Exploring the Use of Virtual Reality Fixtures for Automated Augmented Reality Software Testing

Jim Jazwiecki
New York, US
jim.jazwiecki@colostate.edu

ABSTRACT

Augmented Reality (AR) systems are widely expected to become a significant platform for software development in the near future, but effectively testing that software on those systems is challenging, both because of the size of the potential input set and the difficulty of controlling conditions under repeated tests. Previous research explored using virtual reality (VR) as a theoretical solution to evaluating AR usability under controlled conditions where aspects of a simulated AR application (registration and jitter) could be experimentally controlled. By surveying a subset of available AR application platforms to determine their capabilities, I identified one (Android) which was capable of supporting the use of 3D VR test fixtures in testing AR applications, demonstrating the validity of this approach for testing non-simulated AR applications. A trivial Android ARCore application was written, and the ability of these tests to identify and kill mutants are used to validate this approach. In addition, AR platform requirements for generalizing and improving the use of VR fixtures for automated testing of AR software are proposed. Adopting the proposed requirements to support automated testing is suggested as a necessary factor to foster the development of high-quality, reliable AR software ecosystems.

INTRODUCTION

Although it has not yet enjoyed breakout success with consumers, there is every indication that AR is within a few years of becoming successfully commercialized. Almost all of the major consumer technology companies have introduced AR hardware or software development kids (SDKs) over the past five years, indicating that all of these companies expect the application of AR software to reach widespread commercial acceptance. The list includes Microsoft's HoloLens, Facebook's SparkAR, Apple's ARKit, and Google's ARCore and Glass, as well as a widely-rumored AR headset in development at Apple. Enthusiasm for AR exists not only on the supply side, but also on the demand side, as the US Army has agreed to purchase 100,000 HoloLens headsets from Microsoft for almost half a billion dollars[8]. As widely-deployed software platforms, as with any consumer, industry, or government software projects, effective testing will be critical to ensuring system correctness is effectively maintained through successive rounds of change. The advantages of automated software testing with regard to effectively maintaining software quality have been demonstrated[9], but it is difficult to write automated functional tests to verify the correctness of AR software.

In order for repeated tests to be valid over multiple runs, and to identify faults over successive rounds of changes, they must be carried out under controlled conditions. For most software systems under test, these controlled conditions are created by running the tests using test doubles, such as automatically or manually created mock or stub objects, in identical environments, with a set of fixtures that create an environment sufficient to initialize and run the system. By using fixtures and test doubles, engineers can ensure the repeatable aspects of their tests, which are critical to maintaining their validity. AR systems, however, will typically be expected to operate under far more widely varied conditions than most software, whether with variations in lighting (hue, brightness), atmospheric conditions (smoke, fog, precipitation), or surface materials (transparency, reflectivity). Effectively testing these conditions consistently with a physical device running the software under test, especially through the expected combinatoric explosion of all variations along those real-world continua, could be prohibitively challenging to the resources of most, if not all, organizations developing AR software.

Consider the aforementioned US military purchase, and a hypothetical application such as a field repair guide which displays interactive instructions and uses image recognition to identify parts in need of replacement. To fully test custom software providing those capabilities, in order to provide the level of functional guarantees befitting hypothetical user expectations, manual tests would have to be conducted in a great number of combinations of atmospheric, lighting, and equipment conditions, all carefully controlled between repeated tests to maintain validity, potentially at great expense. Alternately, consider an AR application designed to aid the visually impaired in safely navigating, which might include the ability to extrapolate velocities and trajectories from moving objects to determine if they will intersect the direction of the user's travel. Verifying the correctness of this functionality might involve choreographing a number of independent actors for each test, again in widely varied conditions.

Ragan, Wilkes, and Bowman[10] proposed using VR environments to conduct usability experiments with simulated AR software, inspiring this exploration. For their experiment, they used a four-sided CAVE VR system with simulated 2D markers which would act as registration points for a simulated AR system. Using an AR task introduced by Ellis et al.[7], users were asked to guide a virtual ring along a contorted virtual tube while the simulation introduced errors in registration. Ragan

et al. were able to demonstrate novel results with regard to the relationship between task completion time, input latency, and "jitter", defined as "output noise exhibited as translation and orientation offset" in the simulated AR system. Specifically, they were able to demonstrate jitter outweighed latency as a factor in task completion time, and they plausibly claim these results would have been impossible to produce without using a simulated AR system in which these registration errors were carefully and artificially controlled. Although there is little published research on testing of AR systems apart from that of Ragan et al., their use of a simulated AR application in a VR environment suggested an intriguing possibility: could a VR environment be used to test non-simulated AR applications?

