Common Sensor Model Common Components – A Design Approach

John O'Reilly
CAE USA, Inc.
Tampa, FL
John.OReilly@CAE.com

Mike Kochmann
PEO STRI
Orlando, FL
Mike.Kochmann@peostri.army.mil

Brett Chladny, Jeff Clark, Brad Colbert
Renaissance Sciences Corporation
Chandler, AZ
BChladny@rscusa.com, JeffClark@rscusa.com, BColbert@rscusa.com

ABSTRACT

As one of the tasking orders on the U.S. Army's SE Core Database Virtual Environment Development program, the Common Virtual Components (CVCs) were envisioned as extensions to the database storage and production facilities of the program. As extensions, CVCs will provide added functionality as models which both are easy to use/integrate and are pre-validated. The Common Sensor Model (CSM) CVC has created a new software module that fits into this mold directly as it provides proven yet modular sensor effects simulation for virtually any image generator (IG) built on an OpenGL 2.0 platform. CSM was designed to be a drop-in module that combines the power of modern commercial-off-the-shelf (COTS) graphics processing unit (GPU) architectures with best of breed government-off-the-shelf (GOTS) sensor modeling approaches pioneered under the Night Vision and Electronic Sensors Directorate (NVESD) Night Vision Image Generator (NVIG), Air Force Research Laboratory (AFRL) SensorHost, and AFRL InfraRed Target Scene Simulation (IRTSS) programs. IGs can easily control CSM through a lightweight thread-safe C++ application programming interface (API). Design objectives focused on modular architectures which would be non-invasive to its host application's scene rendering yet facilitate future incorporation of additional math models and new sensor types. These design objectives were realized, in large part, by utilizing a floating point frame buffer object (FBO) to cleanly separate the rendering of quantitative radiance scenes from the rendering of sensor effects. This paper will provide an overview of the design and inner working of the CSM code base and will conclude with an example integration.

ABOUT THE AUTHORS

John O'Reilly is a Principal Engineer and Technical Leader of Sensor Systems at CAE USA in Tampa. He has worked in the simulation industry for over 20 years and has experience in the digital radar simulation and database development environments for ten of those years. He has an MSEE degree from the University of South Florida in Tampa.

Mike Kochmann currently works for the U.S. Army's Program Executive Office for Simulation, Training, and Instrumentation's (PEO STRI) Virtual Engineering Directorate and is matrixed to the Project Manager for Combined Arms Tactical Trainers (PM CATT). As the Lead Engineer for Synthetic Environment Core (SE Core) Database Virtual Environment Development (DVED) Common Virtual Components (CVCs), he oversees efforts among a team of Government, military, and contractor personnel, building products for use in various virtual, live, and constructive simulation systems. Previously, he was the Project Director for all visual aspects of the Close Combat Tactical Trainer (CCTT) program and held several positions within PEO STRI's Engineering Directorate. He has been involved with virtual simulation systems at PEO STRI since 1994. Mr. Kochmann has a Bachelor of Science degree in Industrial Engineering and a Masters of Science degree in Engineering Management, both from the University of Central Florida.

Brett Chladny is currently a Senior System Analyst at Renaissance Sciences Corporation where he is the Principal Engineer of multiple programs and pursues his passion for implementing novel solutions to some of visual simulation's most challenging problems. Over the course of his 10 year career in the real-time graphics industry, he has worked at Silicon Graphics and MultiGen-Paradigm. During that time he has created several new graphics

techniques, four of which have patents pending. Mr. Chladny received a B.S. in Computer Science, with minors in Fine Art and Mathematics, as well as a M.S. in Computer Science from the University of Missouri.

Jeff Clark has been active in the sensor simulation research and development arena for the majority of his professional career. He currently serves as president of Renaissance Sciences Corporation, a leading sensor simulation research and development firm embedded in US Air Force, US Navy and Marine Corp, and US Army government technology teams. Formally trained as a computer scientist, Mr. Clark previously served in software and systems engineering capacities at L-3 Communications, Raytheon, Hughes Aircraft, and Accurate Automation Corporation.

Brad Colbert's visual simulation experience began at the Naval Research Laboratory's VRLab where he helped investigate novel ways for presenting battlefield information. Mr. Colbert later moved into platform and sensor simulations with his work at GreyStone Technologies and then Renaissance Sciences Corporation where he now serves as a vice-president. He has a B.S. in computer science and has worked in the visual and sensor simulation industry for over 10 years.

Common Sensor Model Common Components – A Design Approach

John O'Reilly
CAE USA, Inc.
Tampa, FL
John.OReilly@CAE.com

Mike Kochmann
PEO STRI
Orlando, FL
Mike.Kochmann@peostri.army.mil

Brett Chladny, Jeff Clark, Brad Colbert
Renaissance Sciences Corporation
Chandler, AZ
BChladny@rscusa.com, JeffClark@rscusa.com, BColbert@rscusa.com

INTRODUCTION

Synthetic Environment Core (SE Core) is a program underway at the U.S. Army's Program Executive Office for Simulation, Training, and Instrumentation (PEO STRI) to build and maintain products and services that facilitate interoperability among virtual simulation training systems. The program consists of two parts. The Architecture and Integration effort provides a product line for host applications. The Database Virtual Environment Development (DVED) domain is responsible for the visual products, such as databases and moving models. Another related product of SE Core is Common Virtual Components (CVCs). There are efforts underway on several CVCs, which include environmental CVCs (which will be mentioned later in this paper as important to the models being presented). The Common Sensor Model (CSM) is one of the Common Virtual Components being developed under DVED.

Quite often, simulation programs are developed for one particular mission or customer. Requirements are set, contracts are awarded, the product is tested, and the result is often a very successful training system. Considering the bigger picture, however, this recipe has some pitfalls. Connecting two or more systems together is quite often non-trivial. Even if they are not connected, there is a discontinuity in training created by differences between systems. For good reason, one may depict the world one way and another may present a totally different rendition. One of the cornerstones to successful training is cognitive repetition, so it is important that systems portray a similar "realism" when used to prepare trainees for what they need to do in real life.

