3D Engines for Mobile Platforms

Howard Mall
ECS, Inc.
Orlando, FL
Howard Mall@ecsorl.com

ABSTRACT

3-D, immersive games are no longer just for game machines (like the Xbox 360) or high-end desktop computers with awesome graphics cards. Mobile devices such as smart phones and tablets have now grown powerful enough to render realistic 3-D graphics and supply high-fidelity game play on the go. The release of games like “Dungeon Defenders” on Android and “Epic Citadel” on iOS show the validity of porting an existing game engine (Unreal 3) to mobile platforms. Unity3D has also seen exceptional growth with their “write once, deploy anywhere” approach to game development. This paper explores what it takes to port and distribute a 3-D immersive, combat medic simulation to mobile platforms. We will examine the selection of game engine, explore the decision process for buy versus make, look at changes in interface from a desktop application to a touch interface, detail changes in the art pipeline, take you through the process of getting the game deployed on various app stores, and look at possible alternative delivery venues. We conclude with a post-mortem on the success of the port.

ABOUT THE AUTHORS

Howard Mall is Vice President of Engineering at Engineering and Computer Simulations, Inc. He has spent the last seven years building various kinds of training systems. He led efforts for the Navy to develop training solutions deployed on cell phones and hand-held computers. For the Army, he delivered the Tactical Combat Casualty Care (TC3) Simulation used by combat medics to learn triage and medical decision-making on a virtual battlefield. He led the development of the Emergency Management Nexus, a next-generation synchronous training platform for the National Guard Bureau that has become a virtual world platform serving myriad federal agencies. He currently oversees multiple engineering efforts at ECS.

3D Engines for Mobile Platforms

Howard Mall
ECS, Inc.
Orlando, FL
Howard Mall@ecsorl.com

INTRODUCTION

Mobile gaming is getting very popular and there are now some very high quality 3-D games available for iPhones and Android devices. This is due to the increased power available to handheld devices. Recent iPads, iPhones, iPods and various Android devices all support pretty decent 3-D graphics rendering. Many engines are now available for these platforms to support 3-D games.

This work will explore the issues and efforts in creating a 3-D serious game for mobile platforms. To closely examine the limitations imposed by a mobile platform and to compare and contrast against the capabilities of a desktop game, an existing simulation will be ported to one or more mobile platforms.

The Tactical Combat Casualty Care Simulation (TC3sim) was chosen for its visual and behavioral complexity. This effort began with some experimentation with a number of game engines looking at the difficulties of importing TC3sim assets into their framework. Unity3D was finally selected to port TC3sim in earnest and proved to be quite effective in creating an efficient asset pipeline, recreating the simulation logic of the original TC3sim, and allowing the developer to easily target both Apple and Google Android mobile devices.

BACKGROUND

The Tactical Combat Casualty Care Simulation (TC3sim) is an effective, desktop, first-person serious game for training US Army Combat Medics (Sotomayor, 2010). A version has also been created for the US Marine Corp called the Computer Based Corpsman Training System (CBCTS). This game was originally created using Gamebryo rendering combined

