NOTE TO USERS

This reproduction is the best copy available.

OPEN SOURCE SYNTHETIC TRAINING ENVIRONMENTS AND THE CANADIAN FORCES©

by

Patrick Castonguay

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Applied Science in Technology Innovation Management

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

July 29, 2009

Your file Votre référence
ISBN: 978-0-494-60255-3
Our file Notre référence
ISBN: 978-0-494-60255-3

ACKNOWLEDGEMENTS

I would like to acknowledge and extend my heartfelt gratitude to the following persons who have made the completion of this thesis possible:

My thesis advisor and friend, Professor Trevor Pearce, for his continuous support and guidance, without his inspiration and insight this work would not have been possible.

Murray and John from CogSim Technologies; Andrew from the CIGI team; and Doug from the OpenEaagles team; as well as many others; for providing technical assistance as well as encouragement. They made the learning process much easier and enjoyable.

Capt Peter Dieter, for the many enlightening and interesting discussions which took place before and during my studies. I am certain these discussions will continue to take places wherever either of us end-up in the future.

My parents, and most importantly my wife Kelly as well as my children (Quinten, Alia and Simon) for their understanding and support during the long hours spend away from them working on this achievement. I hope to make you all proud and maybe someday be a source of inspiration myself, just like my father has always been to me.

15 July 2009

DISCLAIMER

The findings and conclusions included in this thesis are not to be construed to reflect the views and/or positions of the Department of National Defence of Canada. Though my studies were sponsored by the Canadian Forces, the work in this document was not governed in any way shape or form by the Canadian Government or any of its agencies.

This document contains color diagrams, a black and white version may be obtained by contacting the author at the following email address: pcaston3 at connect dot carleton dot ca.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PLATES

LIST OF ACRONYMS

Acronym	Definition
AFRL	Air Force Research Laboratory
API	Application Programming Interface
CBC	Canadian Broadcasting Corporation
CCL	CIGI Class Library
CDB	Common Database
CDDL	Common Development and Distribution License
CDR	Canadian Defence Review
CF	Canadian Forces
CFAWC	Canadian Forces Air Warfare Center
CGF	Computer Generated Forces
CIGI	Common Image Generator Interface
COTS	Commercial Off The Shelf
CSP	Combat Simulator Project
CTFT	Continuous Training Federation Testbed
CTL PNL	Control Panel
DAOD	Defence Administrative Order and Directive
DB	Database
DIS	Distributed Interactive Simulation
DISA	Defence Information System Agency
DND	Department of National Defence
DOF	Depth Of Field
DRDC	Defence Research and Development Canada
EDL	Eaagles Definition Language
EO	Electro-Optic
EOIR	Electro-Optic and Infrared
FOSS	Free and Open Source Software
FOV	Field of View
FSF	Free Software Foundation
GL	Graphics Library
GLSL	OpenGL Shading Language
GPL	General Public License
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDD	Head-Down Display
HDR	High Dynamic Ranging
HLA	High Level Architecture
HMI	Human Machine Interface
HUD	Head-Up Display
ICD	Interface Control Document

Acronym	Definition
IG	Image Generator
IOS	Instructor Operator Station
IP	Intellectual Property
IR	Infrared
IT	Information Technology
ITSEC	Interservice/Industry Training, Simulation and Education Conference
LAN	Local Area Network
LGPL	Lesser General Public License
LOC	Lines of Code
LOD	Level of Detail
MCE	Mapping and Charting Establishment
MCU	Master Control Unit
MMD	Moving Map Display
MPV	Multi Purpose Viewer
MS	Microsoft
MSIAC	Modeling and Simulation Analysis Center
NASA	National Aeronautics and Space Administration
NASMP	Navy Aviation Simulation Master Plan
NATO	North Atlantic Treaty Organization
NPS	Naval Postgraduate School
OE	OpenEaagles
OMS	Operational Mission Simulator
OSBR	Open Source Business Resource
OSCON	Open Source Conference
OSG	Open Scene Graph
OSI	Open Source Initiative
OSI-PDMS	Open Source Initiative for Parallel and Distributed Modeling
OSL	Open Source License
OSS	Open Source Software
OTD	Open Technology Development
OTW	Out The Window
PC	Personal Computer
PDU	Protocol Data Units
PIP	Picture-In-Picture
PPU	Post Processing Unit
RAPTOR	Reusable Architecture / Plugin Technology / Open and Reconfigurable
RF	Radio Frequency
RTT	Render-to-Texture
SDO	Standards Development Organization
SE	Synthetic Environment
SECO	Synthetic Environment Coordination Office
SIMAF	Simulation and Analysis Facility

Acronym	Definition
SISO	Simulation Interoperability Standards Organization
SIW	Simulation Interoperability Workshop
SSG	Standing Study Group
SVN	Subversion
TBSC	Treasury Board of Canada Secretariat
TDP	Technology Demonstration Projects
TEC	Topographic Engineering Center
TIM	Technology Innovation Management
TSJ	Training Simulation Journal
TUAS	Tactical Unmanned Aircraft Systems
UDP	User Datagram Protocol
UK	United Kingdom
US	United States
UTM	Universal Time
VRML	Virtual Reality Modeling Language
VRSG	Virtual Reality Scene Generator
VTP	Virtual Terrain Project
XML	eXtensible Markup Language

1 INTRODUCTION

The use of Modeling and Simulation (M&S) to create synthetic environments for military training is quite prominent in the Canadian Forces (CF) (Garnett, Hénault, & Leggat, 2000) and M&S is considered a wealthy industry even through the current recession. It is reported that the United States (US) alone generated close to $2 billion in M&S sales for the year 2008 and it is projected to reach more than $10 billion globally by the year 2018 (Cason, 2009). The CF is a smaller consumer for this technology than the US but is still a relevant participant. This is particularly true for the Air Force, where the use of real equipment for training is often too expensive, too dangerous, impractical, or impossible. For example, the CF is looking at spending an estimated $7.4 Million (CDR, 2006) over the span of two years to provide an interim training solution for the Aurora long range patrol aircraft while awaiting a modernized version of the associated crew Operational Mission Simulator (OMS).

Since the original acquisition of the OMS in the mid seventies, the actual aircraft configuration and role have evolved, but the training system has not. For example, the current OMS lacks the Electro-Optic and Infrared (EOIR) sensor now found on the airplane. This sensor is a collection of three specialized video cameras used for long range surveillance. The disparity between the real aircraft and the training system forces operators to conduct simulator-based training that is not representative of current mission profiles. This is undesirable, and requires the entire crew to fly a higher number of training hours in the real aircraft so that

the EOIR operator becomes proficient. The use of real aircraft to conduct training that might be accomplished in an up-to-date simulator has significant operational and financial impact. Unfortunately, the current CF simulator procurement process is oriented to large infrequent investments rather than regular updates that keep pace with the aircraft, and therefore, future disparity between the real aircraft and the simulator seems inevitable as the aircraft's configuration and role continue to evolve.

Open source software (OSS) is now a recognized practice (Stacy Avery, 2008). Many information technology companies are embracing the benefits of open source as well as proprietary software development and are generating products and services which reflect the evolution in the industry. This is not representative of the Government of Canada (GoC) when it comes to simulation-based training. Most of the training systems in the CF have been, and are still being, acquired as closed proprietary systems, as is the case for the OMS and its proposed replacement. Some identified advantages in using OSS in a military context include decreasing vendor lock-in, shortening response time, enhancing agility; reducing acquisition cost, and increasing quality (Herz, Lucas, & Scott, 2006). Therefore, the CF should start investigating the potential of OSS as an option to help prevent the occurrence of training disparities similar to the OMS.

1.1 Objectives

The objective of this research is to investigate the applicability of OSS in the context of synthetic training environments.

1.2 Contributions

There are four major contributions from this research that complement existing research in the field. The first is the design and implementation of the OpenEOIR project (Castonguay, 2009b), an OSS prototype synthetic training environment for EOIR sensor operators. This prototype takes advantage of mature OSS technologies such as the OpenEaagles framework, Common Image Generator Interface (CIGI), and Multi-Purpose Viewer (MPV). It is a good candidate for continual development into a complete EOIR simulation framework, and its details are discussed in sections 5.

The second contribution is the demonstration of the maturity and relevance of OSS in the context of development of synthetic training environment.

The ecosystem generated by the development of OpenEOIR is the third contribution of this research. It increased the synergy of OSS in the field of simulation based training and facilitated the implementation of the prototype.

Lastly, all modifications made to existing OSS frameworks while developing the OpenEOIR project are contributed back to the respective simulation communities. Software contributions are made to OpenEaagles and MPV, and non-software contributions are made to CIGI.

1.3 Relevance

There are at least three groups to whom this research may be relevant. The first comprises top management teams of the CF who are considering the

use, development or acquisition of simulation-based systems. This research shows that OSS systems and OSS development methods should be strongly considered when making important business decisions related to simulation-based training system acquisition. Project managers in charge of the OMS as well as other training systems in the CF have already shown interest in this research.

Secondly, this research may be relevant to researchers and students with an interest in the field of M&S as it provides them with an example of an OSS stack for use in synthetic environment based training. Furthermore, the prototype can be used as a starting point to develop more advanced solutions to be used in sensor operator related research. This is important as OSS is still in its infancy and sometimes not recognised in the M&S field. Some scientists working for DND believe that OSS cannot even be considered for their projects and this research suggests the contrary.

Thirdly, simulation companies and simulation practitioner will be interested in this research. Even-though OSS in M&S can be considered in its infancy, the importance of the OSS trend in other fields of practice is well recognized (O'Flynn, 2008). This is starting to affect simulation-based companies and some are interested in monitoring this trend. "It is something [CAE Professional Services] discussed many times over the past few months as [we recognise] it is a trend in the industry in general" (Brennan, 2008).

1.4 Organization

This dissertation is organised as follows. A background description of the different graphic processing technologies relevant to the design and implementation of the OpenEOIR prototype are presented in section 2, along with a brief description of terminology. This is followed by a review of the current literature on open source software, EOIR sensor training environments, and business ecosystems. These three areas are related, as OSS was used to create an EOIR system while developing a relevant ecosystem. A description of the particular approach followed during this research is detailed in section 4. The requirements, design, implementation, and testing of the OpenEOIR prototype are described in section 5. The analysis of the OSS used for the development of the prototype is then presented in section 6 and a discussion on the maturity and potential of OSS ensue in section 7. Finally section 8 states the conclusions of this research, underlines the limitations of this work, and provides suggestions for further research.

2 BACKGROUND

The graphics-oriented technologies underlying the OpenEOIR prototype are presented in this section, based on the Open Graphics Library (OpenGL) (SGI, 2009). Furthermore, terminologies providing for a better understanding of the document are presented. Other important OSS software technologies such as OpenSceneGraph, CIGI, OpenEaagles and Delta3D have been published in papers or journals and are discussed in section 3.2 of the literature review covering the state of the art.

2.1 OpenGL

OpenGL is a platform independent software library which provides an Application Programming Interface (API) supporting the generation of two-dimensional (2D) and three-dimensional (3D) graphics. It is the open standard counterpart to the well known (proprietary) Microsoft DirectX (Microsoft, 2009). The details behind OpenGL, the way it approaches graphics rendering, as well as the concepts of shaders, Render-to-Texture (RTT), and Level of Detail (LOD) are presented here. The use of these in the development of OpenEOIR is discussed later in section 5.

OpenGL is a generic software abstraction of the different assembly languages used to control Graphics Processing Unit (GPU). It remains a low-level graphics language, as it does not offer windowing or high-level 3D object descriptions; however, it does provide an extensive set of commands to define and combine geometric primitives such as points, lines and polygons. The

OpenGL abstraction is based on a complex logical structure for generating visual representations, referred to as the OpenGL rendering pipeline.

A simplified version of the pipeline is shown in Figure 1. The goal of the pipeline is to create the *Frame Buffer*, which contains an image in a pixel-oriented format suitable for output to the user display. In the first stage of the pipeline, the *Evaluator* constructs an interim description of the objects to display based on geometric information found in the *Vertex Data* and *Pixel Data*. This data is supplied primarily by the application, but may also be fed back from deeper in the pipeline. At the *Primitive Assembly* stage, the objects are positioned in an abstract 3D scene and each is assigned a texture from the *Texture Memory*, The texture can be thought of as an image wrapped onto the object's 3D shape, similar to a skin. In the *Rasterization* stage, the resulting 3D scene is converted into a 2D view, similar to that which might be obtained by taking a photograph of the scene. At this stage, combinations of related pixels (such as a line), are grouped as logical fragments. The fragments are then

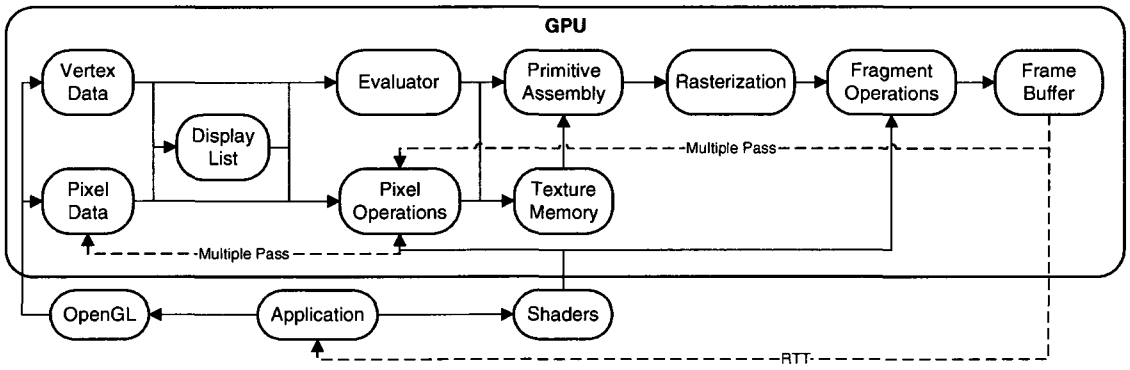


Figure 1: OpenGL Rendering Pipeline

processed by *Fragment Operations* to apply many visual effects, such as shading, transparency (alpha), blending, or even fog. Once this processing is completed, the scene view is written to the *Frame Buffer* which can be directed to the display.

Note that the *Frame Buffer* can also be used as a source of data to be fed back into the pipeline in two ways for further processing. One way, which is internal to the pipeline, allows data to be to be fed back through *Pixel Operations* to influence textures. The other way, which is referred to as Render-to-Texture, allows an external application to select data and feed it back into the pipeline.

A shader is a GPU program which can be inserted into the rendering pipeline dynamically by an application. Shader programs are used to replace or affect portions of the rendering pipeline, and Figure 1 shows that the most common use of shaders affects the fragment and/or pixel operations. For example, a shader could be used to control the brightness of a particular object at run-time without affecting the rest of the scene. Shaders are an effective way to implement customized graphics effects which are specific to the application in use. The GL Shading Language (GLSL) is a standardised abstract language for writing shader programs for OpenGL.

The Level of Detail (LOD) is an indication of the quality of the visual representation of a model and is used to manage the GPU's workload. It is proportional to the number of polygons used, as well as the number of components shown when rendering 3D objects. A common scheme when

implementing a visualisation system is to have an entity supporting multiple LOD within the same definition file. Supporting multiple LOD requires entities to be modeled in such a way that both components of the LOD can be indexed. The most common control mechanisms used are distance-to-the-camera and relative pixel size on the screen. Applications set the mechanism to be used when rendering the 3D model and OpenGL handles the calculation of the indexing parameter. When using the distance mechanism, an aircraft displayed in the horizon with low LOD could be rendered as just one triangle representing the body and the only component displayed could be the collision light. The wings, tail and position lights are not rendered, though available in the aircraft definition file. When displayed relatively close to the camera, at the higher LOD, the body can be represented by thousands of polygons, and can include the details of the wings, tail, antennas, navigation lights, and other parts. This distance-to-the-camera mechanism works well if the airplane is quite far, as the observer would not differentiate between a fully rendered model and one with a low LOD. But it does not lend itself well to the concept of a camera having a powerful zoom. If the distance mechanism is used, an airplane situated far from the observer would be rendered at low LOD even when the zoom is used, and the observer would actually see a triangle and not a plane. Therefore, when considering an optical sensor, the relative pixel size is a better LOD control mechanism.

2.2 Use of terms

To be more concise, certain terms are abbreviated and used in a greater context throughout the remainder of this document. M&S is extended to signify modeling and simulation-based training, and not just modeling and simulation. The term sensor is used to represent an electro-optic and infrared sensor.

3 LITERATURE REVIEW

The literature relevant to this research encompasses open source software, the state of the art in EOIR sensor simulation for use in training, and business ecosystem.

3.1 Open Source Software

Open source software (OSS) is not a new phenomenon, but it can still be considered in its infancy in the field of M&S. This section first defines OSS, places it in the context of M&S, and discuss some of the many ways to participate in OSS projects. An overview of OSS licensing and a discussion on some relevant licenses are then given. This section continues by comparing the positions of different governments, mainly Canada and the US, on the issue of using OSS and shows that the US appears more accepting. Finally a discussion of the state of OSS within the Simulation Interoperability Standard Organization is presented which highlights that the organization does not have a definite position.

3.1.1 What is Open Source Software?

OSS was born out of the Free Software Foundation (FSF), which was established in 1985. According to the Open Source Initiative (OSI), OSS means more than access to the source code; the distribution terms of open-source software must also comply with a selection of criteria ranging from free redistribution, to non-discrimination, to derived work, and integrity of author's work. A complete list of the OSI OSS criteria, along with associated definitions,

can be found in Appendix A: OSI Criteria. In short, OSS allows people to access its source code so that they can study, modify, and redistribute the source code. This software development method also has the goal of accepting contributions from other developers.

In contrast, the proprietary development that is typical in commercial software is a practice where clients and users have to pay a licence fee in order to be able to use or incorporate the software at run-time or during development. This type of distribution scheme does not normally allow users access to the source code, and the typical product received is either an executable program or a set of libraries with defined interfaces.

The term Free Software is championed by the FSF which has a worldwide mission to promote computer user freedom and to defend the rights of all free software users (FSF, 2008). The idea of freedom defended by the FSF is not one of price but one of liberty. It is defined by the freedoms to run the program for any purpose; to study it and adapt it to ones needs; to redistribute the program; and to modify it and distribute the modifications back to the community (Bjorgvinsson & Thorbergsson, 2007; FSF, 2008).

The term OSS is often combined with Free Software and referred to as Free and Open Source Software (FOSS). The two are arguably not the same: open source is a development methodology and free software is viewed a social movement. It has been stressed by the FSF that Free Software (as per their definition) is implicitly OSS but that the inverse is not the case. Depending on

the licence being used, OSS can possibly not respect the four freedoms. The FSF is quite purist in their approach; it will not recognise as '*free*' applications which display, or actively support proprietary commercial solutions. Alternatively, the OSI shows a more business-aware approach and encourages economically viable business creation based on OSS components or applications. For the context of this document OSS is used as a generalization of both terms. It is not meant to be restrictive or exclusive, but includes the overarching idea encompassing both principles independently or together.

OSS frameworks for M&S began emerging approximately four years ago. At that time they were identified as "disruptive innovation" (Darken, McDowell, & Murphy, 2005). OSS was also described as a motivation engine for innovation within the field of M&S (McDowell, 2007) without which companies could charge large amount for software applications based on the magic formula: "you can't get it anywhere else" (Goth, 2005). It was also highlighted that OSS is most successful when focused on underlying infrastructure, and that businesses can profit more by focusing their efforts on differentiators while collaborating on the commodity functionalities (Pearce, 2006). Figure 2 shows that the differentiator can be in technical complexity or in a niche area for which OSS solutions do not yet exist. Figure 3 then shows the motivating effects OSS can have on companies to be more innovative when the commodity functionalities grow over time. This effect can be referred to as innovative seeding, where a small amount of innovation can germinate and grow to be something greater. The appearance

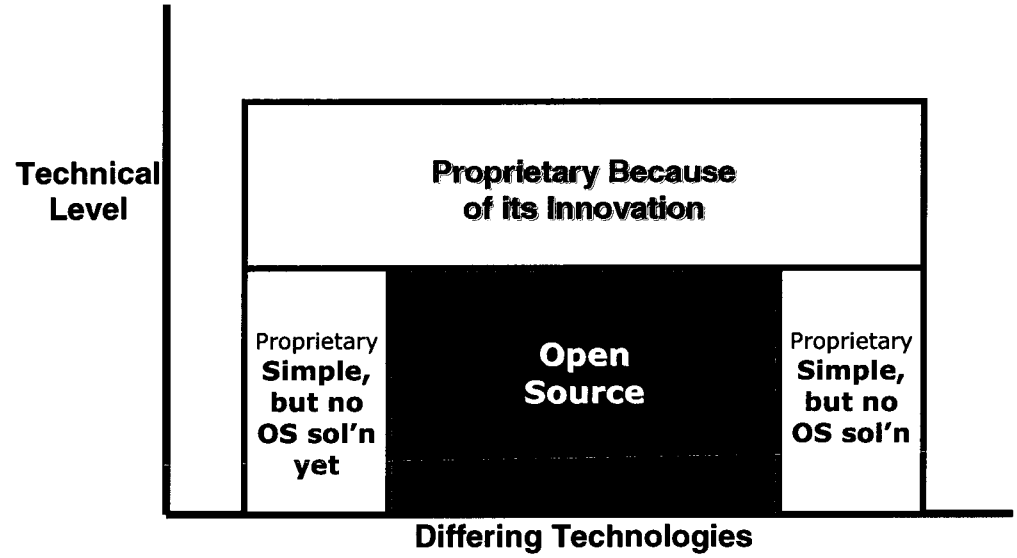


Figure 2: Situating OSS (McDowell, 2007)

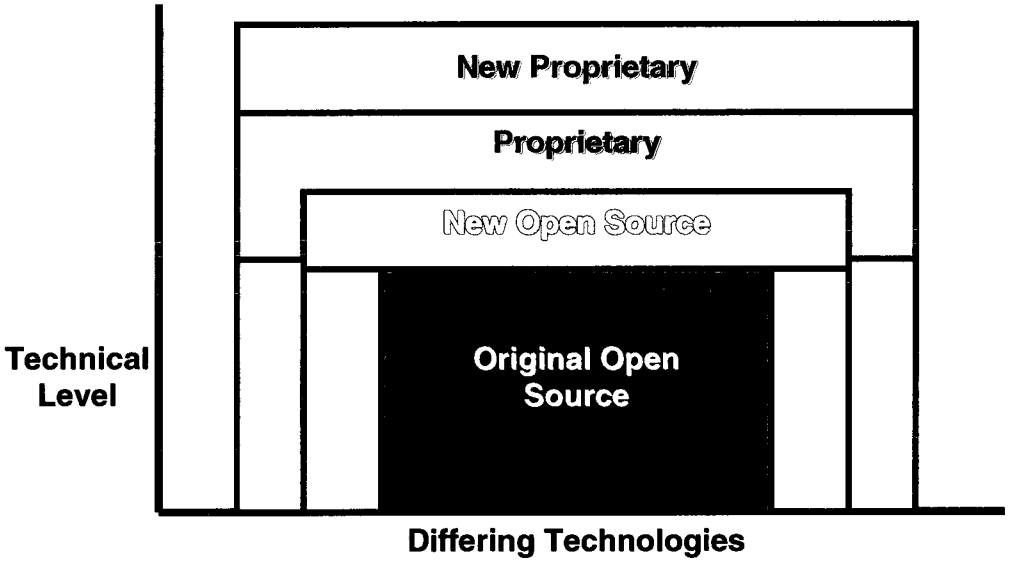


Figure 3: Innovation Effect of OSS (McDowell, 2007)

of OSS is said to normally start with commodity functionality (Milinkovich, 2008). When momentum is gained around a certain project, the technology covered by OSS will most likely grow and displace the proprietary technology, forcing the

vendors to be more innovative, and provide functionality that the OSS does not. Section 3.2.6 introduces two of the OSS M&S frameworks which have been successful thus far.

The source code has a significant role in OSS, but there are many ways way to contribute to open source projects and some do not even involve the source code. It was identified that for OSS to thrive, a high level of activity is paramount (Pokorny, 2008). This activity can range from using the product to contributing enhancements to the code or documentation, but Pokorny (2008) underlines that providing feedback in any way shape or form is of utmost importance. The feedback might include submitting bug reports, requests for new features or even participating on forums. Redistributing projects that are embedded within other projects is also considered a contribution. Finally, letting other people know about a project in any way is considered important.

Participation within a project has been classified by Charpentier et al (2004) to be passive when the code is simply used with no further involvement; or active when the user gets involved beyond using the code. This classification, seen in Figure 4, is further divided into developers and non-developers but missing is the power of having ambassadors publicizing a project as well as participation in forums. One OSS project manager suggests that he considers a project to be successful when a non-project team forum participant has answered questions on the forum for the first time. An interesting dimension presented in Figure 4 is that participation within an OSS project over time (represented by the

blue arrows) typically moves from passive towards more active, but rarely the other way.