The U.S. Army takes great pride in its ability to perform at night, in bad weather, or in any other adverse condition. Key to this capability is its usage of sensor technology. Therefore, it is critical that training devices be able to depict scenes that replicate what is seen in the real world through these sensors. The problem is that sensor simulation has largely been a victim of stove-piped development. Each individual program uses renditions of sensor simulations that are driven by a spectrum of variation between databases, models, the environment, and how the sensors themselves are simulated. The lower end of this range includes static, texture-driven scenes that are essentially pictures painted on terrain and models, with no effects of the environment and no diurnal effects. Some systems do much better, using physics-based algorithms and material encoding on the terrain and models, along with dynamic environmental conditions and time of day. Include the variation of hardware in this cornucopia of software implementations and it is easy to see that there is a lot of room for improvement towards a common sensor simulation. Even if the goal is just to have the scene look the same when a trainee looks at a display, there are just too many variables in all these simulation systems to allow that to happen. One of the initiatives of SE Core is to figure out how to fix this.

This paper uses the term "Image Generator" (IG) to refer to a complete system, both hardware and software, that generates visual images of a synthetic world and is only one component of a visual simulation. The term "host application" refers to the software application that CSM is integrated into.

CSM OBJECTIVES

All the SE Core products and services are designed to work together, as well as to be platform independent. The goal of the Common Sensor Model effort is to have all systems produce the same sensor scene, regardless of which Image Generator (IG) or run-time software it utilizes. The Government's motivation is to create realistic, validated, and consistent sensor simulations that enhance the training experience by fostering interoperability and commonality. The industry, in general, also can benefit from adapting a Common Sensor Model. For the products that don't currently offer sensor effects, or don't do them well, the advantage is obvious. However, even for the companies that already have excellent sensor capabilities, whether it comes as a result of hardware, software, or both, their situation can be enhanced by integrating CSM.

There are many variables to creating a valid sensor scene. The database, models, runtime software, and hardware all factor in and if the scene doesn't look right, any or all of these variables may need to change. It is quite common, that final system testing reveals a problem with the sensor simulation and costly rework on the database, changes to models, or a tweak to the runtime software is required. This is a lot of rework. Sometimes, hardware limitations just won't allow changing the variables to correct the problem. These are all non-trivial situation that most visual system producers have experienced. SE Core products, working together, will alleviate many of these issues. The Common Sensor Model will be a validated set of software that works with a common environment, databases, and models. Integration of CSM will relieve vendors of any eleventh-hour rework because the model itself will be pre-validated. Through the use of plug-ins, IG vendors that use CSM will have a certified ability to produce valid sensor scenes. This shifts the responsibility off them and onto SE Core.

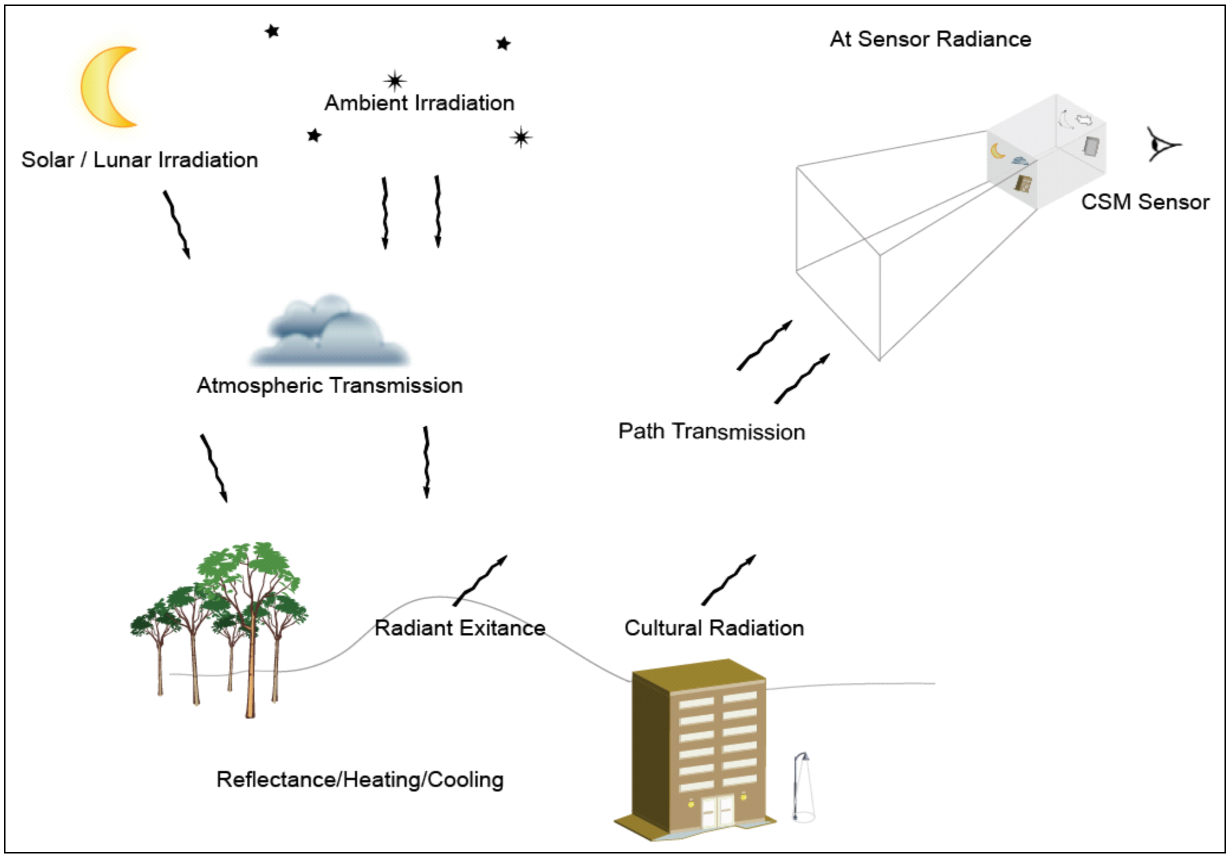


Figure 1. Typical Environment Modeling

Implementing the CSM Objectives

The task of creating a Common Sensor Model was decomposed into three key components: the common environment, sensor simulation, and rendering (Figure 1). The initial scope of CSM is to focus on the last two components, then attack the first in conjunction with other SE Core activities. Under related efforts, the Common Environment CVCs will define the SE Core approaches and specifications for common and portable multi-spectral scene generation, as impacted by simulation databases, atmospherics, and radiative transfer. This environment framework will be developed under future tasks of the SE Core DVED program. The final CSM package will contain the sensor simulation framework (described in this document) in addition to any elements required to integrate CSM with the SE Core environment simulation framework (to be specified in future documents).

