A Survey of Technologies for Building Collaborative Virtual Environments

Timothy E. Wright and Greg Madey

Department of Computer Science & Engineering, University of Notre Dame, United States

***Abstract*—What viable technologies exist to enable the development of so-called desktop virtual reality (desktop-VR) applications? Specifically, which of these are active and capable of helping us to engineer a collaborative, virtual environment (CVE)? A review of the literature and numerous project websites indicates an array of both overlapping and disparate approaches to this problem. In this paper, we review and perform a risk assessment of 16 prominent desktop-VR technologies (some building-blocks, some entire platforms) in an effort to determine the most efficacious tool or tools for constructing a CVE.**

***Index Terms*—Collaborative Virtual Environment, Desktop Virtual Reality, VRML, X3D.**

I. INTRODUCTION

The benefit of virtual reality (VR) has been understood for some time. An array of training uses for VR are cited by Macredie, Taylor, et al. [1]; industrial and Internet visualization techniques are respectively demonstrated by Luckas and Dörner [2], and Benford, Taylor, et al. [3]; Franco, da Cruz, et al. [4] discuss the effective use of VR and other interactive media in teaching and learning; Yan [5] employs VR for speech therapy; and Cobb and Sharkey [6] review a decade of VR technologies used to help those with disabilities. This chorus is enhanced by further examples of VR's utility in artistic, cultural heritage, and scientific visualization deployments [7, 8], as well as the ongoing possibilities for experimentation and research offered by virtual worlds [9]. Despite its many successes and potential, VR remains a technology that has not seamlessly integrated with the average user's computing interface. For example, regarding the compatibility of different VR systems with one and other, Pape, Anstey, et al. [8] point out that in terms of VR applications we are at the Tower of Babel stage. Differences in tracking systems, interface devices, operating systems, VR libraries, graphics libraries, VR authoring systems, and even versions of the same software mean that, unless you have built your VR system with specific applications in mind, not many (or no) existing VR applications will run on it.

This issue is compounded by the fact that different levels of immersion (a participant's sense of residing within a virtual environment) often require different computing resources. Whereas desktop virtual reality (desktop-VR) typically uses nothing more than a keyboard, mouse, and monitor, a Cave Automated Virtual Environment (CAVE) might include several display walls, video projectors, a haptic input device (e.g., a "wand" to provide touch capabilities), and multidimensional sound. The computing platforms to drive these systems also differ: desktop-VR requires a workstation-class computer, mainstream OS, and VR libraries, while a CAVE often runs on a multi-node cluster of servers with specialized VR libraries and drivers. At first, this may seem reasonable: different levels of immersion require different hardware and software. However, the same problems are being solved by both the desktop-VR and CAVE systems, with specific issues including the management and display of a three dimensional environment and user interactions within that environment. Even when we remove the level of immersion as a factor (i.e., ignore the differences between desktop-VR and CAVE interfaces), the system designer still faces an array of VR software options—some overlapping, some different—all geared to accomplish similar tasks.

Regardless of immersion hardware requirements, the incompatibilities and duplications of effort among VR software solutions may hinder the acceptance of VR and its integration as a common computing interface. In an effort to grapple with this problem, we narrow our focus just to the area of collaborative, virtual environments (CVEs) accessible through desktop-VR, and explore 16 software technologies relevant to building CVEs. For our purposes, then, a CVE is taken to be a multiuser, 3D graphical environment accessible from, and potentially hosted on, workstation equipment. Although highly immersive environments, such as CAVEs and head-mounted display (HMD) systems, offer a more realistic user experience [10], we restrict our consideration to desktop-VR for the following reasons:

- As desktops have become less expensive and more powerful (especially with regard to graphics adapters), viewing and interacting with high quality VR environments is more easily managed than in the past [11]
- The use of complex VR worlds for entertainment, commerce, and educational purposes is becoming more popular and effective for desktop users [12, 13]. Typical, modern workstations are able to execute both client and server software, though scalability issues may apply

Manuscript received on January 3, 2008.
E-Mail: twright @nd.edu, gmadey@nd.edu

- Giving attention to the area of desktop-VR may help speed up the overall integration of VR into the average user's digital media interface

In this survey, we will work toward identifying the strengths and limitations of the desktop-VR technologies under review, thereby highlighting the best option or options for building CVEs. To help assess each technology, we propose that, for any CVE to function, basic capabilities must be present in the areas of graphics rendering and networking. Secondarily, audio characteristics are desirable. Graphics rendering should include the display of 2D and 3D images, ranging from simple geometries to complex, texture mapped surfaces. Networking should at least provide the ability to remotely access VR content (e.g., images, sounds, scripts); preferably, it should also include features for sharing a virtual environment among multiple, simultaneous users. Last, audio should permit the playing of spatial sound; that is, sound with location, direction, and intensity. It should be observed that not all of the technologies in our survey include capabilities we consider to be primary (i.e., graphics and networking), or primary and secondary (i.e., graphics, networking, and audio). Less equipped software is also considered if it is commonly used as a CVE building-block.[1]

Finally, some VR technologies include out-of-the-box features that are convenient for building a CVE (e.g., avatar support, multi-user capabilities, virtual collaborative tools [such as whiteboards], asset management,[2] an art path,[3] and rigid-body/particle physics). Having these extra features could mean significantly less work for the CVE developer. Therefore, two additional characteristics that we believe to be significant, avatar support and multi-user capabilities are taken into account later during a risk assessment of the 16 technologies under review.

The remainder of this paper is structured as follows: the next section summarizes related works; Section III outlines the nomenclature and approach we use in our review of VR software technologies; sections IV, V, and VI include assessments of the technologies (divided into three categories), Section VII provides a qualitative risk assessment of the VR technologies, and Section VIII ends the survey with concluding remarks.

II. RELATED WORK

Various desktop-VR technology surveys exist within the literature, a few of which compare tools that can be used to construct a CVE. One such effort, made by Holmberg, Wünsche, et al. [14], outlines a framework for comparing five Web-based visualization technologies, including Scaled Vector Graphics (SVG), Dynamic Hypertext Markup Language (DHTML), Virtual Reality Modeling Language 2 (VRML 2), Extensible 3D (X3D), and Java3D. This framework consists of four high-level categories (technical capabilities, interactivity, support, and application specific) used to measure the strengths and weaknesses of the technologies in question. Although somewhat subjective in places, the framework's metrics do a reasonable job of capturing the utility and functionality of each technology. Ultimately, the authors select X3D as the best option for displaying 3D visualizations in an experimental project they carry out.

In a survey covering similar territory, Lau, Li, et al. [11] offer descriptions of what is termed emerging web graphics standards and technology. Consideration is given to several major areas, including 2D and 3D graphics standards (e.g., SVG and X3D, respectively), approaches to client synchronization and coherency in CVEs, virtual humans represented on the Web (e.g., as highly-defined "talking heads" or fully rendered human bodies through the Humanoid Animation standard [H-Anim]), and alternative approaches to traditional, projective-geometry-based web graphics. The authors do not offer any significant conclusions in their discussion, except to comment on areas of future research important to the technologies under review. Other work in the same vein, though geared toward highly immersive technologies such as CAVEs, includes an outline by van Dam, Forsberg, et al. [10] of the potential and difficulties entailed by using VR for scientific visualization, and an entry level review of VR terminology and technology by Pape, Anstey, et al. [8].

In another survey, Bouras, Panagopoulos, et al. [15] compare and contrast different free and commercial implementations of the X3D standard. Specific technologies include the X3D browsers Flux Player, FreeWRL, OpenWorlds Horizon, and Xj3D, and commercial CVEs from Blaxxun, Bitmanagement Software, and Octaga. The authors focus their discussion on how suitable these technologies are for constructing an X3D-compliant networked virtual environment (NVE).[4] Only the contending CVEs are substantively compared with one and other according to ten simple criteria: X3D support, type of architecture, H-Anim support, streaming video support, voice-over-IP support, extensibility, scripting support, presence of X3D authoring tools, supported operating systems (OSs), and cost. The authors point out the risks of developing with commercial software (i.e., cost and lack of options should the vendor decide to change their product), and offer these as reasons for leveraging open source technology instead. Noting that the Xj3D Java libraries offer a "flexible and cross-platform Java architecture," they use Java and Xj3D in the construction of an X3D browser and the retrofit of an existing, but limited, research CVE.

