Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Infrared Scene Generation (IRSG)

Developer's Guide

M. Eric Rouleau
LTI inc.
2700, De Carthagène,
Québec, QC
G2B 5M4

Contrat Number: W7701-52709
Scientific Authorities:
Jean-François Lepage
(418) 844-4000 Ext.: 4291
Nathalie Harrison
(418) 844-4000 Ext.: 4604

The scientific or technical validity of this Contract Report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of Defence R&D Canada.

Defence R&D Canada – Valcartier
Contract Report
DRDC Valcartier CR 2008-258
September 2008

Canada

LTI Informatique & génie
Software & engineering

Infrared Scene Generation (IRSG)

Developer's Guide

September 2008

Prepared By:

M. Eric Rouleau - LTI

Prepared For:

M. Jean-François Lepage and Ms. Nathalie Harrison

Defence R&D Canada - Valcartier

W7701-52709 (CR 2008-258)

LTI2008DES-3

Authors

M. Eric Rouleau - LTI

Reviewed By

M. Jonathan Richard - LTI

© Sa majesté la reine, représentée par le ministre de la Défense nationale, 2008

© Her Majesty the Queen as represented by the Minister of National Defence, 2008

2700, De Carthagène,
Québec, Qc
Canada
G2B 5M4

Website: www.ltinfo.ca
E-mail: contact@ltinfo.ca

Release date : September 2008

Category : Contract Report

Version : 1.5

Comments :

Abstract

This contract report presents the first implementation phase of 3D infrared signature modelling compatible with the KARMA simulation environment developed on behalf of Defence Research and Development Canada – Valcartier in the framework of Public Works and Government Services Canada contract number W7701-052709/001/QCL "Development and exercise simulation". The main objective of this first implementation is to demonstrate the feasibility and the benefits of 3D infrared signature modelling as opposed to punctual signatures. Free and open source software as well as commercial off-the-shelf tools have been considered, allowing the reduction of the implementation time and increasing reliability.

This report is presented as a developer's guide of the infrared scene generation module and describes the implementation of the scene generation process tailored to KARMA. The first implementation phase of 3D infrared signatures improves signature modelling in KARMA and demonstrates the applicability of a 3D viewer for infrared scene generation and the higher level of fidelity that is reached for infrared guided weapon engagement simulations. However, the current implementation could be improved to offer pre-computations and support the former punctual signatures.

Résumé

Ce rapport de contrat présente l'implémentation de l'outil de modélisation de signature qui a été développé pour la première phase d'implémentation de la modélisation de signatures infrarouges compatible avec l'environnement de simulation KARMA développé dans le cadre du contrat numéro W7701-052709/001/QCL intitulé « Development and exercise simulation » émis par Travaux publics Canada pour le compte de Recherche et développement pour la défense Canada - Valcartier. L'objectif principal de cet outil est de supporter la modélisation 3D des signatures infrarouges. Des logiciels libres de même que des outils commerciaux ont été utilisés pour réduire le temps de développement et pour augmenter la fiabilité.

Ce rapport prend la forme d'un guide de développement de l'outil « Signature Modeling and Analysis Tool » (SMAT) et présente les détails de l'implémentation de cet outil qui permet la gestion de la base de données des propriétés infrarouges d'un modèle 3D qui est utilisée lors de la génération d'une scène infrarouge. SMAT offre un plein contrôle sur les conditions environnementales et sur les caractéristiques d'un senseur pour fins d'analyse. De plus, SMAT permet l'étude des compromis en fonction des choix de modélisation. La génération de scène infrarouge KARMA est utilisée pour générer une scène infrarouge telle que vue par le senseur et des méthodes d'analyses sont offertes pour aider à la modélisation. Tous les résultats d'analyse peuvent être exportés pour effectuer des analyses ultérieurement ou pour fins de référence. Cet outil est très pertinent pour la modélisation de signatures infrarouges, mais des améliorations devront y être apportées pour mieux supporter le processus de modélisation de signatures infrarouges.

Executive Summary

The representation of infrared signatures is the main factor influencing susceptibility of a target being acquired and tracked by an infrared guided weapon. In order to represent a complete infrared signature, the spatial, spectral, and temporal aspects must be taken into account in the signature modelling. However, depending on the parameters, these aspects of the modelling might compromise the accuracy of the resulting signature or be too demanding on computation resources.

This LTI inc. contract report presents the first implementation phase of 3D infrared signature modelling compatible with the KARMA simulation environment developed on behalf of Defence Research and Development Canada – Valcartier in the framework of Public Works and Government Services Canada contract number W7701-052709/001/QCL "Development and exercise simulation". The main objective of this contract was to increase the level of fidelity of infrared guided weapon engagement simulations using the KARMA simulation environment. The equivalent of 78 months/person was invested in that effort.

Sensor models have been improved from low to medium-high fidelity by including signal processing and an infrared scene generation module was integrated to KARMA. This report is presented as a developer's guide of this infrared scene generation module. The signature modelling in KARMA, based on in-band point source intensities, was revisited to introduce 3D infrared signature modelling. The infrared scene generation implementation had to properly interface with the existent KARMA simulation infrastructure. Furthermore, since the infrared signatures would have to be updated several hundreds of times during an engagement, computation resources had to be kept as low as possible to avoid slowing down the entire simulation.

An approach based on 3D models has been implemented to benefit from the huge computing power of commercial off-the-shelf graphics card while reducing the implementation time and demonstrating rapidly its feasibility. The OpenSceneGraph library was integrated to the infrared scene generation module to allow the management of 3D models into a scene. Although this library is intended for rendering in the visible band, the availability of an application programming interface allowed performing additional modifications to represent infrared scenes. OpenSceneGraph relies on OpenGL® for optimized scene rendering but this task is accomplished by OSMesa (similar to the OpenGL® specification) which allows 16-bit rendering. An infrared property database is associated to a 3D model of the supported format (OpenFlight). Indexes of this database are set to each polygon of a 3D model to specify the corresponding temperature and material and the KARMA infrared scene generator produces an infrared image as seen by a sensor that is used by a sensor for signal processing (tracking).

This first implementation phase represents a major step in the modelling of infrared signature since it offers a maximum of flexibility while ensuring accuracy. The infrared properties of the database can be defined to best fit the modelling, spectral parameters can be of any resolution and the number of polygons of a 3D model can be set according to the desired level of detail. The computation of the total apparent

surface radiance of each polygon is performed spectrally, without pre-computations. Improvements shall be considered to better support the varying level of details of the infrared signature modelling (e.g. pre-computations and support the former punctual signatures).

Rouleau, E., 2008. Infrared Scene Generation (IRSG) Developer's Guide. DRDC Valcartier CR 2008-258

Sommaire

La représentation de signatures infrarouges est le facteur principal affectant la susceptibilité d'une cible à être détectée et poursuivie par une arme guidée à l'infrarouge. La modélisation doit considérer les aspects spatiaux, spectraux et temporels pour qu'une représentation de signature infrarouge soit complète. Cependant, en fonction des paramètres utilisés, ces aspects de la modélisation peuvent compromettre la précision des signatures générées ou nécessiter beaucoup de ressources de calcul.

Ce rapport de contrat de LTI Inc. présente la première phase d'implémentation de la modélisation de signatures infrarouges compatible avec l'environnement de simulation KARMA développé dans le cadre du contrat numéro W7701-052709/001/QCL intitulé « Development and exercise simulation » émis par Travaux publics Canada pour le compte de Recherche et développement pour la défense Canada - Valcartier. L'objectif principal de ce contrat est d'augmenter le niveau de fidélité des simulations d'engagements d'armes guidées à l'infrarouge dans l'environnement de simulation KARMA. L'équivalent de 78 mois/personne a été investi dans cet effort.

Les modèles de senseurs ont été améliorés pour passer d'un bas niveau de fidélité à un niveau moyen-élevé en incluant du traitement de signal et un module de génération de scène infrarouge a été intégré à KARMA. Ce rapport prend la forme d'un guide de développement de ce module de génération de scène infrarouge. La modélisation de signature dans KARMA, basée sur des intensités ponctuelles en bandes, a été revue pour ajouter la modélisation de signatures infrarouges 3D. L'implémentation de la génération de scène infrarouge devait s'intégrer adéquatement à l'infrastructure de simulation existante de KARMA. De plus, en raison de la fréquence de mise à jour d'une signature infrarouge lors d'une simulation d'engagement, les ressources de calcul devaient être limitées et optimisées pour éviter de ralentir la simulation.

Une approche basée sur les modèles 3D a été développée pour tirer profit de la puissance de calcul des cartes graphiques commerciales tout en réduisant le temps de développement, et ainsi démontrer rapidement la faisabilité de cette approche. L'implémentation de la génération de scène infrarouge est basée sur des logiciels libres de même que des outils commerciaux. La librairie OpenSceneGraph a été intégrée au module de génération de scène infrarouge pour effectuer la gestion des modèles 3D sous forme de scène. Cette librairie est conçue pour faire de l'affichage dans la bande visible, mais il est possible de faire du traitement additionnel à l'aide d'une interface de programmation pour convertir la scène en infrarouge. OpenSceneGraph utilise la librairie OpenGL® pour l'affichage optimisé via la carte graphique, laquelle est substituée par OSMesa (similaire à la spécification OpenGL®) qui permet la génération d'images en 16 bits. Une base de données de propriétés infrarouges est associée à un modèle 3D du format OpenFlight. Les indexes de cette base de données servent à spécifier une température et un matériau pour tous les polygones du modèle 3D et le module de génération de scène infrarouge est utilisé pour générer une scène infrarouge telle que vue par le senseur utilisée pour le guidage.

Cette première phase d'implémentation représente une étape majeure dans la modélisation des signatures infrarouges puisqu'elle offre un maximum de flexibilité en ce qui a trait à la modélisation de signatures et assure la précision des signatures générées. Les propriétés infrarouges de cette base de données peuvent être définies pour permettre la meilleure adéquation à la modélisation, les paramètres spectraux peuvent avoir différentes résolutions et le nombre de polygones des modèles 3D peut être adapté au niveau de détail désiré. Le calcul de la radiance apparente totale de surface des polygones est fait dans le domaine spectral, sans pré-calcul de simplification. L'implémentation actuelle peut être améliorée en offrant du pré-calcul pour alléger le traitement et en supportant les signatures ponctuelles.

Rouleau, E., 2008. Infrared Scene Generation (IRSG) Developer's Guide. DRDC Valcartier CR 2008-258

Table of Contents

Abstract i

Résumé ii

Executive Summary iii

Sommaire v

Table of Contents vii

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Overview 1

1.1.1 Modelling 2

1.1.2 Simulation 2

1.2 Process 3

1.3 Implementation 4

2 Definitions 6

2.1 Coordinate System 6

2.2 IR Property Database 7

2.3 Spectrum Data Type 9

2.3.1 Analytic 9

2.3.2 In-band 12

2.4 SceneGenerator3D 13

2.4.1 Scene Graph 14

2.4.2 View Frustum 15

2.4.3 Animations 16

3 Initialization 19

3.1 Memory Allocation 19
3.2 Settings 20
3.2.1 Textures 20
3.2.2 Lighting 20
3.2.3 Fog 20
3.2.4 Blending Mode 21
3.2.5 Antialiasing 23
4 Process 25
4.1 Update Entities 26
4.2 Update Scene Graph States 27
4.2.1 Updating Scene 27
4.2.2 Sensor Point of View 27
4.3 Cull Polygons 28
4.4 Update Polygons 28
4.4.1 Compute Radiances 28
4.4.2 Compute Scaling Factor 29
4.4.3 Set Polygons Color/Transparency 30
4.5 Render Scene 30
4.6 Compute Angular Irradiance Distribution 31
5 Modifications to OpenSceneGraph 33
5.1 OpenFlight 33
5.2 Rendering 33
6 Future Work 35
References 37
Abbreviations and Acronyms 39
Appendix 1 41

List of Figures

Figure 1 – Scene generation steps 4
Figure 2 – Suite of tools of the IRSG implementation 5
Figure 3 – Coordinate systems used in the IRSG module, a) NED convention and b) Z-up convention 6
Figure 4 – IR Property Database UML class diagram 8
Figure 5 – Spectrum resampling 10
Figure 6 – Product operation example between analytic spectrums 11
Figure 7 – Error represented as an area for the inferior limit of product operation example when a point with a value of 0 is extrapolated (Approach A) or when the inferior limit is increased (Approach B) 11
Figure 8 – Bands representation of an in-band spectrum 12
Figure 9 – Graphical representation of an in-band spectrum 12
Figure 10 – IRSG UML class diagram 13
Figure 11 – Sequence diagram for the scene generation using the getImageIrradiance method 14
Figure 12 – OSG nodes used to organize entities in the scene graph 15
Figure 13 – Frustum culling using Near and Far planes 16
Figure 14 – Representation of the pre-allocated memory space for 3D rendering 20
Figure 15 – Representation of polygon transmission (surface 1 is behind surface 2) 23
Figure 16 – Scene generation UML sequence diagram 26
Figure 17 – Supersampling and downsampling operations 31

This page intentionally left blank.