Therefore, the front end boundary of the CSM as it stands today is the at-scene radiance from the IG. The back-end boundary is the rendering output of the model back to the IG.

Interservice CSM Collaboration

The approach taken to build this “engine” was to look at government-off-the-shelf (GOTS) solutions to see if there was something to leverage. Collaboration between PEO STRI, the NAVAIR, and the Air Force Research Lab revealed several common objectives for producing common sensor scenes. These related efforts seem to complement the goals of CSM by providing key building blocks. It has become apparent that the model should incorporate a best-of-breed approach. The GOTS building blocks utilized for proof-of-concept development are summarized below.

NAVAIR NPSI

The Naval Aviation Simulation Master Plan (NASMP) Portable Source Initiative (NPSI) primary objectives are to increase visual database reuse, increase standardization, and lower life cycle acquisition costs for new systems acquisition, legacy platform trainer procurements, and major trainer visual upgrades. The NPSI datasets capture the prepared/corrected/refined visual source data in standard COTS formats for reuse (NPSI data sets) by other platforms. An NPSI data set may include imagery, elevation data, feature data, and three-dimensional (3-D) models. The NPSI data standard also includes support for sensors such as FLIR and Night Vision Devices (NVD), if the producer platform requires it. The NPSI effort maximizes the reuse of costly geospecific imagery, and captures the common labor intensive value added effort involved in preparing the various source data layers prior to customization and optimization for a specific image generator's runtime format.

AFRL SensorHost

SensorHost is a GOTS Night Vision Goggle (NVG) math model produced at AFRL/HEA Warfighter Readiness Research Division for real time modeling and simulation applications. SensorHost has been widely fielded previously [Haberman, 2007] and has been paired with a variety of COTS image generation systems. Built on a modular software programming library design, the NVG math model can be integrated with other elements of the CSM design using SensorHost’s open and documented API.

AFRL IRTSS

The InfraRed Target Scene Simulation software was developed at AFRL Hanscom AFB, MA over several years beginning in the mid 1990’s. The primary development contractor was Radex Inc. now a division of AER Inc. The goal of the system was to provide operational users of infrared imagers a means to predict the effects of weather on the performance of imaging sensors by using weather forecasts and modeling of the infrared background and imaging process. There are four major areas of physical modeling in the IRTSS system: terrain modeling, target signature modeling, atmospheric transmission and path radiance modeling, and sensor system modeling. The system ingests weather data from AFWA and has run on a laptop computer since 2002.

IRTSS provides two main operational benefits. The primary operational benefit is enhanced aircrew situational awareness during mission execution. Before flying a mission, aircrews can view a physically accurate representation of the target scene and its relative contrast to broad geographic features in a sensor specific waveband. The net result reduces the amount of effort and time needed for a system operator to carry out a mission.

The secondary benefit allows weather to be systematically injected into the mission planning process to help determine a mission profile (time over target, ingress/egress, weapons selection) that is optimized for both the anticipated tactical situation and environmental conditions over the target. Mission planners can view predictions of how a target will appear (position relative to broad geographic features and contrast against immediate background) for a variety of mission profiles.

In 2003, the system received extensive use in the air strikes in the early part of Operation Iraqi freedom by F-117 aircraft crews. The system was successfully utilized in-theater to anticipate and plan numerous strike missions under varying environmental conditions, with good results and accolades from meteorological support personnel, planners and pilots.

NVESD NVIG

The Night Vision and Electronic Sensors Directorate (NVESD) Night Vision Image Generator (NVIG) is a sensor image generator (IG) of a modular design which consists of two phenomenology/render modules (the sensor shader library and the sensor post-processor library) and an IG framework. The NVIG produces real time simulation of IR scenes and sequences using modeled backgrounds and targets with physics and empirically based IR signatures. Range dependant atmospheric effects are incorporated, realistically degrading the infrared scene impinging on an infrared imaging device.

CSM Technical Approach

CSM is being released as a self-contained package in order to provide initial capabilities under the most efficient circumstances. CSM is designed so that full-scale integration of the new environment efforts will be as straightforward as possible.

SE Core Confederate IGs will control CSM through a lightweight thread-safe C++ Application Programming Interface (API). In many ways, the CSM software plays a role similar to that of traditional hardware sensor post-processor systems, but is largely implemented on modern Graphics Processor Units (GPUs) with an open interface to facilitate portability. The input interface to CSM from the Host Application is largely accomplished by "handing off" at-sensor radiance frames from the host application to CSM. In this interfacing approach, the implementation details associated with scene generation are hidden from CSM, so that CSM portability may be maximized (See Figure 2).