X3D is the focus of work by Anslow, Marshall, et al. [16], too, wherein the standard's structure and composition are

[1] It is noteworthy that all of the software we survey, from the least to most equipped, is graphics oriented. Text-based multi-user environments (e.g., Multi-user Dungeon [MUD] and MUD Object Oriented [MOO] systems) notwithstanding, our definition of desktop-VR requires a 3D graphical display. Thus, one can build a desktop-VR environment without networking or audio—although, absent networking, there would be no collaborative ability; one cannot build a desktop-VR environment without graphics.

[2] In the context of our survey, this term means tracking the inventory and disposition of objects comprising a virtual world.

[3] This refers to the tools and compatibility available for integrating images, textures, meshes, sounds, etc. into a virtual world.

[4] The term *networked virtual environment* should be viewed as synonymous with *collaborative, virtual environment*.

reviewed. Specifically, the authors are interested in applying desktop-VR to the endeavor of software visualization; that is, using a VR context to represent the relationships, statistics, and source code text of software systems. They conclude that the X3D standard offers the means to support software visualization, although they find text rendering (for displays of source code) to be an area of weakness.

Detailed consideration is also given to X3D's capabilities by Segura, Arizkuren, et al. [17], where the authors discuss the development of an application to display user-selectable configurations of a given product in high-quality 3D. Several X3D browsers are compared by the authors who note that the lack of reflection mapping and shadowing in the X3D standard may be overcome through appropriate extensions to the standard.

A broader net is cast by Grimstead, Walker, et al. [18] who compare 42 collaborative visualization systems. While some of the reviewed technologies are CVEs, many comprise middle-ware and tangentially related applications. The authors divide the systems into five basic application areas and then attempt to compare them across five categories (number of users, access control, communication architecture, transmitted data, and user synchronization). The primary conclusion reached is that virtual environments (i.e., CVEs) and online games offer the most capability in terms of scaling.

Some desktop-VR technology surveys are more conceptually oriented, choosing to review relevant precepts and foundations as opposed to specific tools or standards. For example, with a stated objective to "provide an awareness of Virtual Reality with respect to simulation," Barnes [19] offers a concise overview of VR basics, including: a definition of VR; brief, background discussions about VR found in the media and as an effective means of simulation; physical mechanisms for interaction and control within virtual worlds; and a thorough discussion of VR in manufacturing (a combination referred to as "virtual manufacturing") along with several real-world examples. Similarly, Otto, Roberts, et al. [20] survey "various [CVE] technology factors and demonstrate their impact on closely-coupled collaboration." These factors include: task design; immersion, field of view, and navigation; and manipulation technique, workflow, social human communication, user interface, distribution, and task performance. Ultimately, the authors classify these into technology factors, human factors, and application factors. They further note that each grouping must be given attention since many of the contained elements have interrelationships.

While all of these works help to build a picture of desktop-VR technology, little guidance is offered about which technologies are most strategic. We contend, however, that with regard to the engineering of CVEs, there are specific, well suited technologies that developers should strongly consider. Moreover, standardizing on such tools may help deal with the incompatibilities and duplications of effort among some VR systems, thereby leading to better integration with the common digital media experience.

III. NOMENCLATURE AND APPROACH

Our stated goal is to assess 16 desktop-VR technologies applicable to the construction of CVEs. As a result of this survey, we intend to note the most efficacious tool or tools. The following three broad categories offer a convenient way to group and compare the technologies of interest. It is not the case that one category is necessarily better than another: advantages and disadvantages exist for technologies in all three categories.

- Application Programming Interface (API): one or more libraries with documented methods that abstract lower-level resources (e.g., a graphics or audio API)
- Framework: a collection of APIs that provides functionality across multiple domains (e.g., graphics, audio, and networking) and is extended by support tools
- Platform: [5] a specialized and complete environment within which a software system may be built, tested, and executed; the platform must be deployed anywhere software based upon it is installed

Within each of these categories, only active technologies (i.e., maintained roughly within the past year) that are free and open to developers are considered. Thus, we exclude proprietary options such as the Adobe Flash and Shockwave platforms, the Octaga suite of tools, the Bitmanagement Software SDK, the Torque Game Engine, etc. (Adobe claims that its Flash and Shockwave players have been installed on a majority of all Internet-enabled PCs [21]. The Flash player handles content created by Adobe's Flash CS3 Professional (2D animations, web advertisements, web interfaces), while the Shockwave player views content created by Adobe's Macromedia Director (multi-user games, 3D content and product simulations). There is a collection of open source alternatives to CS3 and Director [22], but the Flash and Shockwave specifications remain proprietary and under the control of Adobe.) We restrict ourselves to active, open source technologies as a way to reasonably limit scope and emphasize software in which the end-users are the primary stakeholders. Also, our intention is not to carry out an exhaustive review of open source VR software: such an undertaking is impractical at best. Instead, we select 16 noteworthy, relevant technologies, any of which can play a significant role in the development of a CVE, and provide simple, concise descriptions of each.

Each technology is examined to determine its capabilities, level of maturity, adherence to a formal standard, and miscellaneous facts of interest. All of this information is summarized in Table I. Here, capabilities denote whether or not the technology in question provides a graphics rendering engine (G), network communications (N), and audio support

[5] Below, note that we do not include Java3D under the "platform" category even though the Java virtual machine and runtime libraries must be installed anywhere Java3D software is executed. Although Java offers the necessary resources (e.g., networking and audio) to complement Java3D's graphics rendering and construct a CVE, by itself Java3D is designed to be general purpose in nature. Thus, we believe it does not align well with the platform designation used herein.

(A). Maturity indicates how long a given technology has been in existence—an inception year is provided. Adherence to an open and free standard connotes how easily a technology can interoperate with other tools and 3D models. Of specific interest are VRML 2 and X3D, which became ISO/IEC standards in 1997 and 2004, respectively [23]. Maintained by the Web3D Consortium, X3D is the latest iteration of VRML and is backwards compatible with VRML 2. Finally, important or interesting facts are provided in a comments entry for each of the 16 technologies.

Following our review, in Section VII we undertake a qualitative risk assessment to derive what we believe is the best option or set of options for building a CVE. In this assessment, significant factors include support that a given technology may have for graphics rendering and networking (primarily), as well as for audio (secondarily). In addition, software engineering characteristics such as utility (given by the presence of avatar and multi-user support), project documentation and help resources, support for relevant standards, and deployment challenges are all taken into account.

IV. APIS

As discussed in Section III, an *API* is considered to be one or more well-defined libraries that provide access to underlying hardware/software resources. In other words, the API works to abstract lower-level resources by presenting the software developer with a documented set of methods that operate as a wrapper. Although they do not offer as much convenience as frameworks or platforms, APIs do enable a "best of breed" approach toward software engineering (i.e., a developer can draw from multiple, well-concieved technologies). However, compatibility issues could result if implementations of the same CVE rely upon different, underlying APIs.

In the following subsections we examine APIs for FreeVR, Java3D, OGRE, OpenGL, OpenSceneGraph, OpenSG, OpenVRML, the X3D Tool Kit, and Xj3D.

4.1 FreeVR

Written in C with support for multi-processing, FreeVR is aimed at providing a uniform, cross-platform means of accessing and integrating with a variety of VR systems and configurations [24]. At present, the FreeVR library compiles on various UNIX platforms, OS X, and Cygwin for Microsoft Windows. Although suitable for use in highly immersive environments such as CAVEs, FreeVR can also be deployed in desktop-VR settings. Graphics rendering is handled by the OpenGL library; the OpenSceneGraph and OpenSG scene graph libraries may also be used.

