paper provides a brief related works justifying use of virtual agent in AR. A methodology section provides a detailed explanation of how the program was setup, followed by an explanation of the testing procedure and the results. This paper concludes with a future work and closing thoughts.

2 Related Work

This paper considers virtual agent to mean any projected visualization in an augmented reality application that must change, move, or adapt to outside stimuli, thus, requiring a control script to function. The term virtual agent conjures mental images of brightly colored cartoon characters running around. While such virtual agents have been used in AR applications to assist with user interaction [6]–[10], not all virtual agents are 3D game characters. Many virtual agents will be informational elements that change and adapt. For example, a virtual agent may be nothing more than a label or text message that appears identifying a desired product in the grocery store [11]. Another example in the realm of health applications is visualizing different organs during surgery or education [12]–[14]. This visualization may need to change, update, or provide some form of response due to the user's action requiring some form of control script. Another important field in AR is bridging the gap between robotics and humans [15], [16]. These applications will employ virtual agents to represent robots or robot-human scenarios and goals. Again, these informational virtual agents will need a control script to update, move, and adapt to different input from the user.

Currently many of the augmented reality applications only focus on one or two virtual agents at once. Therefore, processing power for their control scripts is not a large concern; however, as the field grows expanding the scale of these applications, it is foreseeable that an AR application may have hundreds of virtual agents running simultaneously. Due to this expectation, the experiment described in the next section seeks to determine the best practice for designing virtual agent control scripts.

3 Methodology

The experiment outlined in this paper was run on a Microsoft HoloLens 2. The program used in this experiment was written with Unity 2020.3 and used the Mixed Reality Tool Kit (MRTK 2.0). This program would add a new virtual agent at a frequency of 2 Hz for five seconds and then wait for five seconds to determine if the system was stable. The above sequence of spawning and waiting would repeat until there were 100 agents, during which the frame rate was recorded to a file at the frequency of 10 Hz. The frame rate was calculated using the unscaled delta time property to provide the most accurate values possible (Equation 1).

$$\mathit{UnscaledDeltaTime\ is\ an\ independent\ interval\ in\ seconds\ from\ the\ last\ frame\ to\ the\ current}\ [17].$$

$$FrameRate = \frac{1}{UnscaledDeltaTime}$$

Equation 1: Equation Used to Calculate Framerate

The program recorded the framerate every tenth of a second and kept the file writer open to minimize computational overhead. The program would only append information, never delete, or search through the file. This limitation with the file handler was explicitly done to minimize its computational load on the HoloLens 2.

The virtual agents were programmed by employing best practices with Unity 3D; however, no other optimization was done. This programming style imitated a typical game programmer and not necessarily a researcher in computer science. The reason for imitation is to ensure the test script represents a typical program written for this platform.

When a virtual agent was created, it was given a team: red or blue. The agent invoked a control function called handle update, which provided the control algorithm.

Each agent ran the same function to control themselves. However, the frequency this function was invoked varied throughout the experiment, and the resulting framerates were compared. Five individual tests were administered for each following frequency:

- Update (once per frame)
- Fixed Update (20 Hz)
- 20 Hz coroutine
- 10 Hz coroutine
- 5 Hz coroutine
- 2 Hz coroutine

The control function performed the following tasks:

1. Control the nav-mesh agent
2. Fire projectiles at the enemy team
3. Orient and update the score panel

The test had a large arena where there were sixteen pre-determined points the virtual agents could move. If the virtual agent were within 6 centimeters of the goal, it would then randomly choose a new goal and start navigating toward the new nav goal.

The nav-mesh system in Unity was used as it is a common and popular tool amongst developers. This nav-mesh path-finding system is well optimized and would likely be chosen over building a path-finding algorithm from scratch. Please note, even though the logic was updated at different frequencies, the agents still moved continuously due to the nav-mesh.

Initially, the nav-mesh agent would be the only logic the virtual agents performed. However, it is unlikely that a typical application would only have navigation for a virtual agent be the only overhead. Therefore, logic was added to determine if there was a virtual agent on the opposite team within 50 centimeters in front of it. If there were, the agent would then fire a projectile in the same direction it was facing. If the projectile struck the other virtual agent, then the score of the first would increase by one. These projectiles also had a timer on them so that they would be destroyed after one second. The rate of fire was controlled by an additional co-routine that would wait for .3 seconds before allowing the virtual agent to fire again.