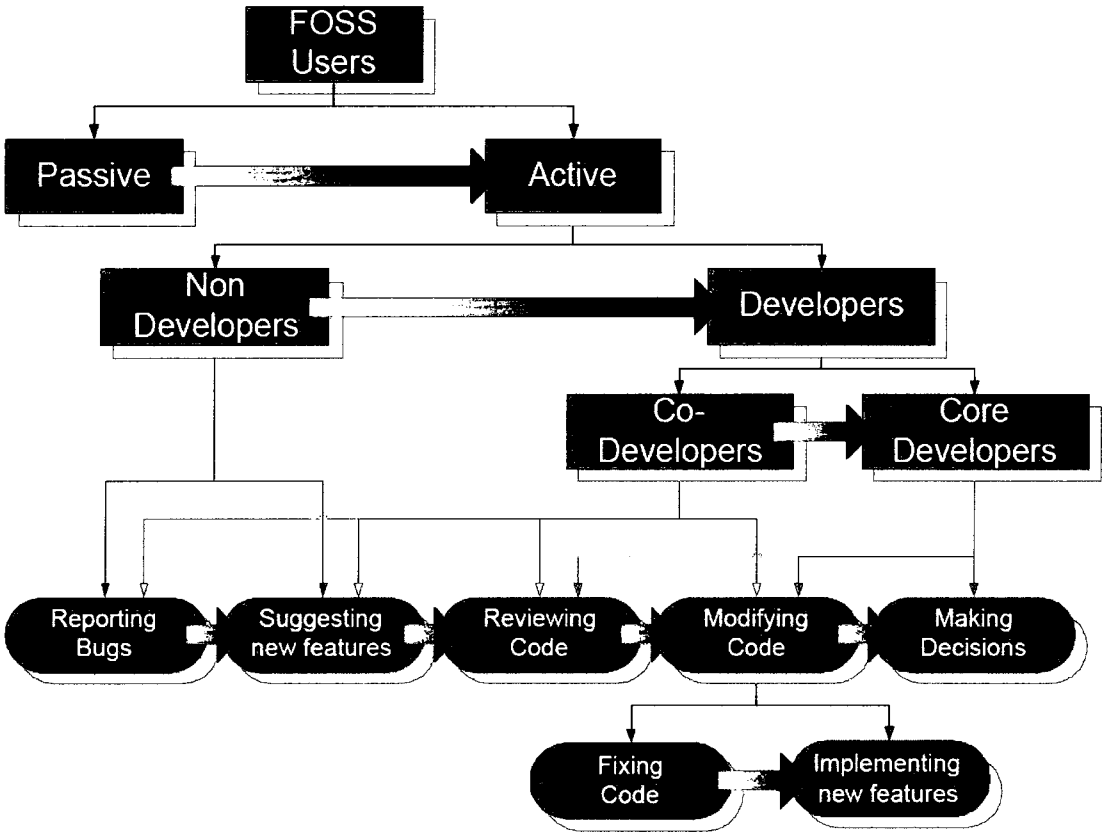


Figure 4: OSS Participation (Charpentier & Carbone, 2004)

3.1.2 Licensing and Release Issues

Open Source Licenses (OSL) introduce the concept of "copyleft" as a method of guaranteeing that free software remains free. This is a form of copyrighting, but adds distribution terms which give users the rights to use, modify and redistribute the products with the caveat that the same distribution

terms are passed along with the new products. Copyleft binds the products and the freedoms in such a way that they may not be separated. Like copyright, it is a general concept and only specific implementations can actually be used or enforced. The most commonly used copyleft implementation is the GNU General Public License (GPL) (FreeSoftwareFoundation, 2008).

Table 1: Open Source License Features

Properties / Licenses	Notice of modification	Redistribution of modified work	Patent peace provision	Third party IP rights	Distribute source code along with the object	Protect legal rights from anti-circumvention law	Linking to closed source code	GPL compatible	Choice of indemnification for charge of fee
CDDL	Y	Y	Y	Y	Y	N	Y	N	Y
MPL	Y	Y	Y	Y	Y	N	Y	N	Y
Apache License 2.0	Y	N	Y	N	N	N	Y	Y	Y
GPL	Y	N	Y	N	Y	Y	N	Y	Y
LGPL	Y	N	Y	N	Y	N	Y	Y	Y
CPL	N	Y	Y	N	N	N	N	N	Y
EPL	N	Y	Y	N	N	N	N	N	Y
MIT	N	Y	N	N	Y	N	Y	Y	N
Mod BSD	N	Y	N	N	N	N	Y	Y	N

The list of OSL available to the OSS community is quite vast and growing constantly. An official list of the current OSL reviewed and approved by the OSI can be found on their website (OSI, 2009), which is reviewed frequently and examined thoroughly (Beard & Kim, 2007; Bhattacharya & Suman, 2007; Cuéllar, 2005; Wu, 2007). Table 1 provides an overview of some of the characteristics of the most popular OSL. This table demonstrates that although some licenses support the basic the idea of freedom, there is a disparity between the different

licenses and the characteristics which they implement. Bhattacharya & Suman (2007) conclude that the oldest license (BSD) was not the most popular one (GPL), and that this most popular license was not found to be the most suitable for their field of study (Common Development and Distribution License: CDDL). While the GPL and LGPL seem to be the most common OSL in the M&S field, some argue that CDDL would provide a better alternative because of the problem of derived work (Gu, 2007). The basis for this argument is that CDDL addresses the concern of derived work by using the term modifications instead. It gives permission for people to include the CDDL covered item without having to release the source code, as long as no modifications to the core were made. The development of plugins or other utilities that link to and use the CDDL covered item are not affected. Migrating to CDDL may appear to be a good option, but may not be possible when working with an OSS project that uses a non-compatible license.

License tracking can be a significant problem when working with OSS projects. Some projects include multiple packages, with each having its own license, and care must be taken to ensure that the licenses do not infringe. This problem is compounded by the fact that some projects combine parts from different licenses to create their own "Frankenstein licenses" (HP, 2008). Tracking and respecting all of the OSL used in a project can become a complex and risky task.

3.1.3 Canada VS US Government Position

More and more governments such as the United States (Herz et al., 2006), United Kingdom (UK, 2009), Croatia (Croatia, 2008), and Iceland (Bjorgvinsson et al., 2007) are releasing formal policies on the use of open source technologies. These policies emphasize equality between OSS and proprietary software in order to prevent discrimination, but some mention that the government should avoid getting locked-in to specific vendors when possible. It is interesting to highlight the strong policy the Icelandic government has taken. They go so far as stating that "Software which is built and financed by public agencies ... must be open source" (Bjorgvinsson et al., 2007). In this section a basic comparison of positions taken toward OSS between the Canadian and the United States Governments is presented.

The Government of Canada (GoC) released their first official OSS position in 2004. The major statement in the position is that OSS is already included in their infrastructure (GoC, 2004). This statement is geared toward the Information Technology (IT) domain, where OSS has been somewhat accepted and used (Chernysh, 2007). Following the release, GoC mandated the Defence Research and Development Canada (DRDC) to conduct a study on OSS and propose a way ahead. This study generated what will be referred to as the Charpentier report (Charpentier et al., 2004). This report concludes that: "even though OSS is not a panacea, it does offer concrete and credible technological opportunities ... and should be considered equally when it comes to contractual issues". They

have identified that OSS has been embraced by the education and health sectors within the GoC, but most other departments are still apprehensive toward OSS. They conclude that Canada is slow in OSS adoption, especially in the public sector, and attribute this to both a lack of clear business cases and the underestimation of the strategic value of OSS. The Charpentier report's recommendations for the way ahead with respect to OSS are summarized in Table 2. Since the publication of the report, the only documented action addressing the recommendation to promote OSS involves a series of lectures, primarily given to the academia. There has been little progress on the recommendation to consider OSS in contractual work since the procurement protocols established and maintained by Public Works are not conducive to participation by the OSS community. It is just at the time of the writing of this document that Public Works has put out a call for information on OSS (Chung, 2009; Franco, 2009). There does not appear to be any evidence of activity addressing the recommendations for support and demonstration of OSS. The recommendation to adopt OSS progressively has been undertaken in certain branches of the GoC (Chernysh, 2007; Chung, 2009). Although the Charpentier

Table 2: DRDC Proposed Way Ahead for OSS (Charpentier et al., 2004)

1	Promote OSS by means of publications, workshops and conferences
2	Consider OSS-based solutions in contractual work when they are technically competitive with other development strategies
3	Support GoC departments in assessing this emerging technology
4	R&D communities should demonstrate leadership in OSS adoption
5	Adopt OSS progressively (see Figure 4 for the suggested adoption method)

report was mandated by GoC, the suggested way ahead was not implemented significantly and the GoC's OSS position has not been modified.

Table 3: DRDC Reported OSS Advantages (Charpentier et al., 2004)

Advantages	Reasoning and conclusions
Greatly eases security enforcement	To increase the reliability and security of code, it is essential to use some complementary mechanisms such as peer review, testing, quality audits, alpha and beta versioning etc. OSS offers the very significant advantage of keeping access to source code. This encourages more peer reviews, testing, and quality audits by a much larger community of users/developers than what would be possible with proprietary code.
Leaner and meaner	"Leaner and meaner" software systems than COTS equivalents that often suffer from feature bloating. Since they are smaller, open source systems are expected to provide fewer opportunities for exploits.
Ease of modification	Source code can be enriched with assertions, complementary safety checks etc.
Less susceptible to cyber attacks	Increased code diversity in the "software ecosystem" that could reduce the speed and the proliferation of cyber attacks.

The Charpentier report also underlines some advantages which OSS can provide, but again, these are geared toward IT and no mention is made of M&S. Table 3 lists these advantages and they suggest that the report focussed on software security as it does not mention other software development areas. The R&D community makes use of M&S in many of their studies and these often culminate in Technology Demonstration Projects (TDP). So far, there has been little evidence of any TDP, be it inside or outside the field of M&S, formally targeting the employment of OSS by the government. Most uses of OSS within the R&D community are at the lower IT level, such as the use of Linux, and not higher level of frameworks. TDPs still largely make use of proprietary

Commercial off the Shelf (COTS) packages, which implies that the Charpentier report has not been accepted by the R&D community.

In the broader field of M&S in the Department of National Defence (DND), there has been little acceptance of OSS. The only documented M&S project making use of OSS is the RAPTOR simulator (CogSim_Technologies, 2006), which was developed to evaluate the state of CIGI (discussed later). The construction of this simulation framework makes use of many OSS projects, but the use of OSS was not mandated by the study. The result itself (somewhat comparable to OpenEaagles which is discussed later) was not released as OSS and is kept as proprietary intellectual property of the crown.

The US government has been much more proactive than the GoC in considering the potentials of OSS. The earliest signs of research into applying OSS within the Department of Defence (DoD) is a 1999 Masters thesis (Seiferth, 1999) conducted at the Air Command and Staff College (which is part of DoD). The research points to the potential for the US Government to move towards an open source business model and stresses that DoD can realize significant gains, both technical and monetary, by the formal adoption, support and use of open licensing systems. Soon after, DoD established an annual Open Technology Conference (AFEI, 2009) with the goal of facilitating the deployment of open technologies and architectures within military systems. The opening keynote of the 2007 conference highlighted that the "Operational utility of Open Source Software continues to be demonstrated in the current fight... when we rolled into

Baghdad, we did it using open source" (Justice, 2007). During the 2008 conference it was noted that DoD is increasingly looking at the open model as a potential way to achieve pieces of its net-centric vision (AFEI, 2009).

DoD released a formal roadmap for Open Technology Development (OTD) (Herz et al., 2006). This document identifies OSS as enabling enormous gains in productivity and efficiency. As well, it highlights that OSS provides many advantages which range from security, to development time, and potential cost effectiveness. Table 4 lists the suggested advantages. Most of these advantages were also confirmed by another independent report on OSS in enterprises (Golden, 2008). The roadmap prescribes that "*the government will need to embrace OTD, integrate it into formal acquisition directives and policies and enforce its application through appropriate procedures and review processes*".

There is evidence that the OTD is being taken seriously within DoD, and non-policy implementation of the OTD is present in many elements. Several successful projects have been released as OSS, such as OpenEaagles, SubrScene and Delta3D (which are discussed in section 3.2) As well, the adoption of other OSS protocols such as CIGI is prominent. DoD has gone even further and established the *Forge.mil* (DISA, 2009) as a collaborative software development environment similar to the well known OSS *SourceForge.net*. This type of infrastructure facilitates the sharing of projects and their source code and is cornerstone to the growth of OSS. Despite these non-policy OTD successes,

Table 4: US OTD OSS Advantages (Herz et al., 2006)

Advantages	Reasoning and conclusions
Encourages software re-use	OSS development allows programmers to cooperate freely with other programmers across time and distance with a minimum of legal friction. Rather than endlessly reinventing wheels, a programmer can just copy someone else's elegant tire from another machine.
Increase code quality and security	With closed source software, it's often difficult to evaluate the quality and security of the code. In addition, closed source software companies have an incentive to delay announcing security flaws or bugs in their product. Often this means that their customers don't learn of security flaws until weeks or months after the security exploit was known internally.
Subject to scrutiny by many eyes	Bugs, security flaws, and poor design cannot hide for long when the software has a community of programmers to support it. Since fixing the code doesn't depend on a single vendor, patches are often distributed much more rapidly than patches to closed source software.
Decreases vendor lock-in	No more paying a vendor for a needless upgrade, simply to maintain compatibility with others using the same software. Business data is also more "future-proof", since most open source programs save text files in ANSI standard ASCII files, instead of proprietary binary formats. If the vendors training materials are inadequate, because they have access to the source code, external vendors can supply as good or better manuals.
Reduces cost of acquisition	Most OSS is available for a nominal cost, often the price of the media, or the time of the download. No more "per-seat" license fees. This means that start-ups don't have to part with precious capital when they need it most. Established companies can try the software with minimal risks. Companies wanting to develop a piece of software that they don't plan to use to differentiate them, they can reduce the cost by collaborating with several companies on the same code base. As well there is no license fee.
Increases customizability	Every organization has unique needs or desires. Linux has been ported to everything from embedded microcontrollers, to IBM mainframes. If there's a nagging bug you want fixed, you can hire someone else to fix it. If two programs don't play well together, one or both can be modified to eliminate the incompatibility.
Meritocratic community	In the open source community, a programmer's status and fame depends on programming skill. This often drives them to deliver better products.
Enhances agility	Agility of IT industries to more rapidly adapt and change to users needed capability in improved by OSS.
Increase competition	OSS strengthens the industrial base by not protecting industry from competition. Makes industry more likely to compete on ideas and execution versus product lock-in.
Helps secure infrastructure	Enables DoD to secure the infrastructure and increase security by understanding what is actually in the source code of software installed in DoD networks.
Shorter response time	Rapidly respond to adversary actions as well as rapid changes in the technology industrial base.

policies mandating the use of open technologies were only found for the US Navy, which states that "OSS has become the cornerstone approach to competition and innovation in the development of information tools (Mullen, 2006)".

A comparison of the OSS positions described above reveals that the US has taken a more aggressive stance than Canada. The US DoD has clearly learned from the OTD as is now stressing the importance of OSS and recognising it as a mature and viable business model. Even though the Charpentier report made similar suggestions to the GoC, evidence of its usage in the M&S field are sparse at best, and there has not been any systems or frameworks released from DND as OSS. Furthermore, comparing Table 3 and Table 4 highlights that the US OTD roadmap has taken a much broader technical emphasis than the Charpentier report. This may have led to the greater acceptance of OSS in the US Government.

3.1.4 SISO Position

The Simulation Interoperability Standards Organization (SISO) was established in 1997 and is the main body for creating and promoting interoperability standards in the M&S community (SISO, 2008a). SISO is recognized as a Standards Development Organization (SDO) by North Atlantic Treaty Organization (NATO), and as a Standards Sponsor by IEEE (Miller, 2008). SISO is in charge of maintaining two major simulation interoperability standards on behalf of the IEEE: Distributed Interactive Simulation (DIS) and the

High Level Architecture (HLA) for Modeling and Simulation. SISO also develops and maintains additional simulation standards which have not been adopted by the IEEE. SISO and its affiliated organisations hold many conferences each year that enable and support M&S standards.

As the international representative body for simulation standards and interoperability, SISO has been struggling with the concept of OSS for quite some time. SISO first began discussing the potential use of OSS while developing the HLA standard in the late 1990's. At that time, a paper presented to SISO identified OSS as having the following shortcomings (Givens, 2000):

- Not many people use open-source software;
- It's hard to get reliable support for open-source software;
- Closed source helps protect against security attacks; and
- Open-source opposes intellectual property rights.

Time has quickly proven these theories wrong and, shortly after, it was identified that SISO should "encourage its members to be involved in the development of software implementing the studied standards including, *potentially*, OSS" (Givens, 2001). However it was felt by some that developing or even encouraging OSS would stifle technological advancement and would only degrade the quality of the software produced (Givens, 2001; Katz, 2001). Most of the concerns against OSS came from companies with proprietary products supporting the HLA, which could explain why OSS has been slow to gain traction within SISO.

It was reported that there had been some interest expressed in launching OSS RTIs (Pearce & Farid, 2004) but not many have reached commercial use. This highlights the fact that the lack of efforts surrounding OSS and the HLA raises the barrier to entry for the domain and restricts those not in a financial position to consider commercial alternatives (Pokorny, 2006).

The first real appearance of OSS within SISO was with the establishment of the CIGI Study Group in 2004 (SISO, 2008a). This was followed by the release of an OSS implementation of the DIS standard protocol named OpenDIS (MOVES, 2009) in 2006 and by the establishment of the Open Source Initiative for Parallel and Distributed Modeling (OSI-PDMS) Study Group (Steinman, 2009).

SISO does not have a strong direction with regards to OSS. Despite holding its first formal session on OSS in 2008 (SISO, 2008b) it is still struggling to include an official position on OSS within the SISO Vision Statement. The draft statement "SISO understands and appreciates the contribution Open Source Software has made to the efforts of many other standards organizations and as such, SISO actively encourages the involvement of the Open Source community in supporting the development and implementation of SISO standards" (Hill, 2008) is still under debate and is not released officially.

3.2 State of the Art for EOIR Sensor Simulation Training

The technologies required for the engineering of a simulation-based sensor training environment are presented in this section. They are applicable to

many simulation systems, and their scope should not be limited to the context of this discussion. The post-processing technique used to emulate an EOIR sensor is presented first, followed by an introduction to CIGI which has become the de-facto communication standard for use in commercial image generators within the field of M&S. An analysis of the CIGI and sensor support provided by a selection of proprietary image generator solutions is then given. This is followed by a discussion of some of open source counterparts and some commercial solutions built using these OSS technologies. Finally, some of the different open source frameworks that could be used to create simulation training systems are presented.

3.2.1 EOIR Processing Pipeline

In order to provide a fully realistic representation of the sensor display, the simulated operator's view of the scene has to approach what is seen on the real equipment. The implementation of the processing described in this section was originally done with special hardware; therefore EOIR simulations were only available for very specific and expensive commercial applications. The advent of powerful GPUs allowed similar results to be achieved on PCs; this has propelled the implementation of sensor effects into the main stream (Harrison, 2003).

A simplified diagram showing what is needed to achieve such EOIR sensor simulation is presented in Figure 5. The figure highlights the need for three major components: an input Scene; the EOIR Sensor processing; and one or more output Displays depending on the system being emulated. The rendered

scene is taken from the Frame Buffer (recall the OpenGL pipeline in Figure 1) and is then processed again (i.e. post-processed) using the RTT technique. This EOIR post-processing pipeline can be divided into three main components: sensor controls; effects which are common to all electro-optical sensors; and finally, effects which are specific to infrared sensors. The result of the EOIR sensor post-processing is then made available for display.

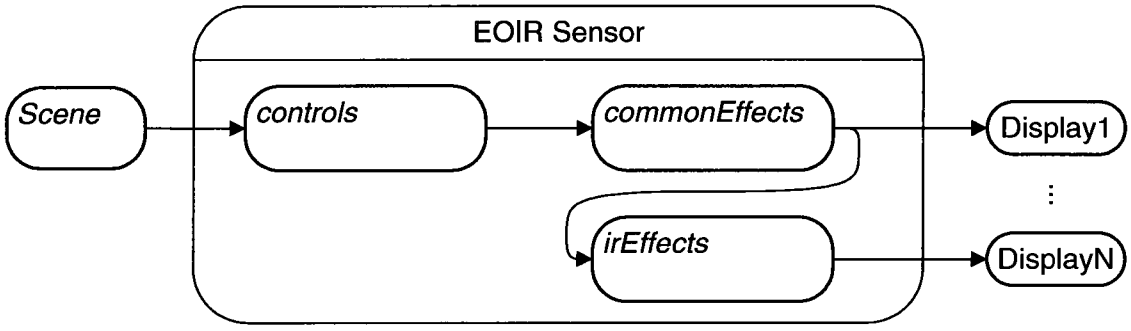


Figure 5: Simplified EOIR Pipeline

A typical Scene is constructed of entities, the environment, as well as a terrain and terrain features such as buildings. The elements needed to emulate scene environmental effects include stars, sun, moon, clouds, rain, fog, snow, and atmosphere. Of note is that environmental effects can have different impacts depending on the optics of the sensor being modeled. For example, the humidity content in the atmosphere, such as clouds or fog, will absorb different portion of the electromagnetic spectrum in different ways which may in turn influence sensors that are tuned to specific frequencies.

Common sensor controls include gain, level and Depth of Field (DOF) (Smith, Murray-Smith, & Hickman, 2007; Sugisaka & Faizal, 2006). Gain and

level are used to enhance the contrast of the image whereas DOF is used to control the focus of the lens.

Common sensor effects include jitter, blurring, High Dynamic Ranging (HDR), scintillation, noise and dead element (Blasband, Bleak, & Schultz, 2004; Liao & Hennessey, 2006; Smith et al., 2007; Sugisaka et al., 2006). Jitter is the effect used to represent vibration in the sensor platform. Blurring is the general inability of the sensor to focus even when adjusting DOF. HDR is a combination of image processing techniques aimed at accurately representing the wide array of intensity levels found in real scenes, ranging from direct sunlight to shadows. Scintillation is caused by a light source travelling through a turbulent atmosphere and can be observed as the twinkling of the stars. Noise presents itself has a random distribution of abnormal variation on the image. Lastly, a dead element has the effect of either a single black pixel on the screen or a complete black line across the screen depending on the sensor. The pipeline described to this point encapsulates the required components to the display associated with an electro-optic sensor but lacks some functionalities need for infrared representation.

The IR sensor specific effects have been described as blooming, persistence, and polarity (Bhatia & Lacy, 1999; Blasband et al., 2004; Liao et al., 2006). Blooming is an effect where portions of the sensor's internal array of cells used to capture data become electrically saturated and some charge spills over into neighbouring cells; this is often observed as a halo around a small intense light source. Persistence arises from the inability of the internal cells to

discharge completely before the next capture pass, and results in the mixing of previously captured images with the new ones. IR sensors also have a unique polarity control where hot items can be displayed in white (WhiteHot) or black (BlackHot). Plate 1 shows a WhiteHot (on the left) and a BlackHot (on the right) image of a frigate.

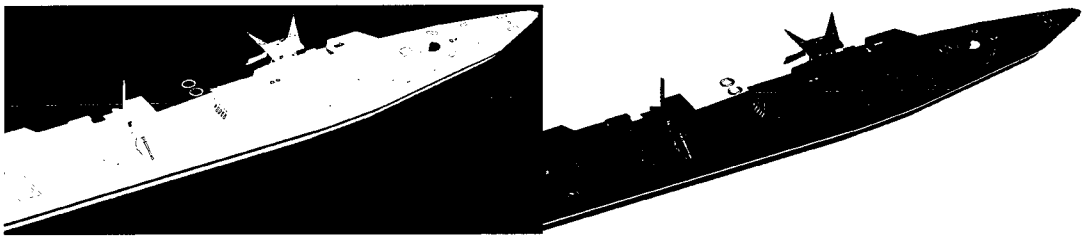


Plate 1: Frigate IR WhiteHot / BlackHot

3.2.2 Common Image Generator Interface

CIGI is an open-standard data interface between a host device (Host) and an Image Generator (IG). A brief historical overview of the emergence of this standard is given followed by an introduction to how this standard is used to control an IG. A highlight of some technical, as well as financial, advantages of CIGI is then presented. This section closes by discussing CIGI's acceptance within the US and Canadian Governments.

CIGI was developed with the intent to lower the burden and the cost of integrating a simulation Host with an IG (Phelps, 2007). Before its release by Boeing under GPL in 2002, IG protocols used in the field of simulation-based training were proprietary and closed in nature (Lechner & Phelps, 2002). The simulations either had to be tightly coupled with the IG, or the distribution of the

visualization work-load to multiple computers used proprietary methods. Most simulation or gaming frameworks such as the Microsoft Flight Simulator (Microsoft, 2009) historically have had the IG as an integrated part of the application. On the other hand, the distributed separation of the Host and IG has been used by large proprietary commercial simulation products such as the ones used by airlines for pilot training.

After the public release of CIGI, a Standing Study Group (SSG) was established within SISO to evaluate industry and government interest in developing a standard image generator interface. This was done to form a consortium of companies with a review board to make CIGI applicable to a greater number of potential users (Durham & Phelps, 2003) and to establish a stronger ecosystem. Many large simulation companies were interested in the standard and, by joining the informal consortium, helped shape the road ahead for CIGI. CIGI has now become the de-facto standard for distributed Host-IG implementations.

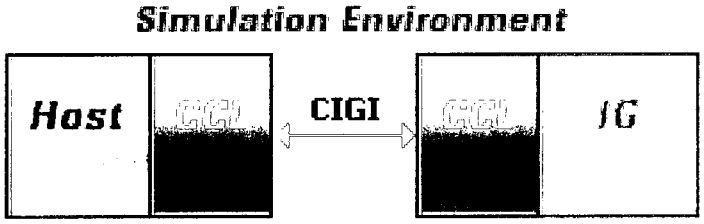


Figure 6: CIGI Diagram (Durham, 2006)

The CIGI project (CIGI, 2008) provides OSS implementations of the CIGI Class Library (CCL), as well as a Host and IG which can be used as surrogates if needed. The CCL implements the CIGI standard and must be incorporated independently by the Host and the IG, as shown in Figure 6, in order to be considered compliant with the standard. The CCL provides an abstraction from the network protocol to be used as well as a common logical, simulation-oriented, packet structure to be used between the Host and IG (Durham, 2006).

A CIGI connection can be used to control multiple views of the world which can be grouped together logically. This feature is used when implementing an immersive simulation where multiple display devices are used to cover as much of the operator's vision as possible. Each view or group of views is assigned a parent entity from which all controlling references are made. Only positional information for the parent needs to be passed, the views are synchronized automatically and only directional information is required. For example, to cover 120 degrees using 3 views: view1 is oriented with the entity centerline observing a 40 degrees angle; view2 is oriented 40 degrees right, and view3 40 degrees left with the same observing angle. This parent-child relationship is supported by most of the CIGI architecture whereas entities can be assigned a parent so that, for example, a missile can be 'connected' with the airplane until it is fired. Until the firing even, only the airplane needs to be positioned and the missile is effectively considered part of the airplane even though its 3D model is contained in a separate file.

Using CIGI provides many technical advantages. The parent-child paradigm effectively minimizes network traffic while maximizing the capabilities supported. As well the workload separation enables the handling of very large and high fidelity visual databases (terrain and entities). Another significant advantage of CIGI is that different CIGI conformant IG can be used with little or no change to the Host (and vice-versa), effectively providing developers and clients with greater flexibility.

CIGI has financial advantages as well. The Boeing Company spent approximately one million dollars developing and maintaining CIGI, and estimates savings in the order of five hundred thousand dollars for each new trainer they implement (Phelps, 2009). Since the number of trainers being developed by Boeing is significant, the company affirms that the spending has been recouped many times over (Phelps, 2009; Walker, 2005b). The establishment of CIGI as an open source standard was even welcomed by large commercial IG implementers. For example, Evans & Sutherland noticed that having implemented the CIGI interface opened up the potential for their systems to be used in more programs (Walker, 2005a), effectively increasing their target market and potential profit.