Some of FreeVR's notable strengths address interfacing with VR equipment and the administration of FreeVR programs. The FreeVR API makes it possible to map physical mechanisms, such as those used for input, to logical devices. This enables easier programatic access to the mechanisms while abstracting and simplifying their use. Also working toward ease of use, FreeVR employs runtime configuration (RC) files to specify details about system resources, processes, and program input/output [24]. By editing an RC file, an administrator can change an executing FreeVR program for purposes of debugging, fine tuning, or interfacing with the user differently. Remote, administrative access to a running FreeVR program is permitted through TELNET or a Tcl/Tk GUI, and includes the ability to view and manipulate information and settings. Although this feature does not also allow the editing of RC files, it does enable debugging and the display of program statistics (e.g., frames per second).

Despite its strengths, FreeVR does face some challenges. First, it is missing support for the network and audio components needed to implement a CVE. A variety of libraries external to FreeVR can be drawn upon to fill these gaps (e.g., the FreeVR source code documentation recommends Virtual Sound Server [VSS] or Bergen Sound Server for audio), but the lack of a uniform, homogeneous approach may result in incompatible FreeVR deployments. Next, FreeVR does not have the option of using a VR standard unless OpenSceneGraph or OpenSG, both of which can support VRML 2, is chosen as the graphics renderer. Unfortunately, the updated and more robust X3D standard is not currently an option with either scene graph API. Finally, the knowledge base for FreeVR is still early in its evolution. Materials such as basic documentation and tutorials appear somewhat rough and sparse.

4.2 Java3D

Java3D offers the ability to manage and render 3D graphics by way of a *scene graph* data structure [25]. A scene graph is a directed acyclic graph that permits efficient storage of, and access/updates to graphics data. More specifically, scene graphs enable "a hierarchical approach to describing objects and their relationship to each other [25]," such that each level of the hierarchy denotes some type of grouping, while the leaves represent the targets to be rendered (e.g., graphics or sound). Because scene graphs enable critical functions like culling (not displaying images that fall outside of the active viewpoint) and sorting, they are used in other desktop-VR technologies, as well (e.g., OpenVRML, the X3D Tool Kit, and Xj3D). One method to implement a scene graph is given by the Composite design pattern, where hierarchical tree structures are built from objects that may be uniformly managed as compositions or individual entities [26]. The Java3D API can use OpenGL or Microsoft's DirectX to render images managed by its scene graph.

In addition to the benefits of a scene graph, Java3D offers the platform independence of the Java Virtual Machine and can execute under Apple OS X, Linux, Sun Solaris, and Microsoft Windows as an application or a web browser applet. Through the use of an extension library, J3D-VRML97 [27], the Java3D API can utilize VRML 2 models; this is also how the API gains basic networking capabilities (i.e., to load remote VR content).

Because Java's extensive library of packages can be drawn upon to fill in many functionality gaps, Java3D is a viable CVE building block despite its limitations. Its weak areas include a lack of support for audio and X3D, as well as size issues. Though its use of the VRML 2 standard is a benefit, at the same time none of the enhancements and improvements of X3D are

available to Java3D (the Xj3D API addresses this—see further below). Also, it may be that a given target host does not possess the Java Runtime Environment (JRE) and/or Java3D. This could make the installation of Java3D-based CVE software more complicated, since the JRE and Java3D would need to be bundled in the install process.

4.3 OGRE

The Object-oriented Graphics Rendering Engine (OGRE) is a C++ API that focuses on modularity and functionality. Available for Linux, Apple OS X, and Windows, OGRE's dedicated purpose is to "make it easier and more intuitive for developers to produce applications utilising hardware-accelerated 3D graphics [28]." To this end, the API is specifically architected to easily interface with projects that might need 3D graphics abilities. In addition, OGRE itself is designed to be extended through plugins: a number of such add-on mechanisms exists within the OGRE community and provides enhancements ranging from indoor and outdoor scene rendering to bindings for physics engines [29]. Finally, it is notable that OGRE's documentation and community support are both highly evolved compared to many other open source projects.

Similar to OpenGL below, OGRE is strictly a graphics engine. As such, it would need to be supplemented with network and audio components if used to build a CVE. Also, it has no direct support for VR modeling standards such as VRML or X3D. Nevertheless, because OGRE has been specifically designed as an encapsulated and extensible API these limitations may really be inconveniences more than drawbacks.

4.4 OpenGL

Our inclusion of OpenGL in this survey is somewhat contrary to one of our goals: to review only open source technologies. Although freely available to developers, OpenGL is not open source. However, its ubiquity and pervasive use in most other graphics-related software make it the elephant in the room that we must address.

A creation of Silicon Graphics, Inc., since 1992 the OpenGL API specification has become a de-facto means of handling 2D and 3D graphics on desktop computers. Nearly all significant OSs (Apple OS X, Linux, FreeBSD, Sun Solaris, IBM AIX, HP-UX, Microsoft Windows, etc.) come bundled with, or can easily compile the OpenGL libraries [30]. Underlining this point, each of the technologies reviewed in this survey either uses OpenGL directly or has the option of doing so. Hence, regardless of the tool or tools a developer might pick from our survey, the OpenGL API is almost certainly involved. In addition to its widespread use, the OpenGL libraries are callable from several languages, including C, C++, Python, Perl, Java, and others [31]. Some additional strengths include the ability to handle computationally intense graphics rendering (e.g., programable shading, lighting, texture mapping, non-uniform rational B-spline [NURBS] curves and surfaces), the flexibility to handle graphical elements such as reflections, support for extensions that take advantage of features in specific graphics cards, and a developed knowledge base [32, 33].

Although its ubiquity, capabilities, and maturity make it an attractive technology for constructing CVEs, OpenGL does have some limitations. First, it is an API that only handles graphics: it offers no networking or audio components, and includes no functionality dedicated to the semantic of a multi-user, VR world. As intimated earlier, however, other VR-motivated technologies may be able to work around this issue by incorporating OpenGL into a more complete solution. Finally, the OpenGL specification is not free to all parties [34] (i.e., hardware platform vendors). Hence, it is possible that software written with OpenGL may not execute on graphics cards from vendors who do not pay to license the API, although this may be a low-risk problem due to OpenGL's widespread nature.

4.5 OpenSceneGraph

The OpenSceneGraph (OSG) project implements a scene graph data structure on top of OpenGL to manage 2D and 3D graphics. Beyond its utility as a scene graph implementation, other strengths of the OSG API reside in its degree of portability, language support, and extensibility. Written in C++ with support for multi-threading and processing, OSG is compatible with the same platforms as OpenGL (see the previous subsection) through multiple languages including Java, Python, Tcl, .Net, and others. OSG can also be expanded through what are termed "node kits:" separate libraries, added at compile- or run-time, that enhance a project with additional graphics features and special effects. Finally, OSG supports a wide range of graphics file formats and, through the OpenVRML API, the VRML 2 standard.

The issues faced by OSG are similar to those of OpenGL. It is an API that only deals with graphics, necessitating the use of other resources for networking and sound that may not be as portable as OSG itself. Also, though unrelated to OpenGL, the utility of supporting the VRML 2 standard is diminished by OSG's lack of support for the improved X3D standard.

4.6 OpenSG

OpenSG is a project with strong similarities to OSG. Both are OpenGL-based, scene graph APIs written in C++. Both are compatible with the same extensive list of platforms, although there appear to be more language bindings for OSG. Both support multi-threading and processing, but OpenSG is explicitly outfitted to handle graphics clusters [35]. Both offer high-performance scene graph management, although OSG offers a compile- and run-time extension feature (node kits). Finally, both support a wide range of 3D file formats, as well as VRML 2—if the OpenVRML API is used.

OpenSG's challenges are identical to those of OSG; see the previous subsection for details.

4.7 OpenVRML

The OpenVRML libraries provide an API for the VRML 2 and X3D standards. Written in C++, OpenVRML is a cross-platform API that uses OpenGL as a graphics renderer and can operate on those platforms where OpenGL is found

(see the subsection on OpenGL). Moreover, if the GIMP Tool Kit Plus (GTK+) user interface libraries are available to OpenVRML, a web browser plugin may also be deployed for Mozilla-based browsers. Built around the use of a scene graph, both VRML and X3D offer a script-based means of describing, rendering, and controlling VR scenes. Complete with support for audio, dynamically changing environments, and avatars, VRML and X3D environments are typically accessed through a web browser equipped with an appropriate plugin. Alternatively, a dedicated application, such as one built on top of OpenVRML, may be used.