List of Tables

Table 1 – List of possible new or enhanced features in the IRSG 36

This page intentionally left blank.

1 Introduction

This contract report presents the first implementation phase of 3D infrared signature modelling compatible with the KARMA simulation environment. This report is presented by LTI inc. as a developer's guide for the infrared scene generation (IRSG) module developed as a part of the contract number W7701-052709/001/QCL "Development and exercise simulation". The main objective of this contract was to increase the level of fidelity of infrared guided weapon engagement simulations using the KARMA simulation environment in order to study the electro-optical self-protection of transport aircraft against infrared (IR) guided threats. The work was carried out from December 2006 to May 2008. Thus, this report describes the development state at the end of the contract.

IRSG is the process of computing IR signature of a scene according to spectral information. It provides a 2D image of the scene, in radiometric units, as seen by a given sensor. The contract mentioned above involves the modelling of high fidelity seeker models that operate in the IR spectral band. The signature modelling in KARMA, based on in-band point source intensities, was then inadequate since the spectral and the spatial aspects of the signature are simplified. Thus an approach was developed to manage detailed signature models that are used to generate an infrared scene dynamically based on 3D models. Although the first implementation phase was aimed at full digital simulations, the goal was to develop an approach that would also fit the purpose of future time-critical hardware-in-the-loop simulations. Therefore, it was important to allow the control of the level of detail as prescribed by the KARMA philosophy [1][2][3][4].

1.1 Overview

KARMA is intended to allow for a varying level of detail, thus the previous punctual signature modelling was kept in the IRSG implementation. In this document, IRSG refers to the 3D signature modelling if not specified otherwise. The IRSG module is used in infrared countermeasure (IRCM) simulations, and also by the Signature Modeling & Analysis Tool (SMAT) for 3D signature modelling. For more specific details about SMAT, the reader is referred to the SMAT user's guide [5] and the SMAT developer's guide [6].

The IRSG is based on the association of an IR Property Database with a visible 3D model (e.g. one used in a 3D viewer). A 3D model is specified for each entity that is a part of the scene (*Scene*) and the information is gathered as a scene graph. This approach eases access to the 3D model information that is used for the *Scene* since the model is organized as a hierarchy of polygons having associated information

such as color and transparency. All information about 3D models is then available in the scene graph and used to generate an image of the *Scene* (also called *Image*) appropriately. This step is referred as scene rendering.

An *Image* is rendered for a given point of view, line of sight (LOS), resolution and field of view (FOV). The sensor spectral response is taken into account as well as the atmospheric transmission between the sensor and each entity into the *Scene*. The IRSG updates the entities position and orientation in the scene graph and updates polygons of each 3D model to reflect the sensor IR detection (apparent surface radiance). At this point, the IRSG acts the same way as to display a visible *Image*, but with updated polygon Red Green Blue Alpha (RGBA) values. The texture and lighting capabilities used for visible display are disabled as discussed in Section 3.2. The IRSG then read back the *Image* from the memory by converting the color values of the resulting image pixels into apparent radiance, giving an apparent radiance image. This image can also be converted into an angular distribution of irradiance (quantity used by the seekers models).

1.1.1 Modelling

3D signature modelling is accomplished using SMAT and Remo3D® [1]. The 3D model is edited using Remo3D to associate a temperature index and a material index to each polygon of the 3D model. Two properties already defined in the OpenFlight format are used to store indexes. These are *IR Color Code* and *IR Material Code*, corresponding respectively to the temperature and material indexes. SMAT allows for the edition of the IR Property Database that maps indexes to specific kind of surfaces with a given temperature and spectral emissivity $\varepsilon(\lambda)$, reflectivity $\rho(\lambda)$ and transmissivity $\tau(\lambda)$.

SMAT is based on the IRSG module of KARMA and shares a few modules that are involved in an IRCM simulation. This does not necessarily provide optimized computations since some operations are repeated for each analysis. SMAT allows controlling the parameters that are used when performing signature analysis of an IR signature model. The point of view is set easily as opposed to a typical simulation where the point of view depends on entities behavior and parameters involved into the scene generation that are provided by a scenario contained in Extensible Markup Language (XML) files.

1.1.2 Simulation

A typical scenario contains entities and some of them include IR sensors. In order to be detected by an IR sensor, an entity must have an IR signature as described in Section 1.1.1. SMAT is used to create a database of temperatures and materials that are associated to a 3D model of the OpenFlight (.flt) file format. This is accomplished by giving the database the same name as the 3D model with the database extension (.db) and saving the database along with the 3D model. The geometry of an entity is specified using the *Model3D* parameter of the *Structure* class.

---

[1] In order to preserve readability, trademark and registered symbols are shown the first time a tradename is encountered.

Engagement scenarios are defined using XML files. A *Scene* composed of 3D models is created during the simulation initialization only if there is at least one sensor that uses the IRSG. A scene graph is created; memory is allocated for the scene generation process and settings for scene rendering are selected. The initialization is presented in Section 3. Each entity having a valid 3D signature model (database associated to an OpenFlight model) is added to the scene graph and the IR Property Database is loaded in memory. When available, animations are set according to the frame rate settings and synchronized to the simulation. The scene graph reflects the KARMA *Theatre*, so 3D models are added to the scene graph as entities are launched or destroyed when a valid 3D signature model is available. When a sensor is being executed, it gets an *Image* of the *Scene* for its current point of view. The *Scene* is updated only when a sensor is executed.

1.2 Process

Figure 1 presents the high-level steps of the scene generation process. These steps are detailed in Section 4. First, the IRSG updates the position and orientation of the entities in the scene graph according to their states in the *Theatre* (Section 4.1). The IRSG updates the scene graph states (e.g. animations) and sets the point of view using the sensor position and orientation relative to the associated KARMA *BaseEntity* (Section 4.2). The IRSG selects the polygons that might be visible in the *Scene* (Section 4.3). This step is referred as polygon culling. Polygons that are not in the view frustum presented in Section 2.4.2 are discarded. Then, the IRSG updates the polygons color and transparency by computing the radiance of the polygons of each entity using the IR Property Database and including the spectral response of the sensor (Section 4.4). Radiance is converted into a color value using a scaling factor that ensures that the whole range of radiance of the *Scene* is represented. Finally, the IRSG generates a 2D image as seen by the sensor (Section 4.5) and converts it into an angular distribution of irradiance also referred as the *Image* (Section 4.6). The average time to generate an *Image*, which depends on many properties such as 3D models, spectrums or dimensions, varies between 100 to 200 ms.

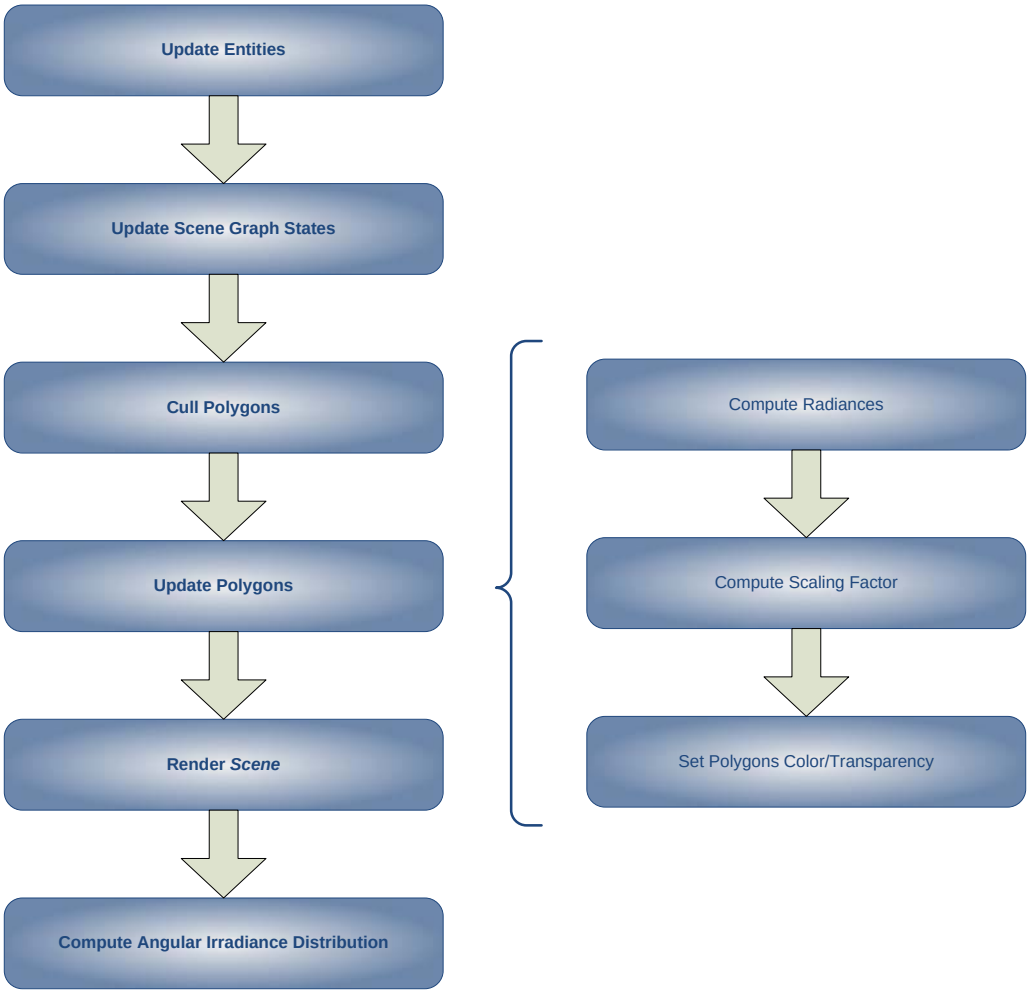

Figure 1 – Scene generation steps

1.3 Implementation

Figure 2 presents the suite of tools that is used for the first implementation phase of the IRSG. The modelling stage is the responsibility of SMAT, which is based on the wxWidgets library (version 2.8) for graphical user interface (GUI) development. Scene generation is performed at the simulation stage and implemented in the KARMA simulation framework. Finally, the 3D rendering stage is based on the OpenSceneGraph (OSG) free and open source library (version 2.2.0) [7]. OSG was selected for the management and rendering of the *Scene* composed of 3D models. Although OSG is intended for rendering in the visible band, the availability of an application programming interface (API) allows the IRSG to perform additional computation to represent IR scenes. OSG relies on the Open Graphics Library (OpenGL®) for rendering optimized computer graphics on any platform [8]. OSG supports many standard 3D model formats including OpenFlight, one of the most widespread formats. The Mesa/OSMesa free and open source library (version 7.0.1) [9] replaces the OpenGL library in order to support 16-bit rendering.