In comparison to the Canadian DND, the US DoD has taken a stronger stance with regards to the use of CIGI. The US Navy has declared CIGI a requirement in its Navy Aviation Simulation Master Plan (NASMP) for all distributed visual system implementations (Glass, Paterson, Andrews, Griffin, &

Seagull, 2002). In Canada, the Defence Administrative Order and Directive (DAOD) on M&S states that the CF should strive to use common, open architectures as well as enduring standards in order to the maximum extent possible in order to minimize effort/cost and maximize interoperability (VCDS, 2006), but it does not specify which standards shall be used. The Canadian Forces Air Warfare Center in charge of the Air Force Synthetic Environment Coordination Office (Air SECO) contracted a prototypal study of CIGI which concluded in 2006 with the following proposition: "the CIGI protocol has been shown to be a mature standard, capable of supporting the needs of the Canadian Air Force in future simulation-based projects" (CogSim_Technologies, 2006). As a result, the Air SECO informally declared it as the way ahead for implementations of new training systems (Johnsrude, 2009) but this has not been transposed into an official document (DND, 2008).

3.2.3 Proprietary IG Solutions

Many commercial IG solutions exist (TSJ, 2008; Witte, 2008), and the following summary is based on the comparison presented in Appendix E: IG Vendor Table. The comparison reveals that many commercial IGs support CIGI, but not all of them implement or provide the possibility of generating credible sensor simulation. For the products stating that CIGI is supported, it was most often found that the version as well as the list of functionality supported was not easily available. This could lead to some confusion where a solution may provide sensor simulation, but only support the basic OTW CIGI functions. A

trend was also noted for most IG systems to support OpenFlight (MultiGen-Paradigm, 2008) and TerraPage (Presagis, 2009) terrain and entity databases format. Finally, it was found that many commercial IG solutions use Open Scene Graph (OSG) which is discussed in the next section.

3.2.4 Open Source IG Solutions

The Open Scene Graph is the most prominent OSS building block for IG in simulation and is presented first in this section. This is followed by a discussion on two of the most prominent OSS IG solutions: SubrScene and MPV. This section is followed by an introduction to some commercial sensor implementations using OSG in section 3.2.5.

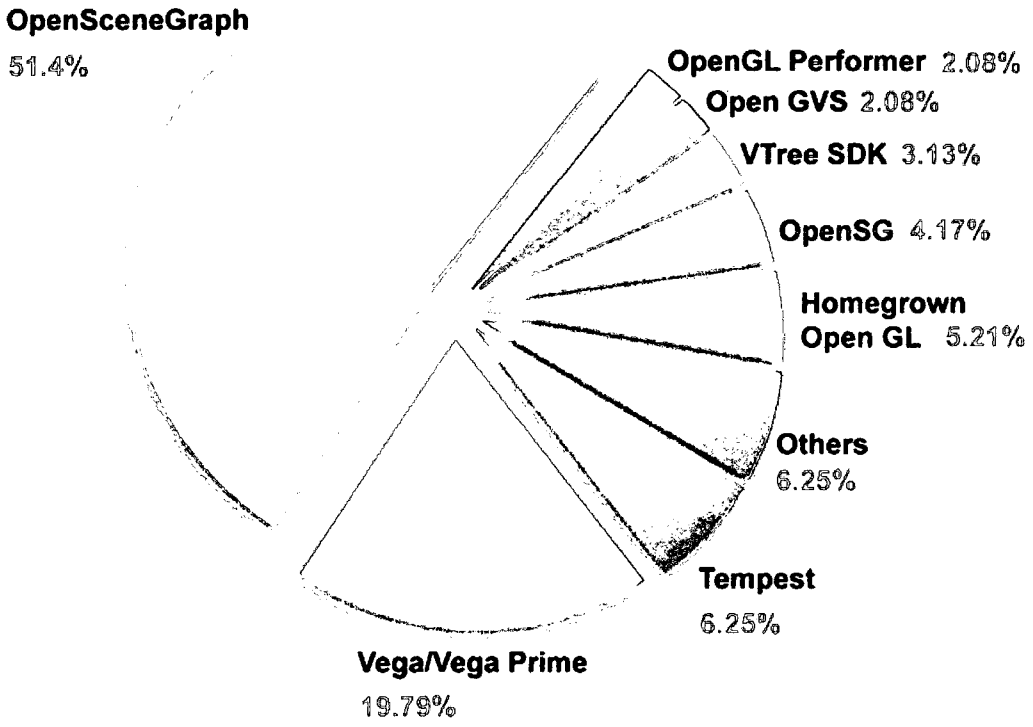


Figure 7: MODSIM Poll on Prefered IG System (MODSIM.org, 2005)

OSG is a high performance 3D graphics toolkit used by many OSS as well as proprietary commercial projects. This project has its own OSGPL licence which closely resembles LGPL and permits usage in a proprietary commercial solution. It was recognized as the most commonly used IG system by the News Portal for Visual Simulation and Training Industry Professionals capturing 51.4% of the reported usage in a 2005 poll, and the use of OSG was reportedly growing (MODSIM.org, 2005). The pollsters note that it is sometimes difficult to identify whether a commercial IG uses OSG, as this information is normally not disclosed on their websites. Figure 7 shows that OSG not only held a majority, but dominated as the second closest IG held less than 20% of the market.

The theory behind this toolkit is that all items to be rendered are stored in a tree like architecture, called a SceneGraph. A SceneGraph is a directed acyclic graph, so it establishes a hierarchical relationship among all the nodes (Barros, 2005). OSG provides an extensive API but some of the most basic components are the *Group*, *Geode*, and *Drawable* classes. A *Group* is a logical container node used to amalgamate other objects. The *Geode* class is a geometric node representing a specific type of container which can be assigned an object to be rendered. *Drawables* are the highest level abstraction for all things which can be rendered and are therefore attached to a *Geode* within a graph. An example of an OSG SceneGraph depicting a *Scene* containing a *Building*, two *Tanks* and an *Airplane* can be seen in Figure 8, this figure also shows that one object (the side *Insignia*) can be referenced multiple times by

adding its reference to a common *Geode*. This object-oriented framework also provides an abstraction from OpenGL and helps the developers keep a high-level programming interface while being able to do complex graphics manipulations on each component independently.

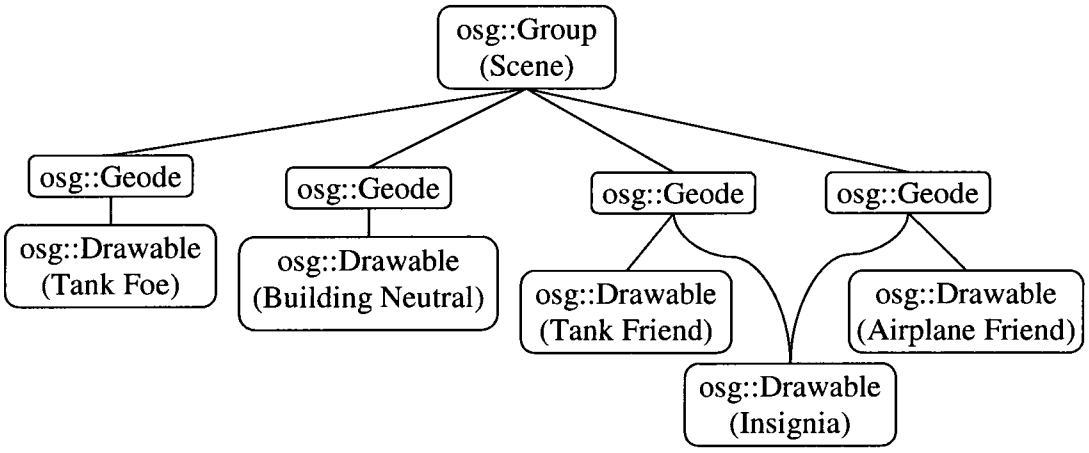


Figure 8: OSG SceneGraph Example

SubrScene is an OSS simulation visualization toolkit originally developed to support flight simulation for the US Air Force Research Laboratory (Subr, 2007). Though originally proprietary to the US Air Force, it was released to the public domain in April 2006 (Subr, 2009). Its framework is built around a client-server architecture, and is meant to be used for distributed display software, similarly to CIGI. It was designed for the server to be tightly coupled with the host and embedded within a simulation application, but was later expanded to support CIGI. The visual depiction of the architecture presented in Figure 9 shows that a duplication of network occurs when using CIGI. One network is needed for the Host to communicate with the server (*ssControl*), while the other

is used by *ssControl* to communicate with the client (*ssClient*) (IG) via another network connection. In contrast, most CIGI compliant Host-IG architectures would not include a controller such as *ssControl*, and the Host would communicate with the client IGs. SubrScene's *ssControl* interface provides functionalities similar to CIGI's CCL. SubrScene currently implements the original CIGI SDK/API and there is no indication of a move to CCL in the near future which limits the functionalities that could be implemented.

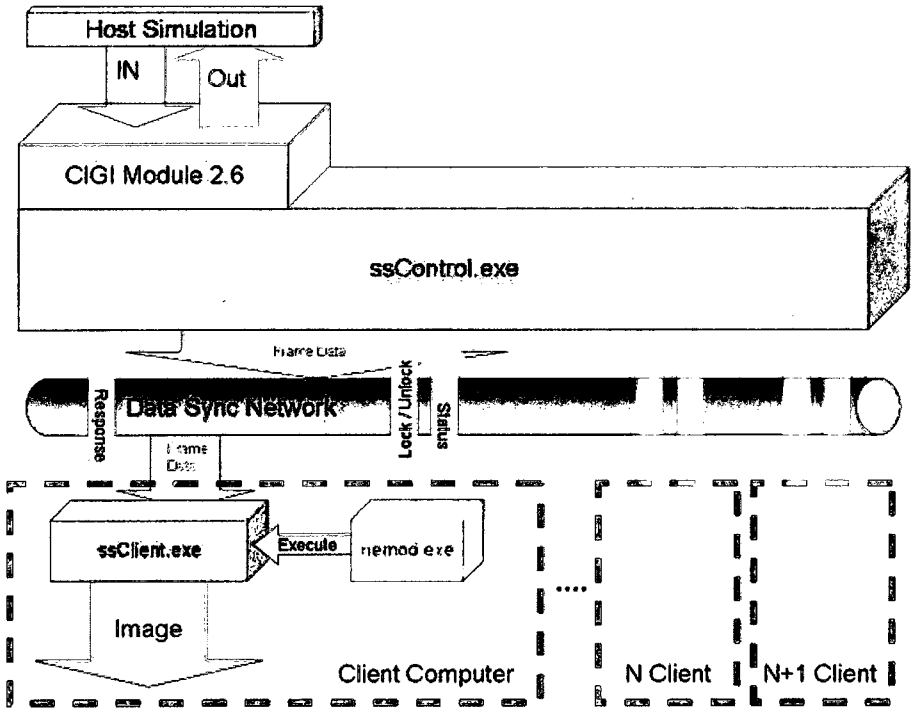


Figure 9: SubrScene Data Flow (Subr, 2007)

The Multi Purpose Viewer (MPV) is an OSS IG developed by The Boeing Company. MPV was developed for internal use but was also released as a free tool for developers to test their Host compliance to CIGI when an IG is not

available (Sampson, 2005). This project was release publicly in 2002 under GPL and is now supported by a strong community. MPV is based on OSG and implements a large subset of the CIGI packets. Support for more packets is expected to grow with the needs of the community. Figure 10 shows the plugin architecture used by this project, which lends itself nicely to such small additions or incremental modifications. The *Kernel* is responsible for loading and initializing the *Plugins*, as well as providing basic services such as networking and timing. The different *Plugins* interacts with each other using a *Blackboard* system where a *Plugin* can post information to be shared with others. One limitation of this architecture at this time is that there is no way to control information access. Information posted to the *Blackboard* can be accessed,

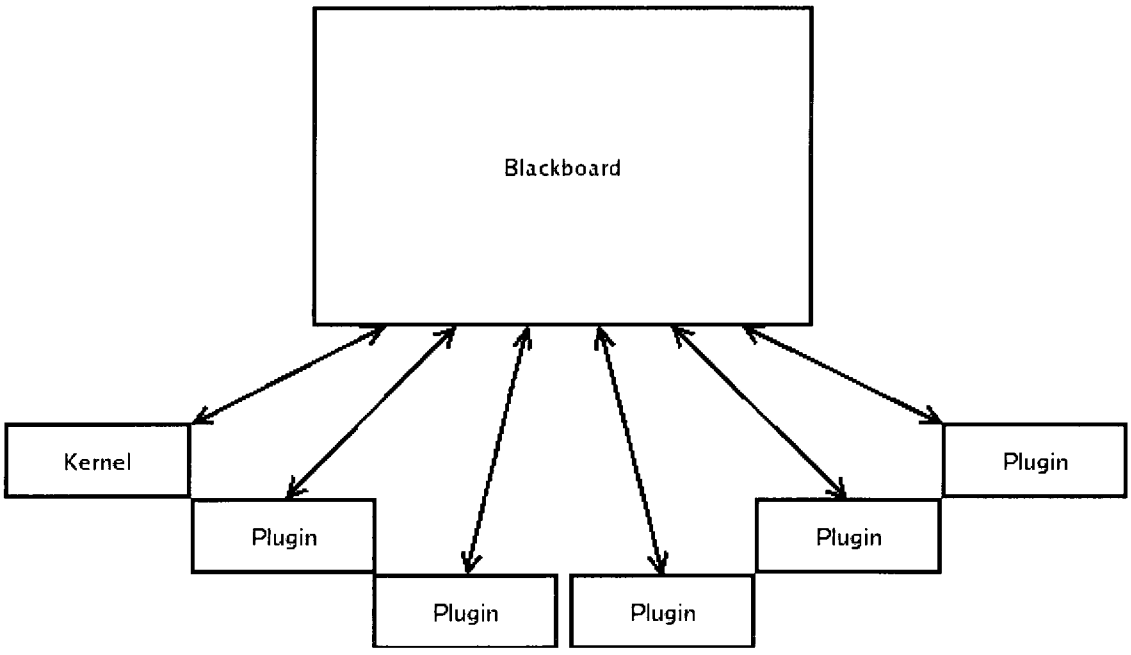


Figure 10: MPV Data Flow (Sampson, 2005)

modified, or even deleted by any other *Plugin* without restriction. A description of the functionality and interaction of each *Plugin* can be found in the Developer's Guide (Boeing, 2006). At the time of this research, MPV did not support multi-threading but work was being conducted by the Boeing team to add this feature.

3.2.5 Open Source Based Proprietary Sensor IG Solutions

The two most prominent commercial products which provide simulation practitioner with complete frameworks to develop Electro-Optic and Infrared (EOIR) simulations are from Oktal-SE (Oktal-SE, 2008) and JRM Technologies (JRM_Technologies, 2005). Both of these companies created commercial products based on OSG and make use of shaders to achieve their impressive results. This section discusses and introduces these two proprietary solutions.

Oktal-SE developed the SE-Workbench (shown in Figure 11) which provides a full set of tools to be used in the modeling and simulation of multi-spectral sensors at the physical level for real-time applications, as well as fully validated non real-time simulations for sensor engineering. This set of tools can be used to classify and pre-process 3D models and terrain databases to create complete representations of sensor effects. This is done by assigning thermal properties to certain materials as well as using complex thermal propagation and interference equations (Latger, Cathal, Joly, & Goff, 2008). The SE-FAST-IR portion of the toolkit provides for real-time rendering of sensor effects to be used in training applications. The tools on the left side of Figure 11 are used to pre-process the models, the atmosphere, and the terrain in order to assign multi-

spectral thermal characteristics and behaviour to each item used in the simulation. Different ranges of the electromagnetic spectrum can be assigned special processing characteristics by the modeller. This pre-processing is then recompiled to merge all the information into a new proprietary database format.

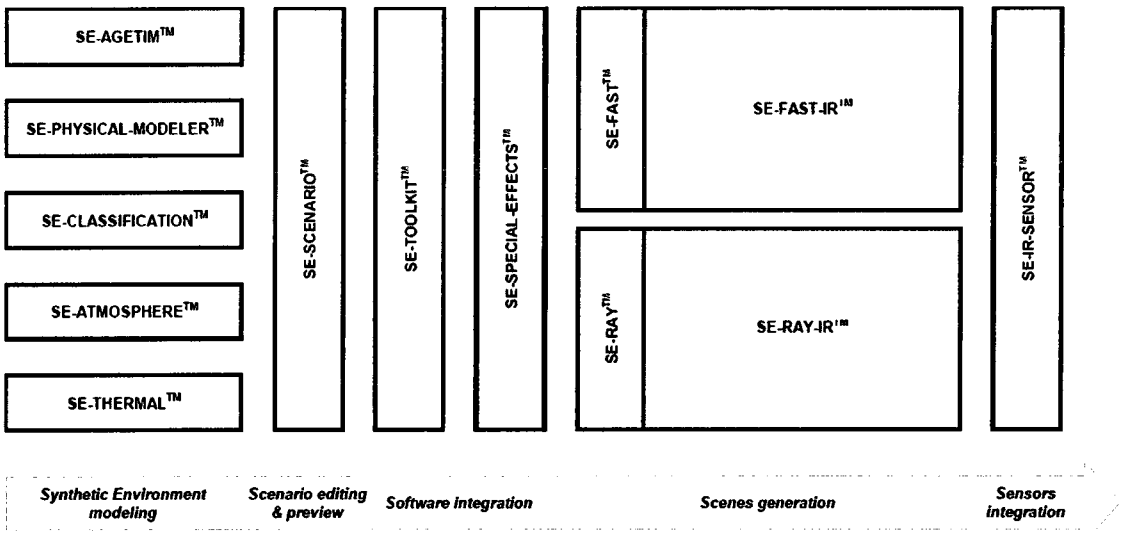


Figure 11: Oktal SE-Workbench (Oktal-SE, 2008)

After this pre-process, there is no way to modify these characteristics and behaviours from outside the framework, but they can be exported to a standard format such as Open-Flight and thereby used by other frameworks or simulation environments. The tools on the right side of the figure are used to model and represent the behaviour of the sensor, such as motion and mode control. The IG provided by Oktal-SE as well as their sensor implementations are CIGI, HLA and UDP compatible, providing the possibility to interface with other simulations or to change the Host simulation. However, a DIS interface has not been implemented as it is not used by their clients so far. Having high fidelity physics

ray tracing tools (non real-time) as well as OpenGL graphics tools (real-time) all within the same framework provides the possibility for validation and cross-comparison of the results in a semi-automated way (Joly, Goff, Latger, Cathala, & Larive, 2006). This validation process is a major differentiator from other products.

JRM Technologies also specialises in advanced sensor modeling and simulation. Their focus is on predictive physics and sensors in real-time training simulators, and they do not provide a sensor engineering approach similar to Oktal. They provide four OSG based libraries which can be used to generate Visual, EO or IR scene rendering. The SigSim, SenSim, Asgard, and Material libraries do not provide an IG per say but can be used to complement a SceneGraph based IG as follows(JRM_Technologies, 2005, 2008a, b, c):

SigSim: An advanced signature synthesis and atmospheric propagation run-time library for radiometrically-correct sensor displays and Out-the-Window (OTW) visuals.

SenSim: An advanced sensor modeling toolkit and run-time library for real-time sensor effects simulation of any optical sensor in the EO or IR pass-band.

Asgard: An advanced physics-based spectral scene generation software package. Its highly realistic image output is ideal for advanced training system applications and tactical sensor studies.

Material Libraries: A library that contains mesh color matching with thermal information providing a semi-autonomous thermal behaviour for terrain and buildings. JRM is currently working on releasing a GUI tool for material classification application for remote sensed data name Genesis. This tool will provide future users with the possibility to create their own material library based on actual sensor source imaging data.

The tools developed by JRM Technologies are found to be used in multiple commercial IG supporting sensors, as shown in Appendix E: IG Vendor Table, and have also been previously used for at least one of DRDC's TDP. On their own, these libraries do not provide an IG or a direct support for CIGI, but if integrated with a CIGI compliant IG, the work required to implement CIGI sensor control would be relatively minor.

3.2.6 Open Source Simulation Engine Frameworks

The emergence of OSS M&S application frameworks was first introduced in section 3.1.1 and the detail of two of the most prominent ones, OpenEaagles and Delta3D, are provided.

The Open Extensible Architecture for the Analysis and Generation of Linked Simulations (OpenEaagles) is an OSS subset of the fully-featured Eaagles simulation framework developed and used by the US Air Force Simulation and Analysis Facility (SIMAF) for more than 10 years (Hodson, 2008; Hodson, Gehl, & Baldwin, 2006b). The OSS version of the framework was released to the public domain in July 2006 under the LGPL and has since been

used by the Mesa US Air Force Research Laboratory (AFRL), universities, and other simulation practitioners. The OpenEaagles broker mentioned that many of the users of the framework desire to maintain anonymity because of the classified nature of the projects and this makes usage tracking quite complex (Hodson, 2009). Nevertheless, in an eight months period the framework alone was downloaded an impressive 4000 times (SourceForge, 2009). This is not including downloads of the developmental copy for which no statistical data is gathered. The OpenEaagles core team consists of five experienced software developers employed by the AFRL, which maintain and administer the framework as well as incorporate changes from the community into the proprietary version used with the US Air Force (Hodson, 2008; Hodson & Buell, 2006a).

At the initial time of the OSS release of the framework, the flow of improvements was from the proprietary version to the open sourced version. The two were considered different applications altogether and maintained separately. Within one year of the OpenEaagles release, the momentum shifted as the OSS version became of a higher quality than the proprietary one. Since then, the flow of updates has been reversed and is now covered by SIMAF funding (Hudson, 2009). The OpenEaagles framework is now viewed as a "common reusable piece of software that is shared for all projects" (Hodson, 2009) both proprietary and OSS.

OpenEaagles has been designed specifically for the development and implementation of real-time, multi-threaded, and distributed simulation systems.

The framework is a platform independent simulation engine which draws on external dependencies for complex 3D mathematics, networking, graphics rendering and complex aerodynamic models (Rao et al., 2007). OpenEaagles also provides a well defined structure for developing simulation applications. Abstract classes which are needed to represent entities such as ships, aircraft, vehicles and lifeforms are an integral part of the framework.

The visual representation of the architecture demonstrating the logical breakdown and object-oriented methodology of the framework can be seen in Figure 12. OpenEaagles comes very well documented with interface documents, published papers, useful examples, as well as a book covering its design philosophy. The backbone of the framework has been tested rigorously in multiple implementations of simulation systems still in use by the SIMAF and AFRL today (Hodson et al., 2006b). Important to this research is the complete to script the movement of a player; *Dynamics Model* for the movement of the entity; and *Gimbals* which are found on EOIR systems (further details are given in section 5.1). Useful APIs for the *DIS* and *HLA* interoperability standards as well as *Input/Ouput* (I/O) *Devices* are native to OpenEaagles. OpenEaagles provides an OpenGL based graphic user interface for implementing instrument Heads-Down Display (HDD). The *Sensor* class structure is used for tactical electronic equipment such as radar and therefore does not permit EOIR sensors implementation. Support for CIGI is not show explicitly on the figure but is encapsulated in *OTW*. At the start of this research, the version of CIGI used in

the framework is the original SDK and API and no APIs were available for EOIR sensors. After the completion of the OpenEOIR prototype and nearing the completion of this document, work was being done to update the CIGI usage within OpenEaagles (Hodson, 2009); this will improve its potential usage with powerful commercial IG as well as open the possibility to use new functionalities not supported by the older version of CIGI.

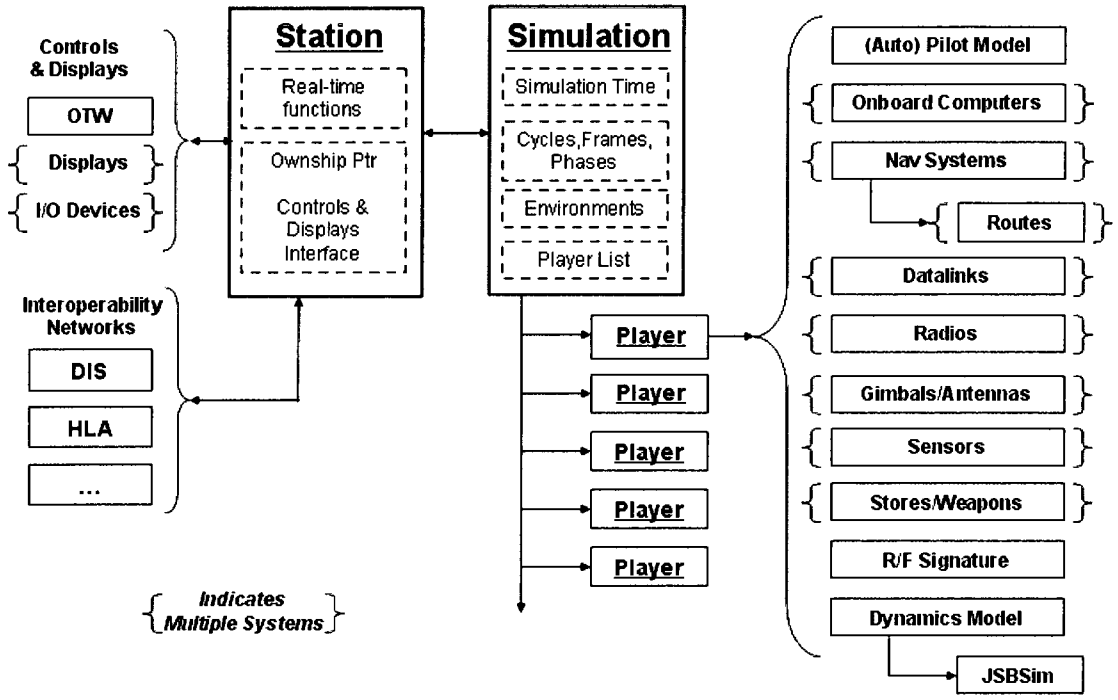


Figure 12: OpenEaagles Simulation Framework (Hodson, 2008)

Another advanced feature that is being investigated by the community surrounding OpenEaagles is the advanced use of multi-threading. The framework is currently based on a dual-threaded architecture but research is under way to make maximum use of multi-core and multi-processor computer

architectures. This work has raised interest from INTEL as they believe it will surpass the level at which even commercial game engines use the growing numbers of cores in a personal computer (Hodson, 2009).

Delta3D is a game engine created at the Naval Postgraduate School (NPS) in the Modeling, Virtual Environments, and Simulation (MOVES) Institute and released under the LGPL in August 2004 (Delta3D, 2008). The project was created to meet the need for an OSS commodity game engine; not to compete with full commercial solutions, but to provide an entry level framework for use by a wide array of developers. In the past years, there has been a trend for the gaming and M&S fields to converge in what is called serious gaming (Roman & Brown, 2006; Roman & Brown, 2009), and this is one area that Delta3D is trying to address in the OSS spectrum (McDowell, Darken, Sullivan, & Johnson, 2006). The second point Delta3D is addressing is the fact that licensing cost and restrictions make it difficult to produce the large number of applications required for the myriad of military training requirements (Darken et al., 2005). With Delta3D, MOVES addresses a niche problem and has been able to generate multiple simulations systems. They appear to be successful within this niche as there have been just over 2000 downloads of the last stable release within a short three month period.