Although inherently suited to much of the CVE building endeavor, OpenVRML does have a few weak areas. First, it does not include a networking capability. It attempts to manage this by providing a C++ abstract class that the CVE developer can use as an interface to their own networking resources. This may be an acceptable approach in some circumstances, but it leaves open the possibility of underlying networking components that operate in one environment but not another (possibly leading to CVEs that are not fully cross-platform). Next, despite its implementation of X3D, there is currently no support for X3D's XML encoding. This is significant since XML is a structured, verbose language capable of being validated against a Document Type Definition (DTD) or XML Schema Reference; it "is the basis of nearly every data language used on the World Wide Web [23]." Finally, as with the FreeVR API, the OpenVRML project's documentation is somewhat thin and in need of expansion and tutorials.

4.8 X3D Tool Kit

The X3D Tool Kit is a once-abandoned project now reborn. From 2002 to 2004 the original author developed the API into an implementation of the X3D standard that offered significant support for static X3D scenes—where sensor and collision nodes were not used [36]. In 2006 the project was revived by a group of developers whose stated emphasis was "on proper X3D behavior as specified by the standards committee [37]." Written in C++, the X3D Tool Kit uses built-in scene graph mechanisms to manage and process 3D graphics information, but relies upon OpenGL for image rendering. Among the X3D Tool Kit's strengths are that its design enables the extension of the API, and its use of C++ and OpenGL permit it to be compiled on the platforms where OpenGL can operate (see the subsection on OpenGL).

The challenges facing this latest incarnation of the X3D Tool Kit are manifold. First and foremost, support for more of the X3D standard, especially collision and sensor capabilities, is needed. Next, the X3D Tool Kit is lacking a network component. As with each of the APIs reviewed to this point, the absence of basic networking capabilities can lead to incompatible deployments of CVE software, since different developers may elect to use different networking mechanisms. Finally, the documentation available for the API exists primarily on the original X3D Tool Kit website, which was apparently frozen in 2004. Thus, any changes or updates to the API are not yet being discussed by the new developers.

4.9 Xj3D

The Xj3D API is maintained by the Web3D Consortium, the same organization responsible for the VRML 2 and X3D standards. Written in Java, Xj3D can operate anywhere the Java Virtual Machine can, provided there is also support for the graphics rendering engine—the developer may elect to use Java3D or OpenGL (through the Java OpenGL [JOGL] libraries) [38]. Xj3D is a mature implementation of the X3D standard, offering not only a graphics rendering mechanism, but audio and basic network components as well. As with Java3D, because the Xj3D API libraries are simply an addition to Java, the use of Xj3D makes the standard Java environment available to the developer. Other notable strengths of Xj3D include the use of a scene graph to manage VR information, support of XML to encode VR scenes, support of the H-Anim standard to facilitate avatars, compatibility with VRML 2 (increasing the base of 3D models available to X3D systems), and the ability to operate as a Java applet or stand-alone application.

The drawbacks of Xj3D are primarily logistical in nature. Similar to Java3D, the basic Java runtime environment (JRE) may or may not be bundled with a given OS. Hence, the install base for a CVE built with Xj3D may need to include the JRE, Xj3D libraries, Java3D and/or JOGL, and the CVE software itself. If a CVE client were deployed as an applet, the collective size of all Xj3D libraries (nearly 30 megabytes at the time of this writing) may be a significant obstacle. An alternative option could be to remotely deploy a client application through Java Web Start; the application could then be executed later without need of further downloads.

V. FRAMEWORKS

As noted earlier in Section III, we define a *framework* as a group of APIs that abstracts multiple domains of software/hardware resources and is extended by various support tools (e.g., content editors, administration utilities). A robust framework for CVEs might cover several areas within graphics, client-server or peer-to-peer networking, multi-user management, and sound. This enables the software developer to work under the auspices of one API/tool collection, possibly reducing documentation and maintenance overhead. One drawback, however, is that the developer may end up committed to all of the framework authors' design decisions—those that are good and those that are poor.

In the following subsections we consider the Crystal Space, Delta3D, Quake III, VR Juggler, and Uni-Verse frameworks.

5.1 Crystal Space

Since its beginning in 1997, Crystal Space has evolved into a highly modular collection of C++ libraries offering CVE-related functionality in graphics (OpenGL-based or a built-in software renderer), networking, spatial sound, and other areas [39]. Compatible with Linux, Apple OS X, and Microsoft Windows, the Crystal Space framework is intended to operate as an application engine: through an extensive system of API plugins, a developer is able to include/exclude

those software components relevant to a project. In addition, through the optional high-level CEL (Crystal Entity Layer) interface, features important to game/multi-user environments (e.g., networking, game logic) are abstracted and made more accessible [40]. Another high-level interface called CELstart makes it possible to program Crystal Space projects entirely in Python or XML, and to bundle projects into standalone, executable packages [41]. Language bindings for Java and PERL are also available for Crystal Space. Finally, an extension for the Blender 3D modeling software allows content to be created for and directly exported to Crystal Space.

The primary area of weakness for Crystal Space is its lack of support for VR standards, making it difficult to share 3D models and environments with other, unrelated systems. Nevertheless, the integration of proprietary 3D model specifications and a convenient means of exporting content from Blender to Crystal Space may be sufficient for some CVE projects.

5.2 Delta3D

Created and maintained by the Modeling Virtual Environment and Simulation (MOVES) Institute at the Naval Postgraduate School in Monterey, California, Delta3D is aimed at VR applications in the realm of defense as well as modeling and simulation. Described on its project web site as "a fully-featured game engine appropriate for a wide variety of uses including training, education, visualization, and entertainment [42]," the Delta3D framework draws upon an array of APIs and support tools. Written in C++, the framework is compatible with Linux and Microsoft Windows with unofficial support for Apple OS X. Operating under the premise that low-level resources should not be obfuscated by their APIs, Delta3D offers a high-level interface across a range of territory while permitting access to low-level functionality if desired [42]. Utilizing a flat intra-framework organization, there are 14 APIs comprising this interface.

Highlights of the Delta3D framework may be found in the areas of graphics, networking, sound, and support utilities. The core API's graphics rendering capabilities are based on OpenGL and make use of OpenSceneGraph, thereby integrating the VRML 2 standard. Environmental effects (e.g., clouds, time of day), particle effects (e.g., smoke), and physics (e.g., gravity, collision detection) are also addressed in the core. Delta3D's networking and game APIs offer client-server or standalone client modes of operation. Sound, both 2D and spatial, is managed by an audio API. Examples of utilities to assist with creating and operating the Delta3D environment include a 3D map editor (the Simulation, Training, and Game Editor [STAGE]), a particle effects editor, and a 3D model viewer. Other strengths of Delta3D include APIs for avatar, weather, and terrain management, and language bindings for Python scripting.

Although Delta3D is a modular framework covering a wide area of functionality it does have limitations regarding VR standards. While it is useful that VRML 2 is supported through OpenSceneGraph, Delta3D is unable to take advantage of the newer X3D standard and accompanying improvements.

5.3 Quake III Engine

Initially powering id Software's Quake III Arena commercial game in 1999, the Quake III engine is also referred to as *id Tech 3* [43]. In 2005, id Software released the game engine as open source under the GNU GPL license, as it had done for the earlier Quake and Quake II engines. We focus on the Quake III Engine because it is the more feature-rich in the series, offering graphics rendering, spatial sound, and network capabilities that all attempt to improve upon the first two engines.

Written entirely in C and compatible with Apple OS X, Linux, and Microsoft Windows, the Quake III Engine comes ready with CVE resources, some of which are also optimized. As a game engine, the graphics rendering and networking functions are designed with performance as a goal: graphics are managed through OpenGL and require 3D hardware acceleration, while networking includes a built-in compression capability to more efficiently transmit data between client and server [44]. Also included in the engine's bundle is the Q3Radiant map compiler and editor, though a newer version, called GtkRadiant, is freely available, too [45]. Another feature of the Quake III Engine is its use of a virtual machine (referred to as the Quake Virtual Machine, or "qvm") to readily and securely enable game modifications by end-users. Virtual machine files are written in ANSI C, compiled into assembly code using the freeware LCC compiler, and then converted into qvm byte code via the *q3asm* assembler—both LCC and q3asm are included with the Quake III Engine bundle [46]. Finally, as a multi-player VR game engine, the Quake III Engine offers full support for avatars and text chat among participants.