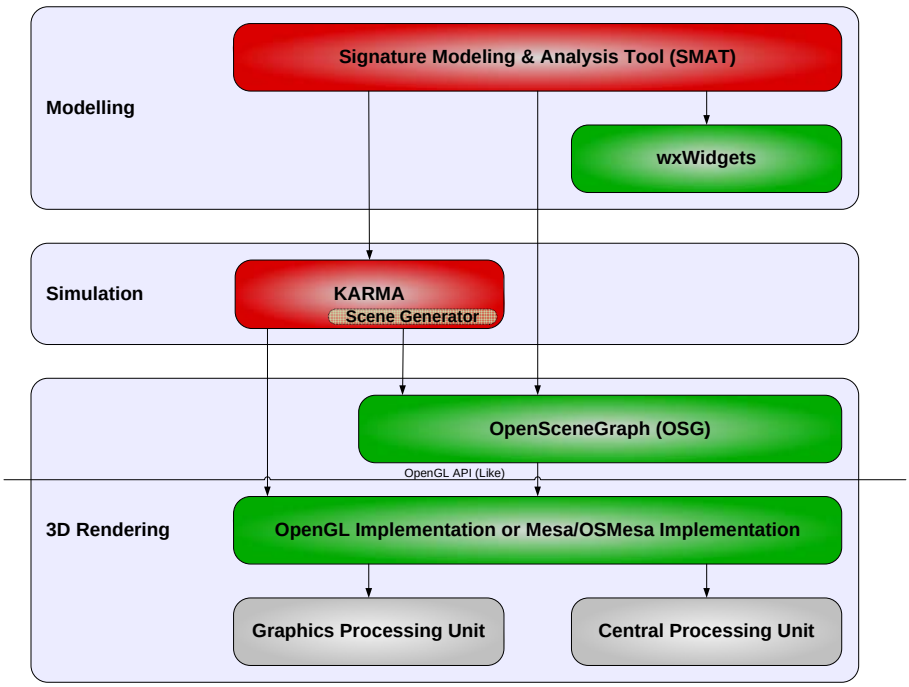

Figure 2 – Suite of tools of the IRSG implementation

2 Definitions

2.1 Coordinate System

Two coordinate systems are used for the entities involved in the IRSG module. The North East Down (NED) convention is used in KARMA. Every entities of the *Theatre* have a position and an orientation that are defined according to the NED reference system. This reference system is shown on Figure 3a). Another convention is used in OSG; the Z-up convention that is shown on Figure 3b). This reference system is used when managing entities (geometries) in the scene graph. The 3D signature models in OpenFlight format must also be defined using this reference system.

a) b)

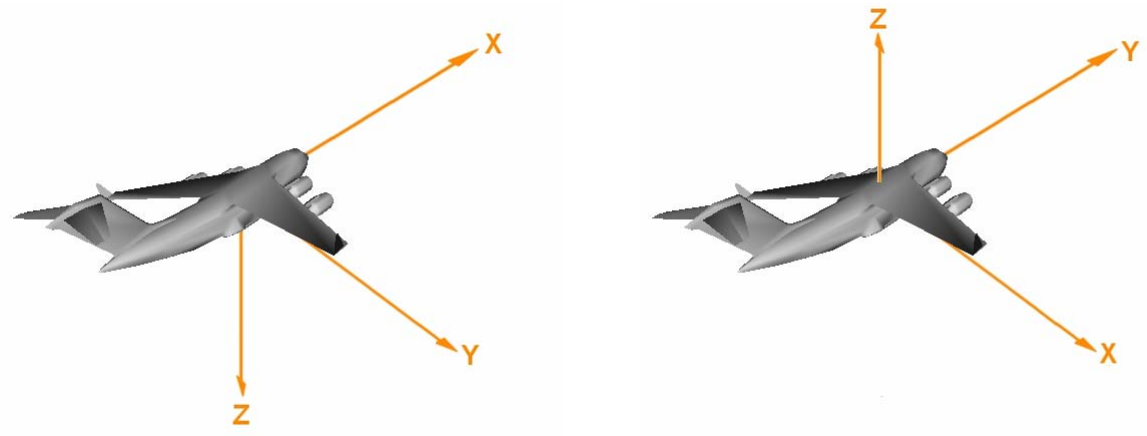

Figure 3 – Coordinate systems used in the IRSG module, a) NED convention and b) Z-up convention

2.2 IR Property Database

This section presents the implementation of the IR Property Database that is used to store radiometric properties of an entity. For more specific details about the properties, the reader is referred to the SMAT user's guide [5].

The *Database* class was developed to manage IR properties that are separated into two categories: temperature and material. The corresponding classes, *Temperature* and *Material*, gather properties for each entry of the database. Each entry is stored in a vector, has a name and has an associated index that corresponds to the one set in the 3D model properties as discussed in Section 1.1.1. A material is defined using three intrinsic properties that are stored using the *Spectrum* (Section 2.3) type: emissivity, reflectivity and transmissivity. Figure 4 shows the Unified Modeling Language™ (UML®) class diagram of these classes.

The scaling properties intended for the spatial scaling of the 3D model are implemented in the *Database* class. Although these properties characterize the shape of an entity according to time, they are not related to the radiometric properties but are stored in the IR Property Database. Refer to Section 4.2.1 for details about their use. The *ScaleList* attribute is a map container of the Standard Template Library (STL) that associates a double (entity time) and a vector (scaling factor for the three axis). The scaling is defined according to the Z-up convention presented in Section 2.1.

A database can be loaded from or stored on disk using a binary format tailored to SMAT and IRSG needs. A description of the format is presented in Appendix 1. Methods were created in the Database class to allow basic operations on different types: integer, double, string, *Spectrum* and temperature lookup table (LUT). The database performs version management by the way of a version tag. This approach allows backward compatibility when properties are added or deleted from the IR Property Database.

Figure 4 – IR Property Database UML class diagram

2.3 Spectrum Data Type

The *Spectrum* class allows spectral operations by associating a list of wavelengths to a corresponding list of values. There are two types of spectrum: analytic and in-band. The analytic type supposes that the step between wavelengths is constant while the in-band type supports variable step. Both types assume a value of 0 outside the spectrum band. Operations can be performed using any combination of spectrum types and the resulting spectrum is in-band only if both spectrums are in-band. The following sections present considerations for each spectrum type.

2.3.1 Analytic

The analytic type is the most precise since operations are performed on every wavelength. The wavelength step (resolution) dictates the number of points and, therefore, the precision. Obviously, higher resolution means slower performances.

2.3.1.1 Resampling

During product and addition operations, analytic spectrums are resampled to maximize the resulting precision. The waveband is selected (intersection or combination) and the wavelength values are aligned to the wavelengths of the spectrum having the highest precision. This way, a minimum of interpolation is performed resulting in higher precision. Resampling is shown in Figure 5 for a product operation between two analytic spectrums. The first step is to determine which *Spectrum* has the smallest wavelength step (Δ_λ). Then, the alignment ($\Delta_{\text{alignment}}$) of that *Spectrum* for its step is calculated using the modulo operator which returns the remainder of the division:

$$\Delta_{alignment} = (\lambda_{\min}) \quad modulo \quad (\Delta_\lambda) \; . \tag{1}$$

Finally, the resulting waveband is computed to preserve alignment with constant wavelength steps. The common waveband (minimum and maximum) might be reduced. The *Spectrum* having the smallest precision is linearly interpolated between its original data points for the whole resulting waveband, using the smallest wavelength steps.

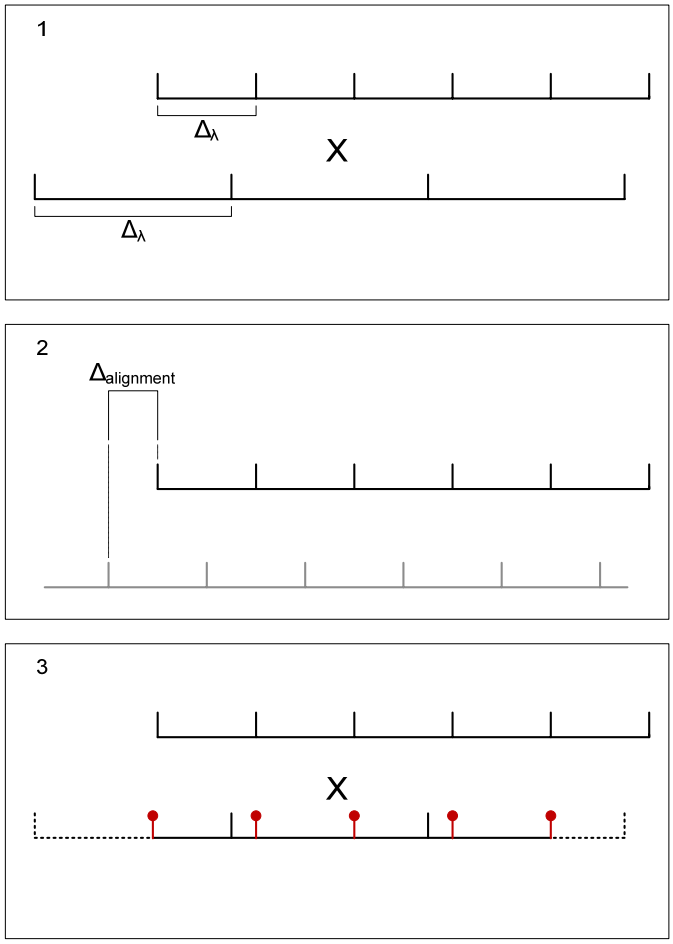

Figure 5 – Spectrum resampling

As an example, Figure 6 presents a product operation over analytic spectrums. The *Curve 1* has a step of 0.03 µm while *Curve 2* has a step of 0.1 µm. The first curve has the highest resolution, so the wavelengths are aligned to this curve (3.98, 4.01, 4.04, 4.07, etc.). Thus, *Curve 2* is resampled at 0.03 µm and the product is computed. At this point, it is important to note the impact of the constant wavelength step constraint when the resulting *Spectrum* has wavelength limits that are not aligned to the step values. As shown in Figure 6, the resulting waveband shall be from 4 to 4.16 µm. However, the inferior limit of 4 µm cannot be represented; only 3.98 or 4.01 µm is possible in order to preserve a constant step of 0.03 µm. Two approaches can be followed: adding a wavelength by extrapolating a point at 3.98 µm with a value of 0 (*Approach A*) or starting with the next valid point at 4.01 µm (*Approach B*). Both ways introduce errors, but errors decrease as the step decreases. Figure 7 presents the area of the product operation, between 3.98 and 4.01 µm, for both approaches. *Approach B* shows the real area that is neglected (0.032685 under the real area) by starting at 4.01 µm instead of 4 µm while *Approach A* shows area estimated (0.01429 over the real area) by extrapolating a point. The *Spectrum* class implements *Approach B*.