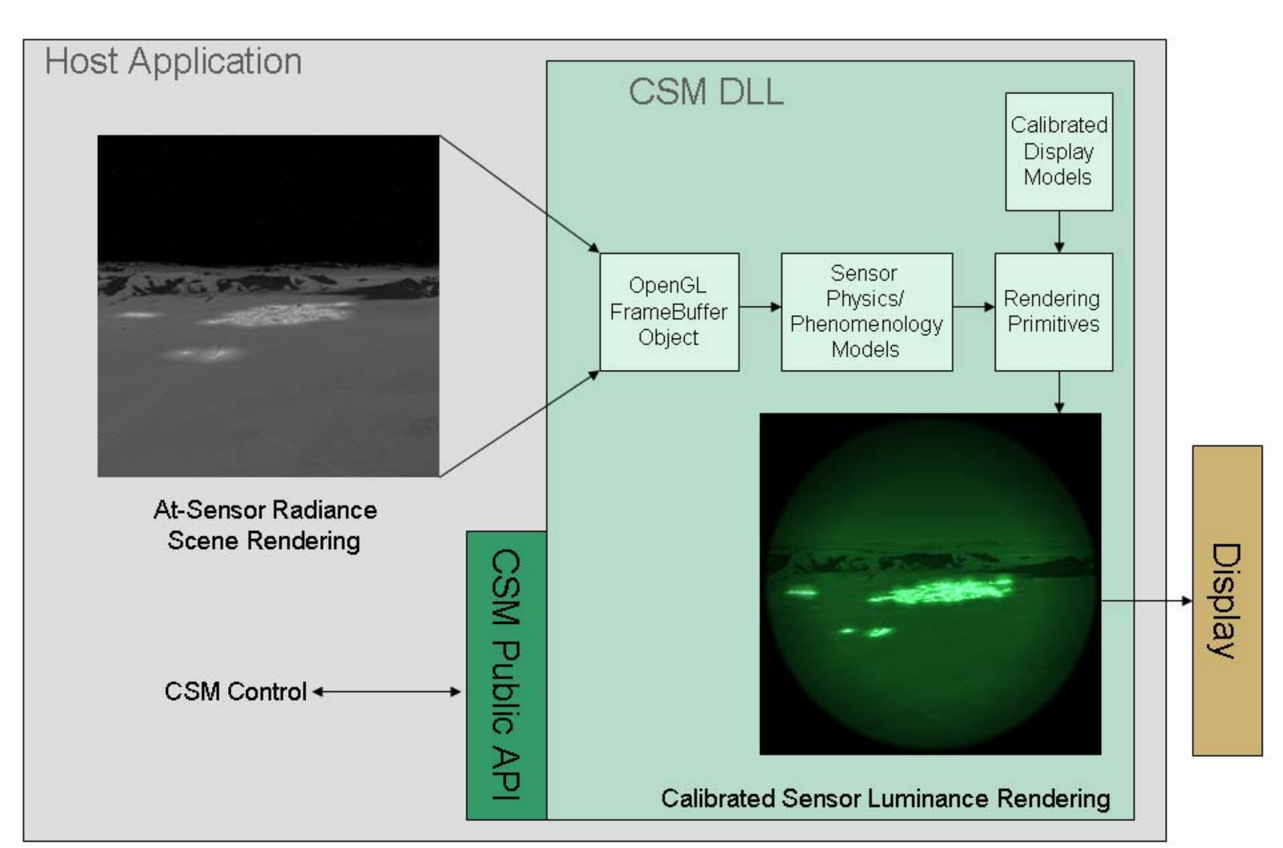

Host Application
CSM DLL
Calibrated Display Models
OpenGL FrameBuffer Object
Sensor Physics/ Phenomenology Models
Rendering Primitives
At-Sensor Radiance Scene Rendering
CSM Public API
CSM Control
Display
Calibrated Sensor Luminance Rendering

Figure 2. CSM Interface Overview

Although CSM plays a role similar to that of a traditional sensor post-processor, it performs this role in a completely different manner. A traditional sensor post-processor uses dedicated hardware that takes two types of input: image data and control data. The IG renders an image of a radiance-based scene and sends the image to the post-processor, instead of the display system, via VGA or DVI cable. After applying the needed effects, the post-processor sends the image to the display system. This design inherently adds latency to each frame. CSM is a software-only solution that performs its work as part of the IG's frame loop, utilizing the IG's programmable graphics hardware. The resulting images can be sent directly to the display system. If an IG is synced to the vertical retrace and CSM does not cause the frame rate to drop, CSM does not add any latency to the rendering process.

The key prerequisites for the CSM host application are that it must:

- Be built on OpenGL 2.0 or greater.
- Be built on Linux or Windows XP.
- Render quantitative, in-band, at-sensor radiance scenes at both high precision and high dynamic range (design minimum threshold: 16 bit floating point).

CSM is built directly on top of OpenGL in order to facilitate ease of integration with host applications, as is illustrated in Figure 3. By avoiding any dependency on a high-level graphics API or scene graph, potential challenges associated with shared dependencies can be avoided. In particular, high-level external dependencies shared by the host application and CSM would need to be managed at the same revision level. For similar reasons, CSM utilizes GLSL (OpenGL's native Shader Language) rather than other vendor specific languages such as CG (NVIDIA's shading language).

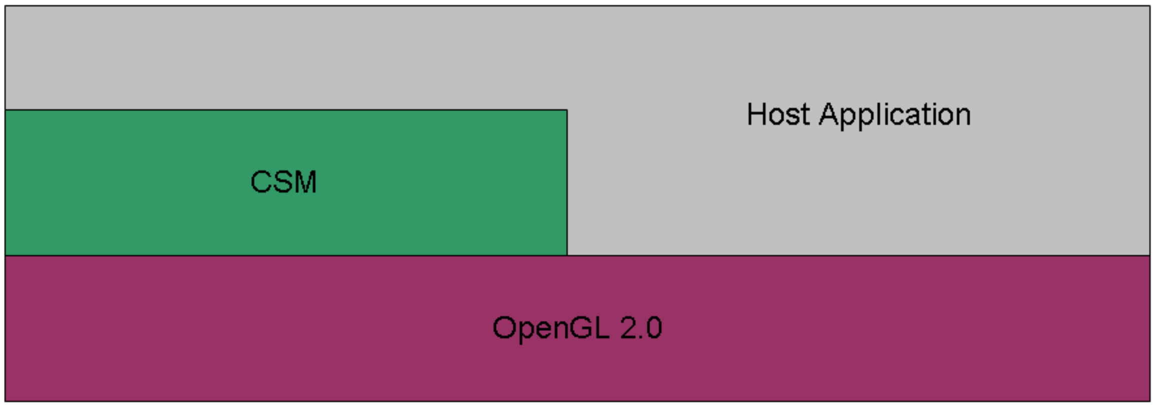


Figure 3. CSM Software Dependencies

CSM Interface Classes

CSM is composed of four major groups of classes: CSM::System, CSM::Higher_Level, CSM::Math_Model and CSM::Base. CSM::System contains the external interface through which the host application communicates with CSM. Its classes are built on top of classes from the CSM Higher Level group. The classes in CSM Higher Level are built on top of two groups of classes: CSM Math Model and CSM Base. CSM Math Model contains classes that are used to wrap various sensor math models. CSM Base contains all of the low level classes needed to support CSM's design, as well as the CSM rendering primitives. This hierarchy is shown in Figure 4.