Challenges of using the Quake III Engine are relegated to a lack of support for VR standards and somewhat dispersed documentation. Regarding standards, proprietary 3D model formats such as MD3 and BSP are used for avatars and maps, respectively. Such models may be difficult to create or manipulate, and are less interchangeable with other VR systems. Also, the Quake III Engine's documentation is abundant, but scattered around many locations on the Internet. When id Software released the Quake III Engine as open source, they included precious little documentation. Presently, most of the information that exists has been voluntarily produced, and there is no authoritative source. Nevertheless, many helpful documents, tutorials, and discussion forums are available, although one must search carefully to find useful resources.

5.4 VR Juggler

VR Juggler is a C++ framework that uses APIs in several areas to provide desktop-VR and other levels of VR immersion. There are two basic framework layers: the Juggler Portable Runtime (JPR) API, and the VR Juggler API and micro-kernel. The JPR resides on top of the OS and works to abstract critical system resources such as threads, I/O, and sockets [47]. Resting on the JPR, the Juggler API/micro-kernel layer "acts as `glue' between all the other Juggler components [48]." This two-layer foundation allows a developer to build application objects in C++ that are executed by the Juggler micro-kernel. One of the paramount goals of this framework is to operate as a middle-ware between applications and underlying VR systems.

In so doing, software can be built against the VR Juggler APIs with little regard for the equipment underneath; such programs become inherently portable among different VR Juggler environments.

Rounding out the framework, several more APIs and tools exist within the Juggler API/mico-kernel layer, including:

- Gadgeteer -- a tool that facilitates VR device management
- The Juggler Configuration and Control Library (JCCL) -- configuration and performance monitoring tools for the Juggler system
- Tweek -- APIs to build extensible, Java-based GUIs that communicate with underlying Juggler applications
- PyJuggler -- language bindings that permit the use of Python Juggler application objects
- VRJ.NET -- language bindings that permit the use of C\# and VB.NET Juggler application objects
- Sonix -- an API for audio hardware and other audio APIs

VR Juggler relies on the use of OpenGL, OpenSceneGraph, or OpenSG to render images. If either OpenSceneGraph or OpenSG are utilized, then support of the VRML 2 standard is brought along. Regarding audio rendering, although the Sonix API is capable of playing sound on its own, other audio APIs, such as OpenAL, may be used for more full-featured sound options. Finally, a tool called Maestro [49] is employed to control the execution of VR Juggler applications on a cluster of machines. Maestro includes a GUI to assist an administrator with the task of overseeing and monitoring Juggler activities.

Despite VR Juggler's breadth of capabilities, it does face some challenges. To begin with, it offers no network component—save that required to facilitate its support of graphics clusters. As discussed in earlier subsections, while it is possible for a developer to utilize a network API of their choosing, doing so may result in software that is incompatible with other VR environments. Also, support for a VR standard is absent in VR Juggler unless OpenSceneGraph or OpenSG is used for rendering graphics. However, as noted for FreeVR and Delta3D, the updated and more robust X3D standard is not currently an option with either of these scene graph APIs.

5.5 Uni-Verse

Founded on the Verse [50] networking protocol, the Uni-Verse framework attempts to "create an open source Internet platform for multi-user, interactive, distributed, high-quality 3D graphics and audio [51]." To realize this premise, Uni-Verse is cross-platform (Apple OS X, Linux, Microsoft Windows), client-server oriented, and includes several C APIs and support tools. The Verse protocol is designed to efficiently share 2D and 3D image data among applications, thereby creating a distributed graphics processing environment. The Uni-Verse server, clients, and various support tools leverage Verse to communicate graphical information, while the Uni-Verse Sound Rendering Protocol is used for audio data. Image rendering is handled through OpenGL with support for OpenSG under Microsoft Windows. With OpenSG, the VRML 2 standard is available if the OpenVRML libraries are used. Sound rendering is facilitated by an audio/video programming environment called Pure Data [52] (a separate project from Uni-Verse) that must be installed on a given Uni-Verse client machine.

The Uni-Verse server and client are collectively made of a Verse server and the client software suite known as Quel Solaar. Noteworthy tools packaged in Quel Solaar include a rendering client also named Quel Solaar, a 3D modeling tool called Loq Airou, and a graphical 3D scene editor called Connector. The Quel Solaar client is based on OpenGL and described by the Uni-Verse project as a high-quality graphics renderer for desktops [53]. Loq Airou, also based on OpenGL, provides a means for creating 3D objects on a Uni-Verse server through an informal "sketch pad" user interface. Finally, Connector enables the editing and management of objects stored on a Uni-Verse server.

In addition to the server and client software, other Uni-Verse system components include, but are not limited to:

- Enough and Ample APIs -- written in C and C++, respectively, these libraries help in the development of Uni-Verse programs
- PyVerse -- a language binding to write Verse programs in Python
- Purple -- a tool that implements a scripting environment for Uni-Verse
- Saver and Loader -- tools to save and load, respectively, a Uni-Verse server's state
- Network security -- clients may be authenticated by a Uni-Verse server using a cryptographically strong cipher (RSA)

Although Uni-Verse offers a full-featured environment, it does face some issues. First, sound rendering is complicated by the need for the Pure Data programming environment to be installed on a Uni-Verse client machine. Second, support for the VRML 2 standard, while useful, falls short of the capabilities offered by the newer X3D standard.

VI. PLATFORMS

Our definition for *platform* picks up where that of *framework* leaves off. In Section III we specify that a platform is a fully contained, specialized environment within which a solution may be developed, tested, and executed. As such, wherever a platform-based solution is deployed, the platform must also be installed; this may be an onerous task for particularly large platforms. Of the 16 technologies surveyed in this paper, only Croquet and Project Wonderland are considered to be platforms.

6.1 Croquet

Built on top of Squeak (an open source derivative of Smalltalk) Croquet's aim is to provide a distributed, virtual environment that emphasizes realtime, multiuser collaboration. The Croquet platform employs OpenGL-based graphics,

OpenAL spatial sound, and a peer-to-peer networking/synchronization architecture (TeaTime) that facilitates object sharing and communication [54]. Due to Squeak's cross-platform nature, Croquet is available for Apple's OS X, Linux, and Microsoft Windows. It leverages Squeak's integrated development environment (IDE) to allow the creation, testing, debugging, and operation of Croquet software. Squeak's late-bound characteristics (inherited from Smalltalk [55]) make it is possible for Croquet program code to be accessed and altered not only prior to, but during execution. Moreover, the TeaTime architecture immediately propagates such changes to all relevant, distributed end-users. Croquet is designed under the premise that its applications will operate across small and large scale computing environments; in TeaTime, this is supported through object replication and synchronization. Another of Croquet's compelling strengths is that, out-of-the-box, it serves as a functional CVE: several of its example programs are fully operational CVEs with support for avatars, text chat, and the collaborative sharing of applications (e.g., text editors).

Relative to the goal of constructing a CVE, Croquet comes armed with an extensive set of features and capabilities. Nevertheless, some challenges and limitations remain. First, the current state of Croquet favors use by developers as opposed to typical end-users: the platform operates from within Squeak's IDE and, thus, may not be suitable for deployment to a general user population. To this point, the project web site notes that "in its current state of development, Croquet should be thought of as an enabling technology intended for use by software developers to create their own applications or customized OpenGL-based game engines—complete with user interfaces and application- specific features and functionalities [56]." Next, the prospect of providing Croquet CVE software to an end-user is troubled by the need for the whole Croquet platform to go along. At the time of this writing, Croquet comes packed into a 72 megabyte archive, regardless of the target OS. In this regard, it is similar to Java3D, Xj3D, or Project Wonderland (below), with the exception of having no Web Start or web browser applet channel for delivery. Last, Croquet requires a more modern workstation to perform well: the use of a late-bound language coupled with the multimedia factors of VR software means that extra graphics power, CPU speed, and memory will lead to better performance. Along these lines, the project FAQ suggests that a workstation "less than two years old" should be adequate to operate Croquet [57].