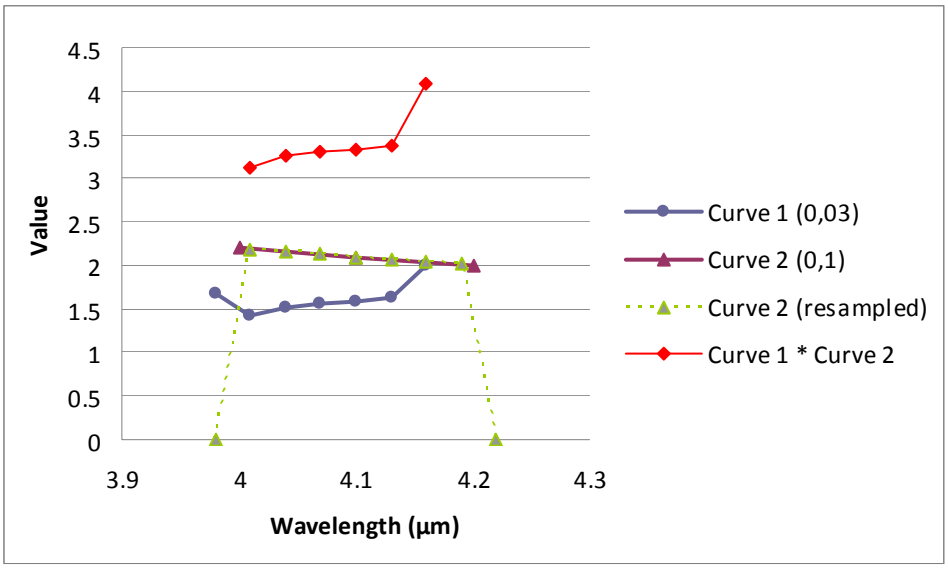

Figure 6 – Product operation example between analytic spectrums

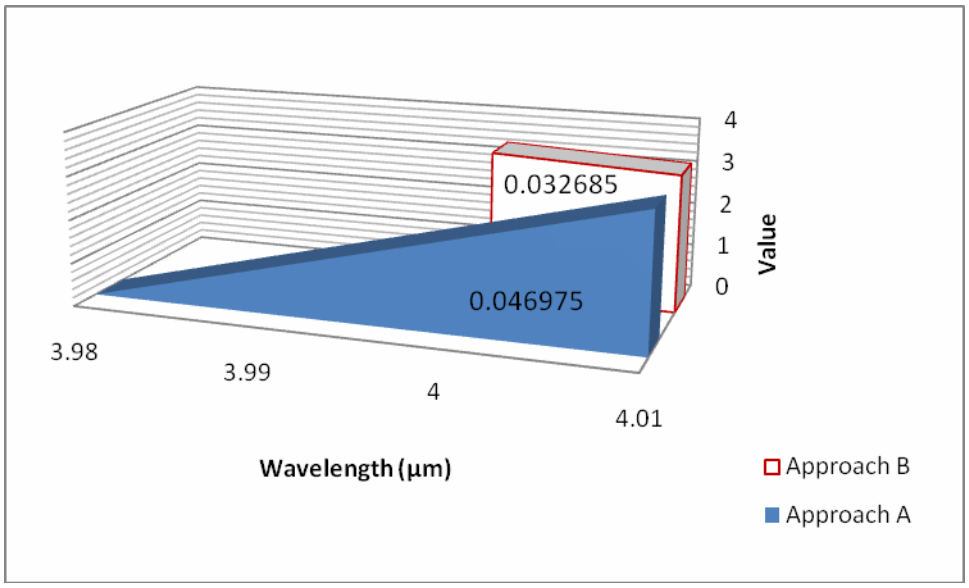

Figure 7 – Error represented as an area for the inferior limit of product operation example when a point with a value of 0 is extrapolated (Approach A) or when the inferior limit is increased (Approach B)

2.3.1.2 Numerical Integration

An analytic spectrum is integrated using the trapezoidal formula:

$$\int_{\lambda_1}^{\lambda_2} f(x)dx \approx (\lambda_2 - \lambda_1)\left[\frac{f(\lambda_1) + f(\lambda_2)}{2}\right], \quad (2)$$

where λ_1 and λ_2 are two successive points of the Spectrum. During the integration process, linear interpolation is performed between wavelengths and 0 is assumed when interpolation is performed outside the waveband (no extrapolation). A discretisation error occurs while performing integration of a spectrum but the smaller the wavelength step, the smaller the discretisation error.

2.3.2 In-band

This type of spectrum is the first spectral representation implemented in KARMA. It allows defining multiple spectral bands easily and operations are performed quickly. Figure 8 presents how a value is associated for a spectral band. A band is defined by two successive wavelengths and the wavelength step can vary within the in-band spectrum. An example of an in-band spectrum having wavelengths 3, 5, 8, 12 µm is shown in Figure 9.

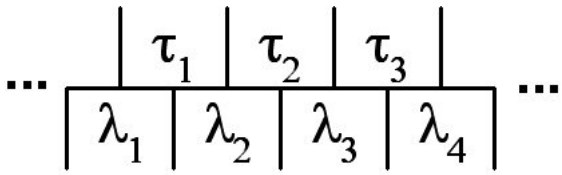

Figure 8 – Bands representation of an in-band spectrum

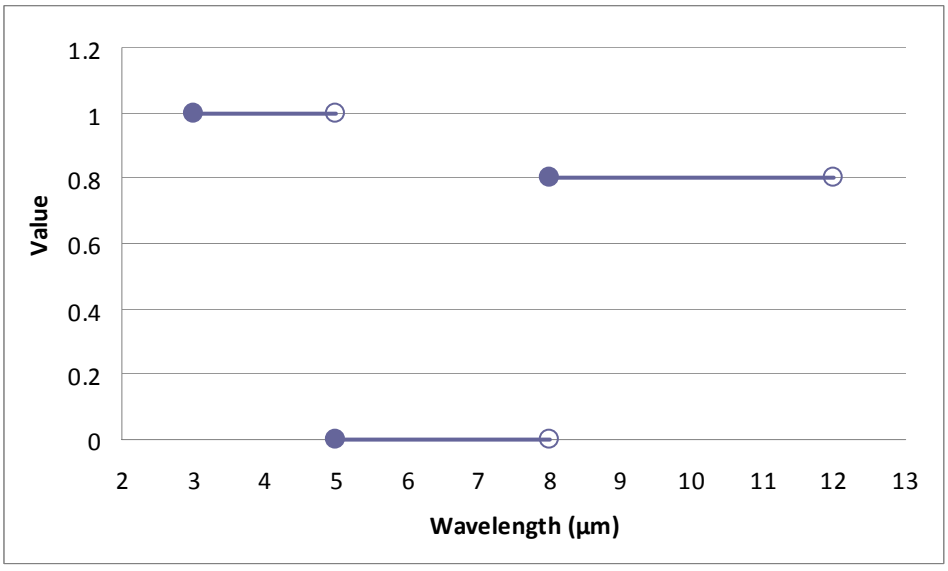

Figure 9 – Graphical representation of an in-band spectrum

2.4 SceneGenerator3D

An IRSG interface (*ISceneManager*) was created to support punctual and 3D approches for scene generation. The punctual scene generation of KARMA is implemented in the *SceneGeneratorPunctual* class while the scene generation based on 3D models is implemented in the *SceneGenerator3D* class. The UML class diagram of the IRSG is shown in Figure 10. A sensor that is derived from the *AbstractSensor* interface has a reference on a scene generator (*ISceneManager*). It is the sensor responsibility to instantiate the appropriate type of scene generator during its initialization. A sensor that performs processing using an *Image* of the *Scene* uses the *SceneGenerator3D* and calls the *getImageIrradiance* method as shown in Figure 11. Refer to Section 4 for more details about the associated process. The *SceneGenerator3D* keeps a scene of 3D models (entities) as a scene graph and entities of that scene are updated according to their orientation and position in the *Theatre*. The classes *GroupVisitor* and *InsertCallbackVisitor*/*CullCallback* are used to manage animations and polygon culling respectively. More details are presented in the following sections.

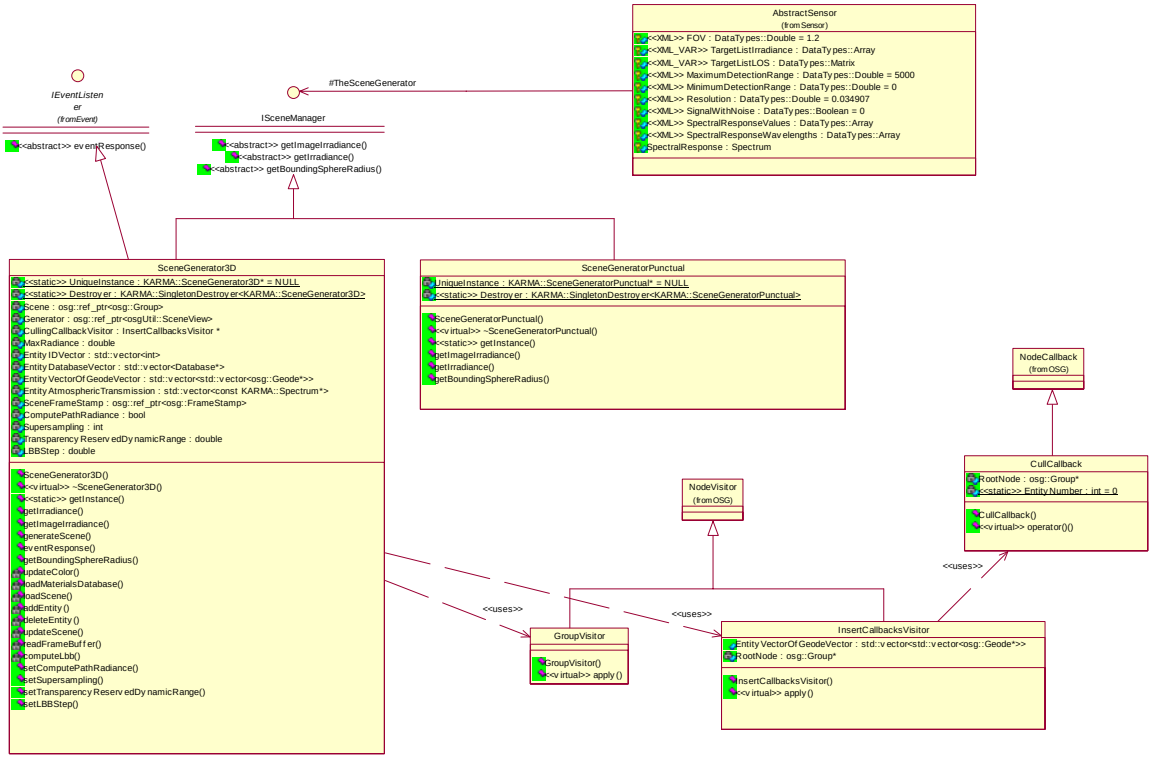

Figure 10 – IRSG UML class diagram

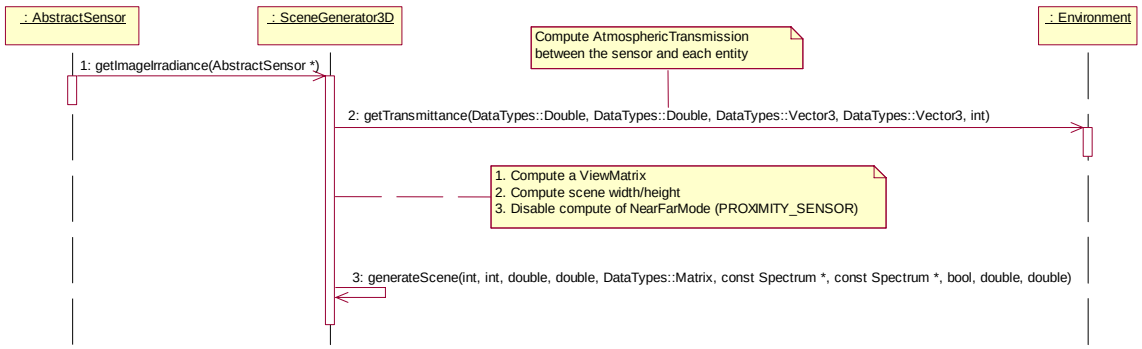

Figure 11 – Sequence diagram for the scene generation using the getImageIrradiance method

2.4.1 Scene Graph

Geometries of the 3D models are managed as a scene graph using OSG. The scene graph reflects the *Theatre* for all entities having an OpenFlight 3D model and an associated IR Property Database.