Delta3D was created around the four-part philosophical credo found in Table 5. The first item of the credo mentions lock-ins but McDowell (2008) also introduces the term lock-outs, which describes when a product is no longer

supported or the company does not exist anymore. These two situations are still found within the military domain of modeling and simulation-based training. The Delta3D architecture is shown in Figure 13. For each major functional block (shown in the third row), the Delta3D project canvassed and selected the most powerful and relevant currently established OSS projects (shown in the fourth row). The result is a very modular engine that leverages pre-existing communities (McDowell et al., 2006).

Table 5: Delta3D Philosophical Credo (McDowell, 2008)

Item	Description
1	Keep everything open to avoid lock-ins and increase flexibility.
2	Make it multi-genre since we never know what app it's going to have to support next.
3	Make it modular so we can swap anything out as technologies mature at different rates.
4	Build a community (or leverage existing ones) so the military doesn't have to pay all the bills.

A direct integration of OSG was chosen over CIGI as the IG portion of Delta3D, since the project was developed to be a game engine with a tightly coupled IG. Support for CIGI has not been implemented, and is not part of the current development plans (Delta3D, 2008). Although there has been an effort to generate a Delta3D base CIGI compliant IG, the inverse is not true.

Since its inception, Delta3D has been used many times to develop commercial products as well as training applications for use by the government (Delta3D, 2008). One of these projects falling in the purview of this research is a single player, classroom level IR target identification game named HuntIR

(Camber, 2007). The project was developed under DoD R&D funding as a proof of concept of the potential of OSS to replace current training received by flash cards. Most of the projects developed using Delta3D, including HuntIR, were released under proprietary license and only minor core modifications to the framework were contributed back while the advanced features were not.

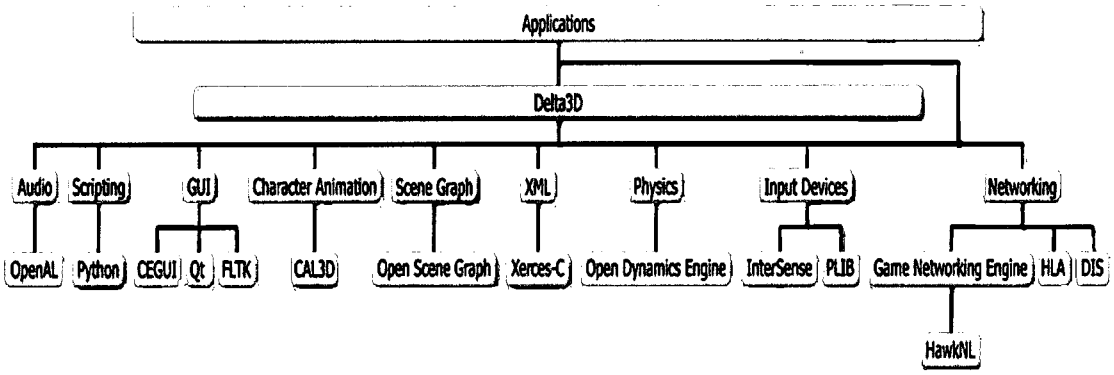


Figure 13: Delta3D Architecture (Delta3D, 2008)

3.3 Business Ecosystem

The concept of a business ecosystem is introduced in this section. After discussing the background and definition of ecosystems, a description of how they evolve is presented. The term ecosystem is then placed in the context of OSS, and finally the advantages of a strong ecosystem are discussed.

Business ecosystem can be defined as an economic community supported by a foundation of interacting organizations and individuals (Moore, 1993) or otherwise stated, an ecosystem is a cooperative approach to developing businesses within a space (Moore, 2006). Economic ecosystems are said to be complex and dynamic (Peltoniemi, 2006) and have been compared to biological

ecosystems (Hannon, 1997; Iansiti & Levien, 2004; Peltoniemi, 2006). While business ecosystems have always existed, managed business ecosystem grew up with the innovation of high technology computer businesses (Moore, 2006) such as some based on OSS.

Business ecosystems, similarly to biological ones, have been observed to gradually move from a random collection of elements towards a more structured community (Moore, 1993). The life-cycle of business ecosystems is said to be divided into four stages (Moore, 2006): (i) pioneering, where a new value is gained; (ii) expansion, where the goal is to gain wide scale acceptance and reach critical mass; (iii) authority, where the particular ecosystem prevails; (iv) and renewal, where the focus is on maintaining the advantage by continuous innovation.

The role of ecosystems in OSS is a topic receiving much attention recently. OSS projects blend well with the ecosystem concept as they co-evolve with their development communities, and this evolution is inter-dependent (Scacchi, 2007). Scacchi underlines that having a strong community and ecosystem surrounding an OSS project leads to quicker technical adoption (Skerrett, 2008). Some go further to say that the ecosystem is a critical element of success (Milinkovich, 2008; Scacchi, 2007). Milinkovich also indentifies OSS as a powerful technique to establish a rapidly growing and strong ecosystem. He also underlines that ecosystems are still misunderstood and need better established metrics to monitor their health. The establishment and nurturing of

ecosystems should be a macroscopical goal and not just a side effect of releasing software as open source (Hao, Zhengang, Chunpei, & Zhuo, 2008).

It has been noted that a strong ecosystem surrounding an OSS project can often be seen as a sign of robustness and longevity which can attract large enterprise involvement (Mitchell, 2009). As well, it was underlined that ecosystems can provide small or start-up companies with the support to compete with larger ones (Bailetti, 2009). Another interesting potential of ecosystems in the OSS context is that exponential growth can be achieved when critical mass is reached and that discontinuous jumps in growth can be observed when OSS projects decide to join forces (Scacchi, 2007).

3.4 Lessons Learned

The lessons that were learned from this literature review are presented in this section.

The primary lesson is that OSS is still not widely accepted in the CF when considering the field of M&S. Also, the CF could be considered to be a passive OSS user as there is no sign of any kind of contribution back to the respective community. This is a noticeable difference with the US approach, where there is an established plan and roadmap to address the integration of open source software in new and current technologies. The primary tangible evidence of OSS acceptance in the field of M&S in the US, is that the DoD has released some of its M&S frameworks as OSS and that these frameworks are being integrated into increasing numbers of DoD M&S projects. In Canada, the use of OSS has been

studied by DRDC but is not actively pursued and no tangible evidence of participation in OSS programs could be found. As well, the GoC does not have any directives for employees to contribute back to OSS communities.

When considering the proprietary nature of most current simulation systems in Canada, lock-in strategies, as well as the potentials for lock-outs, are still visible. This commercial strategy, combined with the current DND M&S procurement process, plays a significant role in the fact that simulation-based training systems in the CF have a tendency to develop significant functional disparities before major effort is spent towards purchasing a new solution meeting the new current needs.

It was found that many commercial sensor simulations are available, most of them implementing the same basic features with some differentiation at the high fidelity spectrum. Users are most interested in realistic sensor fidelity and sensor fusion. This suggests that the basic sensor simulation building blocks should be considered as commodities, and therefore prime candidates for OSS.

Another significant lesson that was observed is the importance that surrounding ecosystems and well established communities play in the success of OSS projects. Without the emergence of such synergy, projects are more likely to stagnate, or at the minimum, fail to reach critical mass. Therefore, OpenEOIR will aim to use such projects.

The literature also points out that there are many different ways to contribute to OSS project and become part of its ecosystem, including: usage of

the software; discussing the existence of the project with others; suggesting improvements; identification of bugs and problems; creating documentation; and implementing improvements to the software are a few way to be involved. It was surprising to note that actual coding involvement is often mentioned as being not as important as other contributions.

Lastly, the literature identifies that the technology needed to implement the OpenEOIR prototype is accessible and that there are enough mature OSS projects available to enable its implementation. That said, no OSS synthetic environment sensor operator training systems were found, and all of the available frameworks require some level of modification to support such a prototype.

4 RESEARCH DESIGN

The design of the research is presented in five parts. First, a description of the research approach is given. Secondly, the unit of analysis is defined. This is followed by a description of the selected study period, and then the procedure to limit the scope of the research is identified. Finally the breakdown of the method used in this research is presented.

4.1 Research Approach

This research is constructive in nature. A working prototype, called OpenEOIR, was developed in order to demonstrate the maturity level of OSS available to the M&S community.

4.2 Unit Of Analysis

The unit of analysis for this research is the CP-140 Aurora community which uses an airborne EOIR system and is currently investigating options for incorporating an environment such as OpenEOIR in their current legacy training system as well as in future replacements. This community has also established a special section to maintain the software used for the mission computer inside the aircraft, and thus have a pool of expertise in software engineering. The Aurora community is therefore a prime candidate by which to introduce the potential OSS has to meet the needs of the CF.

4.3 Study Period

This study was conducted from January 2008 to January 2009.

4.4 Scope

To limit the scope of the research, OpenEOIR is inspired by the system used in the Aurora but addresses only a representative subset of its functionalities. As well, the level of fidelity of the prototype developed is not representative of the real system as some of its specifications could be considered sensitive or classified. The OSS nature of the prototype permits future developers to implement a sensor which could be classified.

4.5 Research Method

The method used to develop the OpenEOIR prototype is described in the paragraphs that follow.

4.5.1 Review State Of The Art

Currently available technologies were identified in order to elicit possible requirements for the system. The commercial and open source solutions and frameworks reviewed, as well as simulation methods currently used for representing EOIR sensor are presented in section 3.2.

4.5.2 Identify OpenEOIR System Requirements

The minimum requirements for the OpenEOIR prototype are presented in section 5.2. The major non-functional requirement is that the prototype maximizes the use of OSS while enabling compatibility with commercially accepted standards and proprietary systems. This requirement is central to the research objective of demonstrating the applicability of OSS in the context of synthetic training environments. The OpenEOIR functional requirements were

elicited from the review of the state of the art as well as this researcher's work experience as an Aurora crew member. The researcher's position within the CF has provided relevant knowledge of EOIR physical systems, as well as contacts within the Aurora community. Conversations with these contacts have allowed requirements to be gathered informally from EOIR operators as well as managers.

4.5.3 OpenEOIR Design and Implementation

The third step in this research was to design and implement the simulation-based training environment prototype. This was done by selecting system components that adhered to widely accepted standards and provided open interfaces. Section 5.3 highlights the design the OpenEOIR prototype and section 5.4 describes its implementation.

4.5.4 OpenEOIR Testing

The prototype was tested to confirm that the implemented functionalities matched the behaviour specified in the requirements. The different scenarios used to test OpenEOIR as well as a discussion of the test results are presented in section 5.5.

4.5.5 OSS Usage Analysis

Once the system was developed and tested, it was analysed with respect to the use of OSS while enabling compatibility with commercially accepted standards and proprietary systems. The amount of effort and knowledge that was required to build the prototype was noted. The implementation code was

analysed to determine how much OSS was reused without modification, how much OSS code required modification, and how much new code was written. The portions of the system where proprietary solutions or information could be used or injected were also identified. Appendix B: Lines of Code presents the complete statistical compilation of the work done, and form the basis for the discussion in section 6.

4.5.6 Analysis of OSS Potential

Based on the data collected and the analysis of the OSS usage, a discussion on the potential of OSS in the context of synthetic training environments is presented in section 7.

5 THE OPEN EOIR PROTOTYPE

This section presents the details of the software engineering process followed during the construction of the OpenEOIR prototype. An introduction to the concept of EOIR sensors is given first in order to better understand the target system. This is followed by the requirements to be fulfilled by OpenEOIR, and then the design and implementation of the prototype. This section concludes by describing the testing which was done to confirm that the prototype indeed meets the OpenEOIR requirements.

5.1 What is an EOIR Sensor System?

An Electro-Optic and Infrared sensor is a collection of specialized video cameras combined for a specific purpose. EOIR sensors are often used by government organisations as part of long range surveillance or targeting systems. A representation of the physical components of an EOIR sensor system similar to the one found in the Aurora aircraft is presented in Figure 14, and the figure is the basis for the rest of the discussion in this sub-section.

The vehicle to which the sensor is mounted is habitually referred to as ownship. Many of the EOIR functionalities, which are discussed later, require the sensor to know its position (*px*). This information is taken from the ownship's Global Position System (*GPS*) and includes geographic coordinates (latitude and longitude), heading, and altitude.

The sensor system is comprised of three different cameras (channels): wide field of view (*EOW*), narrow field of view (*EON*), and infrared (*IR*). Each

camera is optimized for a specific use and has different optical characteristics. The EOW camera is used to capture colour images. The EON camera provides a stronger zoom capability and in order to achieve this, gives up the color capability in favour of black and white images. The IR camera is a heat sensitive camera which operates outside of the visual range to detect heat sources within the infrared portion of the energy frequency spectrum. One of the most used features of the IR camera is the polarity control which determines whether hot items are displayed in white (WhiteHot) or black (BlackHot) color gradient (recall Plate 1). Although not shown in Figure 14, some EOIR systems also include a laser target designator (used for guided ammunition guidance) and a laser range finder. The cameras are contained inside of a *Turret* which is used to control the look angle[1] of all cameras simultaneously as well as isolating them from platform

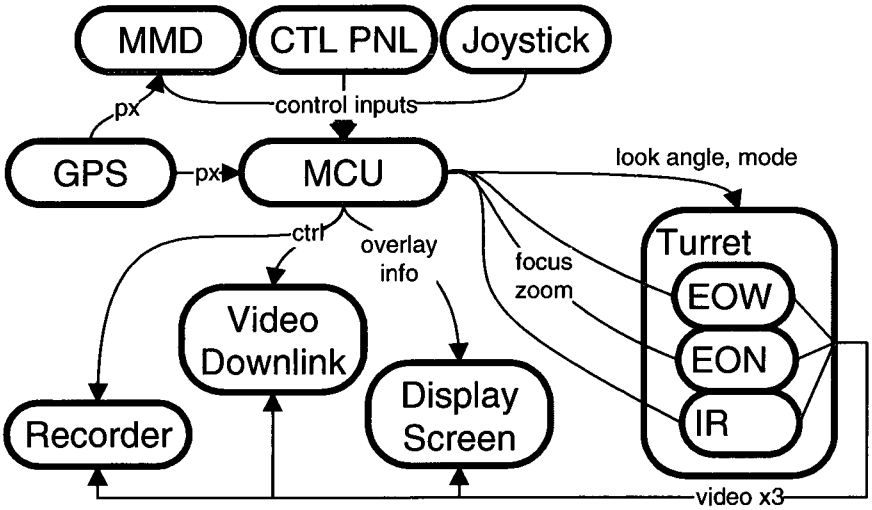


Figure 14: EOIR Physical System

[1] The look angle is the combination of the bearing and pitch based on the camera's line of view in reference to the ownship's nose including its attitude.

vibrations via gimbals. The turret can operate in specific modes such as AUTOSCAN where it is rotated at predefined rates and operator inputs are ignored; MANUAL where only operator inputs control the look angle; FORWARD where the turret is slewed to the ownship's front; and Automatic Video Tracker where the turret follows an identified target.

The sensor is controlled via a *Joystick*, a standalone control panel (*CTL PNL*), and/or an external Moving Map Display (*MMD*) software. The joystick provides many of the operator commands such as look angle control and zoom selection. The CTL PNL is a separate consol which provides the operator with toggle switches as well as rotary dials that provide a finer level of control than the joystick. The MMD receives position information (*px*) and can automate much of the turret cueing information alleviating operator workload such as 'clicking' a target instead of manually directing the sensor to it.

The Master Control Unit (*MCU*) provides look angle and mode commands to the turret as well as zoom and focus information to the different cameras. It acts as a routing station by selecting which signals of the MMD, CTL PNL or Joystick are sent to the sensor depending on which mode is selected. The modes of operation of the MCU and turret can be ONLINE (look angle automatically calculated by the MCU), OFFLINE (look angle directly based on operator inputs), and STOW (turret is set to protect the camera sensors). Some of the advanced processing done by the MCU include: geo-tracking, which is when the sensor remains fixed to a position on the ground; heat-tracking, where

the sensor remains locked to a specified hot spot; and velocity-cueing, when the sensor follows a certain direction and speed on the ground.

The resulting video captured by the cameras is sent to the *Display* output and possibly a *Recording Module* and/or a *Video Downlink* system. The operator can select which channels to display, and are presented with a Head-Up-Display (*HUD*) which is overlaid on the active display. The HUD provides situational awareness information such as aircraft position, sensor look angle, and target position. The configuration of the HUD varies greatly between sensors, and can often be configured by the operator. There is also the possibility of displaying multiple channels in a Picture-In-Picture (PIP) fashion as well as controlling an independent information overlay for each channel. The recording module simply records the defined channel to an independent storage unit and provides immediate playback capability. The video downlink system is used to send the channel selected as the main display to a receiving station for remote viewing.

5.2 OpenEOIR Requirements

This section presents the non-functional requirements first which are mostly OSS research-oriented; and follow by the functional ones which are mostly sensor simulation as well as simulation training-oriented. Typical requirements language is used throughout this section, where the term s*hall* identifies mandatory requirements and the term *should* identifies the ones which are desirable but not mandatory.

As this research is aimed at investigating the potential of OSS, the prototype shall be built by reusing OSS projects and frameworks wherever possible. The OpenEOIR project shall also be released as OSS under a non-restrictive license. Functionality that is not available as OSS shall be included into an existing OSS framework or implemented as an extension to such framework. The prototype shall demonstrate the injection of proprietary solutions and information in at least three cases.

The OpenEOIR prototype is built as a proof of concept and is not intended to be deployed as a practical training environment. It shall, however, implement enough functionality to demonstrate that the development of a practical environment is possible based on OSS. The implemented components (outlined in blue in Figure 14) shall comprise: ownship information including GPS coordinates, joystick control, the three sensor channels, a display system (including a HUD), and the MCU. Each is discussed further bellow. The prototype should also be built such as to permit the future incorporation of the other components and more advanced sensor visual representation and effects.

The sensor's ownship information including position, altitude, attitude[2], velocity[3], and acceleration shall be controlled directly via the simulation framework selected but OpenEOIR should provide for extensibility to support the

[2] Attitude information includes ownship's heading, pitch, and bank angles.
[3] Velocity is a three dimensional vector as it can differ from the attitude to account for wind and turbulence, the same is true for the acceleration.

acceptance of position information from a different simulation via one of the recognised simulation interoperability standards.

OpenEOIR shall implement the EOW, EON and IR channels, which is more than what is found in some proprietary commercial implementations. The complexity of the visual representation of each channel need not be fully realistic, but implement a significant subset of the sensor effects introduced in section 3.2.1 Figure 5. This includes HDR, DOF and continuous zoom for the EOW channel; a grayscale visual representation with stronger continuous zoom shall be supported for the EON channel; and lastly, a grayscale temperature display with polarity switching and a step-zoom for the IR channel.

The prototype shall implement the capability to display a HUD as shown in Plate 2. The HUD shall include the following elements: active channel; cross hair; UTM date and time; turret look angles (pitch and bearing); and ownship information (latitude, longitude, heading and altitude). This configuration implements approximately 45% of what is found in a typical EOIR sensor's HUD overlay as it does not cover target and advance sensor mode information.

The MCU shall support: MANUAL and FORWARD sensor mode; look angle; and a separate zoom control for each camera. Advanced MCU functionalities such as heat-tracking and velocity-cueing are not mandatory requirements. The implementation of these advanced functionalities is not required to demonstrate the feasibility of developing a sensor simulation but should be included in a complete implementation of OpenEOIR. By

implementing only the critical components of the MCU, two of the six possible operation modes (MANUAL, FORWARD, AUTOSCAN, HEAT-TRACKING, VELOCITY-TRACKING, and GEO-TRACKING) approximately 33% of the functionalities are covered.

To be relevant for training, the prototype is required to support the generation of scenarios as well as stimuli from other simulations via a simulation interoperability standard. OpenEOIR shall provide a pre-execution

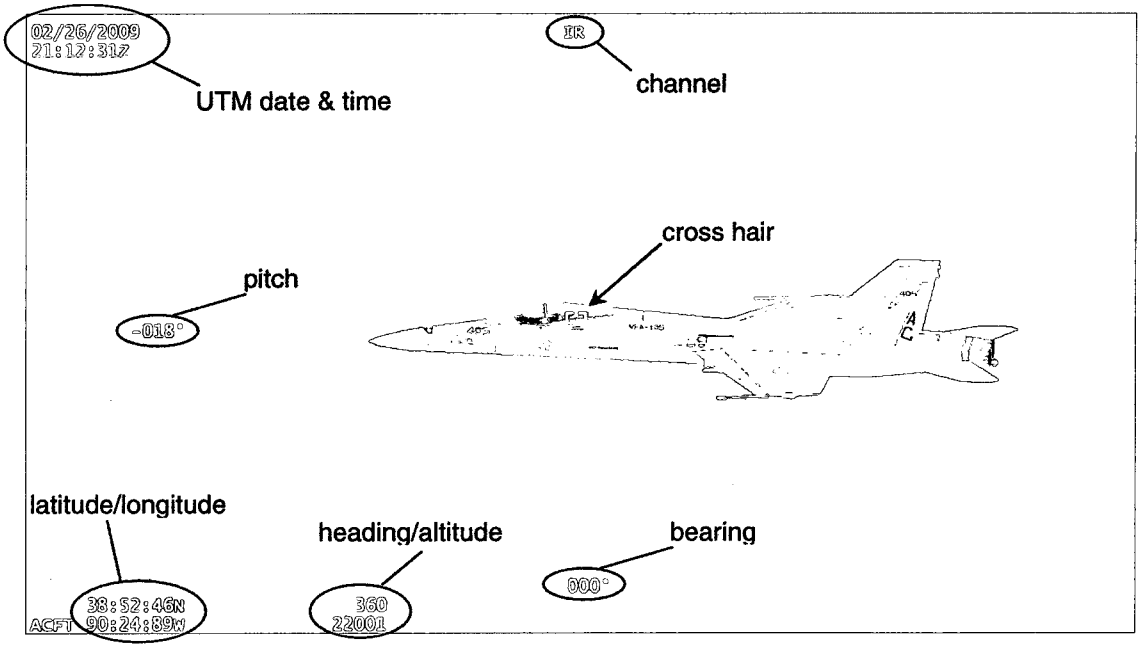


Plate 2: Sensor HUD Overlay

declaration of entity behaviours such as movement and temperature information. The list of entities and their associated 3D models shall be configurable from outside of the application, and therefore, can be tailored for each specific scenario execution.

The scenarios used to test OpenEOIR are required to support a representative subset of entities that would be realistically encountered by the Aurora's EOIR sensor operators, including air, land and navy entities. The MANUAL turret mode shall provide the operator with the ability to control the sensor. The FORWARD turret mode shall slew the sensor to the (0, 0) look angle and disengage to MANUAL mode if the operator tries to move the sensor. All channels shall be tested for their control of look angle and zoom, as well as the visual representation of entities. Different initial temperatures and trends shall be used for each entity to assess both the WhiteHot and BlackHot visual representations for the IR channel. The scenarios shall also test the support for the different communication and interoperability standards by distributing the system to different computers on a network.

An implementation of an Instructor Operator Station (IOS) is ideal but not essential for this prototype, therefore, the pre-execution scripting of scenarios shall suffice. If an IOS is not part of the prototype, OpenEOIR shall provide for the structure to facilitate its implementation.

5.3 OpenEOIR Design

A system design that meets the OpenEOIR requirements is heavily influenced by the standards supported and frameworks selected. Specific standards impact the architecture of components that interact using those standards. The frameworks have associated design philosophies and extension mechanisms that must be integrated effectively. This section presents the

OpenEOIR design. The high-level component architecture is presented first, and the impact of supporting the CIGI and DIS standards is evident. The design of the components is then described, and has been influenced by the selection of the OpenEaagles and MPV frameworks. The rationale for selecting these standards and frameworks is presented when discussing the related portions of the design.

The high-level architecture of OpenEOIR is shown in Figure 15. The *Host* and *IG* components are predicated by support for *CIGI*, as discussed in section 3.2. The *Host* component simulates the sensor turret and its control system. This encompasses the functionalities of the sensor while abstracting the graphical representations of its displays. The *IG* module generates the sensor's view of the world and isolates the graphics calculations from sensor functionalities. The design of the *Host* and *IG* components are elaborated further in sections 5.3.1 and 5.3.2, respectively. Virtual world graphical data is maintained in 3D databases representing the terrain and its components, as well as separate sets of 3D entity models for use in the scenarios. The *Computer Generated Forces* (CGF) component controls the behaviours of the simulated entities that participate in operational scenarios, and communicates this to the *Host* using the *DIS* standard. Further details on these components are given below. The outline colors used in the figure in this section have the following meaning: green indicates components that have been reused without modification; blue represents new features implemented during the development of OpenEOIR;

violet designates components which were reused but required modifications; and red identifies components which can be interchanged between proprietary and/or OSS 3rd party software to show the modular flexibility and potential of the design.

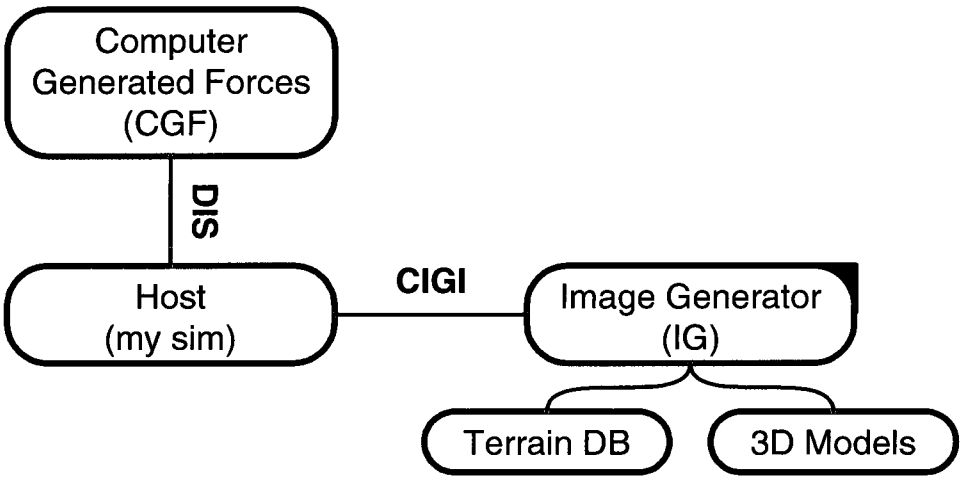


Figure 15: High Level Conceptual Diagram

As revealed in section 3.2.2, CIGI has been recognised as the de-facto standard for distributed visual systems, and is chosen as an integral standard for the distributed OpenEOIR architecture. The distribution model inherent to CIGI was first introduced in Figure 6 and is followed by the Host and IG within OpenEOIR. Furthermore, CIGI is a mature standard with an active and reputable OSS community, and is supported by commercial products as shown in Appendix E: IG Vendor Table. CIGI also provides the modular flexibility to replace the IG or the Host portion of the prototype with a commercial product or another OSS solution.