Each virtual agent had a small canvas above itself in world space. This canvas displayed the current score for each virtual agent and was used to simulate a visualization load that may be required for an AR application. The control function would rotate the canvas to make the visualizations more user-friendly to ensure it was facing the camera regardless of what direction the virtual agent was moving. Typically, this would be done in the update function, but it was added to the control function to keep all tests consistent.

4 Testing

The testing procedure took five individual tests of the above-mentioned frequencies. The user would start each test and disable the default profiler, to keep things consistent, then move outside the arena and sit down. The user's action would be constrained to look around the arena as the different virtual elements were spawned and performed their control logic. This reduction in physical movement is essential as fast or erratic movements by the user will cause the system extra computation to keep the virtual environment aligned with the real environment. It was not the intention of this experiment to put the AR application under stress from user movement. After the number of agents reached 100, the test was stopped, and the framerate was collected in a comma-delimited file. The raw data was very noisy, as demonstrated in Figure 1.

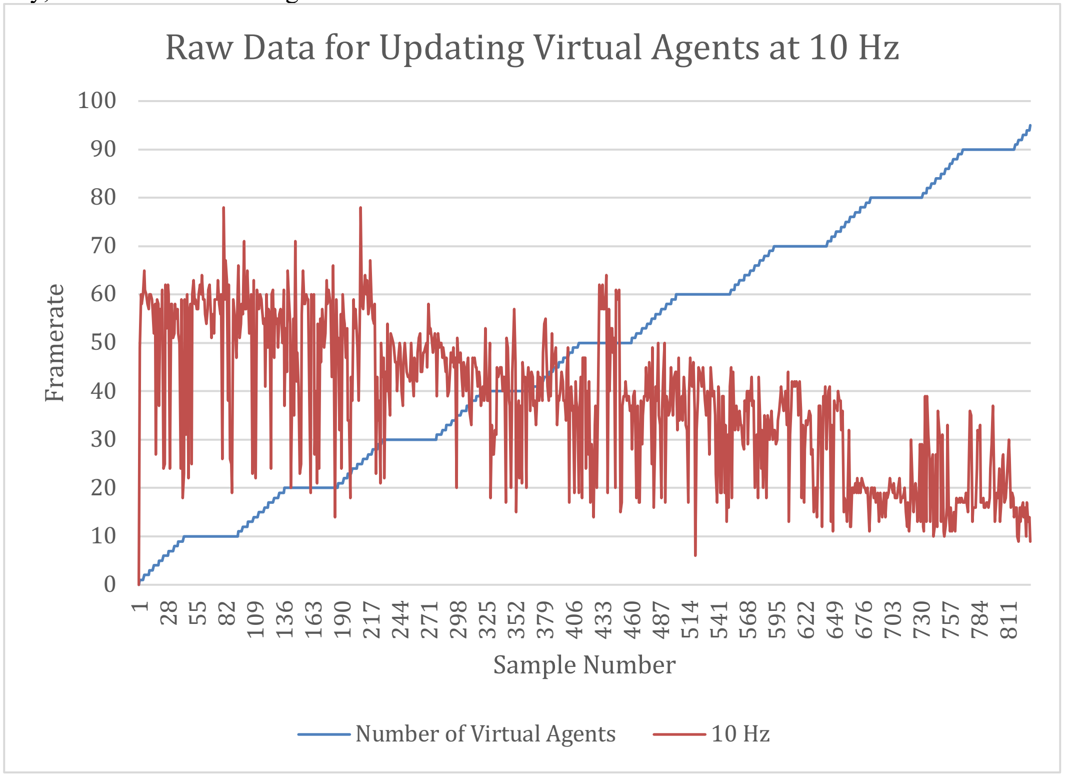

Figure 1: Raw Data from 10 Hz Virtual Agent Update Experiment

Five tests were run for each frequency and then averaged together in order to reduce the noise. The results were still noisy; therefore, a moving average with a sliding window of size ten was used to improve the results further (Equation 2)

$$AverageRecord_r = \frac{\sum_{Test=1}^{5} Value_{r,Test}}{5}$$

$$\forall ma \text{ where } ma = \{10 \ldots Number\ of Records\}.$$

$$MovingAverage_{ma} = \frac{\sum_{n=ma-10}^{ma} AverageRecord_n}{10}$$

Equation 2: Calculation for Moving Average

The improvement can be seen by comparing the above graph with Figure 2. Notice the noise is significantly reduced.