2.4.1.1 Hierarchy

Geometries of a 3D model are gathered as a hierarchy of nodes. Basically, a 3D model has a root node (*osg::Group*) and is composed of *osg::Geode* objects that are composed of geometries (*osg::Drawable*). The root of the scene graph is a node *osg::Group* and is referred as the *Scene*. An intermediate node *osg::PositionAttitudeTransform* is used as a child node when an entity is added to the *Scene*. Figure 12 presents this hierarchy and an example of the code is shown below.

```
// Create a transformation node at the root of the scene
osg::ref_ptr<osg::PositionAttitudeTransform> xform = new osg::PositionAttitudeTransform();

// Insert the entity transformation node as a direct child of the scene
Scene->addChild(xform.get());

// Insert the entity 3D model as a direct child of the transformation node
xform->addChild(loadedModel.get());
```

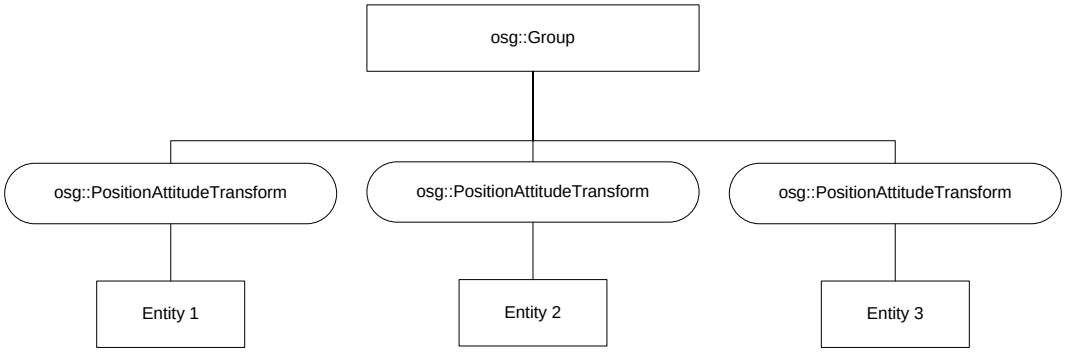

Figure 12 – OSG nodes used to organize entities in the scene graph

The node *osg::PositionAttitudeTransform* offers methods to set the location and the orientation of its children nodes (that represents the entity). Conversion between KARMA and OSG coordinate systems presented in Section 2 is performed while updating entities:

$$\begin{aligned} Position_{OSG} &= Position_{KARMA}(Y, X, -Z); and \\ Euler_{OSG} &= Euler_{KARMA}(-X, Y, Z). \end{aligned} \tag{3}$$

Entities of the *Scene* are associated to the corresponding ones in the *Theatre* using a vector of *BaseEntity* ID attribute (*EntityIDVector*). Each entry of this vector is mapped to the child (*osg::PositionAttitudeTransform*) of the *Scene* having the same index value.

2.4.1.2 States

Nodes of the scene graph have states (or properties) that are used when the 3D rendering is performed. The *osg::StateSet* of the *Scene* allows setting default states for all entities. All the children of the root node, regardless of what the children attribute value is, will inherit the parent node attribute value. However, it is possible for a child node to discard default states by the way of a protection setting (*osg::StateAttribute::PROTECTED*). The main properties are related to textures, lighting, transparency and blending (related to transparency). Settings that are applied are presented in Section 3.2.

2.4.2 View Frustum

Sensor detection ranges can be used when rendering a *Scene*. In the current implementation of the IRSG, it is done only for proximity fuze sensors. OpenGL, and so OSG, offers Near and Far clipping settings. Polygons outside the Near and Far distances are not visible in the *Image*. This process is referred as frustum culling and is shown in Figure 13. However, it is important to note that these settings might compromise the quality of the *Image*. A Near clipping plane that is not set properly can result in artefacts when drawing geometries near the viewpoint, usually permitting the viewer to see into nearby objects. Consequently, the Near clipping plane is often as far as possible toward the viewpoint, in front of the nearest object in the FOV. These settings are computed automatically by OSG to allow for appropriate rendering. Using this mode, OSG attempts to make the most of the available resolution of the depth buffer by tuning the Near and Far clipping planes to closely bound the visible portion of the *Scene*.

The ratio of the Far to Near distances is critical, and unless a corresponding adjustment is made to the Far clipping plane, depth buffer precision is wasted. If the Near distance is set to zero then the standard perspective projection is undefined (refer to the OpenGL online documentation [8]).

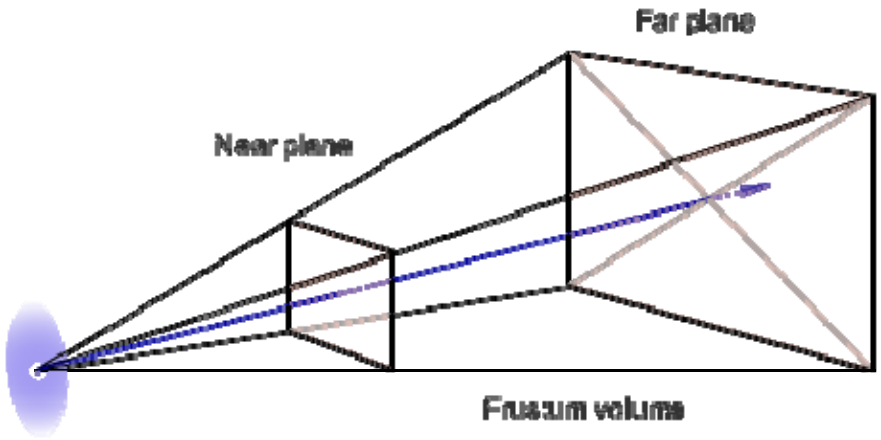

Figure 13 – Frustum culling using Near and Far planes

2.4.3 Animations

Entities of the *Scene* can be animated using 3D models animation offered by the OpenFlight file format [10]. This feature allows flexibility for time varying signature models such as aircraft rotors, plumes, and particles models. The IRSG implementation manages animation settings by the way of the node comments of the OpenFlight format since Remo3D offers a limited support for animation. In addition to associating temperature and material indexes to each polygon of a 3D model, Remo3D is used for defining animation settings (forward or swing animations, repetitions). Using Remo3D, the frame rate of an animation is specified in frames per second (fps) and set in the comment field of the animation node using a specific convention.

```
@dis animation FRAME_RATE
```

An animation is a node *osg::Sequence* and its children nodes represent a frame of the animation. Only one frame (child) is activated at a time. OSG supports animation, but default implementation has a fixed frame rate and is not synchronized with time. Therefore, the next animation frame is selected each time the scene generation is performed. In order to setup animation correctly, a node visitor *GroupVisitor* was created. This class is called each time a model is inserted into the *Scene* to extract animation settings and set the *osg::Sequence* node accordingly. The node visitor scans all nodes of the 3D model that is being inserted into the scene graph. When an *osg::Sequence* node is found, its synchronize option is set and the frame rate is set (default value of 10 fps if not available in the node comment).

```
// Synchronize frames with time
anim->setDuration(1, -1);     // No speed-up and unlimited repetitions
anim->setSync(true);

// Update frames duration according to the frame rate
for (int loop = 0; loop < anim->getNumFrames(); loop++)
    anim->setTime(loop, 1.0/frameRate);      // Frame duration (seconds)
```

The *Scene* time is synchronized to the simulation by the way of an *osg::FrameStamp* attribute.

```
// Allow going back in time (update sequences)
if (DOUBLE_LESS(Theatre::instance()->getTime().getValue()
                + EPSILON, SceneFrameStamp->getSimulationTime()))
{
       SceneFrameStamp->setSimulationTime(0);
       Generator->update();                 // Reset simulation time for all sequences

       GroupVisitor resetGroups;
       resetGroups.traverse(*Scene); // Reset sequences
}

// Force new frame when duration is elapsed
SceneFrameStamp->setSimulationTime(Theatre::instance()->getTime().getValue() + EPSILON);

// Pass frame stamp to the SceneView so that the update, cull and draw
// traversals all use the same FrameStamp
Generator->setFrameStamp(SceneFrameStamp.get());
```

The *if* condition is used for SMAT which allows generating images for any simulation time. The method *update* (*osgUtil::SceneView*) is called to notify animations that the simulation time has changed and then, *GroupVisitor* resets animation to display the first frame. EPSILON is added when setting the simulation time of the *FrameStamp* to the next frame to be activated (double comparison issue).

Entities being inserted during a simulation are also synchronized with an *osg::FrameStamp* attribute but an update is performed using *osgUtil::UpdateVisitor* to allow their animation starting at this simulation time.

```
// Apply current simulation time to the 3D model being inserted into the scene
SceneFrameStamp->setSimulationTime(Theatre::instance()->getTime().getValue() + EPSILON);
SceneUpdateVisitor->reset();
SceneUpdateVisitor->setFrameStamp(SceneFrameStamp.get());

// Use the frame number for the traversal number
if (SceneFrameStamp.valid())
{
    SceneUpdateVisitor->setTraversalNumber(SceneFrameStamp->getFrameNumber());
}

// Update frame stamp (animations will begin on the current simulation time,
// begin at 0 s otherwise)
loadedModel->accept(*SceneUpdateVisitor.get());
```

3 Initialization

This section presents the operations that are performed once, when the IRSG is used for the first time during a simulation.

3.1 Memory Allocation

The *SceneGenerator3D* is a singleton, so there is only one instance for a simulation. The class constructor is called when *SceneGenerator3D* is called for the first time and memory is allocated for the scene generation process. OSMesa is used to create an off-screen context that will be used transparently by OSG to generate an *Image* of the *Scene*. The *OSMesaCreateContextExt* method is used to create this context, the RGBA channel type is selected with 16-bit per channel. This allows representing colors (pixels) using 65536 grey levels. The *OSMesaMakeCurrent* method is used to associate the memory to the context. Memory is allocated for predetermined dimensions and supersampling (SS) settings:

$$TotalSize = \left(Max_{Width} Max_{SS}\right)\left(Max_{Height} Max_{SS}\right) N_{Channels} N_{Bytes} \ , \tag{4}$$

where:

- TotalSize is the number of bytes of memory;
- Max_{Width} and Max_{Height} are the maximum image width and height in pixels (fixed to 1000);
- Max_{SS} is the maximum supersampling factor (fixed to 4);
- $N_{Channels}$ is the number of channels (fixed to 4 since RGBA is used); and
- N_{Bytes} is the number of bytes per pixel (fixed to 2 since 16-bit is used).

By allocating a fixed amount of memory in the constructor, memory is not continuously allocated/deleted since many sensor configurations might be used. The memory necessary for the scene generation is set using a viewport. Refer to Section 4.5 for more details. Figure 14 presents the concept of the pre-allocated memory space. The total memory space is represented by the outer rectangle while the viewport is shown as the inner rectangle. *OSMesaDestroyContext* method is called when the *SceneGenerator3D* instance is destroyed.

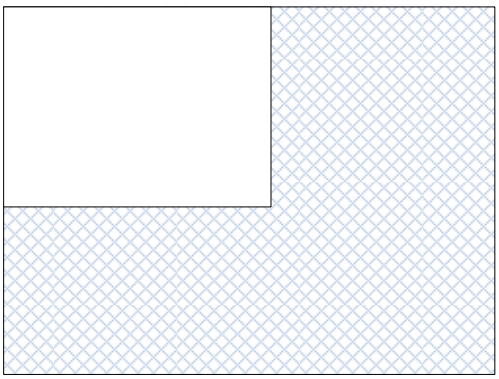

Figure 14 – Representation of the pre-allocated memory space for 3D rendering

3.2 Settings

The settings of the *Scene* are used during the scene generation process and are stored in *osg::StateSet* which is accessed using the *getOrCreateStateSet* method. As presented in Section 2.4.1.2, default settings can be set for the scene graph.

3.2.1 Textures

By default, OpenGL includes textures when rendering 3D models. The IRSG disables textures (*GL_TEXTURE_2D*) since radiometric information is only provided using polygon colors, set according to the total apparent radiance, and is not related to textures which are usually intended for visible effects.

3.2.2 Lighting

The IRSG also disables lighting effects (*GL_LIGHTING*) in order to remove any visible effects (e.g. shadow).