6.2 Project Wonderland

Sun Microsystems' Project Wonderland was motivated by problems in telecommuters' collaborative experiences. Specifically, Sun wanted to leverage CVE technology as a means to better integrate remote employees with their actual place of business and related work resources [58].

Organizations should be able to use Wonderland to create a virtual presence to better communicate with customers, partners, and employees. Individuals should be able to do their real work within a virtual world, eliminating the need for a separate collaboration tool when they wish to work together with others.

Implemented in Java and leveraging the Java3D API for graphics rendering, Project Wonderland offers a substantial platform for building CVEs. Though not as mature as Croquet, it nevertheless includes the same staple components: graphics (OpenGL or DirectX), network support, spatial sound, and multi-user capabilities with avatars. In addition, Java's cross-platform nature makes Project Wonderland available for users of Apple's OS X, Linux, Sun Solaris, and Microsoft Windows. Similar to Croquet, Project Wonderland comes as a fully functional CVE and also includes the ability to collaboratively share X Window applications. However, unlike Croquet, it is largely client-server oriented in its architecture. Its client and server components are based on two other open source Sun products: Project Looking Glass and Project Darkstar. Project Looking Glass and Java3D are used in combination to provide the 3D rendering for Project Wonderland; more specifically, Looking Glass offers APIs for building 3D applications and windowing environments. The Project Darkstar game server provides a "fully distributed, fault tolerant communication and event processing system [59]." It is through Darkstar that state and persistence are managed for all objects comprising a given virtual world.

The challenges facing Project Wonderland continue the similarities with Croquet. First, like Croquet, Project Wonderland is not currently in a polished state. Deployment and troubleshooting of the server components require some level of Java programming and systems administration skill. Also, even though the Wonderland client can be automatically distributed through Java Web Start, there may be need for troubleshooting and system configuration by the end-user. Next, depending on the platform, the client bundle for Project Looking Glass can range from 110 to 120 megabytes—this does not include the size of the Java runtime environment, should it be missing. Finally, Project Wonderland's features make it a resource intensive system: the need for a modern, speedy CPU (1.5 GHz or better), a graphics adapter with hardware acceleration, and a gigabyte or more of RAM precludes the use of older equipment [58].

VII. A SIMPLE RISK ASSESSMENT

As mentioned earlier, the aim of this survey is twofold: first, to concisely review several prominent, active desktop-VR technologies, and, second, to recommend the technology or technologies most well suited to building a CVE. In this section we make our recommendation.

Pressman notes that "the software engineering environment supports the project team, the process, and the product. But if the environment is flawed, it can be the source of significant risk [60]." Hence, it is important that our recommendation be risk savvy—that is, it should consider the development environment risks that effect each of the technologies we survey in earlier sections. To accommodate this, we undertake a simple, qualitative risk assessment of the 16 technologies. Our assessment is not all encompassing: for accessibility and

expedience we look at the major development environment risks comprising the limitations of the technologies. Also, because this is a qualitative exercise, there is some degree of subjectivity involved. Nevertheless, we believe there is value in the assessment, since the risks we measure are easy to understand, gage, and compare across the 16 technologies. Table 2 captures our assessment.

We calculate the overall risk for a given technology by adding up the products of *likelihood* and *impact* across several specific risks:

$$\text{Risk} = \sum_i (\text{Likelihood}_i \bullet \text{Impact}_i)$$

Likelihood indicates the perceived chance of something unwanted happening and impact denotes how significant (on a scale of 1 to 5, with 1 being minimal) the outcome will be. The risks we measure include:

- Primary Features Missing -- likelihood of 50% for either feature absent (graphics [G] and network [N]); impact of 5
- Audio Missing -- likelihood of 100% if audio is absent (A); impact of 2
- Lack of Utility -- likelihood of 50% for each CVE-oriented feature missing (proprietary, H-Anim, or other avatar support [V]; multi-user support [M]); impact of 3
- Support Missing -- likelihood of 25% for each type of support material missing (basic user/administrator/ developer documentation [D], multiple tutorials [T], one or more forums to post questions and interact with a development community [F], and a means to track bugs [B]); impact of 4
- VR Standard Missing or Not Current -- likelihood of 100% if there is no standard, 50% if a standard is not up-to-date; impact of 3
- Immature Technology -- likelihood of 10% for each year less than ten; impact of 2
- Deployment Challenges -- likelihood of 33.3% for each constraint (install package is 20 megabytes or more [L], special expertise required of the end-user [E], extra computing resources recommended [X]); impact of 1

Regarding the Deployment Challenges risk, it is difficult to accurately estimate how large an API, framework, or platform will be when installed: the inclusion or exclusion of optional features, utilities/tools, and optimizations can lead to very different footprints for the same technology. Hence, a best effort is made to determine whether the size of a default installation falls above or below a 20 megabyte watermark. This watermark was arrived at by presuming a download rate of 128,000 bits per second (a very modest high-speed line), which is equivalent to 15.6 kilobytes/sec, or 20 megabytes in a little more than 20 minutes. For a 56K dial-up modem (presuming 52,000 bits per second due to good line conditions), this works out to 20 megabytes in less than an hour. Our belief is that most end-users would not wait more than 20 minutes to download CVE (client) software, and fewer still would wait an hour.

Our desire is to select a technology that minimizes the overall risk. The largest quantity that could theoretically appear in Table II's Risk column is 20—given by the sum of the impacts for each individual risk. Happily, most of the technologies in our survey score around or below half of this maximum, with the least risky being Xj3D (2.43) and the most risky being FreeVR (12.8). Thus, we believe the technology that is most conducive to building and deploying CVEs is Xj3D. There are, however, some remarks to be made about this selection. First, it is noteworthy that among the three least risky technologies (Xj3D, Wonderland, and Croquet), deployment risk counts for much of the overall score. In particular, if we ignore the need for end-user expertise and additional computing power, the risks for Wonderland and Croquet drop to 1.93 and 2.43, respectively. Putting Wonderland in first place and tying Croquet with Xj3D may be a more accurate reflection of overall risk if we are unconcerned about our end-users' capabilities and resources. Next, this risk assessment tries to compare apples with apples across all technologies in the survey; this means an important feature specific to a given technology may not factor into the assessment. In the case of Croquet, for example, atomicity and synchronicity of multi-user activities are guaranteed: such a characteristic may be critical for some CVE applications. Finally, although a best effort is put forth to make reasonable choices, one should bear in mind that this is a qualitative risk assessment. Even a modest change in perspective on the 16 technologies could lead to different selections for likelihood and impact and, thus, a different outcome.

VIII. CONCLUSION

The boon of VR has been somewhat muted as a result of technology solutions that are redundant and/or incompatible. This circumstance has worked against the acceptance and integration of VR as a common computing interface. In an effort to untangle some of the overlap and differences among significant desktop-VR technologies, we have surveyed 16 current, open source solutions (grouped as APIs, frameworks, and platforms). During this review we discussed the strengths and challenges for each technology relative to the goal of constructing a CVE. We then undertook a simple, qualitative risk assessment to make a determination of which technologies were most well-suited to building a CVE. A given technology's risk was found by summing the products of likelihood and impact along an axis of seven development environment risks. This axis was drawn from the issues and limitations uncovered by our earlier review of the 16 technologies, and included consideration for the presence of CVE-oriented features that could aid the development process. Although we concluded that Xj3D is the technology most well-suited to building and deploying a CVE, even a minor shift in what is considered important about the surveyed technologies could change the risk assessment's outcome. For example, Project Wonderland becomes the least risky technology if we ignore the deployment needs of end-user expertise and computing power.