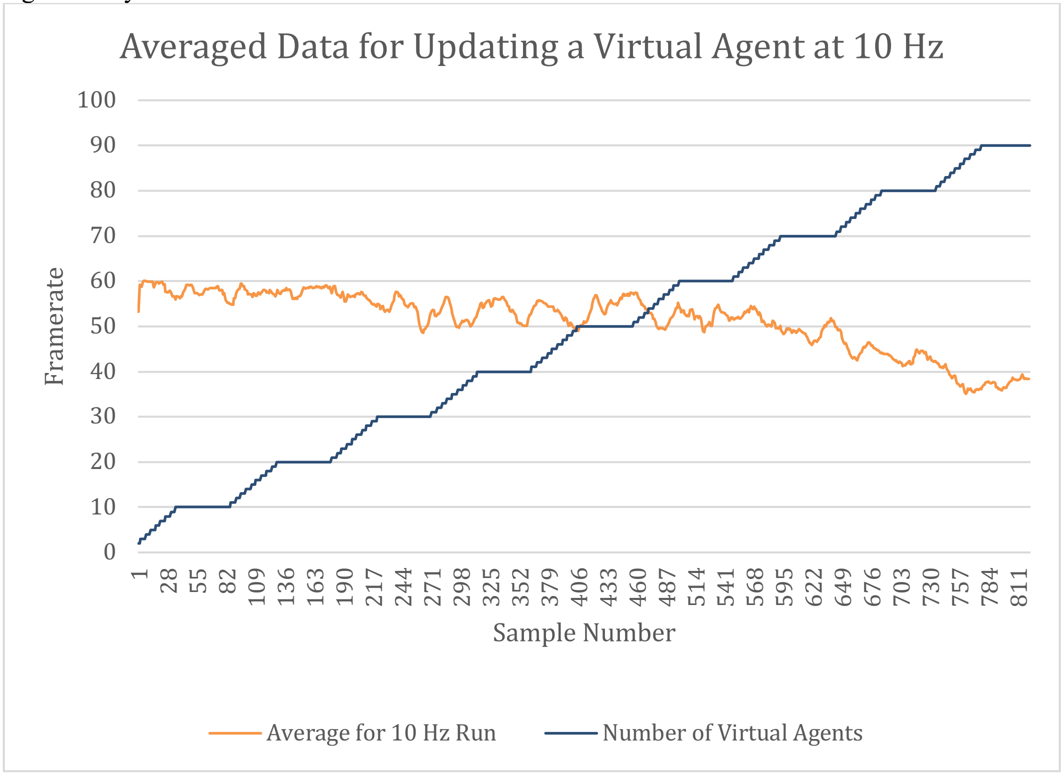

Figure 2: Averaged Data for Updating a Virtual Agent at 10 Hz.

5 Results

At the beginning of the experiment, all frequencies were performed in the acceptable range between 50 and 60 FPS. Around 40 virtual agents, the performances started to diverge, and by the time there were more than 50 virtual agents, all frequencies demonstrated a downward turn (Figure 3). Fixed update performed the worst as it is the most rigid with the time precision; therefore, the CPU could not keep up as more agents were added. This result coincides with the anecdotal wisdom that fixed updates should be used sparingly on mobile devices. Calling the control function once per frame (Update) performed decently. Although, it is worth noting that as the framerate drops, so does the number of calls to the control function, creating an unintended feedback loop. Coroutines were used to provide the 20, 10, and 2 Hz, where the performance was anticipated to improve the slower the frequency. The results differed significantly from the expectation. The 10 Hz coroutine performed the best out of all the tests. The 2 Hz coroutine had trouble maintaining the same performance as calling the control function once per frame. The experiment for 2 Hz was repeated to ensure this data was not an anomaly, and the second set of results was similar. It was hypothesized that the 2 Hz test conflicted with the virtual agent spawning as they are both using the same frequency. Another experiment was run at 5 Hz, and it performed better than 2 Hz but did not outperform the 10 Hz coroutine.