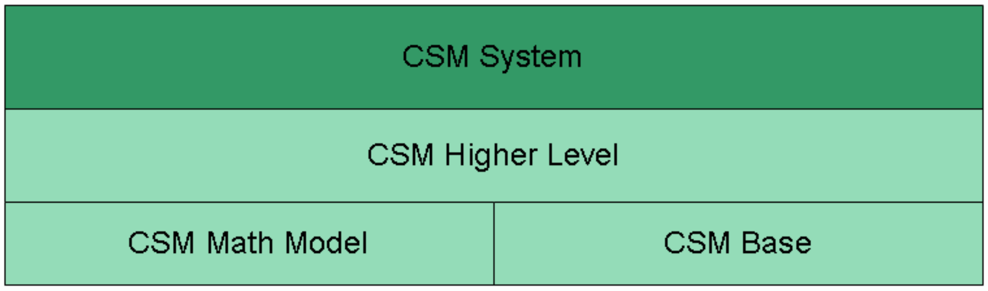


Figure 4. CSM Class Hierarchy

The host application should only instantiate one object from the entire CSM code base, i.e. a sensor-specific object derived from CSM::System. The CSM::System class provides most of the programming interface between the host application and CSM. In conjunction with the classes derived from it, CSM::System performs three main categories of tasks:

- Creates the required internal CSM classes

- Passes data from the external API and configuration files to the internal classes
- Orchestrates the updating and rendering performed by the internal classes

Neither the CSM::System class nor any other CSM class installs callbacks or "hooks" into the host application. It is the responsibility of the host application to send data to CSM as well as calling the CSM functions that need to be executed each frame.

Classes derived from CSM::System will instantiate the appropriate higher level CSM objects to implement a specific type of sensor. Many of these objects will, in turn, create additional internal CSM objects. However, it is the classes derived from CSM::System that will "build" the sensor from the available higher level CSM classes. It is at this level that major components of a sensor will be included or excluded, for example noise or a housing mask for a Night Vision Goggle (NVG) sensor simulation.

The host application does not directly control internal CSM classes. Therefore a mechanism is needed to communicate to CSM what type of sensor CSM is to emulate. The host application passes an enumerated token to one of the interface classes derived from CSM::System. CSM currently provides four such classes:

CSM::NVGSystem,
CSM::FullGainFLIRSystem,
CSM::HorizontalFLIRSystem,and
CSM::PlateauEqualizationFLIRSystem.

Each of these classes provides a unique enumerated list of specific sensors models available for emulation (See Figure 5).

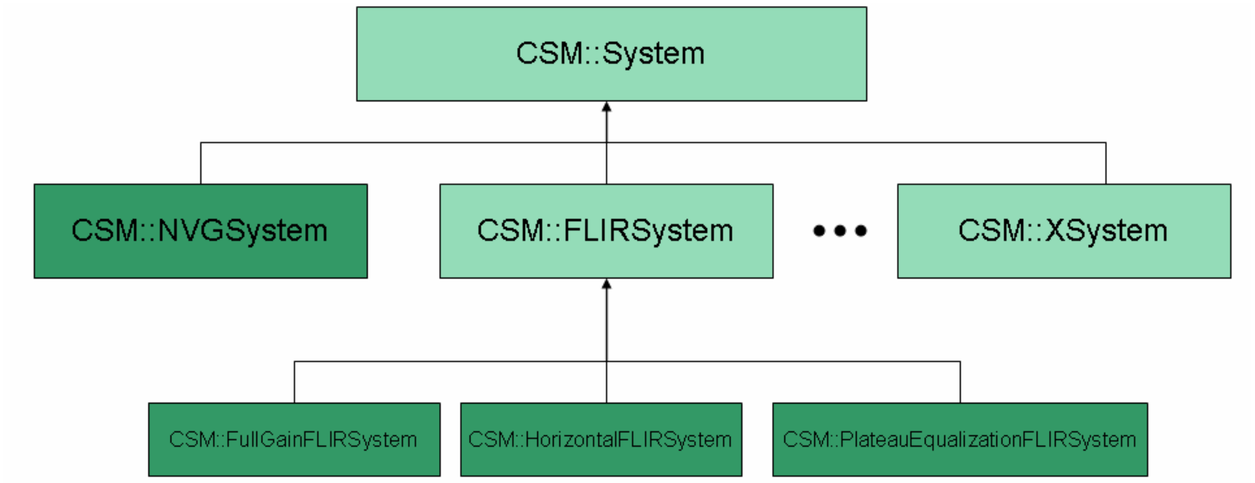


Figure 5. CSM::System Classes

The host application instantiates one of the classes derived from CSM::System (not a CSM::System class). As such, CSM::System provides much of the interface for sensor simulations without actually implementing any specific sensors. The host application can use a pointer to this base class for many of the CSM calls that need to be made each frame.

Thread safe API

Many host applications are multi-threaded and have an application, a cull, and a draw thread. CSM supports multi-threaded applications by allowing them to pass a frame number into almost every API call. When a frame number is passed into a "set" call, the other passed in arguments are buffered until that frame is rendered. Likewise, for "get" calls, the frame number is used to retrieve the data that will be used to render that frame number. For single threaded host applications, frame numbers do not need to be passed in.

The most critical API call that takes a frame number is CSM::System::render(), which tells CSM what data to use when processing the rendered scene and adding sensor effects to it. This style of buffering input data allows the host application to control CSM's ability to perform frame accurate rendering.

Table 1, Table 2, and Table 3 outline the functionality of some member functions in CSM::System and which thread they should typically be called from. CSM does not mandate that these calls be made from these threads, but rather recommends it for performance reasons. The only restriction in this regard is associated with the CSM::System::setBufferEnable() and CSM::System:: render() calls. These must be called from the draw thread with the scene's OpenGL context made current. A detailed specification of the entire public API is presented in the CSM Public API (SE Core DVED 2007 [1]) document.

Table 1. Application Thread Classes provided by CSM::System

Application Thread	
System::System	Default constructor that creates a new Common Sensor Model System.
System::configure	Called after OpenGL window is created but before any rendering is done. It invokes the creation and configuration of the objects necessary to render the sensor effects as well as the appropriate math model.
System::setEnable	Turns all CSM processing on or off.
System::setFOV	Sets the vertical and horizontal field of view of the sensor.
System::setViewport	Sets the window coordinates, in pixels, which CSM objects can read from and write to.

Table 2. Cull Thread Classes provided by CSM::System

Cull Thread	
System::setPointSourceList	Sets a list of bright point sources that the host application will render on the specified frame. Each bright point source should consist of a position and size. Position should be specified in OpenGL's normalized device coordinates.
System::setMoonPosition	Sets the position of the center of the moon in OpenGL's normalized device coordinates.
System::setMoonOrientation	Sets the rotation of the moon.
System::setMoonFOV	Sets the Field of View (FOV) the moon should cover.
System::setMoonPhase	Sets the moon phase. This is used to choose the appropriate effect texture to apply over the top of the moon.

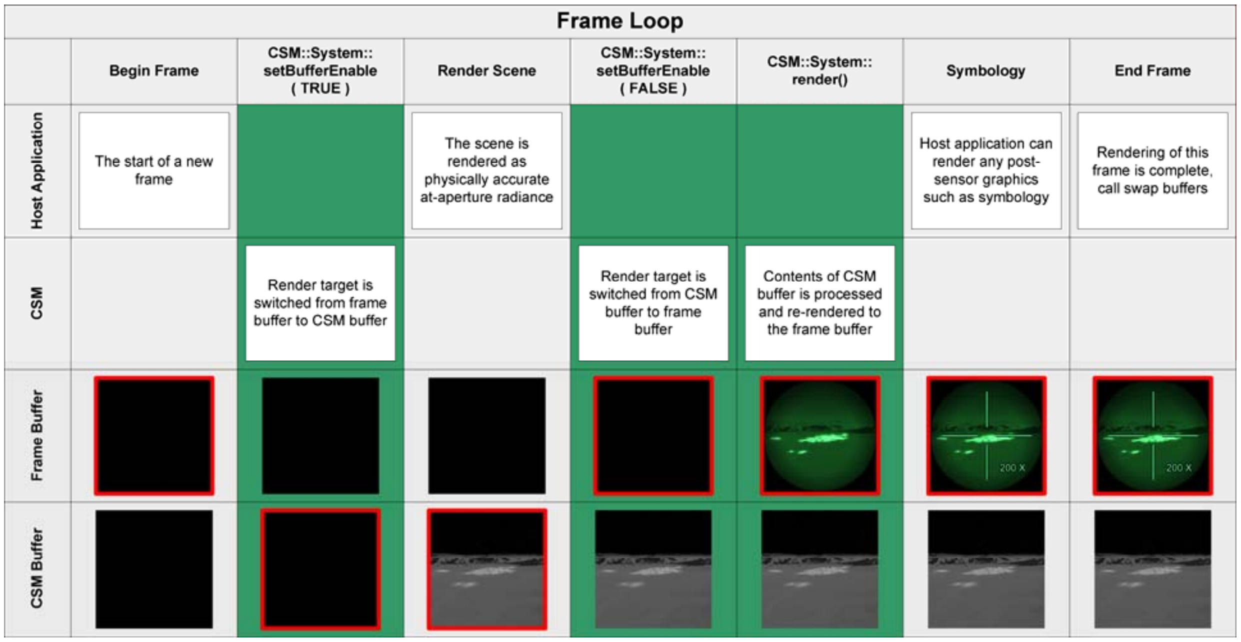
Table 3. Draw Thread Classes provided by CSM::System

Draw Thread	
System::setBufferEnable	When called with a value of TRUE, the internal CSM buffer becomes the OpenGL render target. When called with a value of FALSE, rendering is directed to the normal frame buffer.
System::render	Called after normal scene is rendered. It performs tasks such as: - Compute gain state - Render FBO contents to the frame buffer - Render bright point source effects - Render moon effect - Render noise - Render Housing mask

Rendering Pipeline

The objective is to maximize the flexibility of the CSM design by decoupling the rendering of the scene from the sensor processing of it. Traditionally, the scene rendering and sensor processing are intertwined, i.e. the sensor response is incorporated into the rendering process. This increases the complexity of replacing sensor models and therefore limits the flexibility of an IG. Separating the rending of the scene from the sensor processing facilitates using different sensors models. A common framework further increases the flexibility and facilitates replacing sensor models. This is the motive behind the CSM concept, which can be used with virtually any OpenGL 2.0 base host application. CSM accomplishes this by creating a buffer that the host application can render floating point values into. This buffer is a Frame Buffer Object (FBO), a relatively new type of frame buffer that is specified by an OpenGL Architectural Review Board (ARB) extension, and is supported on most current generation COTS GPUs. Once the true radiance values of the scene are rendered into the FBO, they can be processed by the selected sensor.

Figure 6 illustrates the processing in a typical frame loop of a host application into which CSM has been integrated. The time axis starts on the left. The light gray background indicates that the host application's code is being executed; the dark green background indicates that CSM's code is being executed; a red line around a buffer indicates that buffer is the active OpenGL render target. Which buffer is active, and the contents of the buffers are based on the state of the host application at the end of each block in the diagram. It is important to note that when the host application is rendering the scene, the CSM buffer is active. Similarly, when CSM is rendering, the frame buffer is the active buffer.



Frame Loop
Begin Frame
CSM::System:: setBufferEnable (TRUE)
Render Scene
CSM::System:: setBufferEnable (FALSE)
CSM::System:: render()
Symbology
End Frame
Host Application
CSM
Frame Buffer
CSM Buffer
The start of a new frame
The scene is rendered as physically accurate at-aperture radiance
Host application can render any post-sensor graphics such as symbology
Rendering of this frame is complete, call swap buffers
Render target is switched from frame buffer to CSM buffer
Render target is switched from CSM buffer to frame buffer
Contents of CSM buffer is processed and re-rendered to the frame buffer
200 X
200 X

Figure 6. Overview of CSM Frame Loop

Integrating Mathematical Sensor Models

CSM's purpose is to facilitate the rendering of sensor effects, based on the values provided by a mathematical ('math') sensor model, but done in a way that is independent of any particular math model.

The sensor specific math model classes are derived from a single generic CSM::MathModel class, not unlike sensor specific classes that are derived from a single generic CSM::System class. CSM currently contains two classes derived from CSM::MathModel:

- CSM::NVGMathModel
- CSM::FLIRMathModel

CSM was developed as GOTS software, therefore GOTS math models were also be used. Sensor modeling approaches pioneered under the Night Vision and Electronic Sensor Directorate (NVESD) Night Vision Image Generator (NVIG), Air Force Research Laboratories (AFRL) SensorHost, and AFRL Infra-Red Target Scene Simulation (IRTSS) programs are used to drive the NVG and FLIR math models.

One of CSM's design goals was extensibility for adding alternate math models, which is possible by deriving a new class from CSM::MathModel and populating the relevant "set" and "get" functions. A new math model can be used by deriving a new class from one of the existing CSM::System level classes. The level of effort required to accomplish this depends on the similarity between the parameter of the new math model and those of the most similar pre-existing math model.

A second goal was a sufficiently generic design that would support sensor types other then I2 and FLIR. The math models for these sensors will have different inputs and outputs than the current set of classes. Completely new sensor types can be implemented within the CSM framework by deriving two new classes, one from CSM::System and one from CSM::MathModel. CSM provides a generic rendering framework with an optimal set of rendering primitives designed specifically for sensor simulation. A complete description of the internal CSM classes is provided in the CSM Developers' API and CSM Software Design Document (SE Core DVED 2007 [2]).

Integration Efforts

The CSM package, by design, is meant to be quite portable across image generator (IG) applications and vendors. Therefore, the software can be incorporated into current IG processes using a portable Application Programming Interface (API) defined as part of the CSM scheme. As a portability proof-of-concept, the CSM development team utilizes the open-architecture, open-source, Open Scene Graph and Delta3D simulation platforms as its host scene generation system for internal CSM development and testing purposes. Sample results can be seen in the screenshots below.

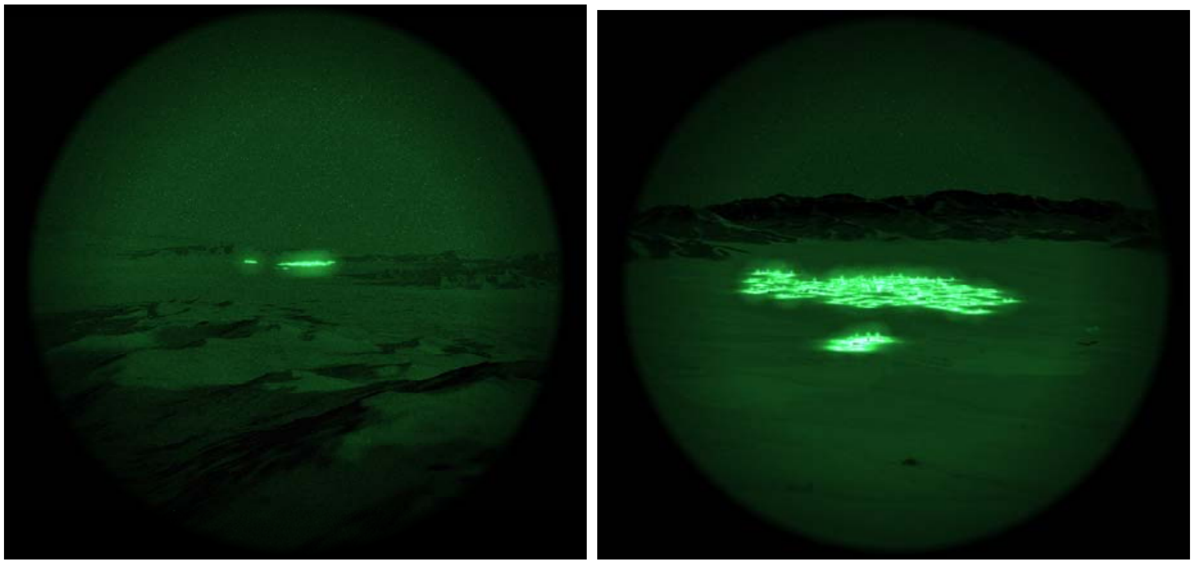


Figure 7. Sample CSM Display Results

In order to test CSM portability on commercial systems, an integration of CSM into a commercial IG is being performed under the SE Core contract. Being one of the SE Core DVED team members, Rockwell Collins Simulation and Training Solutions (RC-STS) has been tasked to perform this prototype integration of the CSM package. The "Environment Processor Technology" (EPX) is the conceptual IG of choice for this effort which will culminate in a Proof of Concept (POC) Demonstration during the primary Government Acceptance Testing phase of the SE Core DVED program.

In order to allow for generic installation of the CSM package, a Public Interface Control Document (ICD) (SE Core DVED 2007 [1]) has been prepared and reviewed. The POC integration effort applies, comments on, and updates this ICD. This provides for a more usable and reliable ICD after this demonstration is complete. This does not complete the document's design and production, though. The ICD is still a living document and is expected to be updated and improved throughout the program (See "How to Get Involved" below.).

It is RC-STS's intent to provide an integration process during its development phase toward the demonstration. This process currently involves generating prototype interfaces and code into their standard IG code to rapidly produce demonstrable results. This is by no means meant to be the final, installed baseline code which RC-STS or any other IG vendor would produce and/or provide to its customers. This "throw-away" prototype is meant to provide exactly what its name implies: experimental code and installation processes upon which later release code will be built. The deliverable code will then be tightly coupled into the EPX IG code, as opposed to being loosely architected "around" the full capability of the IG. This process, which will be available as an SE Core product to the public, is designed to provide more efficient and timely integration efforts to the IG populace.

Conclusion

The main objective for the Common Sensor Model (CSM) software was an abstraction of sensor simulation that can be incorporated into different training systems and still produce the desired sensor effects. The technical challenge was to offer sufficient flexibility of the 'drop-in' design for supporting different mathematical sensor models without interfering with the rendering of the scene. This was accomplished by separating the rendering of the scene from the sensor processing, by using a floating point buffer, and a generic OpenGL-based rendering infrastructure that facilitates the rendering of the needed effects with a wide variety of mathematical sensor models.

By providing this infrastructure of different training systems, differing image generators on the systems, and different (validated) sensor models, the final model gives the training environment a one-stop-shopping solution to sensor training. As this solution progresses in capabilities and usage, the training domain will profit from CSM's abilities and non-exploiters of the architecture will need to provide alternative solutions.

Current Status

The Common Sensor Model proof-of-concept implementation, including foundational NVG and FLIR sensor modeling and effects capabilities, is scheduled to be completed in the fall of 2007. The CSM sensor modeling and effects are being developed as a self-contained software library so that it may be completed independently of the design and development of scene generation portions of the CSM. The design and development of the multi-spectral scene generation elements of the CSM are scheduled to begin in the fall of 2007.

Road Forward

The Common Sensor Model is the foundation for creating common sensor scenes across a variety of systems. It is a robust and elegant model that allows for growth by being modularized and receptive to new sensor models. It is efficient and does not add significant latency. It is portable and generic enough to merge with most IGs' run-time systems. This engine is ready for fuel!

In the upcoming year, the CSM effort will work in two directions:

- To define the common environment in conjunction with other SE Core activities. This will be a non-trivial task, but it will important to approach this with enough breadth and depth to make the Common Sensor Model ready for implementation with databases, models, and environmental conditions that include things like weather, dust, smoke, and other effects.
- To focus on the implementation of larger numbers of real-world sensor systems.

The CSM, as it stands today, will undergo testing and validation as a proof of principle. After the common environment and other sensor systems are selected, designed, and implemented, formal testing will take place to validate the model and how it depicts the target and environment. This will need to involve all potential users of CSM, to allow them to claim that their scene is validated, interoperable, and consistent with real-world sensor views.

How to Get Involved

The Common Sensor Model is required to provide for easy integration on simulation vendors' systems. In order to prove this specification, SE Core is soliciting assistance from both Government and industry participants to review, integrate, and provide supportive commentary on the model. This process involves several directions and efforts:

- The SE Core program provides documentation to the public for their review. This data is in the form of ICDs, integration documentation and commentary, and requirements documentation. These reviews are vital to the CSM effort in order to assure that clear and concise information is provided to ease integration tasks.
- The SE Core program provides assistance to the public in integrating the CSM. Reviews and inputs into documentation are needed, but the proof of a portable product can only be had by readily integrating that product. SE Core is tasked to provide as much technical assistance as is practical in order to assist simulation vendors with that integration effort. This effort is intended to be as small as possible for reasons of budgetary reasons, but mostly because the effort is intended and expected to be minimal due to the common modeling concept.
- The SE Core program solicits simulation vendors to provide their validated sensor models to be incorporated into the CSM package. One of the main tenets of the SE Core program is Technology Insertion. This means that additional sensor models in the I2, IR, and Radar realms are encouraged and required to be inserted into the CSM package. The inclusion of future sensor models can be affected by either the simulation vendor itself (using the CSM Internal ICD) or by SE Core development engineers. Once again, SE Core is required to provide some modicum of assistance if the vendor chooses to insert their sensor model into CSM.

SE Core provides a lengthy list of Points of Contact through which both Government users and industry participants can become involved (http: //www.peostri.army.mil/PM-CATT/ APM_ SECore .jsp). The SE Core program readily accepts and anticipates wide-ranging participation through varying fora:

- Industry/Government Reviews – Review cycles will be on-going throughout the developmental phases. These phases will be running through the Government's 2008 Fiscal Year.
- Integrating CSM into IGs – Incorporation of the CSM into Image Generators is a vital link for the program. SE Core is providing expertise to this end in many forms as discussed above, including: sensor knowledge, CSM Public API, Developers' ICD, and programming assistance.
- Integrating sensor models – CSM integration involves both installation of the models into IGs, but also the integration of validated sensors into the common model. SE Core welcomes and will assist with incorporation of models in order to raise the value of both CSM and the sensor models.

REFERENCES

SE Core DVED (2007). Common Sensor Model ICD, 26 April 2007.

SE Core DVED (2007), Common Sensor Model Software Design Document, 6 March 2007.

Haberman, F.W., and Patrey, J, LCDR, USN, (2007) Navy Aviation Simulation Master Plan Portable Source Initiative (NPSI), A case study in software reuse success, and positive return on investment (ROI), Proceedings of HSIS 2007: ASNE Human Systems Integration Symposium (HSIS), March 19-21, 2007, Annapolis, Maryland, USA

J. Clark, B. Colbert, K. Pribadi, J. Riegler, and G. Anderson, Display Design Concepts For Physics Based Stimulation of Night Vision Goggles, *Proc. Image 2005 Conference*, Scottsdale, Arizona, 10-14 July 2005.

L. Martin and J. Clark, Physics Based Simulation of Night Vision Goggles, *Proceedings IMAGE 2000*, Scottsdale, Arizona, July 2000.