The 3D database formats chosen to be supported are OpenFlight for terrain and entities; and TerraPage for the terrain. These selections are based on the fact that they are the two most commonly found databases within DND.

The *CGF* component is used to support operational scenarios by externally stimulating the sensor simulation. CGF provides computer representations of the simulated entities that exhibit autonomous behaviour (DoD, 1995). For example, an EOIR training scenario might consist of the CGF directing a simulated subject aircraft to fly along a pre-defined series of waypoints while the EOIR operator locates, tracks, and classifies the subject.

The use of interoperability standards allows the reuse and integration of modular simulations. The *DIS* simulation interoperability protocol is selected over the HLA because of it's simplicity of implementation. DIS only needs look up tables and scenario information to be matched for entity types, whereas the HLA requires more complex configuration. DIS is used to supplement the prototype by allowing the collaboration of 3rd party proprietary CFGs with OpenEOIR, which in turn extends the range of possible scenarios.

5.3.1 Host

The Host component of the OpenEOIR simulator supports the sensor MCU and user interface functionality. The review of the state of the art in section 3.2.6 suggests that the two most promising building blocks for this functionality are OpenEaagles and Delta3D. Both have their distinct advantages and provide much of the structure required to build simulations and script scenarios; however,

OpenEaagles is more appropriate for use in OpenEOIR because: (i) CIGI is considered to be a crucial element of the OpenEOIR architecture and OpenEaagles supports CIGI while Delta3D does not; (ii) OpenEaagles provides a more robust DIS interface than Delta3D; and (iii) OpenEaagles was developed for use in a distributed simulation system, whereas Delta3D is targeted toward an all inclusive, single computer, game-style system. Therefore, OpenEaagles was selected as the underlying framework to support the Host component in OpenEOIR.

The resulting design of the OpenEOIR sensor Host component based on OpenEaagles is depicted in Figure 16. Each entity in the simulation has a temperature associated with it. The entities and their associated temperature information are defined via the Eaagles Definition Language (EDL) which provides scripting support. Entity information received via DIS is transposed to the Host with the default temperature of zero, since DIS does not have entity temperature assignment. The temperature of DIS received entities could be controlled via an IOS, if one were to be implemented. The Host then augments the entity information with the entity's temperature, and passes this to the IG via CIGI. The sensor turret accepts mode, steering and zoom commands from the sensor control (MCU). The joystick provides the Human Machine Interface (HMI) to the ownship and sensor. The ownship interface implements the pilot control of attitude and velocity. The sensor commands supported are sensor mode (including polarity), look angle steering (including reset), zoom control, simulation

freeze (used to pause scenario execution), and simulation reset (used to restart the scenario from its initial state). The compiled sensor state and commands are sent to the IG via CIGI.

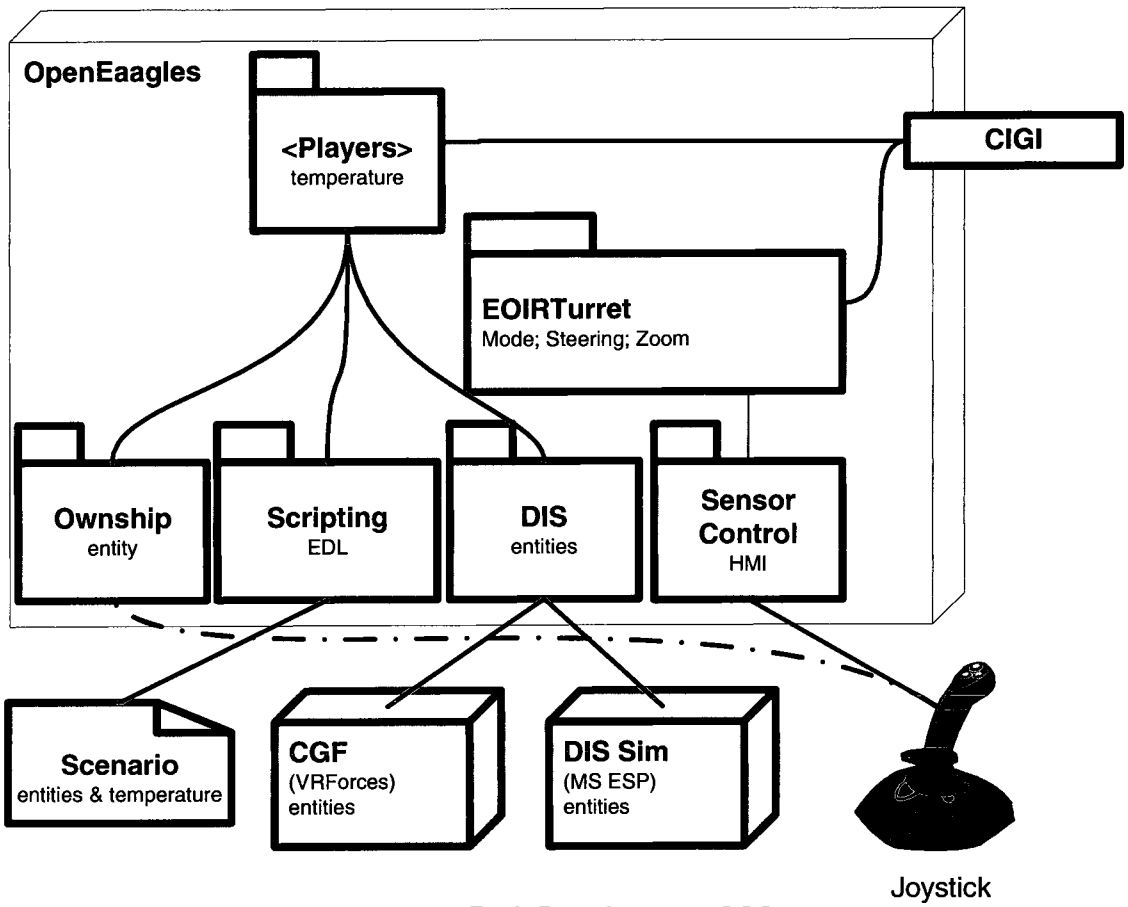


Figure 16: OpenEOIR Host Architecture

5.3.2 IG

The IG component of OpenEOIR supports the visual representation and control of the three channels (EOW, EON, and IR). As discussed in section 3.2.4, the two most prominent OSS IG systems are MPV and SubrScene. They

are both well-established projects, support the required 3D databases formats and provide many of the functionalities required for OpenEOIR including full support for OTW displays. On the other hand, neither of them supports an EOIR sensor simulation which hints to a relatively equal level of development effort for OpenEOIR. MPV was selected because: (i) MPV was originally built with CIGI in mind, where SubrScene had it's own distributed visual system implementation but now has an add-on which supports CIGI; (ii) MPV is always kept to the current CIGI ICD version which is now CCL V3.3 whereas SubrScene only supports the SDK/API V2.0; (iii) MPV is built in a modular plugin architecture which provides the ability to add functionalities and behaviours without making changes to the core of the application while SubrScene is a self-contained application which does not provide this flexibility; (iv) MPV is currently working on high performance multi-threading support whereas the multi-threading future of SubrScene is somewhat uncertain.

Based on MPV's current design, the architecture of the OpenEOIR IG is presented in Figure 17 and highlights the items that are retrieved and published to MPV Blackboard in the inset boxes. The new functionalities incorporated into MPV (shown in blue) are: support for RTT; shader management; EOW, EON, and IR sensor effects; sensor control; and HUD sensor overlay. MPV functionalities requiring customization for OpenEOIR (shown in violet) are: camera scene rendering; terrain rendering; entity rendering; and small changes to MPV core.

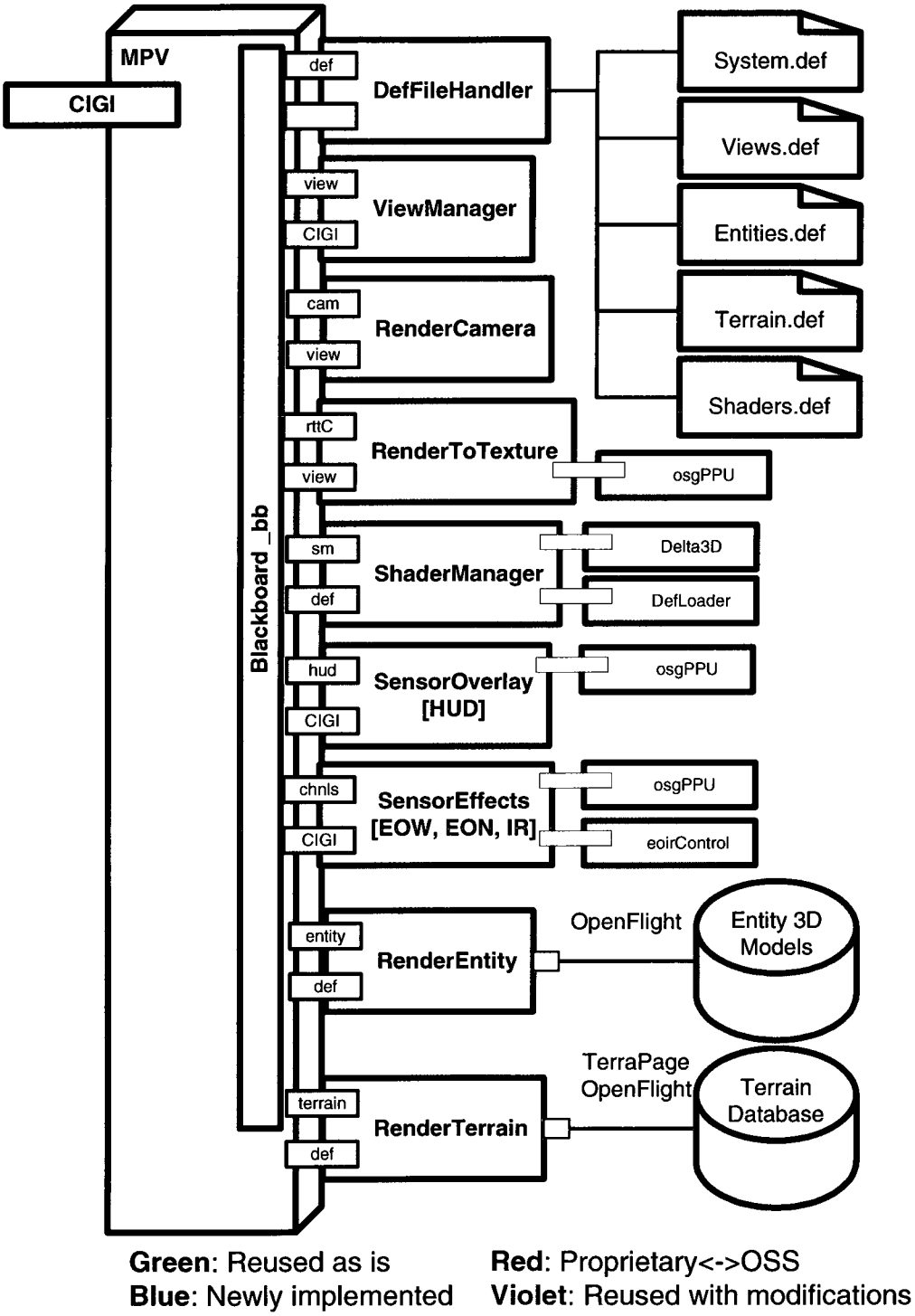


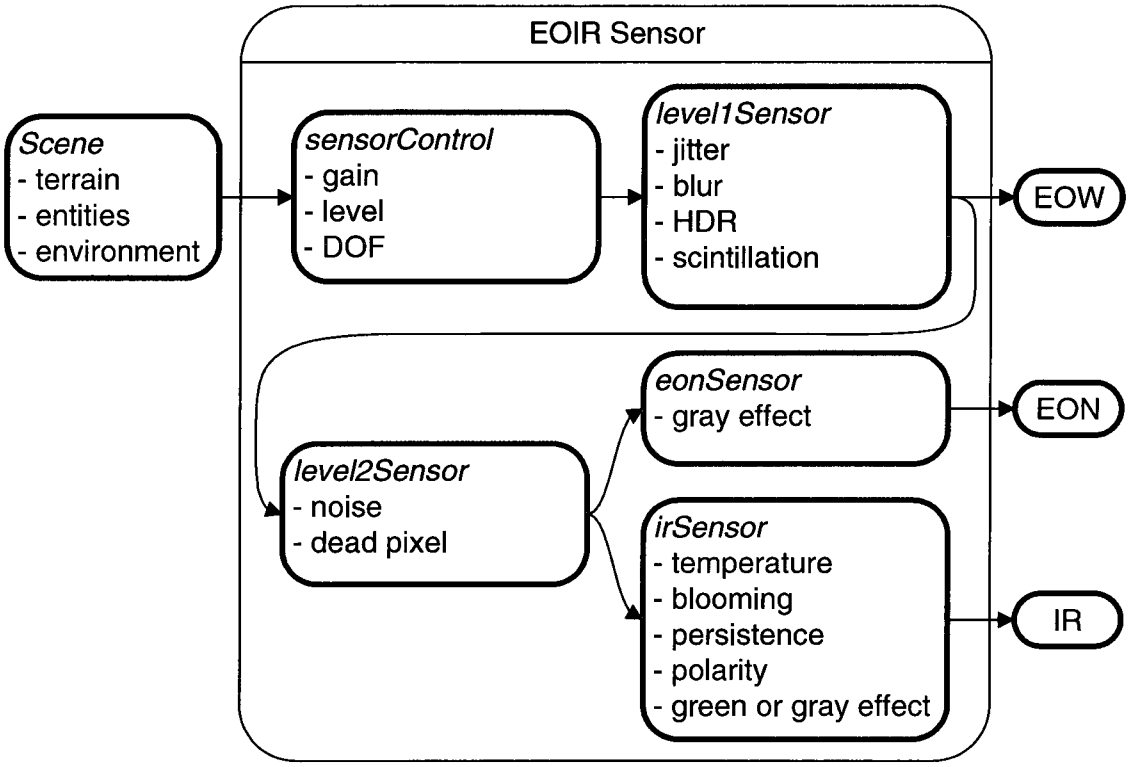
Figure 17: OpenEOIR IG Architecture

To simplify the reuse and modification of the IG component, the design allows definition, manipulation and configuration of the shader files and programs to be done outside of the application. This capability is one of the strong points of the Delta3D design, and the Delta3D ShaderManager structure has been integrated into the design of the OpenEOIR prototype in the form of the ShaderManager plugin to MPV. The plugin is intended to be useful to other projects utilizing MPV, hence it was developed and is managed under its own OSS project: mpvShaderManagerPlugin (Castonguay, 2009a).

The OpenEOIR prototype implements a representative subset of the sensor effects needed to emulate a realistic EOIR sensor, as discussed in sections 3.2.1 and 5.2. Figure 18 extends the simplified EOIR post-processing pipeline in Figure 5 to include the details specific to OpenEOIR. These details include the three sensor channels and the following effects: DOF, HDR, visual effects for each channel, and polarity control for the IR channel. OpenEOIR implements 40% of what is required for a complete sensor representation. The complexity of the post-processing pipeline could be easily increased to emulate a more realistic sensor.

The HUD SensorOverlay is used by the operator to get quick situation awareness information on the current scenario. The HUD composition and disposition is something that varies greatly between systems as well as operators. The new version of the CIGI ICD (3.3 and above), supports the definition and control of the overlay via the Host. Unfortunately, the OpenEOIR

Host is based on OpenEaagles, which currently does not support the latest ICD. It was therefore necessary to implement a plugin to generate the HUD through the IG, as is done in the current configuration of most commercial IG system.



Black: Non implemented **Green**: Reused as is **Red**: Proprietary<->OSS
Blue: Newly implemented **Violet**: Reused with modifications

Figure 18: OpenEOIR Pipeline

5.4 OpenEOIR Implementation

OpenEOIR was developed using Microsoft Visual Studio (VS) Express 2005 (Microsoft, 2009), as both MPV and OpenEaagles are distributed with VS project files defining all required dependencies. VS Express 2005 is not OSS, but is a freely distributed version of the professional suite offered by Microsoft.

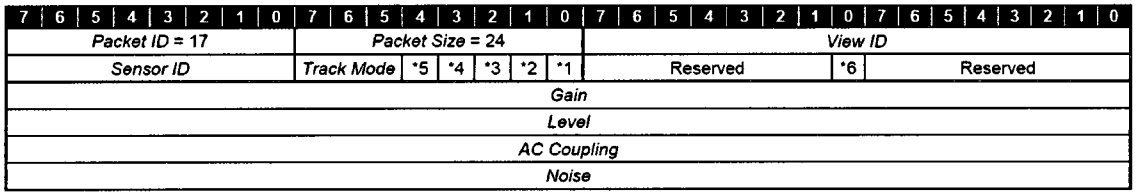
OSS alternatives such as Eclipse could have been used, but associated learning curve would have increased the implementation time enough that it was not felt viable. The implementations of the Host and IG components of OpenEOIR are described in the following sections.

5.4.1 Host

The OpenEaagles framework provides support for much of the Host design (recall Figure 16); however two areas required modifications. The OpenEaagles core was extended to support entity temperature information, temperature control, and CIGI sensor control. These modifications involved the Player class and the CIGI Entity Control Packet. In addition to the core modifications, the introduction of the EOIR sensor requires new functionality to permit a joystick to control sensor mode, look angle, and zoom. These modifications involved the creation of the EOIR Turret and the SensorControl classes as well as the manipulation of the CIGI View Definition Packet.

OpenEaagles encapsulates entities of all kind in the Player class, and this class was modified to incorporate temperature information. For the simple OpenEOIR prototype, the implementation of a single internal temperature for the entity is used by the IG to generate a visual temperature effect on the 3D model. Each resulting Player instance has: a Current Temperature; Temperature Limits (Maximum and Minimum); as well as a Temperature Trend which initiates a rate of temperature change over time until a limit is reached. The Player class was given a public interface to these parameters which could be used by a future IOS

to control temperature parameters at run time. This interface is used by the "Slot" mechanism provided by OpenEaagles to enable the initialization of temperature parameters. Slots are used for pre-execution scenario definition using the EDL. A more advanced implementation could delegate the control of sub-component temperature to the containing entity (e.g. a vehicle entity could control the temperature of its hood, engine, doors, wheels, etc...). This delegation of functionality is supported by CIGI V3 and above which is supported by MPV but was not supported by OpenEaagles at the time of the OpenEOIR implementation.



*1 *Sensor On/Off*
*2 *Polarity*
*3 *Line-by-Line Dropout Enable*
*4 *Automatic Gain*
*5 *Track White/Black*
*6 *Response Type*

Figure 19: CIGI Sensor Control Packet Structure (Durham, 2006)

The current CIGI interface supported by OpenEaagles only provides OTW display, and therefore, support for entity temperature and sensor control needed to be added. When processing each entity for CIGI information, OpenEaagles was leaving the temperature information of the ENTITY_CONTROL packets empty (default to zero). This was rectified by copying the Players' temperature into the control packets. As the SENSOR_CONTROL packet handling was non-

existent in OpenEaagles, its interface to the API needed to be created. The new interface provides access to all of the fields identified in Figure 19. The OpenEOIR prototype is only required to use the following parameters:

- ViewID – the view to which the sensor is attached;
- SensorID – can have multiple sensors per view;
- Sensor On/Off; and
- Polarity;

The Line-by-Line Dropout is always sent as the default disabled value, and could be used to represent dead pixels by generating black lines through the channel's display. The Automatic Gain is used to toggle from automatic to manual gain control but is not required by the OpenEOIR IG. The Track White/Black and Response Type parameters are used to control intelligent sensor target tracking. These last two parameters were not implemented as they are normally used when controlling seekers such as missiles which are outside of the OpenEOIR requirements.

OpenEaagles comes with many examples, one of which is a flight simulation supporting: (i) a HMI via a Joystick; (ii) a local (OpenGL) HDD implementing graphics for flight instruments as well as other sensors; and (iii) a CIGI OTW view. It also demonstrates the use of EDL for connecting the HMI to a dynamic flight model for ownship control; connecting a few sensors defined within OpenEaagles core to the ownship; specifying a DIS interface; and declaring a basic scenario. This example was used as the starting point for the

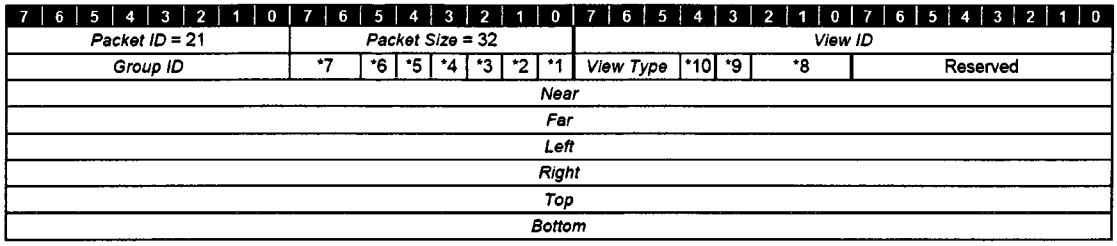
OpenEOIR's Host implementation and, with the modifications made to the core of the framework, only the implementations of an EOIR Sensor Control and a CIGI compliant EOIR Turret was required (recall Figure 16). The Sensor Control class is not discussed in detail as it simply maps specific Joystick buttons to the Turret's look angle control, independent zoom control, and active channel selection functionalities. The remainder of this sub-section describes the implementation of the turret and channel control modifications.

Control of the turret look angle must satisfy the requirement that all cameras within a turret be slewed to the same gimbals. This is accomplished by synchronizing all cameras to follow the same inputs, regardless of whether they are selected. The implemented gimbals are simplistic, and only the bearing and pitch angles of the turret are controlled. Furthermore, behaviours such as vibrations and complex slew rate control are not implemented. The rotation speed of the turret is biased to account for the zoom factor of the currently selected channel so that when a high zoom is used, the movement is not too fast for the operator to observe.

The OpenEOIR operator must be able to select which camera is to be displayed. This is done by using the CIGI SENSOR_CONTROL packets (Figure 19). A single view is managed with appropriate visual effects depending on the camera that is currently active. At this time, CIGI does not support the dynamic control of view positions and view layering which would be necessary to implement Picture-In-Picture (PIP) functionality. Appendix C: shows a non-CIGI

compliant prototype, built with OpenEOIR, to demonstrate the functionality which is found on many EOIR systems. The addition of the support for PIP to CIGI was suggested to the team which will canvas for general interest in the future. The user can control which camera is active by toggling through the channels using a button on the joystick.

Each camera in the turret requires independent control of its zoom level. The zoom is simulated by controlling the Field Of View (FOV) parameters in the CIGI VIEW_DEFINITION packet, see in Figure 20. Reducing the Left, Right, Top, and Bottom FOV gives the impression of a zoom. For example, if the initial FOV is 20 degrees vertical by 15 degrees horizontal, a 2X zoom can be simulated by changing the FOV to 10 degrees vertical by 7.5 degrees horizontal. The zoom could also be implemented via a custom COMPONENT_CONTROL packet but would be less desirable since it would be IG dependent. The chosen implementation is general since it keeps the control of this functionality on the Host side, and therefore enables an easy transition to a different IG if desired. As the system has three different channels, the zoom information of the active channel must be saved each time a new channel is selected. This ensures that the correct zoom can be restored the next time the user switches back to that channel. This is done using a table of information for each channel.



*1 Near Enable
*2 Far Enable
*3 Left Enable
*4 Right Enable
*5 Top Enable
*6 Bottom Enable
*7 Mirror Mode
*8 Pixel Replication Mode
*9 Projection Type
*10 Reorder

Figure 20: CIGI View Definition Packet Structure (Durham, 2006)

5.4.2 IG

The visual representation of the three channels supported by OpenEOIR, as well as their control, is implemented in the IG. MPV provides most of the functionalities required to generate and control an OTW implementation, but lacks the implementation of the image processing required for an EOIR sensor simulation. This is remedied by the addition of RTT, LOD, and Temperature capabilities. These capabilities are implemented by modifying pre-existing RenderCamera, RenderEntity, RenderTerrain, common, and commonOSG plugins (shown in violet) in Figure 18. The common and commonOSG plugins are part of MPV core and are not shown in the figure. New plugins (shown in blue) were developed to provide ShaderManagement, EOIR visual SensorEffects and control, a RenderToTexture post-processing pipeline, as well as handling of the HUD SensorOverlay. The ShaderManager reused the Delta3D project

whereas the RTT, SensorEffect, and SensorOverlay reused the osgPPU project (Tev, 2008) which is discussed later.

The RTT technique is used to create the post-processing pipeline, as described in section 3.2.1. Therefore, the *RenderToTexture* plugin to MPV was developed to insert a new camera (*rttCamera*) in the SceneGraph and direct it output to the Frame Buffer for post-processing. However, the creation of the cameras and viewports is handled in the *RenderCamera* plugin, and that plugin required extension to provide appropriate access to the viewports. Figure 21 presents the OpenEOIR SceneGraph after inserting the cameras, and shows the new nodes in blue. The *Viewer* is the top SceneGraph aggregate and is used to render the scene's different views. Prior to OpenEOIR, the *viewports* were connected directly to the *RootNode*. This was modified to add a *vpNode* for each viewport to which a *rttCamera* is attached. The Frame Buffer of each rttCamera is passed to the associated post-processing unit (*ppu*) which is used and controlled by the *SensorEffect* plugin (discussed later). There are three viewports for the current OpenEOIR sensor configuration; however, the number of viewports is shown as N indicating that further enhancements can easily add additional views to the configuration in a MPV definition file. The implementation was made to be generic to support future OpenEOIR extensions such as the addition of a three channel OTW pilot view and another sensor camera, generating a SceneGraph with seven viewports.