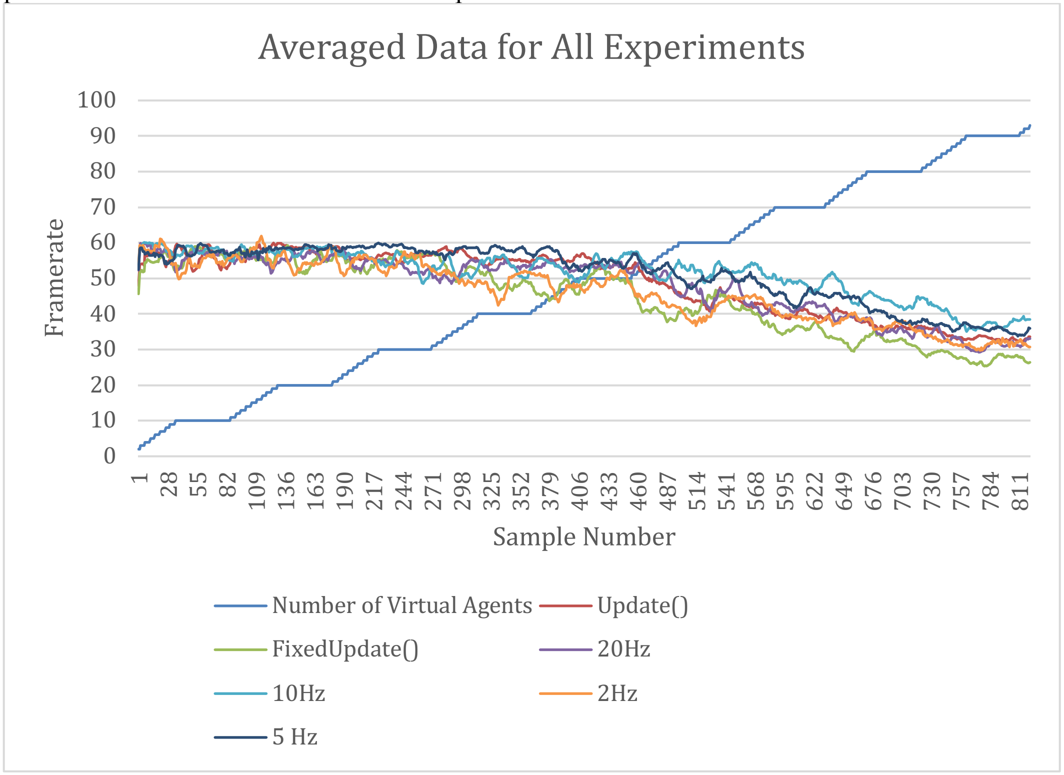

Figure 3: Averaged Data for All Experiments

It is worth noting that all the experiments remained above 24 frames per second, satisfying the real-time constraint. However, to find a strong contrast between the different frequencies, the highest number of agents was taken while the framerate was still above 50 frames per second (Table 1).

Frequency (From Best to Worst Performance)	**Highest Number of Virtual Agents before Dropping Below 50 FPS.**
10 Hz	71
5 Hz	66
20 Hz	58
Update	54
2 Hz	50
Fixed Update	50

Table 1: Highest Number of Virtual Agents while maintaining 50 FPS.

The final framerate was also used to determine the performance rating. Due to the system load, the framerate recorder would cease to record values of around 92 virtual agents. Therefore, the data ends at this point even though the experiment ran to 100 virtual agents. The order of performance is the same except at the end, calling the control function once per frame (update) performed better than a 20 Hz coroutine by one frame per second. Updating the virtual agents at 10 Hz still outperformed all other tests; however, its lead had narrowed significantly (Table 2).

Frequency (From Best to Worst Performance)	**Last Framerate (92 Virtual Agents)**
10 Hz	38
5 Hz	36
Update	34
20 Hz	33
2 Hz	30
Fixed Update	26

Table 2: Highest Number of Virtual Agents at the End of the Experiment

The conclusion from this data clearly shows that a slower update rate on logic does not translate into better performance. Further research is needed to identify why 10 Hz performed noticeably better than all other frequencies. Current hypotheses are the Unity 3D coroutine system favors this frequency due to some underlying design, or 10 Hz has the least conflicts with the HoloLens 2 system processes. Regardless, updating the control script of a virtual agent at 10 Hz will provide the best performance for the HoloLens 2.

6 Future Work

There are two questions that require further investigation. The first question is how this experiment will perform on different platforms. The Mixed Reality Toolkit (MRTK) allows projects to be cross-platform; therefore, this experiment is hypothetically portable. Comparing the results from different platforms would provide an opportunity to see if there was a generalized best practice frequency.

Another modification to the experiment would be removing all coroutine usage except for agent spawning, ensuring the test was not interfering with the framerate. The primary reason this has not already been done was to ensure the virtual agents used common practices with game development. Coroutines are a common practice, and an experienced game developer would use them. This means it is not unreasonable to expect multiple different coroutines to be used by a virtual agent.

7 Conclusion

This paper has detailed an experiment where different frequencies were used to update control scripts on virtual agents in an augmented reality application. The justification was given for why specific programming practices were used, and the experiment isolated one factor: The rate at which a function was invoked. Five different tests were made per frequency, keeping all other factors the same, and the results were averaged together. The data concludes that the ideal update frequency for control scripts in virtual agents for the HoloLens 2 is 10 Hz.