TABLE 1: SUMMARY OF ACTIVE, DESKTOP-VR TECHNOLOGIES

Technology (Version)	API, Framework, or Platform	Capabilities	Maturity	Open, Free VR Standard	Comments
FreeVR (0.5g)	API	G	Since 2003	VRML 2	Found online at *www.freevr.org*. Generally directed at, but by no means restricted to, CAVE environments. In general, serves as wrapper around VR input devices and offers multi-processor management
Java3D (1.5.1)	API	G-N	Since 1998	VRML 2	Found online at *java3d.dev.java.net*. Sun Microsystem's 3D graphics API for Java. Uses scene graph data structure to manage graphics information
OGRE (1.4.5)	API	G	Since 2001	No	Found online at *www.ogre3d.org*. The Object-oriented Graphics Rendering Engine; abstracts OpenGL and Direct3D
OpenGL (2.1)	API	G	Since 1992	No	Found online at *www.opengl.org*. Licensing free for end-users and developers, but not for graphics hardware manufacturers. Libraries packaged with most Oss
OpenSceneGraph (2.0)	API	G	Since 1999	VRML 2	Found online at *www.openscenegraph.com*. Similar to OpenSG. Abstracts OpenGL libraries to provide scene graph data structure
OpenSG (1.8.0)	API	G	Since 1999	VRML 2	Found online at *opensg.vrsource.org*. Similar to OSG. Abstracts OpenGL libraries to provide scene graph data structure
OpenVRML (0.16.6)	API	G-A	Since 2000	X3D	Found online at *www.openvrml.org*. Currently supports only VRML-encoded X3D (i.e., no XML)
X3D Tool Kit (1.2)	API	G-A	Since 2004	X3D	Found online at *sourceforge.net/projects/x3dtoolkit*. Abandoned in 2004 and resurrected in 2006
Xj3D (1.0)	API	G-N-A	Since 2002	X3D	Found online at *www.xj3d.org*. Maintained by Web3D Consortium. OpenGL or Java3D may be used for rendering X3D scene graph
Crystal Space (1.2)	Framework	G-N-A	Since 1997	No	Found online at *www.crystalspace3D.org*. Highly modularized collection of C++ libraries forming an application engine for 3d software and games
Delta3D (1.5.0)	Framework	G-N-A	Since 2004	VRML 2	Found online at *www.delta3d.org*. Suite of libraries and tools developed by MOVES Institute at Naval Postgraduate School. Used largely for DoD computer simulation and training
Quake III Engine (1.32b)	Framework	G-N-A	Since 1999	No	The ioquake3 project was reviewed for this paper: *www.ioquake3.org*; original source found online at *www.idsoftware. com*. First released as the Quake III Arena commercial game in 1999, ID Software made the Quake III Engine's source code available as open source under the GNU GPL license in 2005
VR Juggler (2.0.3)	Framework	G-A	Since 1997	VRML 2	Found online at *www.vrjuggler.org*. Middle-ware framework to support VR environments
Uni-Verse (3)	Framework	G-N-A	Since 2004	VRML 2	Found online at *www.uni-verse.org*. Centered around Verse (network protocol for graphical, 3D environments)
Croquet (1.0.18)	Platform	G-N-A	Since 2002	VRML 2	Found online at *opencroquet.org*. Enables creation of CVEs with Squeak (derivative of Smalltalk). Peer-to-peer environment in which the SDK/IDE acts as viewer
Project Wonderland (0.4)	Platform	G-N-A	Since 2007	X3D	Found online at *lg3d-wonderland.dev. java.net*. Enables creation of CVEs with Java. Largely client-server oriented

TABLE 2: RISK TABLE FOR ACTIVE, DESKTOP-VR TECHNOLOGIES

Technology	Primary Features Missing	Audio Missing	Utility Lacking	Support Missing	VR Standard Missing or Not Current	Immature Technology	Deployment Challenges	Risk
FreeVR	N: 50% * 5 = 2.5	A: 100% * 2 = 2	V-M: 100% * 3 = 3	T-F-B: 75% * 4 = 3	VRML 2: 50% * 3 = 1.5	6 Years Old: 40% * 2 = .8	Negligible	12.8
Java3D	Negligible	A: 100% * 2 = 2	M: 50% * 3 = 1.5	Negligible	VRML 2: 50% * 3 = 1.5	Negligible	L: 33.3% * 1 .33	5.33
OGRE	N: 50% * 5 = 2.5	A: 100% * 2 = 2	M: 50% * 3 = 1.5	Negligible	Missing: 100% * 3 = 3	8 Years Old: 20% * 2 = .4	L: 33.3% * 1 .33	9.73
OpenGL	N: 50% * 5 = 2.5	A: 100% * 2 = 2	V-M: 100% * 3 = 3	B: 25% * 4 = 1	Missing: 100% * 3 = 3	Negligible	Negligible	11.5
OSG	N: 50% * 5 = 2.5	A: 100% * 2 = 2	M: 50% * 3 = 1.5	Negligible	VRML 2: 50% * 3 = 1.5	Negligible	L: 33.3% * 1 .33	7.83
OpenSG	N: 50% * 5 = 2.5	A: 100% * 2 = 2	M: 50% * 3 = 1.5	Negligible	VRML 2: 50% * 3 = 1.5	Negligible	L: 33.3% * 1 .33	7.83
OpenVRML	N: 50% * 5 = 2.5	Negligible	M: 50% * 3 = 1.5	T: 25% * 4 = 1	Negligible	9 Years Old: 10% * 2 = .2	Negligible	5.2
X3D Toolkit	N: 50% * 5 = 2.5	Negligible	V-M: 100% * 3 = 3	D-T: 50% * 4 = 2	Negligible	5 Years Old: 50% * 2 = 1	Negligible	8.5
Xj3D	Negligible	Negligible	M: 50% * 3 = 1.5	Negligible	Negligible	7 Years Old: 30% * 2 = .6	L: 33.3% * 1 .33	2.43
Crystal Space	Negligible	Negligible	Negligible	Negligible	Missing: 100% * 3 = 3	Negligible	L: 33.3% * 1 .33	3.33
Delta3D	Negligible	Negligible	Negligible	B: 25% * 4 = 1	VRML 2: 50% * 3 = 1.5	5 Years Old: 50% * 2 = 1	L: 33.3% * 1 .33	3.83
Quake III	Negligible	Negligible	Negligible	Negligible	Missing: 100% * 3 = 3	Negligible	L: 33.3% * 1 .33	3.33
VR Juggler	N: 50% * 5 = 2.5	Negligible	V-M: 100% * 3 = 3	T: 25% * 4 = 1	VRML 2: 50% * 3 = 1.5	Negligible	L: 33.3% * 1 .33	8.33
Uni-Verse	Negligible	Negligible	V-M: 100% * 3 = 3	F-B: 50% * 4 = 2	VRML 2: 50% * 3 = 1.5	5 Years Old: 50% * 2 = 1	Negligible	7.5
Croquet	Negligible	Negligible	Negligible	Negligible	VRML 2: 50% * 3 = 1.5	7 Years Old: 30% * 2 = .6	L-E-X: 100% * 1 = 1	3.1
Wonderland	Negligible	Negligible	Negligible	Negligible	Negligible	2 Years Old: 80% * 2 = 1.6	L-E-X: 100% * 1 = 1	2.6

REFERENCES

[1] R. Macredie, S. J. E. Taylor, X. Yu *et al.*.Virtual reality and simulation: an overview. pp. 669--674.

[2] V. Luckas and R. Dörner. Experience form the future---using object-orientation concepts for 3D visualization and validation of industrial scenarios," *ACM Comput. Surv.*, vol. 32, no. 1, pp. 38, 2000.

[3] S. Benford, I. Taylor, D. Brailsford *et al.*. Three dimensional visualization of the World Wide Web, *ACM Comput. Surv.*, vol. 31, no. 4, pp. 25, 1999.

[4] J. F. Franco, S. R. da Cruz and R. de Deus Lopes. Computer graphics, interactive technologies and collaborative learning synergy supporting individuals' skills development, pp. 42.

[5] J. Yan. A 3D Pedagogical Agent Enhanced Virtual Therapy System for People with Idiopathic Parkinson Disease, *The International Journal of Virtual Reality*, vol. 7, no. 2, pp. 59-62, April, 2008.

[6] S. V. G. Cobb and P. M. Sharkey. A Decade of Research and Development in Disability, Virtual Reality and Associated Technologies: Review of ICDVRAT 1996--2006, *The International Journal of Virtual Reality*, vol. 6, no. 2, pp. 51-68, June, 2007.