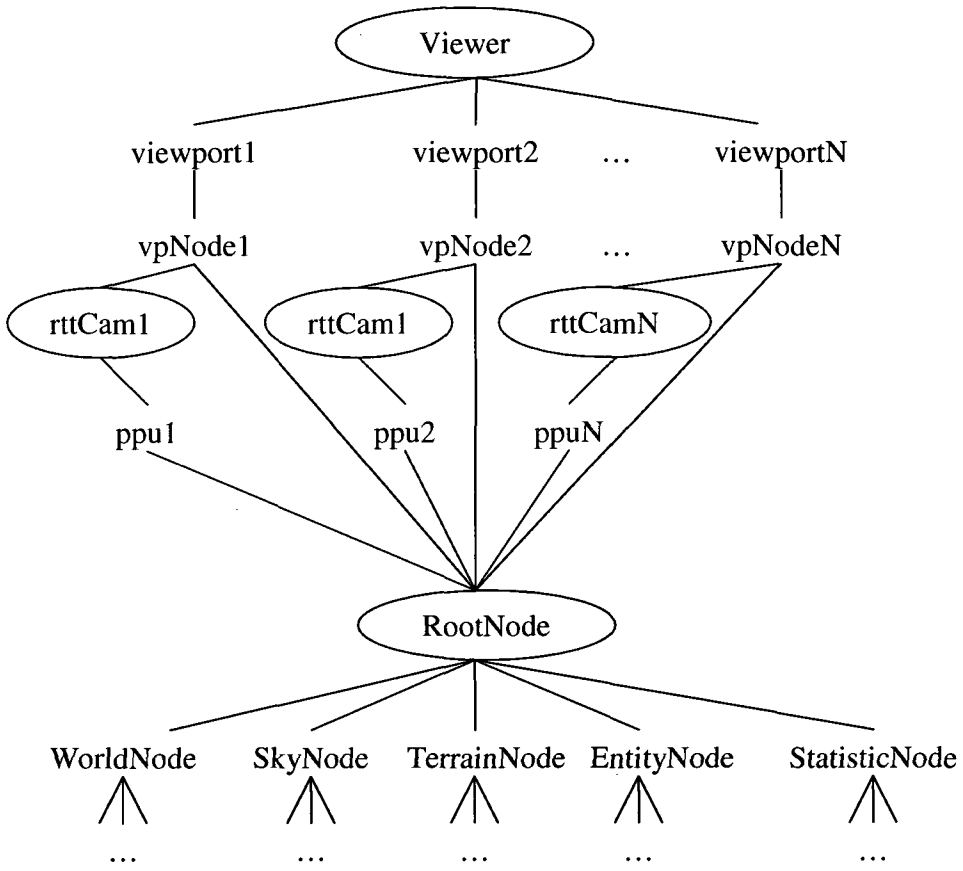


Figure 21: OpenEOIR SceneGraph

The original version of MPV supports only the distance LOD control technique, which is not acceptable for sensor simulations that incorporate zoom functionality (recall section 2.1). The introduction of the relative pixel size control mechanism is appropriately placed in the *commonOSG* plugin, which is used by MPV to store OSG related utility functions such as node searching. The new *setLODMode* function provides control of the LOD at the node level and can be accessed from any plugin. The *RenderTerrain* and *RenderEntity* plugins were

modified to select the relative pixel size control mechanism so that the LOD is synchronized with the sensor zoom.

OpenEaagles currently implements the older version 2 of the ICD, which sends the temperature information via the ENTITY_CONTROL packet. MPV implements version 3 of the ICD, which expects the temperature information to come from a CUSTOM_COMPONENT_CONTROL CIGI packet and disregards the temperature information from the ENTITY_CONTROL packet. The *common* plugin is used by MPV to store general utility functions and classes such as an entity class, and was modified to assign temperature information to entity instances within MPV. This information is then used by the *SensorEffect* plugin to assign the temperature information to the appropriate shader.

The *ShaderManager* plugin incorporates the Delta3D shader management system with the aim of separating the application programming from the graphics manipulation. It abstracts the definition of the shaders and their parameters as well as automates much of their definitions and handling. This included the creation of a parser and a plugin interface. The parser processes the information from the *shader.def* file containing the shader definition which was loaded by the DefFileHandler to the Blackboard. The parser then loads the shader code (GLSL programs) into the GPU memory and inserts references to the programs in the *shaderManager* instance. The actual GLSL code for the shaders is situated in separate files from the *shader.def* and can be changed during runtime and reloaded to the GPU. This gives much greater flexibility to experiment

dynamically with changes to the shader code. The resulting *shaderManager* instance is then posted to the blackboard for other plugins to use and the shader parameters are continually updated in the graphics pipeline. The effectiveness of the ShaderManager is discussed in more detail in section 6.3.

The *SensorEffect* plugin encapsulates the functionality of the EOIR Sensor component of the OpenEOIR Pipeline in Figure 18. The plugin simulates the EOW, OEN, and IR effects using a post-processing pipeline. The osgPPU library (Tev, 2008) is one of the many projects that build on OSG, and provides a mechanism specifically intended to create complex post-processing pipelines. It does so by breaking down the pipeline into smaller Post-Processing Units (PPU) which can be combined in complex ways. The osgPPU library is used in OpenEOIR, and a PPU processor is assigned to each RTT camera in the SceneGraph (recall Figure 21). This processor can be comprised of many interconnected PPUs that implement the desired post-processing.

Existing functionality in the osgPPU library is used to implement the DOF and HDR PPU required to generate the EOW channel. A sample image from the EOW channel is shown in Plate 3. The *eonSensorPPU* adds a simple shader to the pipeline in order to generate a grayscale effect by applying a bias to each of the red, green and blue color components of the entire scene. A sample of the resulting shading is shown in Plate 4.

Plate 3: Destroyer EOW

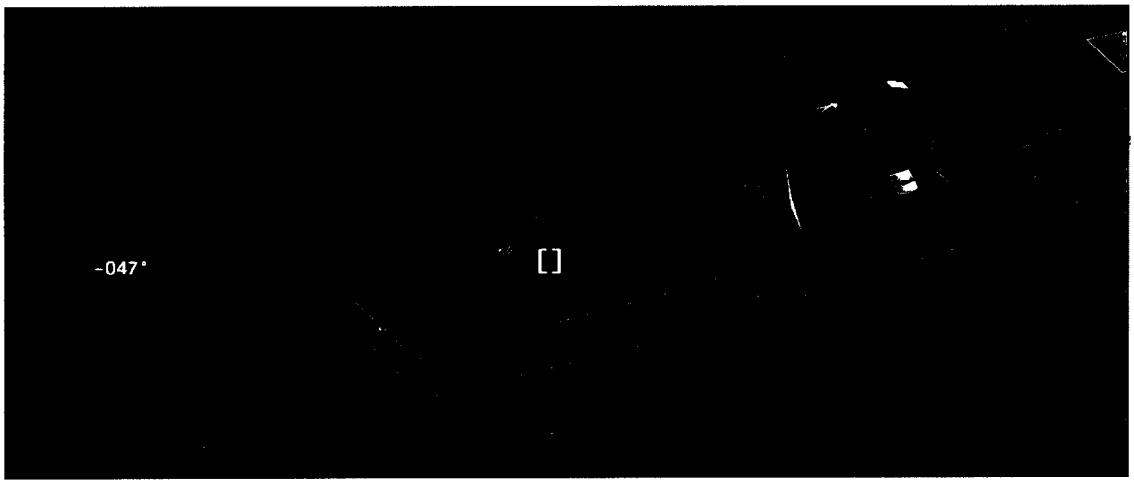


Plate 4: Destroyer EON

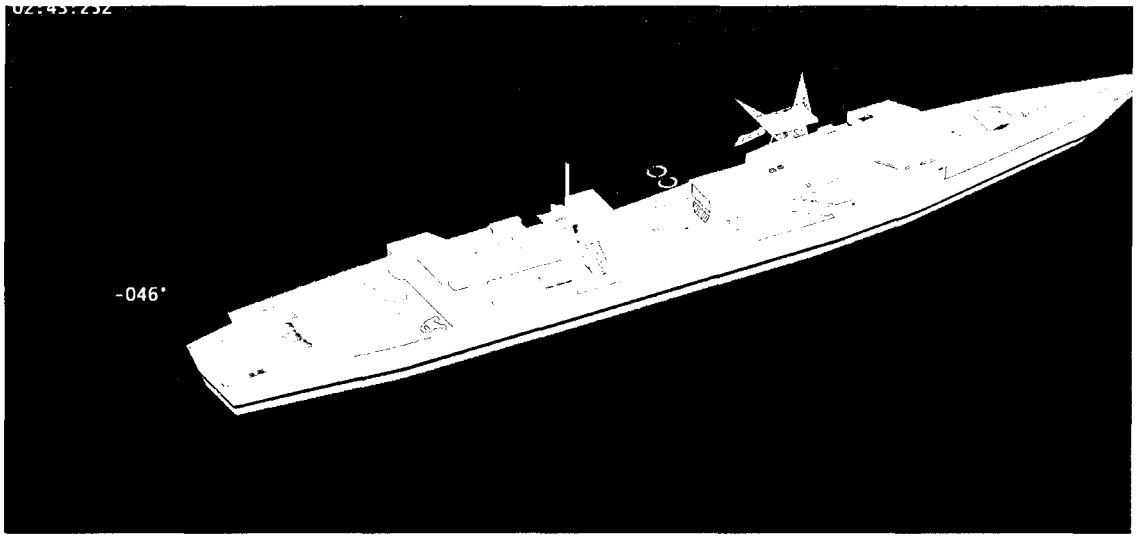


Plate 5: Destroyer IR

The *SensorEffect* plugin also introduces the *irSensorPPU* to implement temperature, polarity, and grayscale effects in the IR channel. Note that the gray effect in the IR channel is different than the one in the EON channel. The temperature effect is implemented via a shader that is applied to individual entities (instead of the entire scene), and a different equation is used to obtain the effect. The temperature shader provides the static definition of maximum and minimum temperatures, which are used along with the entity's current temperature in the heat effect equation to calculate a heat factor. The heat factor is applied to all color components of the entity being processed. The result is a differential appearance among entities based on their current temperature, i.e. their brightness depends on their current temperature. For sake of simplicity the implemented heat effect equation is a simple linear relationship, but could be changed with modifications to the shader code only. The *temperature.glsl*

shader provides an interface to set the current temperature of the entity being processed. This parameter can be updated as often as needed by the application to ensure that entities will be processed with their most current temperature. The updating of shader parameters is accomplished via the *shaderManager* instance. The polarity and grayscale effects are implemented together in the *infrared.glsl* shader, which implements the grayscale effect in much the same way as the EON shader, but with a different factor for the blue color component. This shader implements the polarity switch visual effect by inverting the color factor so that white is represented by black and vice-versa, thereby emulating WhiteHot and BlackHot effects. A sample of the resulting WhiteHot shading obtained by the temperature and infrared shaders is shown in Plate 5.

Sensor control is also implemented as part of the *SensorEffect* plugin using the eoirControl structure (recall Figure 17). A listener was added to the incoming CIGI packet processor to detect SENSOR_CTL_PACKETs (recall Figure 19). When a packet is received, the listener enables a callback to the plugin which processes the packet and makes the enclosed sensor information available to the EOIR post-processing pipeline and its PPUs

The *SensorOverlay* plugin creates and updates the HUD. This plugin also uses the osgPPU library to create a pipeline to inject the HUD information as an overlay on top of the shaded scene. The pipeline takes advantage of the osgPPU library's TextUnit, which is a special type of PPU containing only

characters. The pipeline uses a TextUnit for each HUD element. For example, the date and time are implemented as two TextUnits. With this structure, every HUD component can be configured and updated independently and an interface could easily be added to remotely configure the HUD composition and disposition. The information used for the HUD is derived largely from viewport, camera and entity data, but also uses a SENSOR_CRL_PACKET listener to get information on the state of sensor as well as the active channel selected (EOW, EON, or IR). An illustration of the current configuration has been shown and described previously in Plate 2.

5.5 OpenEOIR Testing

Compliance to the OpenEOIR requirements is demonstrated using four operational scenarios. These scenarios are incremental in complexity and exercise the CIGI and DIS interfaces of the prototype, as well as basic entity control. The first three scenarios run OpenEOIR on a single workstation and CIGI is configured to communicate via the LocalHost IP address, and without a CGF connected. The last scenario is distributed to four workstations and is focussed on exercising the DIS interface to a connected CFG.

Plate 6: Tank IR, Varying Temperatures

The first scenario situates the sensor within a static ownship entity (i.e. one that is fixed in space) and uses an OSS terrain database located in the St-Louis area. Two other static subject entities are situated at a small known altitude and position offset from the ownship entity. One subject was a CF-18 (Canadian fighter airplane), and the other a T-72 (Russian tank). The CF-18 was represented using CF proprietary 3D model whereas the T-72 was an OSS model available through the Delta3D project. The subjects are set (on the Host side) to have a temperature variant from -273 deg C to 1000 deg C with a gradient of 30 deg/sec in order to better demonstrate and visualize the entire temperature spectrum supported by the temperature shader. The sensor turret is then manually directed (MANUAL mode) to the subjects and the different channels are adjusted for optimum zoom. The change in temperature for the tank entity is shown in Plate 6, and is observed in WhiteHot IR mode. The images range from coldest temperature on the left, to hottest on the right. Toggling the polarity from WhiteHot to BlackHot obtained results similar to those for the frigate in Plate 1. The EOW and EON channels demonstrated acceptable visual representation with results similar to the ones shown in Plate 3 and Plate 4. This scenario demonstrates that the sensor graphical representation is acceptable, and that both proprietary and OSS models can be used at the same time. This scenario also highlights that geo-tracking functionality would be useful, as it would alleviate long cueing times with the Joystick.

The second scenario situates the sensor within an air entity flying straight and level at slow speed. This flight profile is configured via EDL and cannot be controlled at run-time. This scenario hosts three dynamic subject entities (a CF-18, a T-72, and a frigate) having different temperature profiles than in the first scenario. The first two entities were the same as in the previous scenario but the frigate was an OSS model taken from SubrScene. The goal is to locate and track each entity during separate runs of the scenario. This scenario tests entity positioning as well as movement of the ownship and the EOIR sensor. It also tests that air, land, and naval entities are supported by the prototype. Executing the scenario demonstrates that these functionalities are supported correctly and highlights that the ONLINE sensor mode with velocity-tracking functionality would be a useful addition, as the constant operator input could be reduced significantly.

The third scenario is similar to the second, but situates the sensor within a dynamic air entity controlled via a Joystick. Furthermore, the scenario uses a CF proprietary database for terrain located on the west-coast of Canada. The pilot and sensor operator controls would be separated in a realistic crew environment, but the OpenEOIR prototype requires the same operator to direct both the vehicle and the sensor turret, which elevates the workload significantly. This scenario demonstrates that with only the ability to operate the sensor in MANUAL mode, it is nearly impossible to track a subject, as the sensor is affected by any movement of the airplane. For example, if the sensor is looking at a tank and the

airplane is banked 10 degrees to the left, the sensor will move off the subject and re-acquisition is a non-trivial task. Without previous knowledge of the position of the subjects, it was difficult to locate them while also accounting for the ownship movement. This scenario highlights that the ONLINE mode providing geo-tacking and heat-source tracking would be required for a more realistic training environment.

These first three scenarios were generated completely within OpenEOIR using the EDL scripting to demonstrate the requirement that the prototype support the generation of scenarios. An example of the use of the EDL can be found in Appendix D: Code Excerpts. The effort required to establish a scenario only requires a few lines within a configuration file and no changes to the code or prototype was needed to migrate between them.

The fourth scenario expands on the third and exercises the DIS interface of the OpenEOIR prototype. This scenario was generated by a combination of EDL and external simulation stimuli. It is intended to test that entities from the different simulations are positioned and updated correctly for all stations. The network and the different commercial software packages used in this scenario as well as conclusions that were drawn from its executions will be presented in the paragraphs to follow.

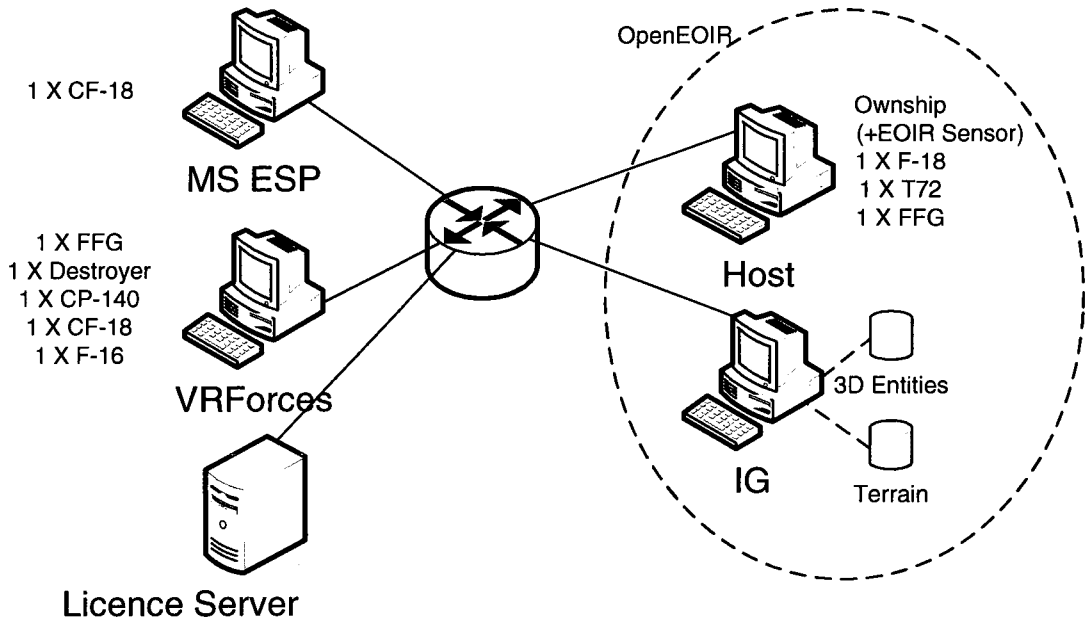


Figure 22: Fourth Scenario Network Configuration

The Continuous Training Federation Testbed (CTFT) provided by DND at Carleton University was used to execute this scenario. CTFT is a laboratory where simulation systems are tested before being deployed for use by other organizations within the CF, and hosts the hardware, software, and licenses required to do so. A Local Area Network (LAN) was established on the CTFT and configured to use four workstations plus the licence server as shown on Figure 22. The deployment of OpenEOIR and the scenario on the LAN is shown in Figure 24. The first station ran the proprietary MaK VRForces (MaK, 2009) CGF. It was used to simulate a frigate, a destroyer, an Aurora, as well as a CF-18 (Canadian) and an F-16 (American) fighter planes. The second station ran Microsoft ESP (Microsoft, 2009) flight simulator, with a DIS plugin (Acron, 2009),

to simulate a remote virtual training device operated by a pilot. The remaining two stations were used to execute the OpenEOIR Host and IG. The 3D entities new to this scenario are the Aurora, which was a CF proprietary 3D model; the F-16, which came from Delta3D; and the destroyer, which was an OSS model taken from the Combat Simulator Project (CSP, 2009). In order to execute instances of VRForces and ESP, licences needed to be checked out at runtime. Fortunately, the stations used were already pre-configured to get licenses from the CTFT server, which alleviated much of the workload. Only slight modifications were needed to the VR Forces and ESP configuration files to enable the use of the desired entities, correlate the DIS identification codes, as well as identify the scenario and network.

As soon as the simulation was initiated, VRForces and ESP displayed and started updating the entities generated by the OpenEOIR Host. As well, entities generated by ESP and VRForces were displayed at the correct positions. After a few seconds of execution, OpenEOIR terminated due to errors in handling EmissionPDUs coming from VRForces. This problem was traced to RF emissions which were by default assigned to air entities within VRForces. The EmissionPDU was not required for the execution of the test scenario, and the RF emissions were disabled so that the scenario could still be executed without the required modifications.

The problem with the EmissionPDU was reported on the OpenEaagles forum, and within hours, the OpenEaagles team indentified that it was caused by

OpenEaagles incorrectly processing the version of the PDU used by VRForces. The OpenEaagles team corrected the code generating the PDU error within a few weeks. The quick response and action from their part disproves the myth that there is no support for well-established OSS projects.

The effort required to get this last scenario working was minimal and was completed within one day, including the installation and configuration of the OpenEOIR prototype onto the CTFT.

To test the compatibility of different CIGI compliant systems, the first scenario was executed with different IGs. An alternate OSS EOIR sensor IG could not be found, and requests for free trial licences of commercial sensor IGs were left unanswered by the two companies contacted (MetaVR, 2009; Presagis, 2009). The system was therefore configured to use only the normal OTW and both MPV and SubrScene were connected as CIGI enabled IGs. With little modification of the configuration files, the simulation ran with different terrain databases and entity 3D models as described in the next paragraph.

Using CIGI, terrain databases and 3D models are assigned a unique identifier number. This allows models and databases to be interchanged easily, and therefore permits the mixed use of OSS and/or proprietary without licence infringement. This enables developers to use OSS models for testing and development and higher fidelity proprietary models for actual training if required. The only restriction is that the formats (proprietary as well as OSS) need to be compatible with the IG system. This was demonstrated in all scenarios and more

specifically by executing the third scenario twice, where the CF proprietary CF-18 model was replaced by a F-18 model (United State version) taken from Delta3D. All of the 3D entities used were in the OpenFlight format but on the second execution of the third scenario a TerraPage format was used instead. Access to the proprietary terrain databases and 3D models was made available through the DND's Mapping and Charting Establishment (MCE) which is slowly becoming the central repository for these resources in the CF.

The OpenEOIR testing indicates that it meets the requirements, but the prototype did not get deployed for thorough user validation. Nevertheless informal feedback by an EOIR operator from the Aurora community indicates that it could meet some training requirements and could be of value to the CF.

6 OSS Usage Analysis

An analysis of the OpenEOIR development effort is presented in this section in terms of: the number of lines of code (LOC) required to implement the modifications; the amount of OSS that was used (or re-used); and the degree to which the prototype enabled the flexibility to access proprietary information and applications without changes to the OpenEOIR code. The tables found in Appendix B give a detailed analysis of the LOC for each component of the system. The numerical LOC values presented in this section represent highlights from those tables and are not simply based on the number of lines, but on the count of logical statements. This was felt to give a better representation as it account for programming style and does not get bloated by comments. It should be remembered that the code created for OpenEOIR was not optimized and sanitized, and that after doing so, the LOC counts for the modifications would be lower. This discussion is organized to follow the high level conceptual diagram in Figure 15 and covers the Host and the IG separately. As the ShaderManager plugin was developed under its own project it is presented within a separate sub-section.

The OpenEOIR is built from more than 1.3 Million LOC, yet only 1400 LOC were developed in this project. The developed code represents a mere 0.1% of the total. These figures do not account for either the OSS configuration files or the terrain and entity models, for which a LOC count could not be compiled.

6.1 OpenEOIR Host

The functionalities added to the core of the OpenEaagles framework was completed with additions representing approximately 0.3% of the total LOC count of OpenEaagles. These modifications were concentrated in two logical section of the framework. The CIGI class structure required an addition of approximately 6% and the Player class structure 5%. These modifications also included the addition of support for the definition of some parameters in EDL which account for a large overhead. The support for the EDL enables the configuration of entity behaviour without changes to the application code and was considered to be worth the effort. The modifications to the framework were submitted to the OpenEaagles team, and will be included in the project development trunk in the future.

The development of the OpenEOIR Host extension to OpenEaagles reused the functionalities provided by an OSS example and was supplemented by an increase of 485 LOC representing 26% of its initial size. The example already provided the basic ownship control and update functionality required to execute a simulation. The new code required was solely for the implementation of the sensor commands to be sent via CIGI.

The DIS interface supported by OpenEOIR permits the interaction between a blend of proprietary and OSS applications. Also, the use of CIGI permits the replacement of the OSS IG developed using MPV by a 3rd party

proprietary application, but the level of effort required for such migration was not compiled as licences could not be obtained.

6.2 OpenEOIR IG

MPV already provided the functionalities required to implement an OTW display. Some additions were made to the framework in order to support the functionalities implemented within OpenEOIR but most of the work was done in implementing the SensorOverlay and SensorEffects plugins, which extended MPV's functionality so to create a specific sensor.

The changes required to the core of MPV accounted for only 63 LOC. The effort was concentrated on support for RTT which accounted for 48 LOC, and the majority of these (36 LOC) were reused from the osgPPU project. As a result, only 27 LOC were developed for this section of the prototype, and this represents less than 0.1% of MPV code base.

The SensorEffects, including its control, were created solely for the OpenEOIR prototype. In total, 797 LOC were required for this plugin. Of these, 560 were reused from osgPPU leaving only 237 new lines, which accounts for a little more than 42% of the plugin. The implementation of SensorEffect required more code to be created than any other section of the IG or Host, and the complexity of the implementation was also the highest. Understandably, this is where the most implementation time was spent as well.

Finally, the SensorOverlay functionality was a new addition to the MPV framework as well. In the development of the plugin, 120 LOC from osgPPU

were reused and 320 sensor-specific LOC were added. The complexity of the coding required for the overlay was very simple when compared to the sensor effects. The code was largely a replication of code already provided by osgPPU, with minor changes such as the assignment of the correct values.

The OpenEOIR IG enabled the use of a blend of OSS and proprietary databases by supporting the widely accepted OpenFlight and TerraPage formats. The different scenarios used to test OpenEOIR in section 5.5 also demonstrated that both OSS and proprietary databases can be used simultaneously.

6.3 ShaderManager Plugin

This analysis of the ShaderManager accounts for more than the implementation of the plugin itself by also analysing its use by other plugins.

Merging the Delta3D ShaderManager architecture with MPV was a relatively simple task once the time and effort had been invested to understand its design pattern. As shown in Appendix B, only 8 LOC were required to incorporate the ShaderManager. The plugin effectively reuses approximately 1700 LOC, and the 8 new LOC represents an impressive 0.4% of the total. Another 39 LOC were needed to create a parser compatible with the definition file structure used by MPV. The original Delta3D ShaderManager uses an XML definition structure instead of the structure used by MPV and this XML functionality was not removed from the ShaderManager plugin developed for OpenEOIR. If MPV was to migrate to a XML structure, the ShaderManager would only need to change one function call to enable this capability. The

ShaderManager plugin also enables an OSS project to access proprietary information by decoupling the application from the shaders. Using the configuration file, one can easily replace the use of an OSS shader file with one that is proprietary with absolutely no modification to the application. For some application, this could be considered a valuable feature.

The effectiveness of the new MPV ShaderManager plugin is significant when compared to direct shader handling. This effectiveness was measured in term of LOC required to implement shader functions within MPV, but no tests were conducted to measure to effect on the frame rate. When running the application, the performance appeared to be equivalent. The difference in coding required to implement shaders within an application is noteworthy, the code analysed here can be found in Appendix D: Code Excerpts. When using the shader manager plugin, the developer requires only 5 LOC but direct shader handling requires 17 LOC. This saving is possible due to the decoupling of the definition of the parameters from the application and moving the definitions into a separate file as found in Figure 25 of Appendix D. When implementing more complex effects, the savings margin would be increased.