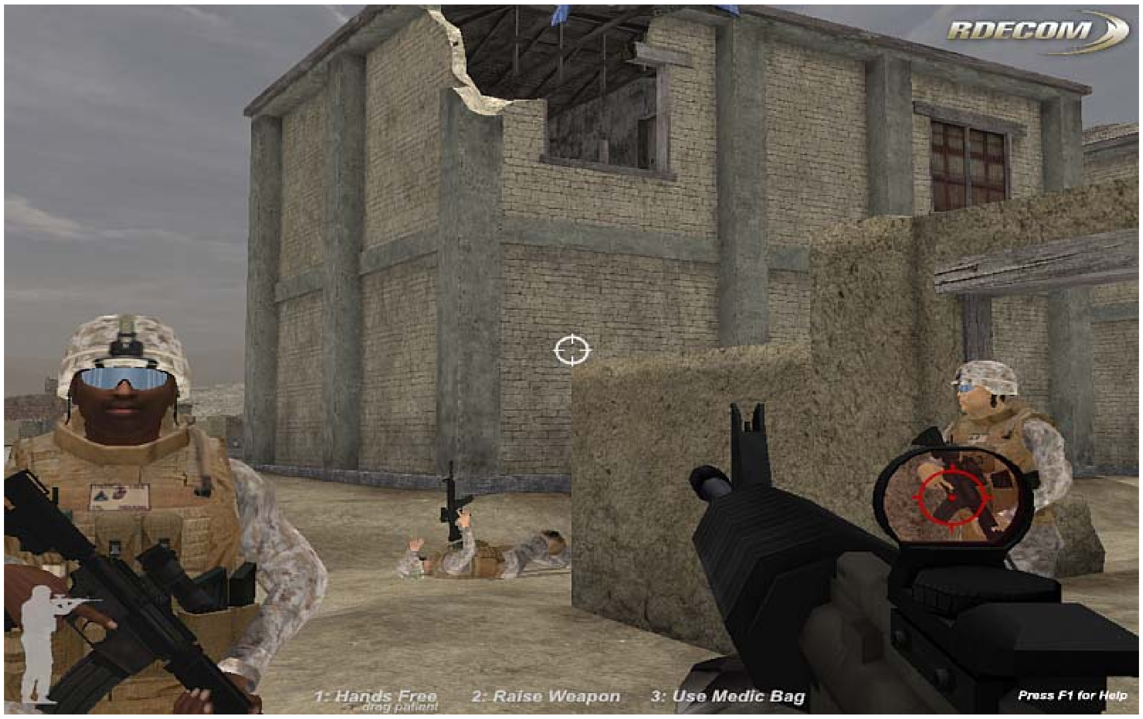

Figure 1: *CBCTS is a version of TC3sim done for the Marine Corp.*

with a game engine using Lua scripting for behaviors. (See Figure 1.) The current version is replacing Gamebryo with the open source (and highly capable) Ogre3D rendering engine.

TC3sim allows the player to choose among a number of scenarios playing the role of a combat medic attached to a fire team. During the course of an exercise many of the combat medic's teammates can be injured by gunshots and explosives. The combat medic must choose how to react to the situation. They must triage their patients, assess their injuries, choose a treatment plan, and keep from getting injured themselves.

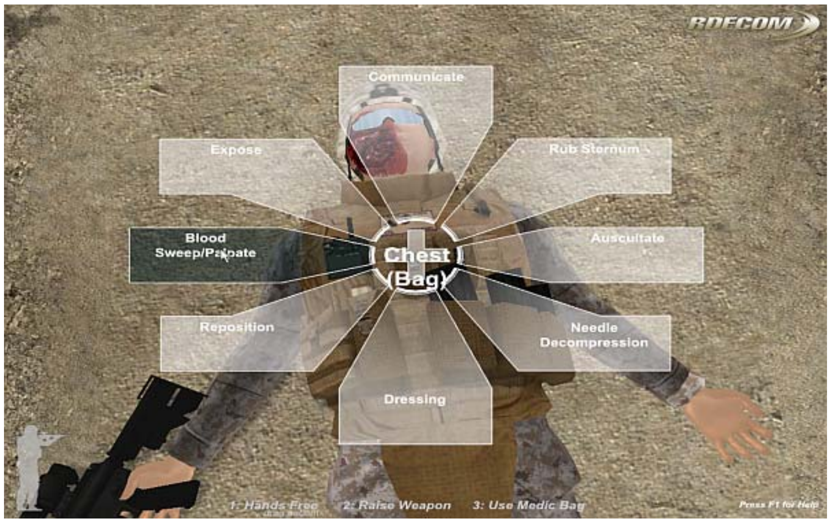

Figure 2: *The Contextual "Pie" Menu used in TC3sim and CBCTS*

The interface is very familiar to those used to playing first person shooter games. The player manipulates their point of view using the mouse. Keyboard keys are mapped to move the combat medic forward, backward, and side-to-side. Keyboard keys can also allow the user to kneel or go prone, raise their weapon, use their medical bag, go hands free, or toggle between running and walking.