VR environments offer the ability to run tests of consistent worlds and precisely, consistently vary conditions over successive tests within the expected tolerances of the AR systems under test, potentially acting as effective test fixtures in a cost-effective fashion. For example, a team developing the above-mentioned audio-based AR wayfinding app for the visually impaired could set up a virtual obstacle course, define three-dimensional areas in which they expect their app to provide appropriate feedback to a user, and then run tests to move the simulated devices through the space at various heights or speeds, and under varied environmental conditions. Alternately, the authors suggest VR environments would provide a safe way to test and prototype AR systems expected to be used in dangerous conditions, such as by firefighters, without requiring testers to navigate an actual burning building until the engineers were relatively certain development had reached the final acceptance testing stages prior to release. This exploration is written to determine the capacity of existing AR development and testing frameworks to support VR fixtures, whether or not those systems are effective in ensuring software correctness, and propose a set of conditions and capabilities necessary for effectively automated tests of AR systems.

EXISTING AR SDK CAPABILITIES

Initial Requirements

Based on widespread commercial and open-source software testing practices, the search for appropriate AR SDKs began with the following requirements against which candidate systems were evaluated:

- Application and test source must, as much as possible, be stored in plain text in order to maintain ease-of-use with common version control systems and software development workflows.
- Builds and tests should be triggered via a command-line interface, rather than taking an action in a GUI, in order to maintain ease-of-use with common continuous integration (CI) environments.
- The fixture should always start in the same ground state even if it is used in multiple tests during the same test run.
- It should be possible to install, configure, and cache the test environment without manual interventions.
- It should be possible both to change the state of the application's environment and access information about application state from within executable test code.
- Pass/fail information should be returned to the same command-line interface used to start builds and tests, ideally with an exit code that can be used to automatically determine whether or not all tests have passed.

Epic Games' Unreal Engine

Considering the market-leading capabilities of Unreal Engine to create VR environments, as well as the more recent addition of native capabilities on the part of Unreal Engine to generate AR applications, this was the first development and test environment evaluated. Unfortunately, the relative paucity of up-to-date documentation on writing and running automated tests as well as the complexity of the build system made it difficult to determine whether it was possible to develop a working proof-of-concept within the time allotted for this project. For example, what documentation there is for the "Automation System" is still tagged with version 4.17[4], which was released over two years ago, and the newer Gauntlet Automation Framework ("Gauntlet is a framework to run things and validate results."), released as part of version 4.21 in November of 2018, has even less public documentation[5]. UE4Editor, the primary VR-editing interface, does not support an integrated code-editing environment, which produced development friction. In order to develop cross-platform code, it relies on a proprietary build system written in C#, and the "mono" framework required to build and run .Net-compliant C# outside of a the Windows environment was found, in one instance, to block further development, which could only be resolved by waiting on the Epic Games team to release updated patch versions of Unreal Engine. While it is possible to create automated tests entirely within UE4Editor, the files generated are binary "Blueprint" files which can only be read or edited within UE4Editor. Although it is possible that, with more time available for study of the source of Unreal Engine and its toolchain, it would be possible to satisfy all of the initial requirements, exploration of this option was halted.

Apple's ARKit

Because working with Apple's ARKit is supported as part of their Xcode IDE, with a simpler, native compiler toolchain, this was chosen as the next candidate for evaluation. This simplified development and testing relative to Unreal Engine. However, early on it was determined that while it is possible to capture[2] and replay[3] sensor data, these features are largely undocumented[1]. Moreover, the "ARWorld" class which is automatically created by ARKit as an AR application runs, has a method ("rawFeaturePoints")[6] which returns a set of 3D points detected by the hardware, but only "notable features", and suggested uses include debugging visualizations and judging the quality of a world map based on the object's raw size. A saved set of feature points can be reloaded, so presumably it would be possible to synthesize an ARWorld object from a 3D environment, but without documentation, reverse-engineering the format effectively would be challenging. It's unclear if a virtual device could be "moved" in that space, or if testing would involve loading a series of ARWorld

objects representing "frames" of navigation and checking the app state. Finally, the emulated devices packaged with Xcode crash when ARKit apps are loaded, with a message stating that ARKit is not available on the device.

Google's ARCore

Fortunately, the emulated devices packaged with Google's Android Studio IDE don't crash when an app using Google's AR SDK, ARCore, is run. Provided that the emulated device has been correctly configured, a set of 3D resources, representing a small apartment (figure 1), is loaded, and the device can "navigate" in that apartment, with all of the sensor data that would be generated moving in a real room. Here, the challenge moved from identifying a method for loading a VR world for a real AR application to writing and running tests that could manipulate the device idempotently.

During manual manipulation of the emulated device, by keeping a modifier key depressed, keyboard and mouse inputs move the device in the virtual space. Initial attempts to send identical sequences to the emulated device revealed that test keystrokes sent by the test runner were passed to the application under test, and ignored by the emulator. Then, direct manipulation of the device's accelerometer sensors was attempted, but this also did not "move" the device in the simulated environment. While it was possible to execute shell commands in the emulated device's environment, a thorough review of the available commands did not turn up any way to move the device.