8 References

[1] C. Arth, R. Grasset, L. Gruber, T. Langlotz, A. Mulloni, and D. Wagner, "The History of Mobile Augmented Reality," *arXiv:1505.01319 [cs]*, Nov. 2015, Accessed: Mar. 30, 2022. [Online]. Available: http://arxiv.org/abs/1505.01319

[2] T. P. Caudell and D. W. Mizell, "Augmented reality: an application of heads-up display technology to manual manufacturing processes," in *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, Jan. 1992, vol. ii, pp. 659–669 vol.2. doi: 10.1109/HICSS.1992.183317.

[3] mattzmsft, "HoloLens (1st gen) hardware." https://docs.microsoft.com/en-us/hololens/hololens1-hardware (accessed Mar. 15, 2022).

[4] "HoloLens 2—Overview, Features, and Specs | Microsoft HoloLens." https://www.microsoft.com/en-us/hololens/hardware (accessed Mar. 15, 2022).

[5] "Magic Leap 1." https://www.magicleap.com/magic-leap-1 (accessed Mar. 15, 2022).

[6] I. Wang, J. Smith, and J. Ruiz, "Exploring Virtual Agents for Augmented Reality," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, Glasgow Scotland Uk, May 2019, pp. 1–12. doi: 10.1145/3290605.3300511.

[7] M. Obaid, I. Damian, F. Kistler, B. Endrass, J. Wagner, and E. André, "Cultural behaviors of virtual agents in an augmented reality environment," in *International Conference on Intelligent Virtual Agents*, 2012, pp. 412–418.

[8] K. Kim, L. Boelling, S. Haesler, J. Bailenson, G. Bruder, and G. F. Welch, "Does a digital assistant need a body? The influence of visual embodiment and social behavior on the perception of intelligent virtual agents in AR," in *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2018, pp. 105–114.

[9] M. Obaid, R. Niewiadomski, and C. Pelachaud, "Perception of spatial relations and of coexistence with virtual agents," in *International Workshop on Intelligent Virtual Agents*, 2011, pp. 363–369.

[10] A. Hartholt *et al.*, "Virtual humans in augmented reality: A first step towards real-world embedded virtual roleplayers," in *Proceedings of the 7th International Conference on Human-Agent Interaction*, 2019, pp. 205–207.

[11] J. Ahn, J. Williamson, M. Gartrell, R. Han, Q. Lv, and S. Mishra, "Supporting Healthy Grocery Shopping via Mobile Augmented Reality," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 12, no. 1s, p. 16:1-16:24, Oct. 2015, doi: 10.1145/2808207.

[12] I. C. S. da Silva, G. Klein, and D. M. Brandão, "Segmented and Detailed Visualization of Anatomical Structures based on Augmented Reality for Health Education and Knowledge Discovery," *Adv. sci. technol. eng. syst. j.*, vol. 2, no. 3, pp. 469–478, May 2017, doi: 10.25046/aj020360.

[13] C. Moro, Z. Štromberga, A. Raikos, and A. Stirling, "The effectiveness of virtual and augmented reality in health sciences and medical anatomy," *Anatomical Sciences Education*, vol. 10, no. 6, pp. 549–559, 2017, doi: 10.1002/ase.1696.

[14] B. Garrett, J. Anthony, and C. Jackson, "Using Mobile Augmented Reality to Enhance Health Professional Practice Education," *Current Issues in Emerging eLearning*, vol. 4, no. 1, Jul. 2018, [Online]. Available: https://scholarworks.umb.edu/ciee/vol4/iss1/10

[15] P. Parashar, L. M. Sanneman, J. A. Shah, and H. I. Christensen, "A Taxonomy for Characterizing Modes of Interactions in Goal-driven, Human-robot Teams," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 2213–2220. doi: 10.1109/IROS40897.2019.8967974.

[16] S. Saeedi *et al.*, “Navigating the Landscape for Real-Time Localization and Mapping for Robotics and Virtual and Augmented Reality,” *Proceedings of the IEEE*, vol. 106, no. 11, pp. 2020–2039, Nov. 2018, doi: 10.1109/JPROC.2018.2856739.

[17] “Unity - Scripting API: Time.unscaledDeltaTime.” https://docs.unity3d.com/ScriptReference/Time-unscaledDeltaTime.html (accessed Apr. 05, 2022).