The possibility of re-loading the shaders during application run-time helped considerably when fine tuning the GLSL code. This feature was used extensively when creating the temperature shader code. Although the ShaderManager has not been included in the MPV trunk, it has been added as a branch on their server (CIGI, 2008). It will remain branched until further testing is

completed, at which point it will be integrated with the trunk, where it will be downloaded by default with the project.

7 Analysis of OSS Potential

The potential of OSS as demonstrated by the development of OpenEOIR is presented in this section. An analysis of the effort required for OpenEOIR's implementation is presented first. This is followed by a proposition of the potential for further development of the prototype. Finally, an indication of the level of interest raised by OpenEOIR is given. This section shows that with relatively little effort it is possible to implement an extendible functioning prototype that raises some interest.

Even though the author is neither a simulation practitioner nor a professional software developer, the OpenEOIR prototype was created with only six months effort. The prototype is built entirely from existing OSS projects, and the necessary modifications amount to 0.1% of the total LOC. The development required relatively little manpower by combining and connecting different OSS projects and only creating new code to address the specifics of the sensor being simulated. The development effort concentrated on the implementation of the sensor effects in the IG, and the work required for the implementation of the Host was much smaller in comparison. In order to make OpenEOIR extendible and support the emulation of different sensors, additional effort was required to enable the use of external configuration files. With the prototype being released as OSS, the software can be fine tuned for aspects that are not supported in the configuration files. This effectively negates the possibility of lock-ins and lock-outs while gaining from inputs from the rest of the community. For example,

nearing the completion of this research, OpenEaagles released a new version of its CIGI interface implemented using the CCL. This is the first step in modernizing the interface to support the newer versions of the ICD and enabling newer more advanced functionalities which could be exploited by the OpenEOIR project in the future. This was done with no effort emanating from this research, and shows that new functionality can be gained with no investment by using OSS improvements.

The main areas of future expansion for OpenEOIR are support for HLA and the implementation of an IOS. These two areas can be developed further with the use of OSS projects. The prototype currently implements a DIS interface, but OSS projects such as Portico (Gu, 2007) can provide infrastructure for migration to HLA. The potential implementation of an IOS could be facilitated by the existence of relevant OSS projects. Examples of these are: WorldWind (NASA, 2009) and TerrainView (ViewTec, 2009), which are open source applications similar to Google Earth (Google, 2009); and OpenVBMS (AFRL, 2009), which soon will be the first OSS CGF[4]. These projects provide basic components that can be extended for use as an IOS, and this could be done with little modification to connect them to OpenEOIR. Successful usage of WorldWind as an IOS was accomplished for the RAPTOR project (CogSim_Technologies, 2006).

[4] OpenVBMS is a proprietary CGF developed by the AFRL and is currently in limited trial distribution before complete OSS release.

Though no direct feedback was collected from potential users or from the Aurora community; it was surprising to see that with no publicity or marketing, the OpenEOIR project ranked 4065 out of the more than 175000 hosted projects according to the SourceForge general activity rankings at the end of May 2009. The prototype was initially released in January 2009, and within four months started to get unsolicited interest. The Aurora community is currently investigating options to have an interim EOIR capability for their legacy OMS and have expressed interest in assessing the potential of the OpenEOIR prototype (Furlong & Dieter, 2009).

8 CONCLUSION

This research investigated the potential and applicability of the use of open source software for the development of simulation-based training environments. This was done by developing a prototype electro-optic and infrared simulation which could be used in a training environment. No open source software that achieved all of the requirements was found to already exist. The resulting prototype is constructed entirely from an OSS base, and required the development of approximately 0.1% of the total number of lines of code. This project and the modifications to existing projects have in turn been released as open source. The conclusions drawn from this research as well as the contributions that helped reach these conclusions are summarized first. This is followed by a discussion of the limitations of this research and of the OpenEOIR prototype, and then suggestions for future research in the field.

8.1 Conclusions

The following conclusions were reached during this research:

1. The design and implementation of the OpenEOIR prototype (recall section 5) as well as the state of the art review (recall section 3.2) demonstrate that the necessary OSS components needed to construct synthetic training environments are not only available, but mature and stable. OpenEOIR is built from over 1.3 million OSS LOC, yet only 1400 LOC, less than 0.1% of the total, were developed for the project. It ensues that OSS in this field is in

such a state that it could and should be considered by the CF when developing and/or acquiring simulation-based training systems.

2. Commercial EOIR simulation solutions share a common set of basic features and are largely differentiated by sensor fidelity and sensor fusion. OpenEOIR has shown that the basic functionalities were implemented from OSS, and that specific differentiating features were added easily for the Aurora sensor. Furthermore, the integration of proprietary information was demonstrated. OpenEOIR shows that it is possible to map the domain of EOIR simulation, and potentially broader applications of simulation-based training, to the OSS innovation model shown in section 3.1.1, Figure 3. In this respect, OpenEOIR might inject the innovative seeding effect described in the same section. Therefore, the EOIR simulation domain is poised for OSS innovation.
3. OpenEOIR has been shown to easily accept format-compliant information from proprietary sources in order to provide more realistic scenarios, and that the quality of the visual output provided to the users is tied to the quality of the databases supplied to the IG. By contracting the development of models and databases for the CF, or creating these in house, instead of buying proprietary versions, the CF would be able to reuse these items across the organization instead of being limited to one simulation system. This could potentially reduce the overall cost of training systems.
4. OSS is supported in a much stronger fashion within the US Government than within the GoC as shown in section 3.1.3. The release of US DoD simulation

frameworks as OSS (such as discussed for OpenEaagles in section 3.2.6) indicates that great benefit can be gained from such activity, and that Canada should investigate the possibility of doing the same. The RAPTOR CF proprietary framework, which was developed to test and investigate the CIGI protocol, is a valid candidate for such activity.

8.2 Contributions

The contributions made from this research are four fold. First, the design and implementation of an OSS synthetic sensor operator training environment is provided in the form of the OpenEOIR prototype. Second, the availability, maturity, and stability of different OSS projects in the field of M&S demonstrates the relevance of such technology. Thirdly, the ecosystem generated by the development of the prototype contributed to the synergy of OSS in the field of simulation based training and eased the implementation of OpenEOIR. Lastly, the projects used to develop OpenEOIR gained from the return of all improvements which ensued from the development of the prototype and they are now better tooled for future sensor application development.

8.3 Limitations

The potential for the use of OSS in simulation-based training system has some limitations. First, OpenEOIR only implements a representative subset of an EOIR operator training system. The host implements 33% of the functionalities found on a real sensor, and no IOS was developed. The IG

implements 45% of a complete HUD and 40% of the sensor effects found on commercial implementations, but did not address environmental effects or terrain IR representation. The focus was kept to a simple vehicle IR representation. As well, the LOC calculations did not account for the difficulty level of the implementation of the statements. This research is focused on the technical aspect of OSS in the context of M&S systems and does not provide a complete business case including financial impact analysis. The prototype code was released on SourceForge near the end of the research, and having done so earlier in development would have helped in monitoring the general interest in the prototype. This interest may have provided some validation of the research. Finally, OpenEOIR was not formally fielded for test by the community and empirical data was not gathered. Again, this testing and feedback this would have helped to validate the perceived value of the system.

8.4 Future research

As OSS in M&S is relatively new, much research in this field remains to be done. A few suggestions for further research related to this research are provided in the following paragraphs.

The current OpenEOIR prototype could be completed into a more robust and feature-rich OSS based training system. While doing so, a more rigorous analysis than the one used for this research could be applied, perhaps using a weighed bias based on complexity of implementation and development time to

give a refined estimate of the effort. A study on the related training needs of the forces could provide the requirements, and an empirical study of how the requirements are fulfilled could follow. The training requirements description from the Aurora training squadron could be used as the basis to construct scenario assessment feedback forms. This would provide a strong case study on how OSS can play a role in the training of Canadian war-fighters.

The CF created its own proprietary RAPTOR framework to test and investigate the CIGI protocol. The analysis of what is required for the release of the RAPTOR framework as OSS in a similar fashion as what was done with Eaagles could provide opportunities for many research projects ranging from business case studies to longitudinal analysis of OSS technology acceptance within the CF and more. Such research could provide great gain for both academia and the CF. For example, if the RAPTOR would have been available as OSS, it would have been strongly considered for basis upon which the OpenEOIR prototype be built.

A longitudinal analysis of different OSS applications in the M&S field could reveal whether the trend which was observed for OpenEaagles applies to other proprietary software released as OSS. Though it appears that OpenEaagles and CIGI have reached the critical mass described by Scacchi (2007), empirical studies identifying how to reach that tipping point could be pursued. Companies could then use the knowledge gained by the research to enable them to reach such critical mass and potentially have a more successful OSS based enterprise.

The analysis matrices provided by (Bhattacharya et al., 2007) could be applied to a variety of OSS simulation projects in order to investigate which licence(s) are most suitable for simulation-based training system development. No such study currently exist in this field and one might show that even though LGPL and GPL are the most commonly found licence in use for M&S development, they may not be the most suitable and projects may gain greater general acceptance from a less restrictive licence.

The development of more technical OSS projects like the OpenEOIR prototype would help the advancement of the status of OSS in the M&S community. These would reinforce OSS's value and potential by providing more case studies as well as support the pyramid of innovation (recall section 3.1.1).

A longitudinal study of M&S placing OSS in the innovation/disruption cycle using Christensen's disruptive innovation model could provide insights as to how vendors and companies can better position themselves. It would also provide empirical data to support the claim made by the Delta3D team in section 3.2.6 related to innovative seeding.

Lastly, the ecosystem of M&S frameworks such as OpenEaagles could be mapped as per the technique shown in (Bailetti, 2009) and studied in order to help identify niche areas that could be exploited and researched. A sample preliminary sketch of such a study is included in Appendix F: Mapping of Ecosystem.

9 REFERENCES

Acron. 2009. Acron Engineering Simx Website. Extracted from: http://www.acroneng.com/SimX.htm on 10 January 2009.

AFEI. 2009. Dod Open Technology Conference Website. Extracted from: http://www.afei.org on 12 January 2009.

AFRL. 2009. Openvbms Project Website. Extracted from: http://openvbms.org/index.html on 10 April 2009.

Bailetti, T. 2009. Business Ecosystems: A New Form of Organizing Creative Individuals Worldwide, Faculty of Carleton University Speaker Series. MaRS Collaboration Centre, Toronto 12 February 2009.

Barros, L. M. 2005. A Short Introduction to the Basic Principles of the Open Scene Graph. Extracted from: http://www.stackedboxes.org/~lmb/OSG/ASIttBPoOSG--02005-10-23.pdf on 10 Septembre 2008.

Beard, A., & Kim, H. 2007. A Survey on Open Source Software Licenses. Journal of Computing Sciences in Colleges (JCSC), 22(4): 205-211.

Bhatia, S. K., & Lacy, G. M. 1999. Infra-Red Sensor Simulation, Interservice/Industry Training, Simulation and Education Conference (I/ITSEC). Orlando, FL November 1999.

Bhattacharya, J., & Suman, S. 2007. Analysis of Popular Open Source Licenses and Their Applicability to E-Governance. Paper presented at the 1st international conference on Theory and practice of electronic governance, Macao, China.254-257.

Bjorgvinsson, T., & Thorbergsson, H. 2007. Software Development for Governmental Use Utilizing Free and Open Source Software. Paper presented at the Proceedings of the 1st international conference on Theory and practice of electronic governance, Macao, China.133-140.

Blasband, C., Bleak, J., & Schultz, G. 2004. High Fidelity, Physics-Based Sensor Simulation for Military and Civil Application. Sensor Review, 24(2): 151-155.

Boeing. 2006. Multi Purpose Viewer (Mpv) Developer Documentation. Extracted from: http://cigi.sourceforge.net on 19 August 2008.

Brennan, J. 2008. Email Concerning Oss Eoir Sim. In P. Castonguay (Ed.). Stittsville, ON 06 June 2008: John Brennan, Manager of Program Development Modelling & Simulation, CAE Professional Services.

Camber. 2007. Huntir Official Website. Extracted from: http://www.huntirgame.com/ on 15 August 2008.

Cason, R. R. 2009. Industry on a Roll: Double-Digit Growth Predicted for M&S Training over 10 Years. Training and Simulation Journal (TSJ), 9(6): 38-40.

Castonguay, P. 2009a. Mpv Shadermanager Plugin Project. Extracted from: http://mpvshaderplugin.sourceforge.net/ on 15 February 2009.

Castonguay, P. 2009b. Openeoir Project Website. Extracted from: https://sourceforge.net/projects/openeoirsensor/ on 25 May 2009.

CDR. 2006. Xwave to Build Trainer for Auroras. Canadian Defence Review, 12(5): 1.

Charpentier, R., & Carbone, R. 2004. Free and Open Source Software : Overview and Preliminary Guidelines for the Governement of Canada. In D. R. a. D. C. Valcartier (Ed.): 1-48.

Chernysh, A. 2007. Examining the Characteristics of Using Open Source Software Products in the Military: The Case of the Department of National Defence, Technology Innovation Management (TIM) Gate 1 Thesis Presentation. Ottawa, ON 16 Novembre 2007.

Chung, E. 2009. An Open Door for Open Source? Federal Government Puts out a Call for Information on Free Software, CBC News, 12 February 2009.

CIGI. 2008. Common Image Generator Interface Website. Extracted from: http://cigi.sourceforge.net on 14 April 2008.

CogSim_Technologies. 2006. Collaborative Simulation Development Project Final Report: 33. Ottawa, ON 23 April 2006: Canadian Forces Air Warefare Center (CFAWC).

Croatia, G. o. t. R. o. 2008. Open Source Software Policy: Government of the Republic of Croatia. http://www.e-hrvatska.hr/sdu/en/Dokumenti/StrategijeIPrograma/categoryParagraph/04/document/Open_Source_Software_Policy.pdf

CSP. 2009. Combat Simulator Project. Extracted from: http://csp.sourceforge.net/wiki/Main_Page on 10 June 2009.

Cuéllar, L. E. 2005. Open Source License Alternatives for Software Applications: Is It a Solution to Stop Software Piracy? Paper presented at the 43rd annual Southeast regional conference, Kennesaw, Georgia.269-274.

Darken, R., McDowell, P., & Murphy, C. 2005. Open Source Game Engines: Disruptive Technologies in Training and Education. Paper presented at the The

Interservice/Industry Training, Simulation & Education Conference (I/ITSEC), 28 November - 1 December 2005, Orlando, FL.

Delta3D. 2008. Delta3d Web Page. Extracted from: http://www.delta3d.org on 20 August 2008.

DISA. 2009. Disa Delivers Dod Collaborative Software Development. Extracted from: http://www.disa.mil/news/pressreleases/2009/forge_042009.html on 20 May 2009.

DND. 2008. Aco 8000-2: Air Force Modelling and Simulation Policy – Synthetic Environments: CF Aerospace Warfare Centre detachment Ottawa (CFAWC (O)).

DoD. 1995. Modeling and Simulation (M&S) Master Plan. In D. O. Defence (Ed.).

Durham, L., & Phelps, B. 2003. Common Image Generator Interface (Cigi) Study Group Meeting, Simulation Interoperability Standad Organization (SISO) 2003 Fall Simulation Interoperability Workshop (SIW). Orlando, FL.

Durham, L. W. 2006. Interface Control Document for the Common Image Generator Interface (Cigi) V3.27 April 2006: The Boeing Company.

Franco, E. 2009. Rfi - No Charge Licensed Software: Public Works and Government Services Canada. http://www.merx.com/English/SUPPLIER_Menu.Asp?WCE=Show&TAB=1&POR

TAL=MERX&State=7&id=PW-%24%24EE-015-18733&FED_ONLY=0&hcode=Au64x22Vv9pVNE3IKtFp3Q%3d%3d

FreeSoftwareFoundation. 2008. What Is Copyleft? Extracted from: http://www.gnu.org/copyleft/copyleft.html on 8 March 2009.

FSF. 2008. Free Software Foundation Website. Extracted from: http://www.fsf.org on 8 March 2009.

Furlong, J., & Dieter, P. 2009. Email: Investigating Open Sourcesimulation Projects. In P. Castonguay (Ed.). Ottawa, ON: Capt Furlong, Senior Program Manager OMS.

Capt Dieter, Directorate of Air Requirement M&S.

Garnett, G. L., Hénault, R. R., & Leggat, L. J. 2000. Creating the Cf of 2020: Concept Development and Experimentation and Modelling and Simulation (Discussion Paper Approved by Vcds, Dcds and Assistant Deputy Minister (Science & Technology)).

Givens, B. 2001. Siso Software Products Study Group (Siso-Ssp-Sg) 2001 Final Report. In T. McLean (Ed.): 21March 2001: Simulation Interoperability Standards Organization (SISO).

Givens, B. R. 2000. Open-Source Rti: Position for and Against, 2000 Spring SISO Simulation Interoperability Workshop. Orlando, FL.

Glass, J., Paterson, D., Andrews, W., Griffin, A., & Seagull, B. 2002. Nasmp: Standards Development in Support of Navy Aviation Simulator Development. Paper presented at the SISO 2002 Fall Simulation Interoperability Workshop (SIW), 8-13 September, 2002, Orlando, FL.

GoC. 2004. Open Source Software Position. Ottawa, ON 29 June 2004: Treasury Board of Canada Secretariat (TBSC). http://www.tbs-sct.gc.ca/fap-paf/oss-ll/position-eng.asp

Golden, B. 2008. Open Source in the Enterprise (Excerpt): O'Reilly Radar Reports.

Google. 2009. Google Earth Product Website. Extracted from: http://earth.google.com/ on 26 June 2009.

Goth, G. 2005. Open Source Business Models: Ready for Prime Time. IEEE Software, 22(6): 98-100.

Gu, P. 2007. Portico Project Website. Extracted from: http://www.porticoproject.org on 25 September 2008.

Hannon, B. 1997. The Use of Analogy in Biology and Economics: From Biology to Economics, and Back. Structural Change and Economic Dynamics, 8(4): 471-488.

Hao, X., Zhengang, Z., Chunpei, L., & Zhuo, D. 2008. The Study on Innovation Mechanism of Open Source Software Community. Paper presented at the Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference.1-5.

Harrison, L. T. 2003. Improved Pc Flir Simulation through Pixel Shader, IMAGE 2003 Conference. Scottsdale, Arizona 14-18 July 2003.

Herz, J. C., Lucas, M., & Scott, J. 2006. Open Technology Development Roadmap. In U. D. D. A. S. Concepts (Ed.): 1-79.

Hill, F. 2008. SISO Forum Thread on OSS Statement. http://discussions.sisostds.org/Default.asp?action=9&read=43550&fid=113&BoardID=2#53321

Hodson, D. 2008. Openeaagles, an Open Source Simulation Framework, American Institute of Aeronautics and Astronautics (AIAA) Modeling and Simulation Newsletter, Vol. 1: 3-4.

Hodson, D. 2009. Telephone Conversation. In P. Castonguay (Ed.). Stittsville, On 2 March 2009.

Hodson, D., & Buell, C. 2006a. Openeaagles Simulation Framework Utilizing Jsbsim. The Quarterly Newsletter For JSBSim, 3(2): 12-13.

Hodson, D. D., Gehl, D. P., & Baldwin, R. O. 2006b. Building Distributed Simulations Utilizing the Eaagles Framework. Paper presented at the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).1-10

HP. 2008. Fossology Webpage. Extracted from: http://fossology.org/ on 14 September 2008.

Hudson, D. 2009. Telephone Conversation. In P. Castonguay (Ed.). Stittsville, On 2 March 2009.

Iansiti, M., & Levien, R. 2004. Strategy as Ecology. Harvard Business Review, 82(3): 68-78.

Johnsrude, E. 2009. Email with Reference to Cigi in the Cf. In P. Castonguay (Ed.). Ottawa, ON 19 February 2009: Major Eric Johnstrude CFAWC SECO.

Joly, A., Goff, A. L., Latger, J., Cathala, T., & Larive, M. 2006. Real Time Optronic Simulation Using Automatic 3d Generated Terrain, and Validation Process, 2nd International IR Target and Background Modeling & Simulation Workshop (ITBM&S). Ettlingen, GR 26-29 June, 2006. http://www.oktal-se.fr/

JRM_Technologies. 2005. Product Technical Descriptions. Extracted from: http://www.jrmtech.com/ on 18 September 2008.

JRM_Technologies. 2008a. Asgard Datasheet. Extracted from: http://www.jrmtech.com on 10 November 2008.

JRM_Technologies. 2008b. Sensim Datasheet. Extracted from: http://www.jrmtech.com on 10 November 2008.

JRM_Technologies. 2008c. Sigsim Datasheet. Extracted from: http://www.jrmtech.com on 10 November 2008.

Justice, N. G. 2007. Opening Presentation by Program Executive Officer, Command Control and Communications Tactical (C3t), Brigadier General Nickolas G. Justice, DoD Open Conference. Vienna, VA.

Katz, W. 2001. Open Source Software Dynamics in the Dod Modeling and Simulation Market Simulation Technology Magazine, 4(3).

Latger, J., Cathal, T., Joly, A., & Goff, A. L. 2008. Improvement of the Se-Workbench Workshop for Rendering Targets, 4th International IR Target and Background Modeling & Simulation Workshop (ITBM&S). Ettlingen, GR 23-26 June 2008. http://www.oktal-se.fr/

Lechner, R., & Phelps, W. 2002. Cigi, a Common Image Generator Interface, The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC) Orlando, FL.

Liao, D., & Hennessey, B. 2006. Using Gpu-Generated Virtual Video Stream for Multi-Sensor System. Paper presented at the Enabling Technologies for Simulation Science (Proceedings of SPIE).1-6

MaK, 2009. MaK Company Website. Extracted from: http://www.mak.com/ on 24 June 2009.

McDowell, P. 2007. Using Open Source Software for Military M&S. Presentation at MOVES Open House 19 July 2007.

McDowell, P. 2008. The Delta3d Open Source Game Engine, American Institute of Aeronautics and Astronautics (AIAA) Modeling and Simulation Newsletter, Vol. 1: 6-9.

McDowell, P., Darken, R., Sullivan, J., & Johnson, E. 2006. Delta3d: A Complete Open Source Game and Simulation Engine for Building Military Training Systems. Journal of Defence Modeling and Simulation, 3(3): 143-154.

MetaVR. 2009. Metavr Comapny Website. Extracted from: http://www.metavr.com/ on 14 June 2009.

Microsoft. 2009. Microsoft Webpage. Extracted from: http://www.microsoft.com/ on 18 June 2009.

Milinkovich, M. 2008. A Practioners Guide to Ecosystem Development, Technology Innovation Management (TIM) Lectures. Ottawa, On 3 September 2008.

Miller, D. C. 2008. A Brief Overview of Siso. Extracted from: http://www.sisostds.org/index.php?tg=articles&idx=More&article=1&topics=1 on 19 September 2008.

Mitchell, T. 2009. Reassuring End Users of Open Source: The Osgeo Example. Open Source Business Resource (OSBR): 10-15.

MODSIM.org. 2005. Homepage on the Internet. Extracted from: http://modsim.org/devrim_extras/poll_piechart_scenegraph2005.png on 12 January 2008.

Moore, J. F. 1993. Predators and Prey: A New Ecology of Competition. Harvard Business Review, 71(3): 75-86.

Moore, J. F. 2006. Business Ecosystems and the View from the Firm. Antitrust Bulletin, 51(1): 31-75.

MOVES. 2009. Opendis Project Webpage. Extracted from: http://open-dis.sourceforge.net/Open-DIS.html on 18 June 2009.

Mullen, M. 2006. Why Open Technologies?, 4th DoD Open Technology Conference: Open Architecture, Open Standards, Open Source Software. Washington, DC 27 November 2006: Admiral Mike Mullen, Chief of Naval Operations.

MultiGen-Paradigm. 2008. Openflight Format Webpage. Extracted from: http://www.multigen.com/products/standards/openflight/index.shtml on 20 June 2009.

NASA. 2009. Nasa World Wind Webpage. Extracted from: http://worldwind.arc.nasa.gov/ on 12 February 2009.

O'Flynn, D. 2008. The Open-Source Identity Revolution, O'Reilley Open Source Convention (OSCON) 2008. Portland, OR 21-25 Jully 2008.

Oktal-SE. 2008. Company Presentation. Extracted from: http://www.oktal-se.fr on 10 January 2009.

OSI. 2009. Open Source Initiative Webpage. Extracted from: http://opensource.org on 12 February 2008.

Pearce, T. 2006. Potentials and Constraints for Open Source and Standards Development, Academic Night Fall 06 SISO Simulation Interoperability Workshop (SIW)12 Septembre 2006.

Pearce, T. W., & Farid, N. B. 2004. If Rti's Have a Standard Api, Why Don't They Interoperate? Paper presented at the 2004 Fall Simulation Interoperability Workshop (SIW).

Peltoniemi, M. 2006. Preliminary Theoretical Framework for the Study of Business Ecosystems. Emergence: Complexity & Organization, 8(1): 10-19.

Phelps, W. B. 2007. Common Image Generator Interface (Cigi) Standing Study Group (Ssg) 2007 Annual Report. Siso-Ref-012-2007.

Phelps, W. B. 2009. Email with Reference to Cigi Questions. In P. Castonguay (Ed.). Ottawa, ON 19 February 2009: Willard B Phelps (Manager Training System Reseach Section, The Boeing Company).

Pokorny, T. 2006. Open Source and the Hla_I Swear It's Here Somewhere, 2006 Fall Simulation Interoperability Workshop. Orlando, FL.

Pokorny, T. 2008. Participation in Open Source Presentation, Simulation Interoperability Standards Organization (SISO) Simulation Interoperability Workshop (SIW). Orlando, FL 17 Septembre 2008.

Presagis. 2009. Presagis Company Website. Extracted from: http://www.presagis.com/ on 14 June 2009.

Rao, D., Hodson, D., Jr, M. S., Johnson, C., Kidambi, P., & Narayanan, S. 2007. Desing and Implementation of Virtual and Constructive Simulation Using Openeaagles. Dayton, Ohio: The Wright State Research Institute.

Roman, P. A., & Brown, D. 2006. Constructive Simulation Versus Serious Games - a Canadian Case Study. Paper presented at the 2007 spring simulation multiconference - Volume 3, Norfolk, Virginia.217-224.

Roman, P. A., & Brown, D. 2009. Games- Just How Serious Arethey? Modeling and Simulation Analysis Center (MSIAC) Journal, 4(1): 9-20.