Fortunately, a close reading of the source code revealed the emulator offers both a simple telnet interface which can be accessed from within the emulated device's environment and a facility for recording and replaying movements through that telnet interface. The navigation macros are stored in a binary format, but the source code suggests they are using a specific Protocol Buffer format to call methods on a "Physics" class, which implies that it might be possible to call these methods directly for more complicated navigation tests, such as tests of AI wayfinding. Little of this is documented or, in the case of the Physics class, publicly available, but with the aid of a trivial socket connection to the host loopback in the emulated network environment, I was able to produce a test environment which met most of the initial requirements, with the exception that the test macros are stored in a binary format.

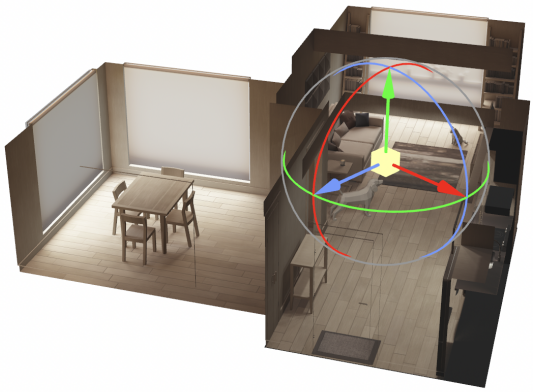

Figure 1. The VR environment.

PROOF OF CONCEPT

The Subject Application

With a suitable environment and test harness established, a very simple AR application (see figure 2), meant to represent a navigation aid for the visually impaired, was created. Every time the phone is moved, it uses the ARCore framework's hit-testing to check for the nearest objects, and updates a simple message displayed on the screen with proximity information. Although the application's presumed users would likely be unable to read the message, this approach was used to simplify testing by constraining the comparison to simple text strings, rather than something more complicated to verify with a test assertion.

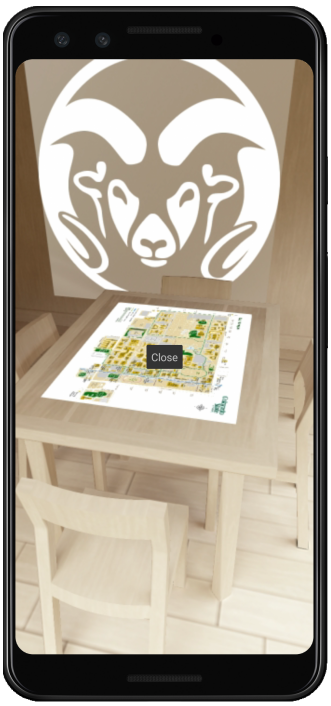

Figure 2. Test AR Application showing a proximity warning in a VR environment running on an emulated device.

Tests

The Android SDK includes an instrumented test runner which starts an emulated device. By using native testing frameworks, setup, builds, and execution were simplified. The macros themselves were created by running a built-in reset macro which teleports the device to the center of the virtual living room. Then, movement is recorded in the emulator and saved locally, but this step does not have to be repeated for each test run. During test execution, the name of the macro is passed to the automator's "play" command through a special bridge function which connects over telnet to the emulator itself. Although currently part of the instrumented test class, it could easily be converted into a helper external to the tests themselves. For the purpose of this exploration, a single macro was created to "move" the device in the VR space in such a way that all feedback messages would be displayed. The tests cover the initial startup state, and moving to a location in the room which will trigger each of the different messages displayed to the user.

RESULTS

Coverage

86% of instructions were covered by the instrumented tests, and 75% of branches, with most of the missed instructions and branches specific to detecting device compatibility and handling potential failure states I was unable to duplicate in the instrumented tests under the time constraints for this project. For the core application logic, which handled conducting hit tests for proximity and displaying the appropriate message, instrumented VR environment test coverage was 100%.

Mutation

Unfortunately, in order to fully use a leading Java mutation-testing tool, PIT, a new plugin would be required to find and run instrumented tests, as PIT only includes plugins for JUnit 4 and 5. However, although PIT could not be used to run the tests, it was possible to use it to generate mutations, manually replicate them, and run the instrumented VR tests. PIT acts on the Java bytecode, rather than the source itself, and generates a report mapping mutation operations it carried out to lines of code. In some cases, it's easy to re-create the mutation in the source code, such as when PIT removes a call to a method or negates a conditional. In others, such as when a conditional boundary is changed, it doesn't report what it was changed *to*, so manual source code mutations were done with a different value meant to meaningfully change the functionality of the application. 94.1% (16 of 17) of manually-created mutations made on the core application logic (e.g. not the main ARCore scaffolding) were killed by the instrumented tests. The one surviving mutation was an ambiguous boundary condition.