[7] M. Hirose.Virtual Reality Technology and Museum Exhibit, *The International Journal of Virtual Reality*, vol. 5, no. 2, pp. 31-36, June, 2006.

[8] D. Pape, J. Anstey and B. Sherman. Commodity-based projection VR.

[9] W. S. Bainbridge. The Scientific Research Potential of Virtual Worlds, *Science*, vol. 317, no. 5873, pp. 421-534, July, 2007.

[10] A. van Dam, A. Forsberg, D. Laidlaw *et al.*. Immersive VR for scientific visualization: a progress report, *Computer Graphics and Applications, IEEE*, vol. 20, no. 6, pp. 26-52, Nov/Dec, 2000.

[11] R. Lau, F. Li, T. Kunii *et al.*. Emerging Web graphics standards and technologies, *Computer Graphics and Applications, IEEE*, vol. 23, no. 1, pp. 66-75, Jan/Feb, 2003.

[12] J. Cross, T. O'Driscoll and E. Trondsen. Another life: virtual worlds as tools for learning, *eLearn*, vol. 2007, no. 3, 2007.

[13] H. Berger, M. Dittenbach, D. Merkl *et al.*. Playing the e-business game in 3D virtual worlds, *OZCHI '06*.

[14] N. Holmberg, B. Wünsche, and E. Tempero. A framework for interactive web-based visualization, pp. 137--144.

[15] C. Bouras, A. Panagopoulos, and T. Tsiatsos. Advances in X3D multi-user virtual environments, pp. 136--142.

[16] C. Anslow, S. Marshall, J. Noble *et al.*. Evaluating X3D for use in software visualization, pp. 161--162.

[17] A. Segura, I. Arizkuren, I. a. Aranburu *et al.*. High quality parametric visual product configuration systems over the web, pp. 159--167.

[18] I. J. Grimstead, D. W. Walker, and N. J. Avis. Collaborative visualization: a review and taxonomy, pp. 61-69.

[19] M. Barnes. Virtual reality and simulation, pp. 101--110.

[20] O. Otto, D. Roberts, and R. Wolff. A review on effective closely-coupled collaboration using immersive CVE's, pp. 145--154.

[21] Adobe Systems Inc. "Flash Player Penetration," http://www.adobe.com/products/player_census/flashplayer.

[22] A. Balkan. "Open Source Flash: Projects," http://osflash.org/projects.

[23] D. Brutzman, and L. Daly, *Extensible 3D Graphics for Web Authors*, San Francisco: Morgan Kaufmann, 2007.

[24] W. R. Sherman. "FreeVR: VR Administrator's Guide," http://www.aces.dri.edu/freevr/vradmin_guide.html.

[25] J. Couch. "Raw J3D: Scene Graph Basics," http://java3d.j3d.org/tutorials/raw_j3d/chapter1/scene_graph_basics.html.

[26] E. Gamma, R. Helm, R. Johnson *et al.*, *Design Patterns: Elements of Reusable Object-Oriented Software*, pp. 163-173, Reading, Massachusetts: Addison-Wesley, 1995.

[27] S. Hong. "J3D-VRML97 Project Page," https://j3d-vrml97.dev.java.net.

[28] The OGRE Team. "About: What Is OGRE?," http://www.ogre3d.org.

[29] The OGRE Team. "Community: Add-on Projects," http://www.ogre3d.org.

[30] Khronos Group. "OpenGL Platform & OS Implementation," http://www.opengl.org/documentation/implementations/.

[31] Khronos Group. "OpenGL Overview," http://www.opengl.org/about/overview/.

[32] Khronos Group. "About OpenGL 2.1," http://www.opengl.org/documentation/current_version.

[33] I. Silicon Graphics. "OpenGL Reference Manual," http://www.glprogramming.com/blue.

[34] I. Silicon Graphics. "OpenGL Licensing and Logos Information," http://www.sgi.com/products/software/opengl/license.html.

[35] D. Reiners, and G. Voss. "OpenSG Clustering," http://opensg.vrsource.org/trac/wiki/FeaturesClustering.

[36] Y. L. Gok. "X3D Tool Kit: Features," http://artis.imag.fr/Software/X3D/features.html.

[37] L. E. Ramey, P. Kerchen, and K. Sons. "X3D Tool Kit: Latest News," http://sourceforge.net/forum/forum.php?forum_id=647928.

[38] Web3D Consortium. "Using Xj3D in your Java Application," http://www.xj3d.org/tutorials/xj3d_application.html.

[39] J. Tyberghein. "Crystal Space: Features," http://www.crystalspace3d.org/main/Features.

[40] J. Tyberghein. "CEL: Concepts," http://crystalspace3d.org/cel/docs/online/manual-1.2/Concepts.html.

[41] J. Tyberghein. "Crystal Space: CELstart," http://www.crystalspace3d.org/main/CELstart.

[42] MOVES Institute. "About Delta3D," http://www.delta3d.org/index.php?topic=about.

[43] id Software. "id Software: Technology Licensing," http://www.idsoftware.com/business/technology/.

[44] B. Hook. "Book of Hook: The Quake3 Networking Model," http://trac.bookofhook.com/bookofhook/trac.cgi/wiki/Quake3Networking.

[45] R. Duffy. "QERadiant Project Home Page," http://www.qeradiant.com.

[46] id Software. "Quake III Arena GPL source release Readme," ftp://ftp.idsoftware.com/idstuff/source/quake3-1.32b-source.zip.

[47] I. S. U. Virtual Reality Applications Center. "VR Juggler Portable Runtime: Programmer's Guide," http://developer.vrjuggler.org/docs/vapor/2.0/programmer.guide/programmer.guide.

[48] I. S. U. Infiscape and Virtual Reality Applications Center. "VR Juggler: Modules," http://www.vrjuggler.org/modules.php.

[49] Infiscape. "Maestro User's Guide," http://www.infiscape.com/documentation/maestro/docs/0.4/userguide.pdf.

[50] E. Steenberg, and E. Brink. "Verse Project Web Page," http://verse.blender.org.

[51] Uni-Verse Consortium. "Uni-Verse Project Web Site," http://www.uni-verse.org.

[52] PD Community. "Pure Data Project Web Site," http://www.puredata.org/.

[53] Uni-Verse Consortium. "Uni-Verse: Tutorials," http://www.uni-verse.org/Tutorials.54.0.html.

[54] Croquet Consortium. "Croquet: System Overview," http://opencroquet.org/index.php/System_Overview.

[55] Squeak Foundation. "Squeak: About," http://www.squeak.org/About/.

[56] Croquet Consortium. "Croquet: A Beginners Guide," http://opencroquet.org/index.php/A_Beginners_Guide.

[57] Croquet Consortium. "Croquet: FAQs," http://opencroquet.org/index.php/FAQs.

[58] Sun Microsystems. "Project Wonderland Web Site," http://lg3d-wonderland.dev.java.net.

[59] Sun Microsystems. "Project Darkstar FAQ," http://www.projectdarkstar.com/index.php?option=com_content&task=view&id=65&Itemid=65.

[60] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, Fourth ed., pp. 139-140: McGraw-Hill, 1997.

Timothy E. Wright Tim is a Ph.D. Candidate in the Computer Science & Engineering program at the University of Notre Dame. His research interests include virtual reality and information security. In the past he has worked as a software engineer, computer fraud and abuse investigator, and an information security professional. While he completes his dissertation at Notre Dame, he is simultaneously a Senior IT Audit Consultant for the University. Tim is also a member of the ACM, ISACA, and ISC^2.

Gregory Madey, Ph.D. Dr. Madey's research focuses on the use of computer science to develop solutions to a wide range of problems. Recent problem domains include environomental science, the open source software phenomenon, decision support for cellular phone networks, mobile ad hoc sensor networks, and bioinformatics. His research is interdisciplinary and draws on theories and topics from database theory, e-Technologies, agent-based simulation, artificial intelligence, emergence and self-organization, chaos and complexity, organizational theory, artificial neural networks, data mining, and operation research.