Sampson, A. 2005. Multi Purpose Viewer (Mpv) User's Guide: 17. St. Louis, MO 14 October 2005: The Boeing Company.

Scacchi, W. 2007. Free/Open Source Software Development: Recent Research Results and Emerging Opportunities. Paper presented at the European Software Engineering Conference. 459-468.

Seiferth, J. 1999. Open Source and These United States, AU/ACSC/94-184/1999-04. Alabama: Air Command and Staff College Air University. http://skyscraper.fortunecity.com/mondo/841/documents/index.html

SGI. 2009. Opengl Website. Extracted from: http://www.opengl.org on 1 June 2009.

SISO. 2008a. Simulation Interoperability Standards Organisation Official Website. Extracted from: http://www.sisostds.org on 20 September 2008.

SISO. 2008b. Special Session on Open Source Simulation, Simulation Interoperability Workshop (SIW) Fall 2008 Agenda: 114. Orlando, FL 15-19 September 2008.

Skerrett, I. 2008. Building Technology Communities, Technology Innovation Management (TIM) Lectures. Ottawa, ON June 4, 2008.

Smith, M. I., Murray-Smith, D. J., & Hickman, D. 2007. Mathematical and Computer Modeling of Electro-Optical System Using a Generic Model Approach. Journal of Defence Modeling and Simulation, 4(1): 3-16.

SourceForge. 2009. Sourceforge Statistics Website. Extracted from: http://sourceforge.net on 22 June 2009.

Stacy Avery, B. 2008. The Heterogeneous World of Proprietary and Open-Source Software. Paper presented at the Proceedings of the 2nd International Conference on Theory and Practice of Electronic Governance, Cairo, Egypt.232-238.

Steinman, J. S. 2009. Open Software Initiative for Parallel and Distributed Modeling & Simulation (Osi-Pdms) Study Group Final Report: 1017 March 2009: SISO. file:///C:/Documents%20and%20Settings/Patrick/My%20Documents/References/SISO/OSI_PDMS%20FR.doc

Subr, R. 2007. Subrscene Image Generation Software User Manual Version 7-15-2007. Extracted from: http://www.subrscene.org/ on 10 Jully 2008.

Subr, R. 2009. Subrscene Home Page. Extracted from: http://www.subrscene.org/ on 10 January 2009.

Sugisaka, M., & Faizal, A. 2006. Computer Simulator for Training Operators of Infrared Camera. Paper presented at the SICE-ICASE, 2006. International Joint Conference.5290 - 5295.

Tev, A. 2008. Osgppu Project Webpage. Extracted from: http://projects.tevs.eu/osgppu/ on 10 September 2008.

TSJ. 2008. Tsj 2009 Global Company Directory. Training and Simulation Journal (TSJ), 9(5): 39-52.

UK, C. O. o. G. C. 2009. Open Source Software – Use within Uk Government: United Kingdom Cabinet Office of Government Commerce. http://www.cio.gov.uk/transformational_government/open_source/policy.asp

VCDS. 2006. Daod2010-1, Modelling and Simulation Management. In DND (Ed.). http://www.smafinsm.forces.gc.ca/dao-doa/2000/2010-1-eng.asp

ViewTec. 2009. Viewtec Company Website. Extracted from: http://www.viewtec.net/ on 10 January 2009.

Walker, K. 2005a. Common Visuals Interface Takes Off. Training and Simulation Journal (TSJ), 6(4): 24.

Walker, K. 2005b. Image Generators Field of Dreams. Training and Simulation Journal (TSJ), 6(4): 50.

Witte, T. 2008. Survey of Terrain Visualization Software: U.S. Army Topographic Engineering Center's (TEC).

Wu, D. 2007. An Empirical Study of the Effects of Open Source Adoption on Software Development Economics. Carleton University, Ottawa, ON.

10 Appendix A: OSI Criteria

Table 6: OSI OSS Criteria (OSI, 2009)

Criteria	Definition
Free Redistribution	The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
Source Code	The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.
Derived Works	The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
Integrity of the Author's Source Code	The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
No Discrimination Against Persons or Groups	The license must not discriminate against any person or group of persons.
No Discrimination Against Fields of Endeavor	The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
Distribution of License	The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
License Must Not Be Specific to a Product	The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
License Must Not Restrict Other Software	The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
License Must Be Technology-Neutral	No provision of the license may be predicated on any individual technology or style of interface.

11 Appendix B: Lines of Code

Table 7: IG Lines of Code Detail

Status	Portion of MPV	New lines	Reused	Function
update	common			Temperature
	Entity.h	1		
	Entity.cpp	1		
		2	***0***	***2***
update	commonOSG			LOD
	MiscOSG.h	1		
	MiscOSG.cpp	10		
		11	***0***	***11***
update	pluginRenderCameraCigiSDLOSG			RTT Capability
	PluginRenderCameraOSG.h	2		
	PluginRenderCameraOSG.cpp	4		
	ViewportWrapper.h	3		
	ViewportWrapper.cpp	3	36	osgPPU
		12	***36***	***48***
update	pluginRenderEntitiesOSG			
	pluginRenderEntitiesOSG.cpp	**1**	**0**	LOD
update	pluginRenderTerrainOSG			
	pluginRenderTerrainOSG.cpp	**1**	**0**	LOD
MPV Core SubTotal		**27**	**36**	***63***
new	pluginShaderManager			
	mathdefines.h		59	Delta3D
	PluginShaderManager.cpp	3	3	New + Delta3D
	PluginShaderManager.h	5		New + Delta3D
	ShaderManager.cpp		277	Delta3D
	ShaderManager.h		73	Delta3D
	ShaderGroup.cpp		58	Delta3D
	ShaderGroup.h		35	Delta3D
	ShaderParameter.cpp		40	Delta3D
	ShaderParameter.h		75	Delta3D
	ShaderParamFloat.cpp		39	Delta3D
	ShaderParamFloat.h		17	Delta3D
	ShaderParamInt.cpp		31	Delta3D
	ShaderParamInt.h		17	Delta3D
	ShaderParamOscillator.cpp		112	Delta3D
	ShaderParamOscillator.h		93	Delta3D
	ShaderParamTexture.cpp		41	Delta3D
	ShaderParamTexture.h		66	Delta3D
	ShaderParamTexture2d.cpp		116	Delta3D
	ShaderParamTexture2d.h		18	Delta3D
	ShaderProgram.cpp		119	Delta3D

	ShaderProgram.h		61	Delta3D
	ShaderParser.cpp	33	294	New + Delta3D
	ShaderParser.h	6	51	New + Delta3D
	SubTotal	***47***	***1695***	***1742***
new	pluginRenderEOIROverlayOSG (osgPPU)	**320**	**120**	Screen Overlay (Text)
new	pluginRenderEOIRSensorOSG (osgPPU & Delta3D ShaderManager)	**237**	**560**	Sensor Effect
	GrandTotal	**631**	**2411**	

Table 8: Lines of Code Analysis

Framework (OpenEaagles)

	OpenEaagles Total				OpenEaagles CIGI			
	Original	New	Diff	%	Original	New	Diff	%
Tot Lines	169434.0	169907.0	473.0	0.3	3761.0	4012.0	251.0	6.7
No Comments	127075.5	127345.3	269.8	0.2	2869.6	2993.0	123.3	4.3
Statement	76133.0	76370.0	237.0	0.3	1991.0	2124.0	133.0	6.7
Average	124214.2	124540.8	326.6	0.3	2873.9	3043.0	169.1	5.9

	EOIR_Host (based on OE example)				OpenEaagles Player			
	Original	New	Diff	%	Original	New	Diff	%
Tot Lines	3397.0	4222.0	825.0	24.3	4349.0	4571.0	222.0	5.1
No Comments	2615.7	3284.7	669.0	25.6	3048.6	3195.1	146.5	4.8
Statement	1696.0	2181.1	485.1	28.6	2241.0	2345.0	104.0	4.6
Average	2569.6	3229.2	659.7	26.2	3212.9	3370.4	157.5	4.9

Image Generator (Multi-Purpose Viewer)

	MPV Total				ShaderManager			
	Original	New	Diff	%	Delta3D	New	Diff	%
Tot Lines	96947.0	N/A	N/A	N/A	3260.0	3415.0	155.0	4.8
No Comments	60010.2	N/A	N/A	N/A	2119.0	2151.5	32.4	1.5
Statement	27560.0	27623.0	63.0	0.2	1695.6	1742.0	46.4	2.7
Average	61505.7	27623.0	63.0	0.2	2358.2	2436.2	78.0	3.0

	Overlay				SensorEffects			
	osgPPU	New	Diff	%	osgPPU	New	Diff	%
Tot Lines	N/A	N/A	N/A	N/A	1234.0	1892.0	658.0	53.3
No Comments	N/A	N/A	N/A	N/A	861.3	1322.5	461.2	53.5
Statement	120.0	440.0	320.0	72.7	560.0	797.0	237.0	42.3
Average	120.0	440.0	320.0	72.7	885.1	1337.2	452.1	49.7

osgPPU

	osgPPU Total			
	Original	New	Diff	%
Tot Lines	11512.0	11524.0	12.0	0.1
No Comments	8392.2	8401.0	8.7	0.1
Statement	4639.0	4643.0	4.0	0.1
Average	8181.1	8189.3	8.2	0.1

OSS Projects Used but not Modified

	Lines	No Comments	Statement
JSBSim	64574.0	45201.8	29491.0
Boost	2121178.0	1754214.2	884508.0
OSG	418362.0	372342.2	190174.0
SDL	149223.0	118632.3	69155.0
CIGI CCL	43707.0	30420.1	15076.0
Total	2732470.0	2320810.5	1188404.0

12 Appendix C: Picture-In-Picture Prototype

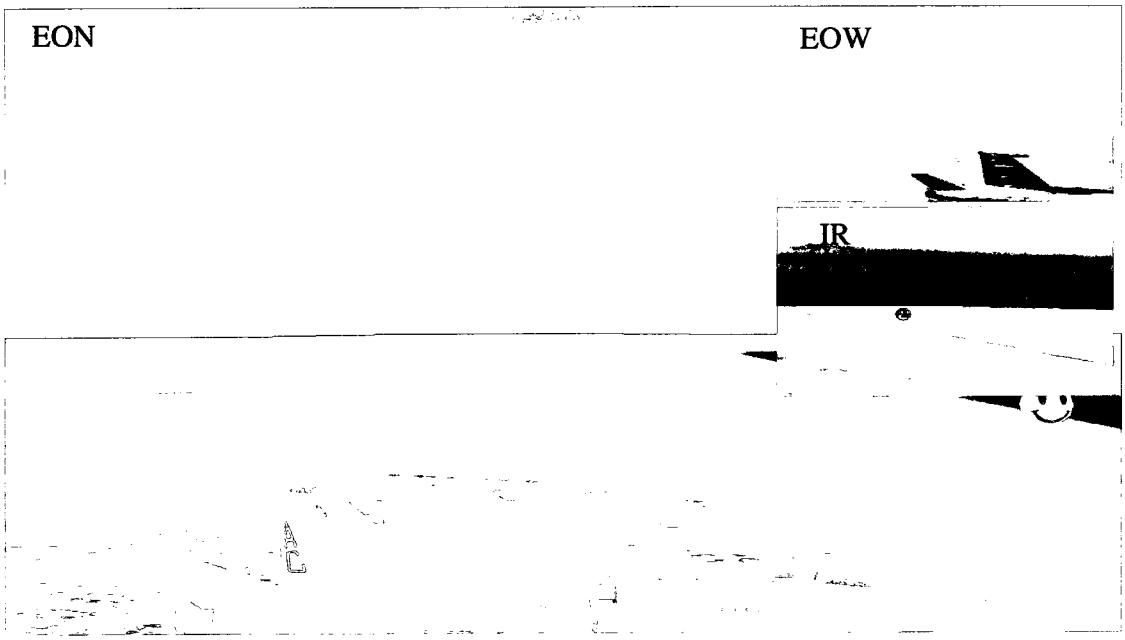


Plate 7: Non-CIGI Multi-Channel View (EON, EOW, IR)

13 Appendix D: Code Excerpts

```
osg::Texture2D* ViewportWrapper::createRenderTexture
                        ( float width, float height, bool depth ){
   static int id = 1;
   // Required parameters
   osg::Texture2D* newTexture = new osg::Texture2D;
   newTexture->setTextureSize(width, height);
   newTexture->setFilter(osg::Texture2D::MIN_FILTER,
                         osg::Texture2D::LINEAR);
   newTexture->setFilter(osg::Texture2D::MAG_FILTER,
                         osg::Texture2D::LINEAR);

   // Extra non essential parameters
   newTexture->setResizeNonPowerOfTwoHint(false);
   newTexture->setWrap(osg::Texture2D::WRAP_S,
                       osg::Texture2D::CLAMP_TO_BORDER);
   newTexture->setWrap(osg::Texture2D::WRAP_T,
                       osg::Texture2D::CLAMP_TO_BORDER);
   newTexture->setBorderColor(osg::Vec4(1.0f,1.0f,1.0f,1.0f));

   // setup float format
   if (!depth)
   {
      newTexture->setInternalFormat(GL_RGBA16F_ARB);
      newTexture->setSourceFormat(GL_RGBA);
      newTexture->setSourceType(GL_FLOAT);
   }else{
      newTexture->setInternalFormat(GL_DEPTH_COMPONENT);
   }
   return newTexture;
}//createRenderTexture
```

Figure 23: CreateRenderTexture Method

```
simulation: ( Simulation
    players: {
        ownship:( SimPlayer
            type: "CP-140"                          //CP-140 Aurora
            side: blue                              // Side
            // Starting position (relative to ref point)
            initXPos: ( NauticalMiles -6 )
            initYPos: ( NauticalMiles -3 )
            initHeading: ( Degrees 0.0 )            // Initial heading
            initAlt: ( Feet 22000 )                 //Initial altitude
            initVelocity: 0                         // Initial velocity
        )   // End ownship

      tank1:( Tank
                  side: red
                  type: "T-72"
                  id: 2000
                  … position info …
                  internalTemp: -273.15             // Initial temp
                  tempChangeRate: 30
                  //minTemp: -273.15                //-273.15 is default
                  maxTemp: 300
      )  // End Tank1

      ffg1:( Ship
                  side: blue
                  type: "CanadianFFG"
                  id: 3000
                  … position info …
                  internalTemp: 0
                  tempChangeRate: 30
                  minTemp: -50
                  //maxTemp: 1000                   //1000 is default
      )    // End FFG1
   }  // End player list
)  // End simulation
```

Figure 24: EDL Scenario Example

```
Shaders {
  Program {
     name = "temperatureEffect";
     viewID = 12;
     fragment = "temperature.glsl";

     integer {//Texture
       name = "texUnit0";
       defaultValue = 0;
     }
     float { //Temperature [-273.15 .. 1000]
       name = "temperature";
       defaultvalue = -270.0;
     }
     float { //Temp trasparency [0.0 .. 1.0]
       name = "tempAlpha";
       defaultvalue = 1.0;
     }
  }//Program Temperature Effects
}//Shader definition
```

Figure 25: Shader Definition

```
// Get the temperature prototype
ShaderProgram* sp =
  shaderManager->FindShaderPrototype("temperatureEffect", "eoir");

// Load it
ShaderProgram* tempShader =
  shaderManager->AssignShaderFromPrototype( *sp, *node );

// Retreive pointer to temperature parameter
ShaderParamFloat* entityTemp = dynamic_cast< ShaderParamFloat*>
 (tempShader->FindParameter("temperature"));

// Store it for later update
temperatureMapSM[ent->GetEntityID()] = entityTemp;

// Set the initial value
entityTemp->SetValue( ent->GetTemp() );
```

Figure 26: Shader Management Using Plugin

```
//Setup temperature shader
osg::Shader *fragTemp = osg::Shader::readShaderFile(
      osg::Shader::FRAGMENT, "../data/shaders/temperature.glsl" );
osg::Program *progTemp = new osg::Program;
progTemp->setName( "TemperatureShader" );
progTemp->addShader( fragTemp );

osg::StateSet* stateSet = node->getOrCreateStateSet();

//Assign it to the stateSet and node
stateSet->setAttributeAndModes( progTemp );
node->setStateSet( stateSet );

//Declare uniform used to control shader and set default values
//Texture
osg::Uniform *texUnit0 = new osg::Uniform( osg::Uniform::SAMPLER_2D,
      "texUnit0" );
texUnit0->set(0);
stateSet->addUniform( texUnit0 );

//Temperature
osg::Uniform *temperature = new osg::Uniform( "temperature", 0.0f );
temperature->setType( osg::Uniform::FLOAT );
temperature->set( ent->GetTemp() );
stateSet->addUniform( temperature );
temperatureMap[ent->GetEntityID()] = temperature;

//Temperature transparency
osg::Uniform *tempAlpha = new osg::Uniform( "tempAlpha", 1.0f );
tempAlpha->setType( osg::Uniform::FLOAT );
      stateSet->addUniform( tempAlpha );
```

Figure 27: Direct Shader Management

```
void main()
{
  vec4 color = texture2D(texUnit0, gl_TexCoord[0].xy);
  float gcolor=(0.299 * color.r)+(0.587 * color.g)+(0.184 * color.b);

  gl_FragColor = vec4(gcolor, gcolor, gcolor, color.a);
}
```

Figure 28: EON Shader Program

```
void main()
{
      vec4 color = texture2D(texUnit0, gl_TexCoord[0].xy);
      float gcolor = (0.299 * color.r) + (0.587 * color.g) + (0.114 *
color.b);
      if(whiteHot == 0)
            gcolor = 1.0 - gcolor;
      gl_FragColor = vec4(gcolor, gcolor, gcolor, color.a);
      }
```

Figure 29: Infrared Shader Program

```
/*
 *  0 Deg Kelvin (absolute 0) is -273.15 Deg C
 *  The max temp for our use was set to 1000 Deg C (for any entity)
 *  For simplicity we use a linear progression
 *
 *  If the polarity is white the hotter temperatures will show white,
 *  if not they will show black
 */

uniform sampler2D texUnit0;
uniform float temperature; //degrees Celcius
uniform float tempAlpha;   //factor used for temperature transparency
//uniform int whiteHot;      //sensor polarity at the scene level...

float maxTemp = 1000;    //Default max
float minTemp = -273.15; //Absolute ZERO

float factor = 1; //[0..1] will be overwritten

void main()
{
      factor = (temperature - minTemp)/(maxTemp - minTemp);

      vec4 color = texture2D(texUnit0, gl_TexCoord[0].xy);
      float gcolor = (factor * color.r) +
                        (factor * color.g) + (factor * color.b);

      gl_FragColor = vec4(gcolor, gcolor, gcolor, tempAlpha *
color.a);
}
```

Figure 30: Temperature Shader Program

14 Appendix E: IG Vendor Table

Table 9: IG Vendors

Company	Systems	Platform	Terrain Input	Models Input	DIS HLA	CIGI	Sensor
Antycip http://www.antycipsimulation.com	Antycip	PC	Proprietary+ OSG OpenFlight And more	Proprietary+ IVE (OSG)	Y	N	Y JRM Tech
Aechelon Technology http://www.aechelon.com	pC-Nova	PC, Multi	Proprietary Tool needed	Proprietary Tool needed	?	Y	Y C-Radiant
Alion Science and Technologies http://www.alionscience.com/	CATI	PC, Multi	OpenFlight	OpenFlight	Y	Y	Y CATI TUAS
Boeing http://www.boeing.com/	Training system development incorporating IG from other companies which implement CIGI. They develop Hosts and do integration.					Y	?
Binghamton Simulator Company http://www.bsc.com/	Training system development incorporating IG from other companies which implement CIGI. They develop Hosts and do integration.					Y	?
CAE http://www.cae.com/	Tropos	Proprietary	Proprietary	Proprietary	N	Y CCL 3.1	Y EVS
	Simfinity	PC	Proprietary	Proprietary	N	N	
Dynamic Animation Systems http://www.d-a-s.com/	DasIG	Windows	Proprietary TerraFirma	Proprietary ApexSDK	Y Add-on	Y	Y Add-on
Diamond Visionics http://www.diamondvisionics.com	Genesis3D	PC	Source data	Proprietary	N	Y CCL 3.2	Y JRM Tech
Equipe Simulation http://www.equipe-simulation.com	Generation	Own SGI	Proprietary	Proprietary	Y Add-on	N	Y Advanced
Evans & Sutherland http://www.es.com/	EPX	PC	Proprietary	Proprietary	N	Y	?

FAAC Incorporated http://www.faac.com	?Only full system integration not IG itself				DIS	Y	N
General Dynamics Land Systems http://www.gdls.com/	Host Implementation and training systems Integration. They play an active role in CIGI maintenance and development as well.					Y	?
Indra http://www.indra.es/	Mentioned on CIGI website as Host and IG implementer, as well as System integrator. Not other information was found.					Y	?
MetaVR http://www.metavr.com	VRSG	Windows	Proprietary (Asia 3D) + OpenFlight	Proprietary + OpenFlight	Y	Y CCL 3.2	Y IRServer
Presagis http://www.presagis.com/	Vega Prime	PC	OpenFlight CDB	OpenFlight	Y Add-on	N	Y Multiple add-ons
	Lyra	PC	OpenFlight CDB ?TerraPage	OpenFlight	N	Y	Y Lyra-Sensor
	SOFVis	Open source based turn-key solution for US DoD. No other info was available.			?	?	?
Quantum 3D http://www.quantum3d.com/	CatalystSE	PC	Proprietary+ TerraPage OpenFlight	OpenFlight	Y Add-on	Y Add-on	Y JRM Tech
Realtime Technologies Inc http://simcreator.com/			Proprietary+ OpenFlight	OpenFlight VRML OBJ 3DS	N		
Rockwell Collins http://www.rockwellcollins.com	Training simulation system integrator.						
Rheinmetall Defence http://www.rheinmetall-detec.de	Avior NTS	PC	?Proprietary	>Proprietary	N	N	Y Avior VisIR
Simultec http://www.simultec.ro/	-	?	Proprietary	Proprietary	N	N	Y Advanced

Tiltan Systems Engineering http://www.tiltan-se.co.il/	TView	Windows	Proprietary+ OpenFlight GeoTiff	Proprietary + OpenFlight 3DMax 3DS	DIS HLA	Y+ Flight TVie w	Y+ IRView
Transas http://www.transas.com	Aurora	PC	Proprietary+ Import	Proprietary + Import	N	N	N
ViewTec AG http://www.viewtec.net	TerrainView	Windows	OpenFlight TerraPage VTP (OSG)	Proprietary + OpenFlight IVE (OSG) And more	N	Y 80%	N
Laminar Research http://www.x-plane.com/	X-Plane	PC, Multi	Proprietary FlightSim		HLA Add- on	Y Add- on CCL 3.2	N
Microsoft	FSX	Windows	Proprietary	Proprietary	N	N	N
http://www.microsoft.com/esp/	ESP	Windows	Proprietary	Proprietary	Y Acro n	N	N
Delta3D http://sourceforge.net/projects/openig	OpenIG	PC, Multi	OpenFlight TerraPage Possibly more	OpenFlight IVE (OSG) Possibly more	N	Y Basic	N
OGRE www.ogre3d.org	OGRE 3D	PC, Multi	It is a 3D rendering engine and not an IG per say but could be used to create a game engine or an IG. There are many 3D engines but OGRE is recognized as the most successful OSS one.				
US AFRL http://www.subrscene.org	SubrScene	PC, Multi	OpenFlight TerraPage VTP (OSG)	OpenFlight IVE (OSG)	N By Host	Y SDK	Single Channel B&W

Boeing http://cigi.sourceforge.net/product_mpv.php	MPV	PC, Multi	OpenFlight TerraPage	OpenFlight IVE (OSG)	N By Host	Y CCL 3.x	Y Basic
Oktal-SE http://www.oktal-se.fr/	SE-FAST	PC, Multi	Proprietary+ OpenFlight TerraPage	Proprietary + OpenFlight IVE (OSG) And more	HLA	Y CCL 3.x	Y Advanced SE-FAST-IR

15 Appendix F: Mapping of Ecosystem

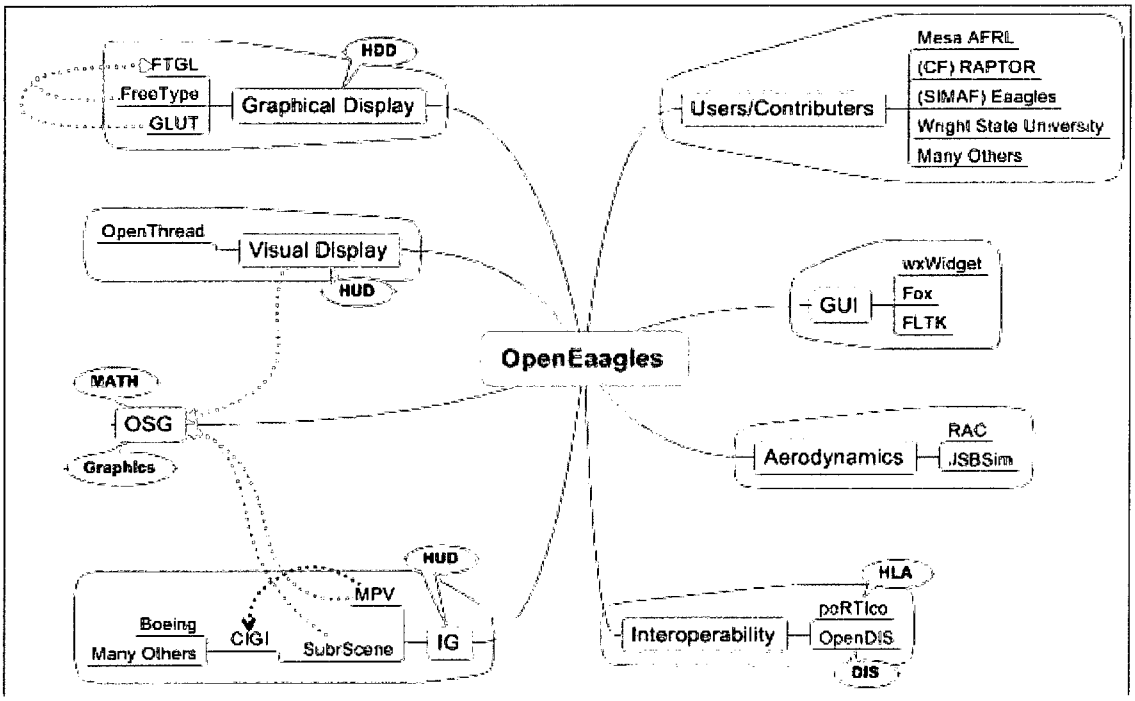


Figure 31: OpenEaagles Preliminary Ecosystem Map