CONCLUSION

Based on the ease of use, test coverage, and mutation results, using VR fixtures to conduct functional testing of AR applications is validated. Although a great deal of effort went into identifying a relatively straightforward test approach, and it is currently limited to a single AR platform, these results could be generalized to support testing for any AR platform. It is not yet possible to achieve the level of AR application testing outlined in the introduction, including simulation of varied weather conditions and multiple actors, but in principle these are achievable if existing AR platform makers improve the tooling of their test environments. Just as better software development outcomes can be achieved by integrating test planning through every step of the process, rather than at the end, better platform development outcomes would be supported by fully supporting integrated automated testing during platform development, rather than as an afterthought.

Limitations

As mentioned above, this approach is only strictly validated for a single platform, and within that platform, there remain some limitations to this approach. First, it is unclear how to change the 3D model used for the VR environment, although the fact that the resources are available in common formats as part of the SDK offers some promise that it should be possible. Second, there is currently no simple way to tell that traversal of a path in the VR environment has reached a point where the state of the application should be checked. It would be possible to determine whether or not the device's accelerometer indicates it has stopped moving, but this is a weak connection at best, as a test might require that the navigation pause for some amount of time. The current approach also does not provide any ability to change the speed of travel or move other objects in the environment.

Proposed Requirements

In addition to the initial requirements outlined above, based in part on the limitations of this approach, the following additional requirements for supporting functional AR tests in VR environments on AR platforms are proposed:

- It should be possible to control discrete manipulations of the emulated device in the emulated environment in all six measured degrees of freedom, at a given speed, and suspend execution of the test until the movement is complete.
- The 3D environment should be replaceable, using common file formats, ideally at runtime, in order to support more complicated test suites (e.g. testing an app in both simulated indoors and outdoors environments) without requiring multiple distinct emulated devices for each environment.
- Tests should be able to alter environmental variables, such as global lighting, during execution.
- It should be possible to manipulate the movement of other objects in the 3D environment in order to support more complicated tests.

Future Work

There are promising signs that the makers of existing AR platforms are increasing the testability of software designed for their platforms, such as Apple's private methods for loading arbitrary "world" objects and Epic Games' new test automation framework, but the lack of comprehensive, high-quality documentation of public methods hampers their efforts. It may also be possible to develop third-party systems capable of supporting the requirements for a full-featured VR testing environment for AR applications, including both customized

emulators and the ability to address application state without using native code.

REFERENCES

[1] 2019a. ARConfiguration – ARKit | Apple Developer Documentation. (6 December 2019). Retrieved December 6, 2019 from https://developer.apple.com/documentation/arkit/arconfiguration.

[2] 2019b. ARConfiguration.h. (6 December 2019). Retrieved December 6, 2019 from https://github.com/xybp888/iOS-Header/blob/master/13.0/Frameworks/ARKit.framework/ARConfiguration.h.

[3] 2019c. ARReplaySensorPublic.h. (6 December 2019). Retrieved December 6, 2019 from https://github.com/xybp888/iOS-Header/blob/master/13.0/Frameworks/ARKit.framework/ARReplaySensorPublic.h.

[4] 2019a. Automation System Overview. (1 December 2019). Retrieved December 1, 2019 from https://docs.unrealengine.com/en-US/Programming/Automation/index.html.

[5] 2019b. Automation System Overview. (1 December 2019). Retrieved December 1, 2019 from https://docs.unrealengine.com/en-US/Programming/Automation/Gauntlet/index.html.

[6] 2019d. rawFeaturePoints – ARWorldMap | Apple Developer Documentation. (9 December 2019). Retrieved December 9, 2019 from https://developer.apple.com/documentation/arkit/arworldmap/2968213-rawfeaturepoints.

[7] Stephen R. Ellis, Francois Bréant, Brian Menges, Richard Jacoby, and Bernard D. Adelstein. 1997. Factors Influencing Operator Interaction with Virtual Objects Viewed via Head-Mounted See-Through Displays: Viewing Conditions and Rendering Latency. In *Proceedings of IEEE 1997 Annual International Symposium on Virtual Reality*. 138–145.

[8] Charlotte Jee. 2018. US Army soldiers will soon wear Microsoft's HoloLens AR goggles in combat. (29 November 2018). https://www.technologyreview.com/f/612490/us-army-soldiers-will-soon-wear-microsofts-hololens-ar-goggles-in-combat/.

[9] Divya Kumar and K. K. Mishra. 2016. The Impacts of Test Automation on Software's Cost, Quality and Time to Market. In *Procedia Computer Science*, Vol. 79. 8–15.

[10] Eric Ragan, Curtis Wilkes, and Doug A Bowman. 2009. Simulation of Augmented Reality Systems in Purely Virtual Environments. In *2009 IEEE Virtual Reality Conference*. 287–288.