A reticule sits in the center of the screen. This can be used to aim their weapon, but more importantly, as the player places the reticule over objects in the environment they can right click their mouse to access a contextual menu (see Figure 2.) In this way, the player can select parts of a synthetic casualty's body and choose tests or treatments (e.g. take pulse, apply bandage, conduct a blood sweep, etc.) These options appear as a "pie menu" around the reticule. The user clicks on one of eight sections to select that option and perform that action on the synthetic casualty.

The interface to the synthetic casualty is the most significant part of the TC3sim. Representing it on a mobile device was a major part of this endeavor and required creativity and an understanding of what could be accomplished on a small screen with a touch interface.

PLATFORMS

Two platforms were selected for this work due to their popularity and capability: iOS and Android. Two platforms were selected over one to assure that there was some contrast against which the work could be evaluated. In the development of any mobile application the selection of platform is major factor in the success of you project (Chludzinski, 2010.)

iOS

iOS is Apple's operating system for iPhones, iPads, and iPod Touches. iPhones and iPod Touches are nearly identical in capability except iPod Touches do not have phone capability and some other subtle differences. The iPad is their larger cousin with a 10 inch screen.

You must register as a developer in order to be able to access their Software Development Kit (SDK) and their developer program. A normal registration (at the time of this writing) is $99 USD a year for a normal license and $299 USD for an enterprise license. The enterprise license allows you the ability to distribute applications to designated hardware within an enterprise without going through Apple's App Store approval process. The normal license allows you to register hardware only for development and to install your software for testing purposes. Either license allows you to submit applications to Apple's App Store.

Typically, you must also own an Apple made computer (such as the MacBook Pro) in order to develop for iOS. (There are other "fringe" options that allow you to develop on Linux but they can prove very difficult to implement and maintain.) Xcode is the Integrated Development Environment (IDE) under Apple's OS X through which development for iOS devices is conducted. Xcode uses a C-derived language called

Objective C that is a contemporary of C++. However, Xcode also supports C++ and C development quite well allowing for parallel development. Objective-C projects can compile and use C and C++. There are also many third party development tools that allow development in other languages, but they eventually generate Objective-C and are compiled by Xcode.

For 3-D development, iOS supports OpenGL ES. OpenGL is a low-level Application Programmer's Interface (API) for graphics rendering that has become a standard across many platforms and graphics cards. OpenGL ES (Embedded System) is a subset of OpenGL for smaller footprint hardware such as cellphones and appliances.

Android

The Android operating system is actively developed by Google but is also *mostly* open source. That allows for mobile phone vendors to create distributions of Android for their particular hardware without having to write everything from scratch. In addition to an operating system, Android includes layers for user interface and a number of native applications. As it is open source, there are some developers who create their own distributions (i.e. cyanogen) for use on "un-locked" phones.

There are many options for building programs for Android, but officially, Google supports a Java SDK and a C/C++ Native Development Kit (NDK) for coding more complex, CPU-intensive applications. You can develop on any number of desktop platforms (Windows, OS X, Linux, etc.) and there are a host of tools available to the developer. There are now also many scripting languages supported on Android like Python and Lua.

Anyone can write Android applications and they can be installed very simply by loading an application package file onto the device. You do have to register as a developer (a one-time fee of $25 USD) if you wish to distribute applications via Google's Android Market. There are also alternatives to Google's market available on the internet as well as downloading application package files directly from developers' websites.

Android's commitment to platform diversity is a strength but also a weakness. Unlike Apple that controls their hardware; Android developers must assure their application works on a variety of devices. However, installing software and testing on hardware is much simpler with Android than with iOS.

GAME ENGINES

A short list of rendering/game engines was examined to provide a small survey of capabilities across iOS and Android. They were selected based on how close their asset pipelines match the current TC3sim engine and their capability on iOS, Android, or both.