3.2.3 Fog

By default, OpenGL disables fog effects (*GL_FOG*) when rendering 3D models. For the same reasons that the textures and lighting effect were disabled, the fog effects are kept disabled.

3.2.4 Blending Mode

By default, OpenGL does not perform blending and the source color overwrites the destination color. In order to use blending, *glEnable(GL_BLEND)* must be called while defining the proper blending function by *glBlendFunction(GLenum sourceFactor, GLenum destinationFactor)*. The possible values for the *sourceFactor* and the *destinationFactor* are described in Table 1 and the equation used by the blend function to compute the final destination color is given by:

$$\begin{aligned} FDC &= SC \cdot SF + CuDC \cdot DF, \\ &= (R_s, G_s, B_s, A_s) \cdot (R_{sF}, G_{sF}, B_{sF}, A_{sF}) + (R_d, G_d, B_d, A_d) \cdot (R_{dF}, G_{dF}, B_{dF}, A_{dF}), \\ &= (R_s R_{sF} + R_d R_{dF}, G_s G_{sF} + G_d G_{dF}, B_s B_{sF} + B_d B_{dF}, A_s A_{sF} + A_d A_{dF}). \end{aligned} \tag{5}$$

Where:

- *FDC* is the final destination color;
- *SC* is the source color;
- *SF* is the source factor;
- *CuDC* is the current destination color;
- *DF* is the destination color;
- (R_s, G_s, B_s, A_s) are the red, green, blue and alpha components of the source color;
- (R_d, G_d, B_d, A_d) are the red, green, blue and alpha components of the destination color;
- (R_{sF}, G_{sF}, B_{sF}, A_{sF}) are the red, green, blue and alpha components of source factor; and
- (R_{dF}, G_{dF}, B_{dF}, A_{dF}) are the red, green, blue and alpha components of destination factor.

Table 1: List of possible source and destination factors for color blending.

Factor Name	**Relevant for**	**Factor Expression (R,G,B,A)**
GL_ZERO	Source or destination	(0, 0 , 0, 0)
GL_ONE	Source or destination	(1, 1, 1, 1)
GL_DST_COLOR	Source	(R_d, G_d, B_d, A_d)
GL_SRC_COLOR	Destination	(R_s, G_s, B_s, A_s)
GL_ONE_MINUS_DST_COLOR	Source	(1-R_d, 1-G_d, 1-B_d, 1-A_d)
GL_ONE_MINUS_SRC_COLOR	Destination	(1-R_s, 1-G_s, 1-B_s, 1-A_s)
GL_SRC_ALPHA	Source or destination	(A_s, A_s, A_s, A_s)
GL_ONE_MINUS_SRC_ALPHA	Source or destination	(1-A_s, 1-A_s, 1-A_s, 1-A_s)
GL_DST_ALPHA	Source or destination	(A_d, A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPHA	Source or destination	(1-A_d, 1-A_d, 1-A_d, 1-A_d)
GL_SRC_ALPHA_SATURATE	Source	(X, X, X, 1) where X = min(A_s,1-A_d)

By using the proper blending mode, the OpenGL blending function can be used to render polygons with transparency effects. The source factor currently used in the IRSG is *GL_ONE* while the destination factor is *GL_ONE_MINUS_SRC_ALPHA*. The blending equation is then given by:

$$FDC = \left(R_s + R_d\left(1 - A_s\right), G_s + G_d\left(1 - A_s\right), B_s + B_d\left(1 - A_s\right), A_s + A_d\left(1 - A_s\right)\right) . \quad (6)$$

The rendering is done from back to front. Then, as shown in Figure 15, when rendering a transparent polygon (surface 2) placed in front of another polygon (surface 1), the final *R* value is given by:

$$R = R_2 + R_1\left(1 - A_2\right) . \quad (7)$$

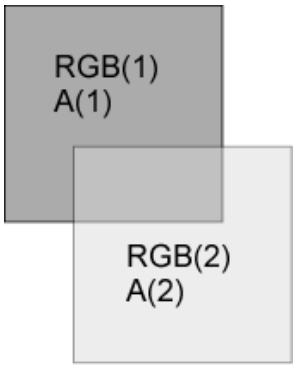

Figure 15 – Representation of polygon transmission (surface 1 is behind surface 2)

3.2.5 Antialiasing

Antialiasing is a technique to create smoother transition between the pixels at the edge of an object and the surrounding pixels. It also attempts to keep a maximum of information from the original data during pixelisation. When generating the *Image*, aliasing could results in significant radiometric inaccuracy.

3.2.5.1 Polygon antialiasing

The polygon antialiasing function (*GL_POLYGON_SMOOTH*) is defined in the basic OpenGL API. It allows to antialias polygons that are in *GL_FILL* polygon mode by using a coverage algorithm. However this technique only works with specific polygon sorting and blending parameters. In particular the polygons must be sorted from front to back and the blending source/destination factors must be *GL_SRC_ALPHA_SATURATE and GL_ONE*. Since these settings are incompatible with the one chosen to handle the transparency (polygon sorted back to front and blending source/destination factors of *GL_ONE* and *GL_ONE_MINUS_SRC_ALPHA*), the polygon antialiasing cannot be used.

3.2.5.2 Multisampling and supersampling

Supersampling is a technique that consists of rendering an image (in the framebuffer) at a higher resolution than the one being displayed. This image is then downsampled to the desired size by averaging the framebuffer pixels in a corresponding pixel of the displayed image.

The multisampling is essentially an optimization of the supersampling since it only supersamples the edges of the polygons (not the interior). It also samples textures and perform light computation only once per group of samples.

The *GL_ARB_multisample* extension defines both multisampling and supersampling implementations. However this extension is not supported by OSMesa. Furthermore the implementation of GLUT on Microsoft Windows does not support this extension either. In order to be able to use the *GL_ARB_multisample* extension, the Microsoft implementation of OpenGL should be used alone or with another toolkit than GLUT. It seems that Simple DirectMedia Layer (SDL) supports the multisample extension but it has not been tested.

Since the need for performing 16-bit rendering was more important than the need for performing antialiasing, the development team has decided to keep going with OSMesa. The simple software implementation of supersampling presented in Section 4.5 has been realized to provide two levels of supersampling.

4 Process

This section describes how the *Scene* is managed to generate an IR output (*Image*). Figure 16 shows the UML sequence diagram of the scene generation process. The *generateScene* method is used to update the IR output of the entities in the *Scene* and generate an *Image* (as seen by the sensor). Entities states are updated from the *Theatre*, the sensor point of view is computed, the apparent radiance of all visible polygons is computed and an image is generated.

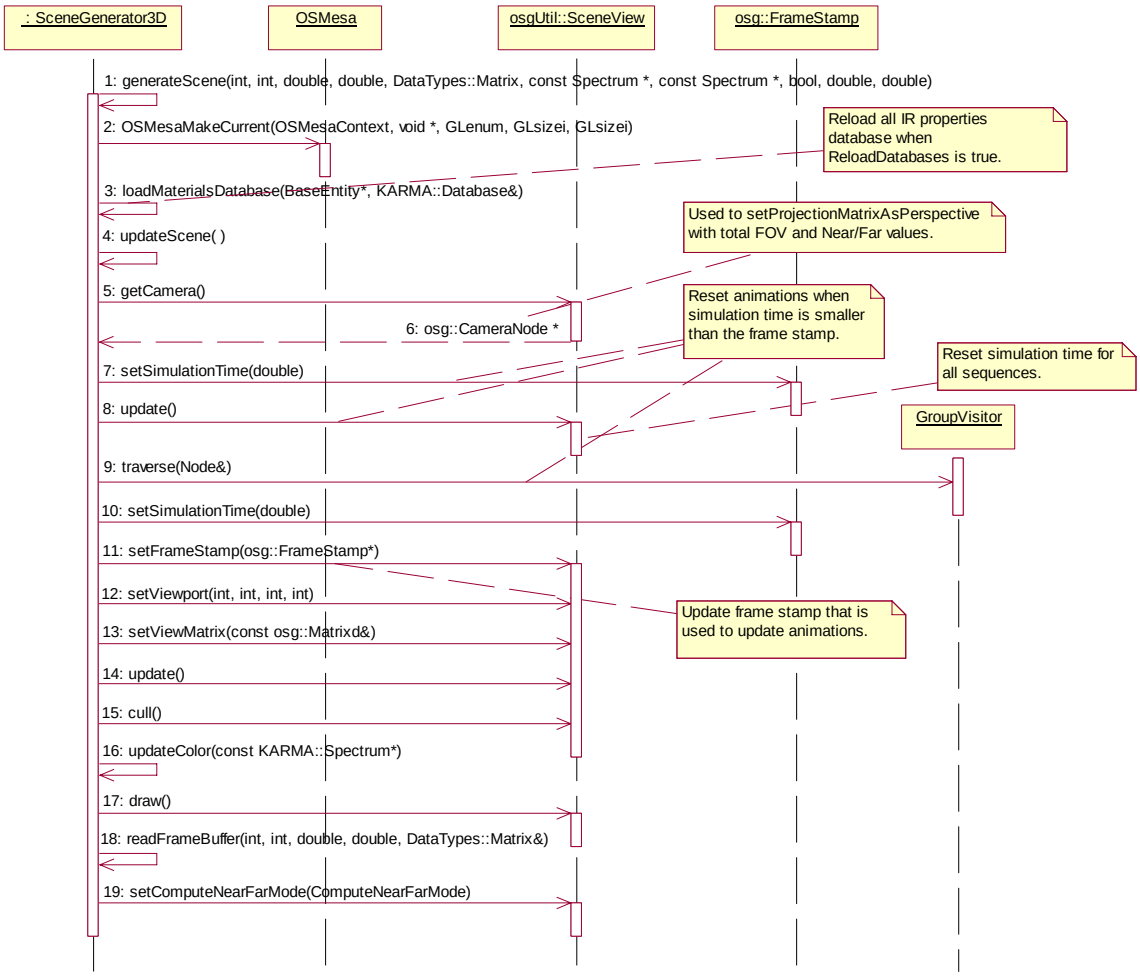

Figure 16 – Scene generation UML sequence diagram

4.1 Update Entities

The *Scene* is updated at the beginning of the scene generation process. As mentioned in Section 2.4.1.1, the position and orientation is updated to reflect the corresponding entities in the *Theatre*. A scaling factor is also applied to the *osg::PositionAttitudeTransform* to represent variation of the entity size with time. The scaling properties of the IR Property Database presented in Section 2.2 are used. When scaling is enabled, the 3D model is also shifted by "-Y Scale" along its y-axis (Z-up convention) in order to represent the trailing aspect of a flare plume.

The IR Property Database (*Database*) is loaded when an entity is added in the *Scene* using the *loadMaterialsDatabase* method. However, there is an argument to the *generateScene* method to allow reloading a database when SMAT uses the IRSG. A reference to the entity database is stored using a

vector of *Database* (*EntityDatabaseVector*). Each entry of this vector is mapped to the child (*osg::PositionAttitudeTransform*) of the *Scene* having the same index value.

Temperature and material properties are associated to polygons of a 3D model using indexes. These indexes are stored in a user defined attribute (*_userData*) that is available for all *osg::Node*. This attribute is a reference to any object that is derived from *osg::Referenced*. Indexes are stored in the user defined attribute when the OpenFlight 3D model is loaded. The OSG implementation is presented in Section 5.1. An object of type *osg::IntArray* contains the temperature and material indexes in the elements 0 and 1 respectively.

```
aDrawable = aGeode->getDrawable(i);
aDrawableIndices = dynamic_cast<osg::IntArray *>(aDrawable->getUserData());
ASSERT(aDrawableIndices != NULL, "Database indices are not defined in the 3D model!");
temperatureIndex = (*aDrawableIndices)[0];
materialIndex = (*aDrawableIndices)[1];
```

4.2 Update Scene Graph States

During this step of the scene generation process, the simulation time is updated at the *Scene* level and the sensor point of view is applied.

