5-5-2017

Best practices for cross-platform virtual reality development

Jonathan Schlueter
Iowa State University, jschlu@iastate.edu

Holly Baiotto
Iowa State University, hbaiotto@iastate.edu

Melynda Hoover
Iowa State University, mthoover@iastate.edu

Vijay K. Kalivarapu
Iowa State University, vkk2@iastate.edu

Gabriel Evans
Iowa State University, gjevans@iastate.edu

See next page for additional authors

Best practices for cross-platform virtual reality development

Abstract
Virtual Reality (VR) simulations have become a major component of the US Military and commercial training. VR is an attractive training method because it is readily available at a lower cost than traditional training methods. This has led to a staggering increase in demand for VR technology and research. To meet this demand, game engines such as Unity3D and Unreal have made substantial efforts to support various forms of VR, including the HTC Vive, smartphone-enabled devices like the GearVR, and with appropriate plugins, even fully-immersive Cave Automatic Virtual Environment (CAVETM) systems. Because of this hardware diversity, there is a need to develop VR applications that can operate on several systems, also known as cross-platform development. The goal in developing applications for all these types of systems is to create a consistent user experience across the devices. It is challenging to maintain this consistent user experience, because many VR devices have fundamental differences. Research has begun to explore ways of developing one application for multiple system. The Virtual Reality Applications Center (VRAC) developed a VR football "Game Day" simulation that was deployed to three devices: CAVETM, Oculus Rift HMD and a mobile HMD. Development of this application presented many learning opportunities regarding cross-platform development. There is no single approach to achieving consistency across VR systems, but the authors hope to disseminate these best practices in cross-platform VR development through the game day application example. This research will help the US Military develop applications to be deployed across many VR systems.

Disciplines
Applied Mechanics | Electrical and Computer Engineering | Mechanical Engineering

Comments
This proceeding is published as Jonathan Schlueter, Holly Baiotto, Melynda Hoover, Vijay Kalivarapu, Gabriel Evans, Eliot Winer, "Best practices for cross-platform virtual reality development," Proc. SPIE 10197, Degraded Environments: Sensing, Processing, and Display 2017, 1019709 (5 May 2017); doi: 10.1117/12.2262718. Posted with permission.

Authors
Jonathan Schlueter, Holly Baiotto, Melynda Hoover, Vijay K. Kalivarapu, Gabriel Evans, and Eliot H. Winer

PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Best practices for cross-platform virtual reality development

Jonathan Schlueter, Holly Baiotto, Melynda Hoover, Vijay Kalivarapu, Gabriel Evans, et al.

Jonathan Schlueter, Holly Baiotto, Melynda Hoover, Vijay Kalivarapu, Gabriel Evans, Eliot Winer, "Best practices for cross-platform virtual reality development," Proc. SPIE 10197, Degraded Environments: Sensing, Processing, and Display 2017, 1019709 (5 May 2017); doi: 10.1117/12.2262718

SPIE.

Event: SPIE Defense + Security, 2017, Anaheim, California, United States

Best practices for cross-platform virtual reality development

Jonathan Schlueter[a], Holly Baiotto[a], Melynda Hoover[a], Vijay Kalivarapu[a],
Gabriel Evans[a], Eliot Winer[a]
[a]Iowa State University, 1620 Howe Hall, Ames, IA, USA 50011;

ABSTRACT

Virtual Reality (VR) simulations have become a major component of the US Military and commercial training. VR is an attractive training method because it is readily available at a lower cost than traditional training methods. This has led to a staggering increase in demand for VR technology and research. To meet this demand, game engines such as Unity3D and Unreal have made substantial efforts to support various forms of VR, including the HTC Vive, smartphone-enabled devices like the GearVR, and with appropriate plugins, even fully-immersive Cave Automatic Virtual Environment (CAVE™) systems. Because of this hardware diversity, there is a need to develop VR applications that can operate on several systems, also known as cross-platform development. The goal in developing applications for all these types of systems is to create a consistent user experience across the devices. It is challenging to maintain this consistent user experience, because many VR devices have fundamental differences. Research has begun to explore ways of developing one application for multiple system. The Virtual Reality Applications Center (VRAC) developed a VR football "Game Day" simulation that was deployed to three devices: CAVE™, Oculus Rift HMD and a mobile HMD. Development of this application presented many learning opportunities regarding cross-platform development. There is no single approach to achieving consistency across VR systems, but the authors hope to disseminate these best practices in cross-platform VR development through the game day application example. This research will help the US Military develop applications to be deployed across many VR systems.

Keywords: Best Practices, Cross-platform, Development, Military, Virtual Reality

1. INTRODUCTION

With new threats emerging that require unique forms of training, US Military personnel have started to recognize the benefits of simulation based training using Virtual Reality (VR). As a result, the US Military's demand for VR technology and research has been steadily increasing. In 2015, Strategic Defense Intelligence estimated the global simulation and training market to be worth US$10.4 billion. The market is projected to grow to US$15.8 billion by 2025.[1] Recent research with VR technology has given the US Military an opportunity to provide less expensive simulation training, compared to live field exercises. Traditional live field training requires lengthy planning, facilitator personnel, and resources. Simulation training can offer trainees the same experience in less time with fewer resources. Additionally, simulation training allows users to experience intense situations with less physical risk. For example, supply truck drivers can initially learn skills and tactics for a hostile war zone through virtual simulation. The simulation training provides a safe opportunity for trainees to experience being an ambush target and then allows them to apply their knowledge in a life-like situation.[2] These simulations should not be limited by the accessibility and compatibility of the hardware, instead the applications should be made to support multiple hardware platforms. This type of development will be referred to as cross-platform development throughout this paper. Another goal of effective cross-platform development is to provide the same user experience throughout all VR systems, allowing a similar learning opportunity for trainees.

As VR technology quickly progresses, it is important to be able to develop and run applications on different systems and receive a similar user experience. Some of the challenges associated with cross platform VR development are dissimilarity between the hardware devices and a lack of software support. Being able to overcome these challenges will allow for organizations, like the military, to develop applications which fit the available VR hardware. This can save organizations money by not having to purchase new VR technology. Additionally, developers will save time by assessing the design concerns of multiple systems at the beginning of the development process instead of later, when the current VR hardware is inevitably rendered obsolete. Unfortunately, very little existing research addresses the issues of developing for several dissimilar VR systems simultaneously. Therefore, it is necessary for developers of cross-platform VR systems to share

Degraded Environments: Sensing, Processing, and Display 2017, edited by John (Jack) N. Sanders-Reed,
Jarvis (Trey) J. Arthur III, Proceedings of SPIE Vol. 10197, 1019709 · © 2017 SPIE
CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2262718

best practices based off personal experiences. Sharing this knowledge can, ultimately, help the US Military utilize resources to the best of their ability.

The purpose of this paper is to provide advice for cross-platform development and contribute to developmental research for the VR academic community. It is important to understand the past progression of VR technology to help shape the future of simultaneous development. Once a solid understanding of VR hardware and software is established, then ways of implementing cross-platform development will be discussed. The researchers at Iowa State University's (ISU) Virtual Reality Application Center (VRAC) have developed a cross-platform football recruitment application referred to as the "Game Day" simulation, which will serve as the case study for this paper. The application was deployed on three different VR systems and was evaluated through user studies. Previous user testing of the Game Day application deployed on multiple platforms indicated users had similar immersion and presence experiences amongst the three systems[3]. Therefore, it can be inferred that all three hardware systems provided a similar user experience. The cross-platform development was done through a game engine, which allowed for the development of a similar application on all three different systems. This research illustrates the cross-platform development is feasible. The US Military's already successful training simulations, could be deployed to accessible VR devices that would allow users to have similar experiences. The work presented in this paper summarizes the researchers' experiences developing a cross-platform VR application for the use of future developers.

2. BACKGROUND

Throughout the development of VR, there has been no standard for hardware or software applications. Because of this, a diverse variety of VR hardware can be seen today and each system has a unique development process. These dissimilar systems have created huge challenges for developers who wish to create applications which operate on multiple hardware systems. Before discussing the challenges posed by cross-platform VR development and their possible solutions, it is important to understand the progression of VR hardware and software development tools as well as what attempts have been made at cross-platform development in the past.

2.1 Progression of virtual reality hardware

Some of the earliest virtual reality systems were developed in the 1960s. However, these first systems could display little more than stereoscopic film and wireframe objects. Space was also a limiting factor. Due to the immense amount of computing power required to run such a device, a reasonable form factor could not be achieved, and VR was not revisited again until the early 1990s, when technology could again attempt to accommodate VR systems. With vastly more powerful processors came a new wave of virtual reality systems[4]. The most notable of these was the Cave Automatic Virtual Environment (CAVE™) system introduced in 1992,[5] which is still used by many VR research centers today. Other devices, commercially produced by the Virtuality Group, were networked virtual reality arcade machines. These machines allowed users to strap themselves into special chairs and experience a seated multiplayer stereoscopic video game. However, despite significant advancements over the devices seen in the 1960s, VR systems again failed to find any widespread traction. The main reason for this was that computing hardware had not yet reached a level where real-time generation of convincing virtual environments was possible. Except for several CAVE™ systems being used for research purposes, VR did not see the spotlight again until 2012 when the first Oculus Rift head mounted display (HMD) was announced.[6]

The Oculus Rift HMD is one of many VR systems in use today. The most popular systems that are currently used are the CAVE™, PC-tethered HMDs, and smartphone-enabled mobile HMDs. While varied, a CAVE™ system is generally a collection of several large-scale displays that make up some or all of a fully-immersive cube. This paper will focus on the C6, the world's highest resolution CAVE™ at Iowa State University. The C6 consists of six 10'x10' rear projection display screens arranged in a cube shape, see Figure 1 on the next page. The user wears active stereo glasses and is tracked based on head position and orientation. The C6 utilizes an array of 24 projectors and 48 computing nodes to synchronously render images to the user. Systems like the C6 also have many military applications, such as acting as a controller for a swarm of UAVs.[7] However, due to the high number of nodes, the system requires a large amount of calibration to generate a seamless experience for the user. CAVE™ systems produce the most immersive VR experience,[8] but are very expensive and require a large area for operation making them less ideal for certain applications.

Figure 1. Individual experiencing the Game Day application in Iowa State University's six-sided CAVE™, C6.

HMDs provide a more consumer-friendly approach to VR hardware, as opposed to the expensive and cumbersome CAVE™ system. More specifically, the Oculus Rift DK2 is a PC-tethered HMD that aims to provide high-quality VR at a price accessible to consumers. This class of HMD relies on a near-eye display. Stereo images are split between each eye and shown to the user through magnification lenses. Research suggests that a user's level of immersion and presence are similar between CAVE™ systems and HMDs in some cases.[3] The DK2 costs around US$450, but it must be tethered to a powerful computer to run.[9] Seen below in Figure 2 is the Game Day simulation running in the Oculus Rift DK2. More recently, HMDs such as the HTC Vive and the Oculus Rift CV1 are enabling even more immersive VR experiences. While these tethered HMDs are certainly more portable than a CAVE™, they still require some amount of hardware and software setup to run properly.

Figure 2. Individual experiencing the Game Day application in the Oculus Rift DK2.

Mobile HMDs can provide the most portable and inexpensive experience, compared to the tethered HMDs connected to a powerful computer. Generally, mobile HMDs consist of a housing for a user's smartphone, and a set of magnification lenses. The smartphone is responsible for generating and displaying the visual content, as well as using the built-in accelerometer for head orientation data. It is by far the most accessible of the three VR display methods,[10] and major companies such as Google (Cardboard/Daydream) Samsung (GearVR), and many others have invested in mobile HMD technology. This paper will focus on a mobile HMD called the MergeVR, seen in Figure 3 on the next page. This low-cost (US$59) headset is made from a flexible, comfortable foam material and is compatible with a variety of smartphone devices.[11] The ability of many mobile HMDs to accommodate a wide range of smartphones makes them more accessible and simple to use by the public. This type of system also provides users with immersive VR requiring little to no setup. However, visual fidelity and head tracking are much lower quality than tethered HMDs or CAVE™ systems.

Figure 3. Game day application on the mobile HMD, Merge VR.

VR research for the US Military has already proven to be beneficial. A study conducted by Loftin, Scerbo, et al., evaluated immersive CAVE™ technology against desktop simulation technology. Trainees were instructed to stand guard at a base entrance and decide whether to allow vehicles onto the premises. Both methods had results that showed trainees effectively learned how to evaluate the vehicles, but the overall level of performance was better in the fully immersive VR technology.[12] This study is an example where the application could have been deployed on multiple VR systems to truly gauge the effectiveness of immersive technology. Through cross-platform development the study could have been done with more participants on portable HMD systems. The unique qualities of these different VR systems provide a range of functionality which can be beneficial to the US Military in many different capacities. For example, mobile HMDs could be used in the field to quickly visualize essential three-dimensional data such as building plans or combat strategies. Whereas an office environment would be better suited to utilize a tethered HMD or CAVE™ to offer a more detailed or high powered simulation for training. More research needs to be completed to determine unique roles and benefits of each VR system.

2.2 Progression of graphical software toolkits

While creating a fully-featured VR system is no small task, the quality of the supporting software framework used to create applications ultimately determines how commercially viable a system becomes. In the 1980s, software developers had to create custom interfaces for each different piece of hardware that linked directly to the operating system and CPU. For virtual reality devices of the time, a developer would have to manually render different perspectives for each eye, and deal with issues such as lens distortion and ocular distance. As expected, creating even the simplest graphical scenes took developers a substantial amount of time and required a team of skilled developers.

Some of the problems of early VR development were resolved in the 1990s, when the standard graphics library OpenGL was released. This library formed the basis for modern computer graphics.[13] Its goal was to simplify the complex interface of the operating system and allow developers to create computer graphics applications without having to worry about the specific hardware configuration of each computer. This significantly reduced the amount of development time required for graphical applications, and enabled developers to focus more on creating elaborate visual simulations. Along with increasingly powerful computer hardware, OpenGL helped spur the boom of virtual reality systems such as the CAVE™. While OpenGL significantly simplified the process of creating graphical applications, it utilizes the lower-level C programming language, and is thus not inherently object-oriented. As such, primitive geometry and object management must be procedurally created, causing a significant burden on the developer, and making it difficult to manage a multitude of complex models and animations.

To help make graphical object management more efficient, software development kits (SDKs) such as OpenSceneGraph (OSG) and OpenSG were introduced in the late 1990s. Specifically known as scene graphs, these tools further simplified complex graphical calculations, but still used OpenGL for rendering capabilities. Unlike direct OpenGL, these scene graphs are object-oriented, meaning complex geometry data is treated as a single object. By enabling easier management of complex geometry data, it became much simpler for developers to integrate large collections of models and animations into a single development project. Scene graph development tools became widely used for existing VR systems such as the CAVE™ due to their fast content creation and the customizability of their build and render methods.

One step beyond scene graph tools like OSG was the introduction of fully-featured game engines such as the Unreal Engine, Unity3D, and CryEngine beginning in the early 2000s.[14,15] These game engines combined important graphics features such as object collisions, multi-user networking, object management, graphical object editor, and artificial intelligence in a way that made them easy for developers to incorporate into virtual environments. Prior to these integrated solutions, developers had to manually implement these features, adding to the development cycle time. However, the software behind these game engines is often proprietary, meaning support for specific VR systems can only be added by the company behind the engine, not an external developer. Due to this limitation, support for VR devices only began to arrive once the Oculus Rift DK1 HMD found popularity in 2013.[9] Several VR hardware companies such as Google, HTC, and Oculus VR have worked closely with game engine creators to add game engine support for their respective VR hardware systems. In addition, they have released standalone SDKs that can be used directly, without the need for a game engine. By directly using the SDKs without a game engine, developers have much more control over rendering techniques, but lose access to convenient features such as a graphical object editor found in game engines. However, by directly using these standalone SDKs, no device support existed for other platforms, making cross-platform development challenging.

In addition to generic game engines available to consumers, several other development tools exist that are tailored specifically for building military training systems. Game engines such as Delta3D[16] and Virtual Battlespace have a more targeted focus on features essential to US Military training simulations, such as after-action review, learning management systems, and distributed interactive simulation networking. While these are less popular than commercial game engines such as Unity3D, their unique feature set may allow a simpler and more targeted development process, depending on the final goals of an application.

2.3 Cross-platform development

With the increasing adoption of both VR hardware and game engines, the ability to deploy an application across multiple systems has become a viable goal. Ideally, a developer would have the ability to simply create one application that is used on all VR platforms, but due to fundamental hardware differences, this is not yet practical. As such, development for one VR system is traditionally done entirely independent from other VR systems. Current VR systems handle essential elements, such as head tracking, very differently. For example, a CAVE™ usually uses infrared tracking nodes to determine head rotation, while a smartphone-enabled mobile HMD uses the phone's internal accelerometer to retrieve this information. In addition, developing commercial VR applications has only recently become common outside of research institutions. Therefore, no set of standards currently exists for deploying a single application across multiple VR platforms. At the end of 2016, some of the major names in virtual reality, including Google, HTC, Oculus, Sony, and Samsung announced a partnership to form a non-profit Global VR Association (GVRA) whose goal is to promote responsible development and adoption of VR globally by establishing best practices.[17] It is unclear, thus far, what set of standards will be created by the GVRA and how beneficial it will be to the academic community. In the meantime, academic researchers will continue to explore and study the idea of deploying a single application across multiple platforms.

Within the last 15 years, several researchers have developed and published studies focused on deploying the same VR application to different platforms. One such study, from Krijin, Emmelkamp, et al. looked at treatment methods through CAVE™ systems and HMDs for patients with acrophobia (fear of heights). While the results between the two VR devices did not show any significant differences, the data suggested that VR therapy sessions produced a similar result to traditional treatment methods.[18] If an application deployed to multiple different VR devices truly does yield similar results, then organizations like the US Military can deploy VR application on whichever VR system is readily available without sacrificing user experience. The paper did not discuss the details on the development of the application, but the research shows that there is a need for cross-platform VR development. The VR community can benefit from researchers sharing their methods and ideas to keep motivating simultaneous cross-platform development.

Unfortunately, most VR cross-platform research has only been functionally tested on no more than two VR platforms at a time. Fewer studies look at user collaboration using many different virtual environments. One of the few examples was an experiment published by Thalmann et al. They conducted a multiplayer user study volleyball game on four different platforms: CAVE™, HMD, stereo screen, and mono screen. The CAVE™ and HMD presented users with a better experience due to a higher sense of presence and immersion. More importantly, this study showed that players were able to effectively communicate and work with each other when using different VR systems.[19] This provides a good indication that different VR platforms can be coherently linked to work toward a common goal. Furthermore, it shows that VR collaboration need not be approached with a "one-size-fits-all" mentality. In terms of the US Military, the ability to effectively collaborate in a virtual environment across separate platforms could prove to be very valuable. By effectively utilizing cross-platform development, US Military personnel could minimize new specialized equipment purchases and

instead focus on currently-available hardware. As more cross-platform research is performed, different VR systems could be used interchangeably, depending on the situation. Through the maturation of the GVRA and with further research, VR development standards will ultimately help organizations such as the US Military to realize the true potential behind collaborative cross-platform virtual environments.

3. DEVICE SELECTION

Given the increasing number of different VR systems becoming available, it is important to determine which systems best fit the goal of the project. Would the project benefit more from the mobility of a smartphone-enabled HMD, or the visual fidelity afforded by an HMD tethered to a PC? If a CAVE™ system is available, is it worth the time to implement an entirely separate interaction system? Using the considerations discussed in this paper, it is possible to produce an application that runs across all of these VR systems. Despite this, with the current state of available development tools, it still requires a targeted effort to tailor-make and maintain an application for each desired platform.

The current VR hardware offerings described in section 2.1 highlight the different systems used by the Game Day simulation developed by the VRAC. By creating a single VR experience that is rendered in the C6, the Oculus Rift DK2, and the MergeVR (using iOS), all possible demonstration situations were accommodated. The C6 and similar CAVE™ systems provide users with the most natural and immersive experience. It also enables a shared view that allows direct interaction between users. This is beneficial for training small groups of warfighters requiring a large amount of physical interaction. In contrast, the Oculus Rift DK2 provides a high-quality individual VR experience for users with no access to a CAVE™. Using high-performance laptops to power the DK2, users are given an experience very similar to the one experienced in a CAVE™, without the accompanying costs or location constraints. This allows geographically-distributed warfighters to be given highly immersive VR training without the need for travel or expensive systems. Lastly, the MergeVR is used to provide portable VR experiences in almost any location. The MergeVR HMD and iPhone require no setup, and allow for quick 3D visualization. This could prove invaluable to delivering spatially-accurate data to warfighters in the field, such as a building layout or combat plans.

4. DEVELOPMENT PLATFORM SELECTION

While choosing the right VR hardware platforms for a project is vital, selecting the proper development platform is likely the most important decision of the entire design process. If chosen incorrectly, the development time may be extended unnecessarily due to any number of reasons, from lack of device build support to lack of sufficient customization options. Currently there are many viable VR development platforms, spanning all levels of customizability and ease of use. When beginning to develop a complex simulation for multiple platforms, a choice must be made between using a highly customizable scene graph kit or a fully-featured game engine.

Generally, scene graph kits are highly customizable, and provide developers with a large amount of control over the graphics rendering pipeline. In the past, the VRAC built most of its 3D applications using the scene graph kit, OpenSceneGraph (OSG). OSG is a powerful open-source graphics programming interface that utilizes C++ and OpenGL to efficiently render 3D geometries across a range of operating systems. In addition, OSG works well with VR Juggler, an application development framework created by researchers at Iowa State University to provide a system-independent operating environment for VR[20]. Using VR Juggler with the customizability of OSG, applications can be deployed to the C6. Since multi-node CAVE™ systems are uncommon, very few software solutions are available to aid in the deployment of applications to the C6. Because of this, the synergy between OSG and VR Juggler makes OSG an appealing option. While OSG is a powerful scene graph kit capable of near limitless customization, the complexity of application development drives many to look for other developmental solutions. In the US Military, new hardware and situation training protocols are continually being developed. If an accompanying VR simulation takes a year or more to be created, the simulation will be outdated the moment it is released, leading to a substantial amount of wasted development time. This makes choosing the most efficient development platform for cross-platform VR systems all the more crucial.

Over the last several years, major game engines such as Unity3D, Unreal Engine, and CryEngine have become increasingly popular. By integrating features such as object collisions, networking, a visual editor, and much more, all in one place, they have become an attractive offering to creating all types of 3D applications. However, each game engine has

differences that distinguish it from the others. For example, Unreal and CryEngine both required a paid subscription for access to the engine at the time of the development of the Game Day application, while Unity3D was offered at no cost. Because of this open access, development resources such as forums and documentation, were substantially more available for Unity3D than for Unreal or CryEngine. Additionally, Unity3D offers the ability to deploy to more systems than either Unreal or CryEngine, and even allows CAVE™ support through external plugins. However, both the Unreal Engine and CryEngine are known to have superior photorealistic rendering capabilities over Unity3D. Despite this, Unity3D was the game engine chosen to be evaluated, based on cheaper costs, superior platform support, and available development resources. Thus, an evaluation was needed to choose between using OSG or Unity3D for the Game Day simulation development. Several categories were investigated, including developer resources and tools, customization capabilities, support for different build platforms, and maintenance.

4.1 Resources and content creation

OSG and Unity3D are both well-known in the graphics community, and each have actively contributing developers and ample documentation. However, the features of each are vastly different. First, the Unity3D editor is built around a graphical user interface that developers can use to move, add, or modify any object or setting in an application. This interface also includes a powerful content management system that allows developers to easily add, categorize, and utilize all assets necessary for an application. To compliment this, Unity Technologies maintains an asset store that holds an immense collection of free and paid game-ready assets that can be added to any application. OSG lacks any such editor, and all modifications and assets must be handled directly in the source code. Lastly, due to Unity3D's immense popularity,[21] primary GPU manufacturers such as NVidia work closely with Unity Technologies to help them integrate the newest rendering techniques into their game engine.[22,23] This is especially important for VR applications, as high framerates and visual fidelity are necessary to give users a comfortable and immersive experience. For US Military training applications, accurate visual information and responsiveness are critical to replicating the real-life situations in a VR simulation.

4.2 Customizability

In terms of raw customizability, OSG is the clear winner due to its transparency, allowing a developer to modify any part of the graphics rendering pipeline. While this could potentially lead to a more efficient renderer, this also means that much of the display and interaction logic is left to the developer to implement. Without an exhaustive collection of templates to pull from, this results in a long development cycle, as each application has a unique set of models and interactions. In contrast, Unity3D's base engine takes care of the graphics rendering process. Thus, the developer does not need to spend any time implementing a rendering procedure for an application. Unity3D does still provide several ways to modify its renderer with the use of custom shaders and material properties. While this simplification makes Unity3D very easy to learn and use, it gives developers much less control over how the application is ultimately displayed. If deploying an application to an embedded system such as a warfighter's HUD helmet, the overhead of Unity3D may be too much for the system to handle. In these situations, the customized and optimized rendering techniques of OSG may prove advantageous.

4.3 Build support

Both OSG and Unity3D support numerous different platforms, albeit in very different ways. In the case of OSG, build support is added by manually writing the libraries needed to interface with the target system. Due to OSG's open-source nature, build support can theoretically be added for any display platform and various operating systems, but it can be an extremely complicated process. Given enough time, some precompiled libraries for OSG might be found for popular systems, but they are often largely untested and take many months to be released. To contrast this, Unity3D provides a large selection of simple precompiled build modules available for popular platforms, including Windows, Mac, iOS, Android, SteamVR, and Oculus to name a few.[24] Since Unity Technologies is a for-profit company with paid employees, build support often arrives relatively soon after a popular system release. However, since the Unity3D game engine is proprietary, an external developer cannot add any custom build support, and is limited by what the company deems useful to support.

While mobile and PC-tethered HMDs have become popular enough to receive widespread build support, CAVE™ systems remain uncommon and highly complex. The number of walls and nodes, as well as protocols for clusterization, serialization, and synchronization must all be considered. There are several plugins available to deploy VR applications to a CAVE™ system. Firstly, VR Juggler is a virtual development environment that provides a system-independent operating environment for VR applications. Just like OSG, it is a very powerful tool, but requires a lot of development and maintenance to efficiently utilize it. Another offering is a Unity3D plugin called getReal3D. Created by Mechdyne, this

plugin integrates directly into Unity3D and allows VR applications to be deployed to compatible CAVE™ systems running Windows operating system. Other tools such as MiddleVR and RUIS are possible alternative offerings.[22] However, analyzing the performance and usability of these different CAVE™ development tools is beyond the scope of this paper. An in-depth review of popular Unity3D toolkits for CAVE™ development has concluded that, while some toolkits are better in certain areas, each CAVE™ system may benefit most from a different Unity3D toolkit, based on application constraints.[22]

Ultimately, less time devoted to providing build support for a specific platform means more time spent creating and polishing the actual simulation. As such, if developing for a popular platform already supported by Unity3D, deployment simplicity is far superior with Unity3D than with OSG. However, if the target platform is not publicly well known, and uses a unique display system, the open and customizable build system of OSG could prove more effective. For the US Military, both OSG and Unity3D may be advantageous. On the one hand, proprietary display technologies will not have Unity3D build support, and the engine cannot be used to deploy applications to the device. On the other, if the US Military utilizes popular consumer VR devices such as the HTC Vive for training, Unity3D readily supports many of them, greatly simplifying development and deployment.

4.4 Maintenance

Once an application is fully developed, the software maintenance cycle begins. The time, effort, and skill required to perform this maintenance needs to be considered and planned for early in the software development cycle. For OSG, it is very difficult and time-intensive to update an application between system updates or OSG version updates. In either of these cases, many of the libraries must be recompiled by hand, and the project needs to be rebuilt and redeployed to the target system. While precompiled libraries can sometimes be found, they are often very generic, unoptimized, and slow to arrive. On the other hand, updates to the Unity3D game engine remain backward-compatible across several versions, and come packaged with an updater that automatically attempts to update an old Unity3D application to the newest version.

The maintenance and update cycle is far from streamlined, meaning that many undocumented or unknown issues could arise. As such, skilled technicians are often needed for complex update and maintenance work. In addition, these training systems are often spread across a wide geographical area. It could be a very challenging task to have enough skilled technicians on-site capable of working with these distributed platforms. As such, the complexity of the maintenance process must be kept as low as possible to avoid substantial costs and system down time.

4.5 Final selection

After the analysis, the Unity3D game engine was selected to be the primary development platform for the VR Game Day application. As seen in the table on the next page (Table 1), Unity3D supports the largest number of platforms, has a huge collection of development resources, and can be used many different types of experiences, from 2D games to fully-immersive 3D VR games. The superior content management system of Unity3D also allows for easy integration of the dozens of time-sensitive animations and rigged humanoid models that were part of the application. Without the use of Unity3D's GUI editor window, the application would have taken many more months to properly set up and orient the complex geometries. Since the Game Day application was being deployed to three separate VR systems, cross-platform compatibility made Unity3D very appealing. The getReal3D toolkit was selected to clusterize the application for the C6 for several reasons. Mechdyne built the VRAC's C6, and they developed the getReal3D plugin so that their customers can deploy applications to their CAVE™ systems. Since the C6 is one of the largest clusters across their customers, Mechdyne often uses it as their testbed platform. As such, compatibility with the C6 was very likely when using getReal3D. In addition, at 48 individual nodes, the C6 contains more nodes than most CAVE™ systems. Some toolkits are not built to handle this amount of network traffic and content synchronization, leading to slow rendering speeds and frequent graphical glitches. With the rich feature set provided by Unity3D and getReal3D, all developmental constraints of the Game Day simulation could be met.

Table 1. Comparison of common graphics engine features for Unity3D, Unreal, CryEngine, and OSG.

Graphics Engine				
	Unity3D	Unreal Engine	CryEngine	OpenSceneGraph
Programming Language	C#, JavaScript	C++	C++	C++
Supported Platforms	25	13	6	Unlimited (added manually)
Price	Free (<$100k revenue)	Free (< $50k revenue)	Subscription ($10/mo)	Free, open-source
Development Resources	Vast collection of assets and tutorials	Growing collection of assets, but limited coding resources	No asset store, limited developer resources	No asset store, no editor GUI
Optimized for Game Type	2D, 3D, VR	2D, 3D, VR, primarily for first-person	3D	Variable

5. IMPLEMENTATION

Once the development platform for the Game Day application was chosen, the actual software development began. For the C6 CAVE™ system, development was straightforward. The getReal3D plugin was obtained directly from Mechdyne. This plugin was imported into the Unity3D project, and several simple adjustments were made to the camera object. Adding support for the Oculus Rift DK2 was equally as simple. After installing the base Oculus application to the PC that powered the DK2, it was a simple matter of downloading and importing the Oculus Utilities package into the Unity3D project. This package includes a basic camera object that is dropped into the scene, allowing full support for the Oculus Rift DK2. The process to deploy the Game Day simulation to an iPhone was slightly more complicated, due to Apple's restrictions on deploying content to iOS devices. Once the MergeVR Unity3D package was added to the project, the application was compiled into an Xcode solution. Once a valid developer license was obtained, the application was built in Xcode and deployed to the iPhone.

While developing the Game Day simulation for multiple VR platforms, several issues and considerations were discovered regarding input, rendering difficulties, and hardware support. This section discusses some of these problems, and the solutions used to address them.

5.1 Input

Implementing user input and interaction is a major part of any application, and is especially important for providing a sense of immersion in VR simulations. Input for the C6 is handled in a unique way. A background program called "trackd", also developed by Mechdyne, is used by getReal3D to handle all interaction and input. It is primarily responsible for tracking head position and synchronizing user input between all C6 nodes. When development on the VR Game Day simulation began, the way "trackd" handled input serialization was not known. As such, the default Unity3D input module was used for input management. However, this led to very odd behavior in which only the master C6 node would receive input, but no other nodes would. After analyzing the serialization scripts, it was determined that getReal3D implements its own unique input system, separate from Unity3D's default module. By modifying the app to use the input system provided by getReal3D, the issue was resolved, and input was delivered synchronously to all C6 nodes. Beyond just CAVE™ systems, it's important to learn how any Unity3D device plugin functions, and what system is in place to handle input. In addition, even though build support may exist for a platform, there is no guarantee that Unity3D's built-in input module will be able to understand or propagate user input events without the use of a device-specific plugin.

Input for the Oculus Rift worked very differently than the C6. At the time of development, the Oculus Rift did not have any Touch controllers, so an RF Logitech gamepad was used to control the scene and the player. Since the Logitech Rumblepad gamepad was a popular input device, Unity3D could recognize the gamepad, and provide a full default button profile for the gamepad. Since the Oculus Rift is tethered directly to a Windows computer, no external plugin was needed to understand the gamepad input, and the built-in input system of Unity3D was sufficient for controlling the player and the scene.

Compared to the C6 and the Oculus Rift, managing user input for iOS proved the most challenging. Since smartphones do not have the capability to receive input from an RF gamepad, a separate Steelseries Bluetooth gamepad was used for navigation and to trigger animations. Unfortunately, Unity3D was not able to recognize or produce a button mapping profile for this gamepad. In this particular situation, this meant that Unity3D treated all different button presses as the same button value. Therefore, even though Unity3D's built-in input system could be used, it was rendered unusable. To fix the issue, there were two potential resolutions. One method would be to develop a driver that worked at the operating system level, and would act to correctly differentiate button input when passed to Unity3D. However, developing a driver is a complicated process, and was not feasible for the development timeframe. The fix that was selected was to modify the interaction to be sequential, allowing any button press to advance the animation sequence one step. This method was successful because the animation events in the VR Game Day simulation were already designed to be sequential. Therefore, this fix may not work for all applications. As such, it is imperative to make sure that the chosen input controller is compatible with the selected development platform.

The US Military employs a vast array of input technologies, from speech recognition to gaze detection.[25,26] With such a spread of unique input methods, the ability to effectively manage and utilize them is imperative. While Unity3D does not natively support these complex input mechanics, plugins can be purchased or created that can seamlessly add custom input support to any Unity3D application. This has already been done for many popular systems, such as Tobii EyeTracking or a speech-to-text plugin directly from Unity Technologies.[25] By leveraging these existing resources, powerful interaction techniques are integrated into a project with little to no effort.

5.2 Visual display

While meaningful user interaction furthers the feeling of immersion, nothing is more important than the visual elements of a VR application. Larger applications often contain a substantial number of active assets such as models and animations. The large amount of geometry poses the risk of lowering the overall framerate. In the case of the C6, this could cause the connection between the nodes to saturate, meaning there is more data being sent over the network than can be received. The first several versions of getReal3D suffered from unoptimized node synchronization, causing the framerate to drop as each node had to wait for the status of all other nodes. A major reason for this was that Unity3D had an entirely closed-loop graphics architecture in previous versions. Unfortunately, this did not allow Mechdyne to provide custom edits to the getReal3D plugin outside of configuration files, so no in-house synchronization or rendering optimizations could be done. However, newer versions of Unity3D are slowly allowing more access to the internal graphics pipeline. This has allowed Mechdyne to substantially reduce the network synchronization burden between the master node and the cluster nodes. With the release of several new getReal3D versions, the framerate of the C6 rose from an unstable 10fps to well over 40fps. While this is still considered a low framerate for many VR applications, the getReal3D plugin is actively being improved upon.

For the Oculus Rift DK2, framerate is much more dependent on GPU power than communication bandwidth. The better the graphics card, the more geometry that can be rendered at the same framerate. This must be done on a per-application basis, as there is no metric for pre-evaluating graphics card performance. However, a developer should always aim for the (currently) recommended 90 Hz refresh rate, which is critical for maintaining a comfortable VR experience.[27] Whether that means purchasing a more powerful graphics card or diminishing visual effects, not achieving this framerate often leads to a user experiencing simulation sickness.

Once again, mobile iOS development had the largest number of issues regarding rendering capabilities. Perhaps one of the largest drawbacks of mobile VR is its limited computing power compared to non-mobile systems. Specifically, the amount of geometry had to be significantly reduced to allow the application to even render on an iPhone. This was primarily done by decimating many of the included 3D models to a level that a mobile smartphone could handle. For example, one of the humanoid models was reduced from almost 273,000 polygons to just under 14,000 polygons. Even though the higher polygon model is more realistic, the increased framerate and responsiveness that came from using the lower polygon model led to an overall more immersive experience. In addition to geometry limitations, Unity3D does not natively support non-full screen video playback on mobile devices. Since the VR Game Day simulation used video as an integral part of the experience, an external plugin was needed to overcome this issue. A plugin was purchased directly from the Unity Asset Store that contained specific libraries used to play mp4 videos on iOS. With little effort, this plugin was integrated into the Unity3D scene, and video could be accurately played on the iPhone simulation.

The visual accuracy of US Military VR training simulations is vital to their overall effectiveness. If a simulation is improperly displayed, a warfighter may be trained on a situation that does not accurately represent reality, ultimately

harming performance. However, as previously discussed, there are limits to the visual fidelity and geometry that can be displayed. Even with continuous improvements to computing power, a balance must always be struck between visual fidelity and simulation responsiveness. This balance may be difficult to identify, but will lead to a more effective training experience.[28]

5.3 Hardware support

Even though Unity3D boasts a large list of supported build platforms, it is not always a straightforward process. For example, Unity3D can use getReal3D to deploy applications to CAVE™ systems, but the application is actually built as a 64-bit Windows application. This means that any CAVE™ system which utilizes getReal3D must also run the Windows operating system. In addition to software-enforced build platforms, hardware limitations also need to be considered.

While developing for the Oculus Rift DK2, it was determined that at the time, very few laptops supported virtual reality. This was due to two main issues. First, the video output on most laptops is tied directly to the motherboard's onboard GPU, which is significantly less powerful than any dedicated GPU found in the laptop. This cannot be circumvented, and is inherent to the design of the laptop. For example, Dell Inspiron laptops can be custom-built to have a dedicated NVidia GPU, but this is not required. To maintain a streamlined assembly process, Dell tied the weaker integrated GPU directly to the display output (e.g., HDMI), regardless of whether an additional NVidia GPU was installed. Such a configuration would let applications use the NVidia GPU when displaying to the native laptop screen, but external displays such as an Oculus Rift were forced to route through the weaker integrated GPU. Since the power of the integrated GPU was not near capable of rendering stereo 3D applications at 90Hz, this architecture effectively disallowed Dell Inspiron laptops from being used to power the Oculus Rift DK2. The second issue, even if the display output was linked directly to the dedicated GPU, accompanying VR-ready drivers were also needed. In the case of the NVidia Quadro series, no such drivers were available, not allowing applications to be displayed in the Oculus Rift DK2. In the last year, many companies have begun offering "VR-ready" laptops that contain dedicated GPUs designed specifically for VR devices. However, these topics must still be considered if deploying to older machines, or to ultra-portable laptops.

If a game engine such as Unity3D is chosen for mobile VR development, it is important to use the plugin associated with the specific device being used. Most mobile VR plugins do essentially the same thing – split the camera into two viewports, and show one to each eye. However, each different mobile VR plugin often comes optimized for the corresponding device, taking into account parameters such as ocular distance and lens distortion. If the hardware and software are mismatched, the user will likely become disoriented in the virtual environment, and simulator sickness may even occur.

6. CONCLUSION

When building a single VR application that will be deployed to multiple vastly different VR systems, there are a number of issues that must be considered. From development platform selection to CAVE™ toolkits, each choice could have a dramatic effect on the overall development process. Game engines have become integral to 3D and VR application development, from their powerful graphical interface editor and content management system, to their growing list of supported devices. For most applications, using a fully-featured game engine such as Unity3D will lead to vastly shorter development times and a simpler maintenance cycle. However, if an application needs to be built with efficiency in mind, the full and direct control over the rendering pipeline provided by OSG may prove more effective. Topics such as user input, visual fidelity, and hardware support also need to be considered when developing cross-platform VR applications. When managing user input, it is important to recognize which input methods are most effective, and if the input devices are natively supported on the chosen development platform. Additionally, a developer needs to actively balance the quality of models with the performance of the application.

Operating systems also present a significant challenge when deploying a VR simulation across different devices. Even though Unity3D allows clusterization through plugins such as getReal3D, they often only run on a specific operating system. Since getReal3D requires a Windows platform to function, the CAVE™ nodes must also be running Windows, which may not be feasible for all configurations. Due to the high variability in CAVE™ systems, clusterization and node count heavily impact the performance of deployment tools like getReal3D.

Given the diverse visual systems available to the US Military, from fully-immersive training simulations to embedded HUD helmets, the ability to efficiently deploy an application across multiple platforms could prove invaluable. By following the best practices and considerations presented in this paper, development time and deployment complexity can be effectively managed, allowing VR to become an even more valuable asset for military training and simulation.

REFERENCES

[1] strategicdefence intelligance., "The Global Military Simulation and Virtual Training Market 2015-2025 Now Available at Market Reports Store," 2015, <http://marketreportsstore.com/the-global-military-simulation-and-virtual-training-market-2015-2025/> (26 January 2017).

[2] "Training Convoy Skills - Military Training International.", Def. House Publ., 2016, <http://www.mti-dhp.com/defense-news/training-convoy-skills/> (1 March 2017).

[3] Miller, J., Baiotto, H., MacAllister, A., Hoover, M., Evans, G., Schlueter, J., Kalivarapu, V.., Winer, E., "Comparison of a Virtual Game-Day Experience on Varying Devices," Electron. Imaging **2017**(16), 30–37 (2017).

[4] Mazuryk, T.., Gervautz, M., "Virtual Reality History, Applications, Technology and Future."

[5] Cruz-Neira, C., Sandin, D. J.., DeFanti, T. A., "Surround-screen projection-based virtual reality," Proc. 20th Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH '93, 135–142, ACM Press, New York, New York, USA (1993).

[6] Kushner, D., "Virtual reality's moment," IEEE Spectr. **51**(1), 34–37 (2014).

[7] Foo, J. L., Knutzon, J., Oliver, J.., Winer, E., "Three-Dimensional Path Planning of Unmanned Aerial Vehicles Using Particle Swarm Optimization," 11th AIAA/ISSMO Multidiscip. Anal. Optim. Conf., American Institute of Aeronautics and Astronautics, Reston, Virigina (2006).

[8] Juan, M. C.., Pérez, D., "Comparison of the Levels of Presence and Anxiety in an Acrophobic Environment Viewed via HMD or CAVE," Presence Teleoperators Virtual Environ. **18**(3), 232–248 (2009).

[9] "Oculus Rift DK1 Release Date, News & Reviews - Releases.com.", Releases, 2014, <https://www.releases.com/p/oculus-rift-dk1> (1 March 2017).

[10] Lohrmann, D., "Will a Smartphone Replace Your PC?," Gov. Technol., 2016, <http://www.govtech.com/blogs/lohrmann-on-cybersecurity/will-a-smartphone-replace-your-pc.html> (22 February 2017).

[11] "Shop - Merge VR - Virtual Reality, powered by your smartphone.", Merge Labs, Inc., 2017, <https://mergevr.com/shop> (1 March 2017).

[12] Loftin, R. B., Scerbo, M. W., McKenzie, R., Catanzaro, J. M., Bailey, N. R., Phillips, M. A.., Perry, G., "Training in Peacekeeping Operations Using Virtual Environments" (2004).

[13] Segal, M.., Akeley, K., "The OpenGL R Graphics System: A Speciication Version 1.2.1" (1998).

[14] Fritsch, D.., Kada, M., "Visualisation Using Game Engines" (2004).

[15] Al-Najdawi, N., "Introduction to Visualization using Game Engines" (2007).

[16] McDowell, P., Darken, R., Sullivan, J.., Johnson, E., "Delta3D: A Complete Open Source Game and Simulation Engine for Building Military Training Systems," J. Def. Model. Simul. Appl. Methodol. Technol. **3**(3), 143–154, SAGE PublicationsSage UK: London, England (2006).

[17] "Virtual Reality Industry Leaders Come Together to Create New Association - Global Virtual Reality Association.", Glob. Virtual Real. Assoc., 2016, <https://www.gvra.com/virtual-reality-industry-leaders-come-together-to-create-new-association/> (1 March 2017).

[18] Krijn, M., Emmelkamp, P. M. ., Biemond, R., de Wilde de Ligny, C., Schuemie, M. J.., van der Mast, C. A. P. ., "Treatment of acrophobia in virtual reality: The role of immersion and presence," Behav. Res. Ther. **42**(2), 229–239 (2004).

[19] Thalmann, D., Lee, J.., Thalmann, N. M., "An evaluation of spatial presence, social presence, and interactions with various 3D displays," Proc. 29th Int. Conf. Comput. Animat. Soc. Agents - CASA '16, 197–204, ACM Press, New York, New York, USA (2016).

[20] Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A.., Cruz-Neira, C., “VR Juggler: a virtual platform for virtual reality application development,” Proc. IEEE Virtual Real. 2001, 89–96, IEEE Comput. Soc.

[21] Unity Technologies., “Unity - Fast Facts,” <https://unity3d.com/public-relations> (22 January 2017).

[22] Ritter Iii, K. A., Borst, C. W.., Chambers, T. L., “Overview and Assessment of Unity Toolkits for Cave Automatic Virtual Environments and Wand Interaction,” Int. J. Innov. Educ. Res., 3–7 (2015).

[23] Lang, B., “Unreal Engine & Unity to get Simultaneous Multi-Projection Rendering,” Road to VR, 2016, <http://www.roadtovr.com/unreal-engine-unity-vr-simultaneous-multi-projection-smp-nvidia/> (6 March 2017).

[24] “Unity - Multiplatform - Publish your game to over 25 platforms.”, Unity Technol., 2017, <https://unity3d.com/unity/multiplatform> (1 March 2017).

[25] Jacob, R. J. K.., Karn, K. S., “Eye Tracking in Human–Computer Interaction and Usability Research: Ready to Deliver the Promises,” Mind 2.3 (2003).

[26] Beek, B., Neuberg, E.., Hodge, D., “An assessment of the technology of automatic speech recognition for military applications,” IEEE Trans. Acoust. **25**(4), 310–322 (1977).

[27] “Rendering Techniques.”, Oculus VR, LLC., 2016, <https://developer3.oculus.com/documentation/intro-vr/latest/concepts/bp_app_rendering/>.

[28] McMahan, R. P., Bowman, D. A., Zielinski, D. J.., Brady, R. B., “Evaluating Display Fidelity and Interaction Fidelity in a Virtual Reality Game,” IEEE Trans. Vis. Comput. Graph. **18**(4), 626–633 (2012).