Irrlicht

Irrlicht is an open source 3D graphics engine written in C++ that has been ported to Android. It offers direct import of 3D Studio Max files. 3D Studio Max is the application in which all TC3sim assets are created. There were also a number of demos and tutorials for building a 3D application in Irrlicht to run on Android (Renzhi, 2011).

It proved not too difficult to build a demonstration capability into Irrlicht that allowed one to view and navigate within a 3D scene. Bringing TC3sim assets into the Irrlicht engine was also not too difficult.

The greatest difficulty with Irrlicht on Android was implementing the advanced control logic and game logic. In Android, user activity is handled in the Java layer and communicated to the C/C++ code through something called the Java Native Interface (JNI). This affected the performance of the application. Programming and debugging for Android (using Irrlicht) was proving quite difficult.

Also, during our exploration there was no native iOS port. The Proton SDK (Robinson, 2010) has since come to light but is not addressed in this paper.

Ogre3D

The current generation of TC3sim will use Ogre3D and our internal art department has a lot of experience with Ogre3D. Ogre3D, like Irrlicht, is an open source 3D engine written in C/C++ (although it has many alternate language bindings including .NET). It is considered the more mature open source offering with a large community, a significant amount of documentation, a number of features and special effects, and cross-platform capability (Benei, 2011). It has also been used as the basis for some commercial game titles.

It has been ported to iOS and it was not too difficult to get the Ogre3D demonstrations working on an iPad. Ogre3D and the iOS port were both built using Xcode. An application was then written and compiled as an iOS application. Using iTunes the application could be installed on a registered development device (in this case an iPad.)

Art assets were exported from 3D Studio Max using OgreMax. OgreMax is a plugin to 3D Studio Max that exports models, textures, materials, and animations into file formats native to Ogre3D.

TC3sim materials had to be slightly altered to account for the lack of multi-pass shader capability in OpenGL ES. However, for the most part, the art pipeline was the same as that for desktop TC3sim. These material changes could also be made after delivery of the asset by the art department using a simple script.

TC3sim assets in Ogre3D ported to iOS looked fantastic. The special effects and graphics capabilities required by TC3sim worked in iOS. However, installation of the program on the iPad with Apple's process is tedious at best. Ogre3D also proved to require a large memory footprint. Also, while there is some work in porting Ogre3D to Android (Jacmoe, 2010) it does not look like it is well supported or recently updated.

Unity3D

Unity3D has grown into a very capable game engine and authoring environment. You utilize a graphical user interface to import content and to write your game. All behaviors, triggers, game flow, etc. is scripted within the Unity3D authoring environment. Once you are done creating your game, you select your deployment target (Windows, Web Player, Mac OS X, Android, iOS, etc.) and it produces an executable and packages the assets for your game. Mac OS X and iOS targets require that you build on Apple hardware.

One of Unity3D's most popular and publicized features is its Web Player (Fenandez, 2009). This is a plugin that allows a Unity3D game to be played in a web browser. This operates similarly to the ubiquitous Flash plug-in. It has similar performance to a native application but downloads and stores its content within the browser's "sandbox" cache.

The base version is free to use for independent game developers. Otherwise, there is a reasonable per-seat license with additional licensing for mobile build targets for iOS and Android. It is well-supported by the company and it comes with some very good samples and tutorials. Many games for both Android and iOS have been created using Unity3D.

It natively imports 3D Studio Max files which fits the TC3sim asset pipeline. Many of the shaders and materials originally created for use in Ogre3D imported very easily into Unity3D making it a nearly painless process. One can also specify post processing rules for assets to change things like texture resolutions for different platforms.

There is a graphical user interface (GUI) toolset that allows 2D buttons and menus to be easily specified and tied to actions within the system. Multi-touch is also supported within Unity3D for platforms that facilitate it. The iPhone standard package includes code to set up a touch-screen, multi-touch joystick.

Android phones support multi-touch differently depending on the hardware. Some phones do not support it at all, while some support only two touches (i.e. pinching). However, the iPhone tutorials available from Unity3D can be (for the most part) used on Android phones and many of the handheld APIs in Unity3D have been unified with iOS for cross-platform projects. You may need to set up conditionals though for compiling different code for different platforms when necessary.

Buy vs. Make

Irrlicht and Ogre3D were clearly in the "make" category of the engines evaluated. Both are open source and free, but both are rendering engines and not game engines. The game logic of TC3sim would need to be recreated for these platforms. Both also seemed to be well established on only one platform: Irrlicht for Android and Ogre3D for iOS. Unity3D became the clear winner in the evaluation due to its maturity and low cost.

Unity3D's cost would pay for itself in its ability to do both iOS and Android development, in its easy ability to import existing art assets, and in the ease with which you can write behaviors and game logic within its IDE. The other engines would require more engineering effort than the cost of a number of Unity3D seat licenses. While there would still be a lot of "make" to be done in Unity3D, buying it has a significant return on investment in terms of developer efficiency.

INTERFACE CHANGES

TC3sim is a serious game for Windows PC. It is controlled by a combination of keyboard and mouse. It also employs a contextual "pie" menu. To run on touch screen devices like the iPhone and Android mobile phones the interface had to be redesigned to accommodate a smaller screen and lack of mouse and keyboard.

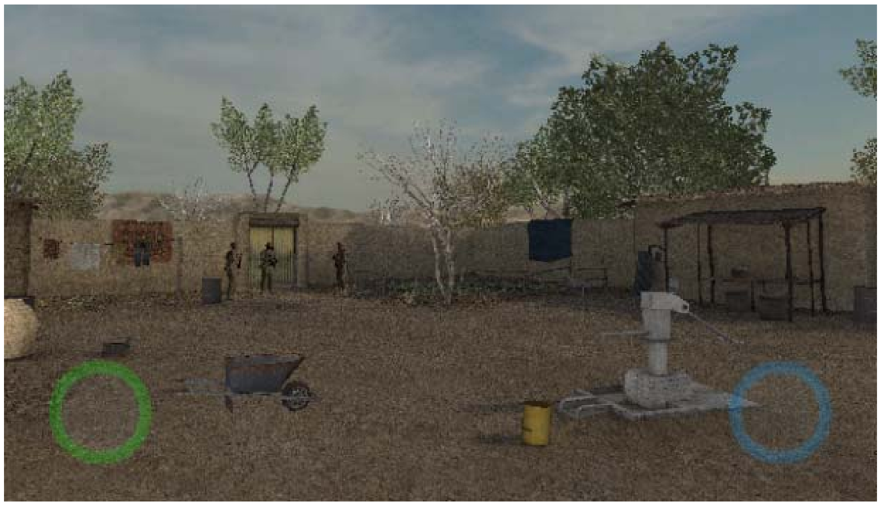
Figure 3: *Unity3D with multi-touch navigation circles*

A common approach to 3D first person games on touch screen devices is to include controller pads at the lower left and right corners of the screen in landscape mode. This emulates common game controllers in ergonomics and operation whereby a joystick is operated by the right thumb and a multi-button rocker is operated by the left.

Unity3D includes an example of how to create this kind of controller for touch screen devices and this was leveraged to create the control scheme for the TC3sim port (see Figure 3.) The circle on the right controls the viewpoint of the player the same as the mouse in the desktop version of TC3sim. The circle on the left controls forward, backward, left, and right movement as the 'W', 'S', 'A', 'D' keys do in the desktop version of TC3sim.

Casualty interaction required some creativity. To interact with the casualty the user would be required to move into close proximity to the casualty and then tap on the body part with which they were to interact. This would bring up an array of nine buttons around the center of the screen (similar to the pie menu from the desktop version). The ninth button at the bottom is a close button. The other eight are actions to be performed on the casualty relevant to the particular body part.

The small screen size makes precise selection difficult. To ameliorate this problem, the interface for interacting with the synthetic casualty registers the finger lift event rather then the finger press event. When the user places their finger on a body part, the name of the body part appears at the top of the screen (see Figure 4.) This provides feedback to the user to assure that their selection and their intended selection is correct. If not they can drag their finger over to another body part or to a "blank" space without activating the menu. The menu will activate only when the user lifts their finger. The menu buttons work in the same way.

This approach to the user interface provides for a similar feel as the desktop version of TC3sim while taking advantage of the affordances of a touch screen interface. At the same time, it attenuates the drawbacks of a small screen. This works well on iPad as well although it has a large screen. This proves a good model for touch screen interfaces and is expected to work on larger Android tablets as well.

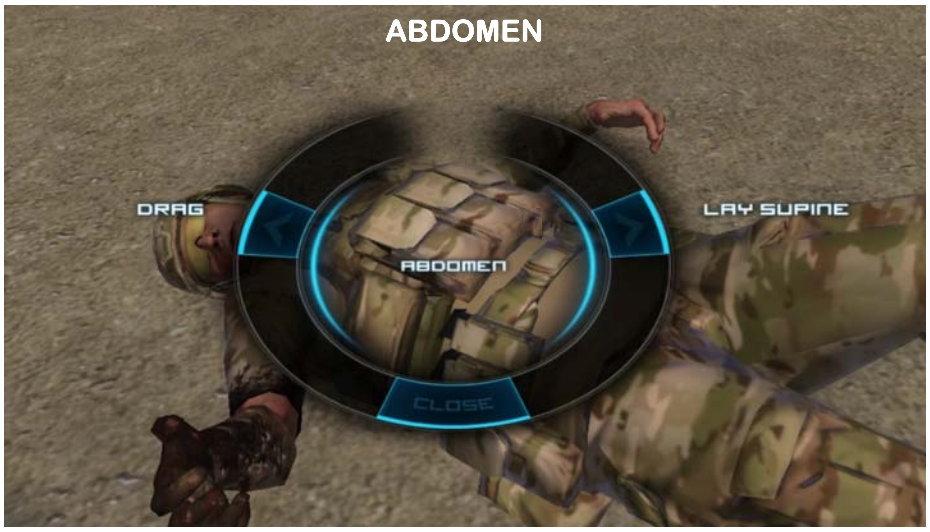

Figure 4: *"Pie" Menu implemented for touch screens*

ART PIPELINE CHANGES

While Unity3D makes it very easy to bring in models from 3D Studio Max and use them for development in iOS and Android platforms. The biggest issues had to do with shaders and memory.

Shaders had to be optimized for the particular device and the particular version of OpenGL ES being implemented. However, the test devices supported OpenGL ES 2.x which allows for simple shaders. Unity3D provides a selection of shaders to get up and running and they work well, but custom pixel shaders can be written and incorporated for better looking scenes and characters. However, because of the performance restrictions of the hardware, shaders must be fast and efficient, thus not too complex. Lightmaps using the integrated Beast system provided for some of the best visual appeal.

Because of the complexity of the port of TC3sim, memory was an issue due to the size of some of the environments and the memory footprint of the synthetic casualties. Typically the casualty models contain a significant amount of extra hidden geometry to allow for the representation of a number of casualty types. However, for use on mobile platforms the synthetic casualties had to be pre-optimized to allow them to run in memory. The injury had to be selected ahead of time and then exported without all the extraneous data. Flexibility was traded for performance. Finally, we kept the synthetic casualties within a “courtyard” environment that limited the amount of scene data being rendered in the environment.

APP STORE DEPLOYMENT

Apple

Apple requires a great deal of control over the development process and the distribution of applications for iOS devices. You must register as a developer on their development portal and develop on Apple hardware. You must create, download, and install a special profile in Xcode to allow it to compile for iOS devices. You must also register all devices you intend to use for testing and development via their portal and creating “provisioning profiles” for the application and the hardware to which it is to be installed. This can be a very tedious process simply to develop the software.

Submitting an application to the App Store requires that you sign up for iTunes Connect. This is the portal for uploading your applications and managing contracts, taxing, banking, and sales information. In addition to submitting the binary you must include a set of metadata about the application that includes ratings, keywords, and screenshots. All applications are reviewed by Apple and they require you to comply with Apple’s content guidelines and that the application has been thoroughly tested on iOS devices. This review process can take hours to a couple of weeks. Your application will “go live” without notice so be prepared.

Google

Publishing to the Android Market requires you to register using a Google account and agree to their terms of service. After that you can upload and update your application as much as you like, as long as certain requirements are met:

1. signed with a cryptographic private key,
2. Android version defined in a manifest, and
3. an icon and label must be identified.

The process is very straightforward and there is no approval process, although there is an after the fact reporting process for potentially offensive material.

Other Publishing Options

Android applications can be installed easily outside of the Android Market. They can be installed via a website or the phone’s Secure Digital (SD) card. One must simply activate the capability within an Android phone’s Settings menu.

Installing applications on iOS devices outside of the App Store requires an Enterprise Developer License from Apple. This license allows for the distribution of in-house applications within an organization of over five hundred employees. Or, if your distribution is to less than one hundred devices you have the option to distribute your applications *ad hoc* under your developer license.

Another option is to “jailbreak” your iOS device. This involves removing Apple imposed limitations on the operating system by gaining administrator rights through some clever hacks. This, however, voids the warranty and (in rare cases) can make the hardware unusable (“bricking it” in the parlance.)

CONCLUSION

Porting TC3sim to the mobile platform turned out to be easier than expected for a number of reasons.
Mobile hardware capabilities are increasing every year. The prevalence of OpenGL ES 2.x on many recent platforms such as iOS devices and Android 2.2 devices brings graphics capabilities closer to those on the desktop. It also made the asset pipeline not too far from the desktop version in terms of shader selection.

The selection of Unity3D as the development platform proved to be a good one. Its development environment and various integrated tools made porting game logic, physiology simulations, and interactivity very easy. The per-seat cost is quickly made up in increased developer effectiveness.

Interface design was key to success and it helped that the original game did not rely on a significant number of key presses. The contextual menu is intuitive, provides for easy discovery of interactivity, and was easily translated to a touch screen device. It is a good exemplar of interface design for mobile devices.

FUTURE WORK

The port of TC3sim to iOS and Android could be further enhanced by providing multi-player, networking capability. Unity3D has networking support and it would be good to explore these capabilities with regard to mobile platforms.

The ability to deliver more expansive environments is an area to explore as well. Are there ways of constructing the environments in such a way to account for the performance limitations of mobile devices?

The visual quality of the system could also benefit from shader and lighting enhancements. This work was able to provide some good visual quality; but there are, most likely, techniques yet to be developed that could create great visual depth without sacrificing performance. There are still “tricks” in the system to be discovered!

REFERENCES

Benei, V. (2011). Reviews for OGRE (O-O Graphics Rendering Engine). Retrieved from *http://sourceforge.net/projects/ogre/reviews/*

Chludzinski, J. and Mall, H. (2010). Challenges to Putting the Real-time Web on Mobile Platforms. *Proceedings of the International Interservice Training Simulation and Education Conference. 2010.*

Fernandez, J. (2009). Software Informer Review of Unity Web Player. Retrieved from *http://unity-web-player.software.informer.com/*.

Jacmoe and Spacegaier (2010). Ogre Android. Retrieved from *http://www.ogre3d.org/tikiwiki/Ogre+Android&structure=Development*.

Renzhi (2011). Programming 3D games on Android with Irrlicht and Bullet (Part 1). Retrieved from *http://renzhi.ca/2011/05/19/programming-3d-games-on-android-with-irrlicht-and-bullet-part-1/*.

Robinson, Seth A. (2010). Proton SDK - component based C++ framework for iOS, Android. Retrieved from *http://irrlicht.sourceforge.net/forum/viewtopic.php?t=40371&sid=f54c5d8bed3b309c3922db62b016d7b9*.

Sotomayor, T. (2010). Teaching tactical combat casualty care using the TC3 sim game-based simulation: a study to measure training effectiveness. *Studies in Health Information and Bioinformatics 2010*;154:176-9.