4.2.1 Updating Scene

The *Scene* time is synchronized to the simulation as presented in Section 2.4.3. The time is updated during traversal of the *Scene* by a *GroupVisitor* (when rewinding time for SMAT) or when the method *update* (*osgUtil::SceneView*) is called.

4.2.2 Sensor Point of View

OSG uses a matrix of transformation when rendering an image and this matrix is an argument of the *generateScene* method. SMAT generates such a matrix using its GUI while it is computed according to the *AbstractSensor* interface in the *getImageIrradiance* (simulation). The sensor position and orientation are expressed in the inertial reference system using the *BaseEntity* information. Then, a matrix of transformation is created.

$$M_T = M_{Rot} * M_{Trans} \ , \tag{8}$$

where $\mathbf{M_T}$ is the matrix of transformation, $\mathbf{M_{Rot}}$ is a matrix of rotation and $\mathbf{M_{Trans}}$ is a matrix of translation.

The final matrix is inverted to adapt to OpenGL Y-up and Z-up conventions. The *osgGA::MatrixManipulator* (used in SMAT) uses the Y-up convention while all other classes, including matrix manipulators, use the Z-up convention.

The view matrix is applied by calling the method *setViewMatrix* (*osgUtil::SceneView*) and the dimensions of the *Image* that will be produced by the end of the scene generation steps is set by calling

the method *setViewport* (*osgUtil::SceneView*). The viewport allows selecting a subset of the pre-allocated memory space (refer to Section 3.1 for more details).

4.3 Cull Polygons

This step discards all entities polygons that are outside the view frustum presented in Section 2.4.2. The polygon culling process is accomplished by OSG, the *cull* method of *osgUtil::SceneView*, but the IRSG needs to be aware of which polygon might be visible in order to reduce the computation cost of the total apparent radiance.

A notification is made to the IRSG using the class *CullCallBack* that inherits from *osg::NodeCallback*. A node visitor *CullingCallbackVisitor* was implemented to set a reference to a *CullCallBack* when an entity is added to the *Scene*. These classes were shown in Figure 10. During polygon culling, its method *operator()* is called by the cull visitor if the node has not been culled (i.e. node might be visible) and a reference is kept to compute the apparent radiance of the polygons. Refer to Section 4.4 for more details. A reference to an *osg::Geode* object is gathered in the *EntityVectorOfGeodeVector* that is a vector of entities visible polygons (vector of geodes):

```
std::vector<std::vector<osg::Geode*>> EntityVectorOfGeodeVector;
```

Culling is limited to *osg::Geode* nodes, which was proven to be the necessary node of the nodes hierarchy involved in the culling process. A *CullCallBack* is associated to *osg::Transform* nodes to associate the *osg::Geode* nodes to the corresponding entity in the *Scene*. The vector is cleared before each culling operation.

4.4 Update Polygons

The apparent radiance is computed in the *updateColor* method that is separated in two steps. The total apparent radiance is calculated for all polygons located in the view frustum (stored in the *EntityVectorOfGeodeVector* attribute), and then it is converted in a color value. For more specific details about radiometric equations, the reader is referred to the document IRSG for countermeasure simulations [11].

4.4.1 Compute Radiances

During the first step, the total apparent surface radiance ($\mathbf{L}^{\mathbf{surf}}_{\mathbf{app}}$) of all entities polygons located in the view frustum is calculated as well as the apparent background radiance. The total apparent radiance is given by:

$$L_{app}^{surf} = L_{app}^{therm} + L_{app}^{refl} + L_{app}^{path} \quad . \tag{9}$$

It is calculated using properties from the IR Property Database presented in Section 2.2 and specified by temperature and material indexes. More details about these formulas are presented in [11].

In order to reduce the computation cost of this process, $\mathbf{L}^{\mathbf{surf}}_{\mathbf{app}}$ is calculated once for each combination of temperature and material indexes at a given simulation time. This approach is allowed since the atmospheric transmission is assumed constant for all the polygons of an entity. The resulting value is used each time the same pair of temperature and material indexes is found.

The temperature and material indexes, $\mathbf{Index_T}$ and $\mathbf{Index_M}$ respectively, are combined into a unique integer value ($\mathbf{Map_{Key}}$) that represents the key to access the appropriate value:

$$Map_{Key} = (1000 Index_M) + Index_T \quad . \tag{10}$$

Therefore, it is assumed that the value of temperature indexes is less than 1000. A local variable (*mapEntitiesColor*) is used to store $\mathbf{L}^{\mathbf{surf}}_{\mathbf{app}}$ in a map container that is indexed using the previous key and stores values in a map container for each entity in the *Scene*.

```
std::vector<std::map<int, double*>> mapEntitiesColor;
```

Each entry of this vector is mapped to the child (*osg::PositionAttitudeTransform*) of the *Scene* having the same index value.

From the surface temperatures determined by Equation (1) of the SMAT user's guide [5], the spectral blackbody radiance ($\mathbf{L_{bb}}$) is calculated in the *computeLbb* method. An attribute (*LBBStep*) is used to specify the wavelength steps of $\mathbf{L_{bb}}$; the default value is 0.01 µm. Once again, a local variable (*mapEntityLbb*) is used to reduce the computation cost. A map container associates a $\mathbf{L_{bb}}$ *Spectrum* to a temperature index.

4.4.2 Compute Scaling Factor

Once $\mathbf{L}^{\mathbf{surf}}_{\mathbf{app}}$ of all entities polygons located in the view frustum is calculated, radiance values must be converted into a polygon color. Indeed, at a lower level, the scene generation mechanism manages a polygon color instead of a radiance value. Therefore, a scaling factor is computed for this conversion and maximizes the use of the total color span:

$$AR_{Scaling} = AR_{Max}\left(1 + \frac{D_{Range}}{100}\right) , \tag{11}$$

and

$$Color = \frac{AR}{AR_{Scaling}} \ . \tag{12}$$

Where:

- $AR_{Scaling}$ is the scaling factor for total apparent radiance to color conversion ($W/m^2/sr$);
- AR_{Max} is the maximum total apparent radiance in the sensor FOV ($W/m^2/sr$);
- D_{Range} is the dynamic range reserved for transparency;
- AR is the total apparent radiance of a given polygon ($W/m^2/sr$); and
- Color is the color value of a given polygon (between 0 and 1).

$\mathbf{AR_{Max}}$ is stored in the *MaxRadiance* attribute and is determined once all $\mathbf{L^{surf}_{app}}$ are computed. An attribute (*TransparencyReservedDynamicRange*) is used to specify $\mathbf{D_{Range}}$; the default value is 5. This leads to a default value of 5% of the total dynamic range being reserved for transparency effects. The final scaling factor is stored in the *MaxRadiance* attribute.

4.4.3 Set Polygons Color/Transparency

The last step of the polygon update is to update the polygons color. All entities polygons are revisited to set their color according to the value computed in the first step (stored in the *mapEntitiesColor*). Colors are computed using a scaling factor, as shown in Equation (12).

A uniform background is defined using the *setClearColor* method (*osgUtil::SceneView*). This color depends on the scaling factor and cannot be set earlier in the scene generation. Therefore, an additional polygon culling operation is required in order to update the background color for the current rendering.

The polygon transparency ($\boldsymbol{\tau_{eff}}$) is computed as presented in Ref. [11]. When a polygon is transparent, the alpha channel is less than 1, meaning that it is not completely opaque. $\boldsymbol{\tau_{eff}}$ is equivalent to (1 – alpha). When the *Scene* is rendered, the resulting color of a transparent polygon is as presented in Section 3.2.4.

4.5 Render Scene

When the *Scene* is updated as seen by the sensor, rendering is performed. Rendering is accomplished using an instance (*Generator*) of the *osgUtil::SceneView* class. The dimensions of the *Image* to be rendered are set using a *Supersampling* attribute and using the *setViewport* method. The supersampling consists of rendering an image in a higher resolution than the one being displayed and to downsample it afterward to the desired size by taking an average of the sub-pixels. Figure 17 shows an example of a 2x supersampling where $L1 = (L1_1 + L1_2 + L1_3 + L1_4)/4$. The modes currently supported are 1x, 2x and 4x. *N*x means that the image is rendered at *N* times the resolution and each pixel displayed is an average of the N^2 corresponding pixels rendered. Higher supersampling results in smoother image and longer computation.

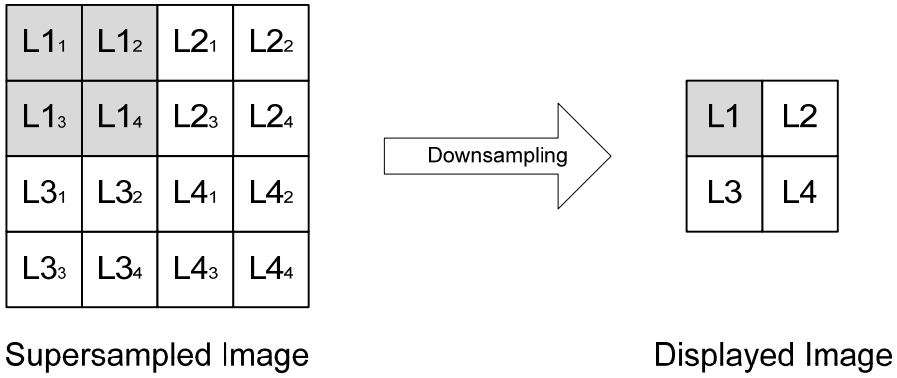

Figure 17 – Supersampling and downsampling operations

4.6 Compute Angular Irradiance Distribution

The *Image* is rendered into the framebuffer by the *Generator* during the execution of the *draw* method. The *readFrameBuffer* method extracts an angular distribution of irradiance from the framebuffer into an image (*DataTypes::Matrix*).

This method converts a pixel color into a pixel radiance:

$$L_{pixel} = Color \frac{AR_{Scaling}}{2^{N_{Bits}}} \ , \tag{13}$$

where:

- L_{Pixel} is the apparent radiance of a pixel ($W/m^2/sr$);
- Color is the color of a pixel (between 0 and 2^{N}_{Bits}-1);
- $AR_{Scaling}$ is the scaling factor for apparent radiance to color conversion ($W/m^2/sr$); and
- N_{Bits} is the number of bits per pixel (fixed to 2 since 16-bit is used).

As presented earlier, the scaling factor is stored in the *MaxRadiance* attribute. Equation (13) introduces the number of bits per pixel since the pixels of the *Image* are not limited to 1 as opposed to the polygon color. The pixel irradiance is then calculated:

$$E_{pixel} = L_{pixel} \frac{FOV_H FOV_V}{N_H N_V} \ , \tag{14}$$

where:

- E_{Pixel} is the apparent irradiance of a pixel (W/m^2);
- L_{Pixel} is the apparent radiance of a pixel ($W/m^2/sr$);
- FOV_H and FOV_V are the horizontal and vertical total FOV of the image (rad); and

- N_H and N_V are the horizontal and vertical dimensions of the image (pixels).

5 Modifications to OpenSceneGraph

Some modifications to the OSG external library are required to allow proper behavior mostly for scene generation. The current version of OSG is 2.2.

5.1 OpenFlight

IR Color Code and *IR Material Code* fields of the OpenFlight file format are used as indices to the IR Property Database for the scene generation. However, those fields are not supported in OSG nodes (read but discarded). The *readRecord* method of the *Face* class (file "GeometryRecords.cpp" of the *osgdb::OpenFlight* package) has been modified to read those fields and store them as user data. The following code has been added.

```
// Set IRColor and IRMaterial properties (not available in OSG)
osg::IntArray * pArray = new osg::IntArray(2);
(*pArray)[0] = IRColor;
(*pArray)[1] = IRMaterial;

_geometry.get()->setUserData(pArray);
```

5.2 Rendering

A warning message was displayed in the application console when running a KARMA simulation each time that a draw was performed (scene generation). This message has been disabled since the draw operation seems to work correctly. The *drawInner* method of the *RenderStage* class (file "RenderStage.cpp" of the *osgUtil* package) has been modified to disable the warning message.

```
if(state.getCheckForGLErrors()!=osg::State::NEVER_CHECK_GL_ERRORS)
 {
   GLenum errorNo = glGetError();
   if (errorNo!=GL_NO_ERROR)
    {
      const char* error = (char*)gluErrorString(errorNo);
      // if (error)  osg::notify(osg::NOTICE)<<"Warning: detected OpenGL error '"<<error<<"'
      // after    RenderBin::draw(,)"<<std::endl;
      // else        osg::notify(osg::NOTICE)<<"Warning: detected OpenGL errorNo=
      // 0x"<<std::hex<<errorNo<<" after RenderBin::draw(,)"<<std::endl;

      if (fbo_ext)
            osg::notify(osg::NOTICE)<<"RenderStage::drawInner(,) FBO status=0x"
            <<std::hex<<fbo_ext>glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT)<<std::endl;
    }
}
```

6 Future Work

The first implementation phase of the IRSG has demonstrated the benefits of modelling IR signatures using 3D models. However, there are aspects that can be enhanced or added. There is also some part of the code that could be reviewed before continuing the development. The envisioned improvements are listed in Table 1.

Table 1 – List of possible new or enhanced features in the IRSG

Feature
Enhance the antialiasing techniques.
Investigate *GL_FOG* settings on the scene generation.
Investigate incompatibilities of some 3D model shapes (e.g. landing gear cover issue) in order to avoid crashes during scene generation (those geometries have been removed at this time and incompatibilities are misunderstood).
Optimize the computations and pre-computations to increase the execution speed (includes a deep analysis of the performances of the scene generation steps using typical settings for dimensions, supersampling, IR Database Properties and details of the 3D models).
Implement the thrust variable in the temperature equation.
Revisit the Near/Far usage to maximize the rendering resolution as long as it is within detection range (instead of having the Near/Far, set to the minimum/maximum detection range).
Investigate the 16-bit implementation using OpenGL (would avoid DLL conflicts with OSMesa DLLs and take advantage of hardware acceleration).
Implement a dynamic selection of the precision of the scene generation process (8-bit or 16-bit) instead of replacing DLLs manually.
Investigate the optimization of the culling process.
Revisit the scene generation programming interface implemented for SMAT to generate an *Image* using a sensor point of view (i.e. using *AbstractSensor* and transmittance model within *Environment* composition).
Implement an interface to the IRSG in order to decouple KARMA from OSG and other dependencies.
Compute the atmospheric transmittance for the main group nodes of the *BaseEntity* instead of the *BaseEntity* itself.
Investigate the externalization of the IRSG: separate in an application that could be used outside KARMA (e.g. MATLAB), which would eliminate the 8-bit/16-bit constraint (3D viewer versus IRSG). Otherwise, implement an interface to the IRSG in order to decouple KARMA from OSG and other dependencies.
Migrate to OpenSceneGraph 2.4 released on 25th April 2008 which features writing OpenFlight format as opposed to the current version (OpenSceneGraph 2.2).

References

[1] Harrison, N. 2005. KARMA: Materializing the Soul of Technologies into Models. DRDC Valcartier Fact Sheet OS-333-A.

[2] Harrison, N., Gilbert, B., Jeffrey, A., Lestage, R., Lauzon, M., and Morin, A. 2005. KARMA: Materializing the Soul of Technologies into Models. Proceedings of the I/ITSEC 2005 Conference, Orlando, Florida, USA.

[3] Harrison, N., Gilbert, B., Jeffrey, A., Lauzon, M. and Lestage, R. 2004. Adaptive and Modular M&S Configuration for Increased Reusability. Proceedings of the I/ITSEC 2004 Conference, Orlando, Florida, USA.

[4] Harrison, N., Gilbert, B., Lauzon, M., Jeffrey, A., Lalancette, C., Lestage, R. and Morin, A. 2002. A M&S Process to Achieve Reusability and Interoperability. Proceedings of the NATO M&S Conference 2002, RTO-MP-094-11, Paris, France.

[5] Richard, J. 2008. SMAT User's Guide. DRDC Valcartier CR 2008-260, LTI, Quebec City, Quebec.

[6] Richard, J. 2008. SMAT Developer's Guide. DRDC Valcartier CR 2008-259, LTI, Quebec City, Quebec.

[7] osg – Trac (online). http://www.openscenegraph.org/projects/osg (access date: April 23, 2008).

[8] OpenGL – The Industry Standard for High Performance Graphics. http://www.opengl.org/ (access date: April 23, 2008).

[9] Mesa Home Page. http://www.mesa3d.org/ (access date: April 23, 2008).

[10] OpenFlight standard. http://www.multigen.com/products/standards/openflight/index.shtml (access date: May 15, 2008).

[11] Lepage, J. F. 2008. Infrared scene generation for countermeasures simulations, Implementation in the KARMA framework, phase 1. DRDC Valcartier (report being published).

This page intentionally left blank.

Abbreviations and Acronyms

API	Application Programming Interface
CSV	Comma Separated Value
DLL	Dynamic Link Library
GUI	Graphical User Interface
IR	Infrared
IRCM	Infrared Countermeasure
IRSG	Infrared Scene Generation
LUT	Lookup Table
OpenGL	Open Graphic Library
OSG	OpenSceneGraph
OSMesa	Off-Screen Mesa
R&D	Research & Development
RGBA	Red Green Blue Alpha
SMAT	Signature Modeling and Analysis Tool
STL	Standard Template Library
UML	Unified Modeling Language
XML	Extensible Markup Language

This page intentionally left blank.

Appendix 1

IR Property Database Description

Data Type	Section Offset	Length	Description
String	0	8	ASCII ID including NULL terminator
Integer	8	4	Version
Integer	0	4	Temperature section ID (100)
Integer	4	4	Number of temperature entries
Integer	8	4	Temperature index
Double	12	8	Temperature (K)
Double	20	8	Reserved
Double	28	8	Reserved
Integer	36	4	Length of temperature name
String	40	N	ASCII name including NULL terminator
Double	40 + N	8	Coefficient A 0 = absolute temperature 1 = relative to ambient temperature
Double	40 + N + 8	8	Offset to the ambient temperature (K)
Double	40 + N + 16	8	Aerodynamic recovery factor
Integer	40 + N + 24	4	Number of entries (i) in temperature LUT
Double	68 + N + (i * 16)	8	Time n (s)
Double	76 + N + (i * 16)	8	Temperature n (K)

Data Type	Section Offset	Length	Description
Integer	0	4	Material section ID (101)
Integer	4	4	Number of material entries
Integer	8	4	Material index
Integer	12	4	Length of material name
String	16	N	ASCII name including NULL terminator
Integer	16 + N	4	Number of points (j) in the emissivity curve 0 = NULL 1 = constant emissivity N = emissivity fonction of wavelength
Double	20 + N + (n * 16)	8	Wavelength n (µm)
Double	28 + N + (n * 16)	8	Emissivity n
Integer	20 + N + (j * 16)	4	Number of points (k) in the transmissivity curve 0 = NULL 1 = constant transmissivity N = transmissivity fonction of wavelength
Double	24 + N + ((j + n) * 16)	8	Wavelength n (µm)
Double	32 + N + ((j + n) * 16)	8	Transmissivity n
Integer	24 + N + ((j + k) * 16)	4	Number of points (l) in the reflectivity curve 0 = NULL 1 = constant reflectivity N = reflectivity fonction of wavelength
Double	28 + N + ((j + k + n) * 16)	8	Wavelength n (µm)
Double	36 + N + ((j + k + n) * 16)	8	Reflectivity n
Integer	28 + N + ((j + k + l) * 16)	4	Scale section ID (102)
Integer	32 + N + ((j + k + l) * 16)	4	Use scaling flag 0 = FALSE 1 = TRUE
Integer	36 + N + ((j + k + l) * 16)	4	Number of scaling values (m)
Double	40 + N + ((j + k + l) * 16) + (m * 32)	8	Time n (s)
Double	48 + N + ((j + k + l) * 16) + (m * 32)	8	Scaling along X axis
Double	56 + N + ((j + k + l) * 16) + (m * 32)	8	Scaling along Y axis
Double	64 + N + ((j + k + l) * 16) + (m * 32)	8	Scaling along Z axis

Distribution list

Document No.: DRDC Valcartier CR 2008-258

LIST PART 1: Internal Distribution by Centre

2 Document Library

1 J.-F. Lepage
1 N. Harrison
1 P. Brière
1 B. Gilbert
1 C. Belhumeur
1 C. Lemelin
1 M. Lambert
1 F. Dinel
1 M. Lauzon

11 TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

1 Director Research and Development Knowledge and Information Management (DRDKIM)

1 Library and Archives Canada

1 Director Science and Technology Air (DSTA)
305 Rideau Street
Ottawa Ontario K1A 0K2

1 Director Science and Technology Air (DSTA) 6
305 Rideau Street
Ottawa Ontario K1A 0K2

1 Directorate of Technical Airworthiness and Engineering Support (DTAES) 8
National Defense Headquarters
400 Cumberland
Ottawa Ontario K1A 0K2

5 TOTAL LIST PART 2

16 TOTAL COPIES REQUIRED

DOCUMENT CONTROL DATA

1. ORIGINATOR (name and address) LTI inc. 2700, De Carthagène, Québec, QC G2B 5M4	2. SECURITY CLASSIFICATION (Including special warning terms if applicable) UNCLASSIFIED

3. TITLE (Its classification should be indicated by the appropriate abbreviation (S, C, R or U)
Infrared Scene Generation (IRSG) Developer's Guide (U)

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)
Rouleau, Eric

5. DATE OF PUBLICATION (month and year)	6a. NO. OF PAGES	6b .NO. OF REFERENCES
September 2008	42	11

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. Give the inclusive dates when a specific reporting period is covered.)
Contract report

8. SPONSORING ACTIVITY (name and address)
Defence R&D Canada - Valcartier
a/s Jean-Francois Lepage and Nathalie Harrison
2459, boul. Pie-XI Nord, Québec, QC G3J 1X5

9a. PROJECT OR GRANT NO. (Please specify whether project or grant)	9b. CONTRACT NO.
Project 13et01	W7701-052709/001/QCL

10a. ORIGINATOR'S DOCUMENT NUMBER	10b. OTHER DOCUMENT NOS
DRDC Valcartier CR 2008-258	LTI2008DES-3 N/A

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

- ☒ Unlimited distribution
- ☐ Restricted to contractors in approved countries (specify)
- ☐ Restricted to Canadian contractors (with need-to-know)
- ☐ Restricted to Government (with need-to-know)
- ☐ Restricted to Defense departments
- ☐ Others

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)
Unlimited announcement

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

This contract report presents the first implementation phase of 3D infrared signature modelling compatible with the KARMA simulation environment developed on behalf of Defence Research and Development Canada – Valcartier in the framework of Public Works and Government Services Canada contract number W7701-052709/001/QCL "Development and exercise simulation". The main objective of this first implementation is to demonstrate the feasibility and the benefits of 3D infrared signature modelling as opposed to punctual signatures. Free and open source software as well as commercial off-the-shelf tools have been considered, allowing the reduction of the implementation time and increasing reliability.

This report is presented as a developer's guide of the infrared scene generation module and describes the implementation of the scene generation process tailored to KARMA. The first implementation phase of 3D infrared signatures improves signature modelling in KARMA and demonstrates the applicability of a 3D viewer for infrared scene generation and the higher level of fidelity that is reached for infrared guided weapon engagement simulations. However, the current implementation could be improved to offer pre-computations and support the former punctual signatures.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Infrared scene generation

Infrared signature

Modeling and simulation (M&S)

Defence R&D Canada

Canada's Leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca