Volume 7

OGC CDB Data Model Guidance (Best Practice)

Open Geospatial Consortium

Submission Date: 2020-xx-xx

Approval Date: 2020-xx-xx

Publication Date: 2020-xx-xx

External identifier of this OGC® document: http://www.opengis.net/doc/BP/CDB-model-guidance/1.2

Additional Formats (informative):

Internal reference number of this OGC® document: 16-010r5

Version: 1.2

Category: OGC® Best Practice

Editor: Carl Reed

Volume 7: OGC CDB Data Model Guidance (Best Practice)

Warning

This document defines an OGC Best Practices on a particular technology or approach related to an OGC standard. This document is **not** an OGC Standard and may not be referred to as an OGC Standard. It is subject to change without notice. However, this document is an **official** position of the OGC membership on this particular technology topic.

Document type:	OGC® Best Practice
Document subtype:	
Document stage:	Approved
Document language:	English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Table of Contents

i. Abstract

This CDB Volume provides Guidelines, Clarifications, Rationales, Primers, and additional information for the definition and use of various models that can be stored in a CDB compliant data store.

Please note that the term “lineal” has been replaced with the term “line” or “linear” throughout this document

Please note that the term “areal” has been replaced with the term “polygon” throughout this document.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, ogcdoc, cdb, models, guidance, simulation

iii. Preface

This document contains material from Annex A in the original CDB 3.2 specification submission to the OGC. These sections provide guidance for model builders to build and maintain model content in a CDB data store.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

iv. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- CAE Inc.
- Carl Reed, OGC Individual Member
- Envitia, Ltd
- Glen Johnson, OGC Individual Member
- KaDSci, LLC
- Laval University
- Open Site Plan
- University of Calgary
- UK Met Office

The OGC CDB standard is based on and derived from an industry developed and maintained specification, which has been approved and published as OGC Document 15-003: OGC Common Data Base Volume 1 Main Body. An extensive listing of contributors to the legacy industry-led CDB specification is at Chapter 11, pp 475-476 in that OGC Best Practices Document (https://portal.opengeospatial.org/files/?artifact_id=61935)

v. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

Name	Affiliation
Carl Reed	Carl Reed & Associates
David Graham	CAE Inc.

Chapter 1. Scope

The CDB Standard defines a standardized model and structure for a single, versionable, simulation-rich, virtual representation of the earth. A CDB structured data store provides for a synthetic environment repository that is plug-and-play interoperable between data store authoring workstations. Moreover, a CDB structured data store can be used as a common online (or runtime) repository from which various simulator client-devices can simultaneously retrieve and modify, in real-time, relevant information to perform their respective runtime simulation tasks. In this case, a CDB data store is plug-and-play interoperable between CDB-compliant simulators. A CDB data store can be readily used by existing simulation client-devices (legacy Image Generators, Radar simulator, Computer Generated Forces, etc.) through a data publishing process that is performed on-demand in real-time.

Please note that this document was Annex A, Volume Part 2 in the pre-OGC 3.2 version.

Chapter 2. Conformance

This is an informative document. There are no normative clauses

Chapter 3. References

For ease of editing and review, the standard has been separated into 16 Volumes, one being a schema repository.

- Volume 0: OGC CDB Companion Primer for the CDB standard (Best Practice).
- Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure. The main body (core) of the CDB standard (Normative).
- Volume 2: OGC CDB Core Model and Physical Structure Annexes (Best Practice).
- Volume 3: OGC CDB Terms and Definitions (Normative).
- Volume 4: OGC CDB Rules for Encoding CDB Vector Data using Shapefiles (Best Practice).
- Volume 5: OGC CDB Radar Cross Section (RCS) Models (Best Practice).
- Volume 6: OGC CDB Rules for Encoding CDB Models using OpenFlight (Best Practice).
- Volume 7: OGC CDB Data Model Guidance (Best Practice).
- Volume 8: OGC CDB Spatial Reference System Guidance (Best Practice).
- Volume 9: OGC CDB Schema Package: http://schemas.opengis.net/cdb/ provides the normative schemas for key features types required in the synthetic modelling environment. Essentially, these schemas are designed to enable semantic interoperability within the simulation context (Normative).
- Volume 10: OGC CDB Implementation Guidance (Best Practice).
- Volume 11: OGC CDB Core Standard Conceptual Model (Normative).
- Volume 12: OGC CDB Navaids Attribution and Navaids Attribution Enumeration Values (Best Practice).
- Volume 13: OGC CDB Rules for Encoding CDB Vector Data using GeoPackage (Normative, Optional Extension).
- Volume 14: OGC CDB Guidance on Conversion of CDB Shapefiles into CDB GeoPackages (Best Practice).
- Volume 15: OGC CDB Optional Multi-Spectral Imagery Extension (Normative).

Chapter 4. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

For the purposes of this document, the following additional terms and definitions apply:

- Volume 3: OGC CDB Terms and Definitions (normative).

Chapter 5. Conventions

This section provides details and examples for any conventions used in the document. Examples of conventions are symbols, abbreviations, use of XML schema, or special notes regarding how to read the document.

Chapter 6. Data Model Guidance

Please note that whenever there is reference to Volume 1: Core, the full reference is Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure.

6.1. Guideline: Creating a 3D Model for a Powerline Pylon

Formerly Annex A.1 Volume 2 of the CDB Best Practice

The goal of this guidance is to model a typical high voltage electrical pylon resembling the one in the following figure. This guidance is based originally on CDB version 3.1 prior to submission to the OGC, but is applicable to version 3.0 as well [1].

Figure A-1: Typical Electrical Pylon

6.1.1. Pylon Model Orientation

The front (and back) of a powerline pylon is aligned with the general direction of the attached wires as illustrated below.

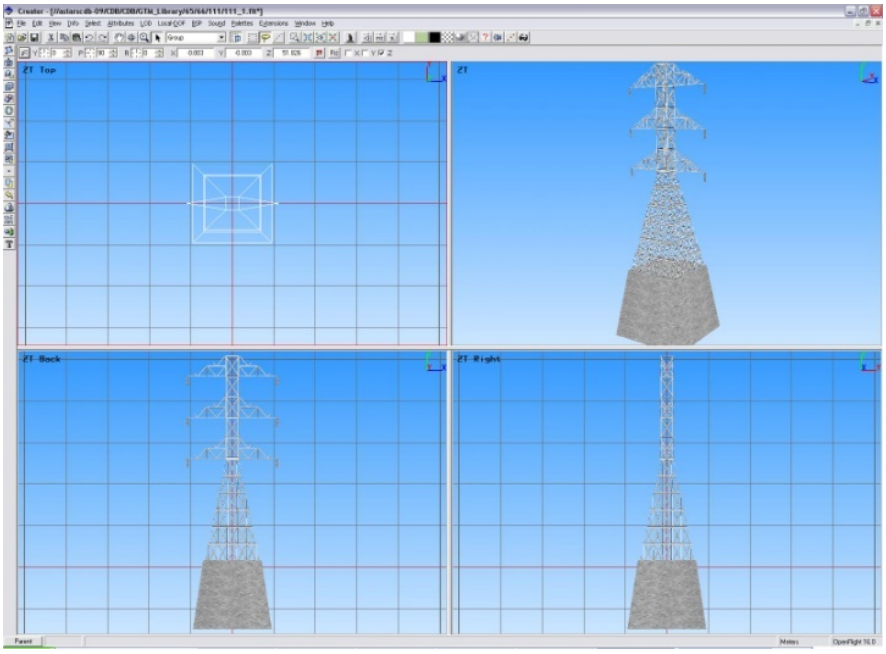


Figure A-2: Pylon Orientation

The above snapshot is similar to the one found in Figure 6–10 of CDB Best Practice - Volume 6: OGC CDB Rules for Encoding Data using OpenFlight.

6.1.2. OpenFlight Graph

The OpenFlight graph of the above pylon exposes the 3 cross-arms, each with 2 insulators where wires are attached. Here are the names of the components that are used to model this power pylon:

- Pylon (global zone)
- Arm (horizontal cross-arm at the top of the structure)
- Insulator (ceramic insulator string attached at the end of each arm)

These 3 names are used to create CDB Zones as explained in section 6.5 of the CDB Best Practice: Volume 6 OGC CDB Rules for Encoding Data using OpenFlight. Here is the first level of the resulting graph.

The rounded rectangle named Object is an OpenFlight object node containing the geometry of the concrete base and lattice steel structure of the pylon, but excluding the geometry of the cross-arms. Arm[1] is the lowest cross-arm; Arm[2] is the middle one; Arm[3] is the top one. Each arm is then made of a steel structure and 2 insulators.

Again, Object represents the steel structure of the cross-arm without the insulators. When looking at one of the cross-arm of the power pylon from the back, Insulator[1] is to the left while Insulator[2] is to the right. Finally, each insulator has an attach point to indicate where to connect an eventual wire.

The node Object contains the geometry of the insulator.

As explained in section 6.8 of the CDB Best Practice Volume 6: OGC CDB Rules for Encoding Data using OpenFlight, the resulting list of paths is as follow:

- \Pylon
- \Pylon\Arm[1]
- \Pylon\Arm[1]\Insulator[1]
- \Pylon\Arm[1]\Insulator[1]\Attach_Point
- \Pylon\Arm[1]\Insulator[2]
- \Pylon\Arm[1]\Insulator[2]\Attach_Point
- \Pylon\Arm[2]
- \Pylon\Arm[2]\Insulator[1]
- \Pylon\Arm[2]\Insulator[1]\Attach_Point
- \Pylon\Arm[2]\Insulator[2]
- \Pylon\Arm[2]\Insulator[2]\Attach_Point
- \Pylon\Arm[3]
- \Pylon\Arm[3]\Insulator[1]
- \Pylon\Arm[3]\Insulator[1]\Attach_Point
- \Pylon\Arm[3]\Insulator[2]
- \Pylon\Arm[3]\Insulator[2]\Attach_Point

Note the presence of a total of 6 attach points (1 attach point per insulator × 2 insulators per cross-arm × 3 cross-arms per pylon = 6 attach points per pylon). Even though all attach points have the same name, there is a unique path to reach each one. For this reason, there is no ambiguity identifying and locating each point.

6.1.3. Attach Point Orientation

When creating the attach point of the insulator, pay attention to its orientation. Since the cable attaches underneath the insulator, the Z-axis of the local coordinate system (LCS) must be pointing down. To achieve a proper positioning of the attach point, the modeler usually inserts two transformations in the node, one translation and one rotation. The translation positions the point underneath the insulator while the rotation changes the orientation of the Z-axis. Make sure to leave the Y-axis in the direction of the wire as in the figure below.

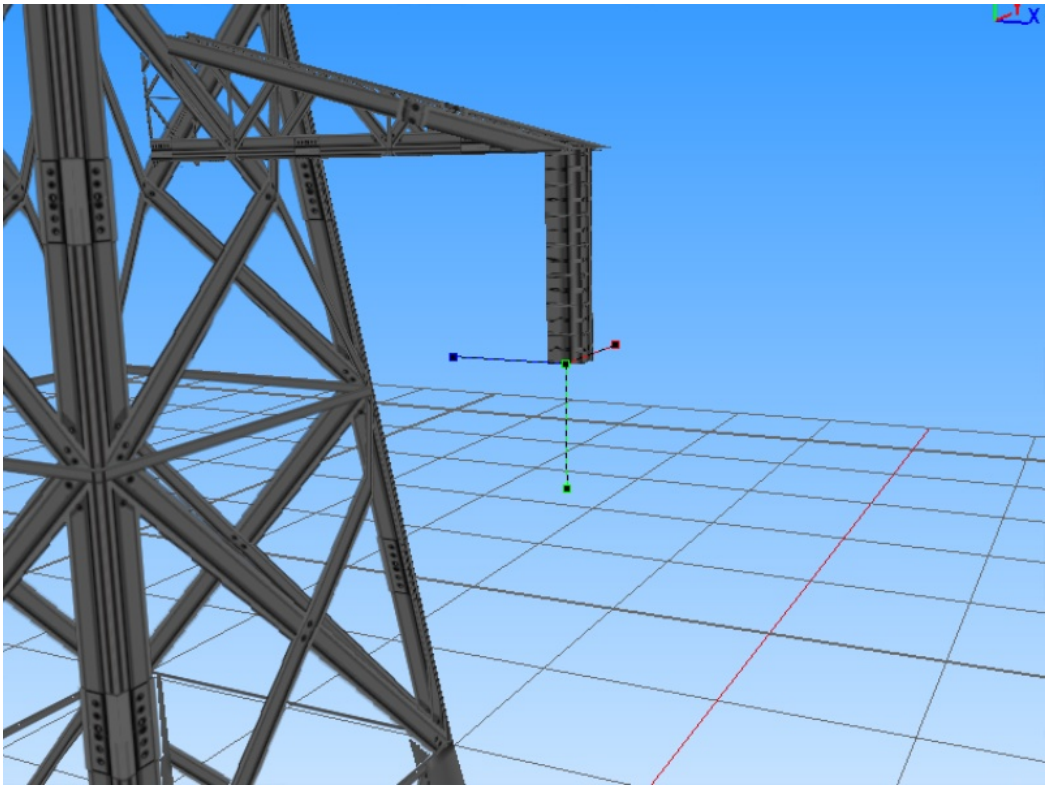


Figure A-3: Attach Point Orientation

In this figure, the position and orientation of the attach point is identified by the blue-red-green axis system beneath the insulator. The Y-axis is in red and points in the same direction as the model's Y-axis, which is toward the front of the model. The Z-axis is in green and points down indicating that wires attach under the insulator.

6.2. Guideline: Generating Wires between Pylons of a Powerline

Formerly Annex A.2 in the CDB Best Practice, Volume 2.

This guideline is intended for both modelers and developers responsible for the creation of:

- CDB content such as 3D models representing pylons
- Tools used to generate the Powerline Network datasets
- Client-devices that use the Powerline Network datasets to generate pylons and wires along the transmission line.

6.2.1. Powerline Network Attributes

The table below is the collection of class and instance-level attributes from tables 5-46 and 5-47 of Volume 1: OGC CDB standard [2].

Table A-1: Powerline Attributes

Required Attributes	Optional Attributes
CMIX	AHGT
CNAM	AO1
DIR	BBH
EJID	BBL
FACC	BBW
FSC	BSR
JID	HGT
LENL	MODL
RTAI	MODT
SJID	SCALn
WGP	

The occurrence of some of the optional attributes depends on the occurrence of other optional attributes. In particular, when MODL is present, other attributes become required while others remain optional. The table below provides the relation between MODL and other attributes.

Table A-2: MODL-related Attributes

Required	Optional
BSR	AO1
HGT	BBH, BBL, BBW
MODT	SCALn

As a result of the above tables, a CDB-compliant Powerline Network dataset requires 11 mandatory attributes (listed in the first column of Table A-1). Optionally, when a 3D model representing a pylon is provided, 4 additional attributes are required (MODL obviously, plus BSR, HGT, and MODT) and 5 others remain optional (AO1, BBH, BBL, BBW, and SCALn).

6.2.2. Generation of HGT

The HGT attribute represents a special case because table 5-47 in Volume 1 suggests that the attribute is optional while, in fact, it should always be present. If you carefully read its description in paragraph 5.3.1.2.3.17, you realize that HGT is required in both the line and figure point features of the Powerline Network.

For line features, HGT represents the average height above ground of the powerline when no MODL is specified, as suggested by the discussion about HGT in section 5.3.1.17 of the Volume 1: Core. In the figure point features, HGT represents the height above ground of the pylon, whether or not a MODL is provided. In either file, when MODL is supplied, HGT represents the height of the 3D model above the ground.

You should read guideline (6.3 – old Annex A.3) for a complete discussion about HGT

6.2.3. Pylon Orientation

If the orientation of the pylon is specified by AO1, then use the value as-is. If the orientation is not specified, the client device *must* compute its value using the orientation of the segments of the line that are adjacent to the pylon. In the case of the first and last segments, the orientation of the segment is also the orientation of the pylon. For the other segments, the orientation of the pylon is the average of the orientation of the two adjacent segments.

6.2.4. Number of Wires

When no MODL is provided at all – meaning no MODL for the line and none for the figure points – and because there is no attribute specifying the number of wires along the transmission line, the client device *must* assume a generic powerline with two wires separated by a width of WGP meters connecting generic posts (simple pylons) of HGT meters high.

When a common MODL is specified for the whole line and no figure points are provided, it is possible to determine the number of wires by counting the number of attach points in the 3D model. Refer to guideline 6.1.2 (old A.1.2) for details on how to detect attach points.

If specific MODLs are defined through figure points, the number of attach points in each 3D model of the collection of all MODLs referenced by the powerline network *must* be identical. For instance, if the line refers to a generic pylon supporting 4 wires, then all specific pylons referenced as figure points *must* also support 4 wires. Furthermore, the general configuration of all pylons *must* be identical. If the general pylon supports 6 wires configured as a matrix of 2 wires horizontally by 3 wires vertically, then all specific pylons *must* also share the same configuration.

6.2.5. How to Connect Wires to Attach Points

If the client device has a single generic pylon along the line, then there is no problem connecting wires and attach points. That is when multiple pylons are used along the same line that problems arise. The client has to match attach points from one type of pylon to attach points on another pylon that may be of a different type. The algorithm to determine how to connect pylons of different types is left to the client device. A future version of CDB Standard will provide a robust and deterministic approach on how to connect the wires.

6.3. Guideline: How to Interpret the AHGT, HGT, BSR, BBH, and Z Attributes

Formerly Annex A.3 in the OGC CDB Best Practice, Volume 2.

The goal of this guideline is to promote a correct use of five CDB attributes: AHGT, HGT, BSR, BBH, and Z. The article is aimed to both developers and users of content creation tools as well as developers of client applications.

A picture being worth a thousand words, the following diagram should help understand the relations between the AHGT, HGT, BSR, BBH, and Z attributes.

Here is a reminder of what these attributes are. The complete definitions can be found in Section 5.3.1.3, CDB Attributes in the CDB Standard Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure.

- AHGT (Absolute Height) is a flag to interpret correctly the value of the Z coordinate of a feature. When false, the value of Z is relative to the ground (Zr); when true, Z is the absolute altitude (Za).
- AHGT is not related with HGT even though their names are similar.
- HGT (Height Above Surface Level) is the distance from the top of the model to the ground.
- BBH (Bounding Box Height) is the distance from the top of the model to its XY plane.
- BSR (Bounding Sphere Radius) encompasses the portion of the model that is above its XY plane.
- Z is the altitude of a feature, either absolute or relative to the ground.

In the diagram above, a model (MODL) is positioned above the ground. This is indicated by the fact that the model's XY plane does not lie directly on the ground. The distance above the ground is represented by Zr. The diagram clearly shows the relation between HGT, BBH, and Zr.

$$HGT = BBH + Zr$$

When the value of Zr is not readily available from the instance of the feature itself (because AHGT is true), it can be computed using the ground height (Gh).

$$Zr = Za - Gh$$

The BBH attribute is optional and defaults to twice the value of BSR, which is mandatory for a MODL model.

$$default\ BBH = 2 \times BSR$$

$$default\ BBH \geq real\ BBH$$

6.3.1. Typical Use-case

Typically, a model is positioned relative to the ground without any offset. As a result, AHGT is false, and Zr is set to zero. Hence…

$$HGT = BBH$$

6.3.2. Light Points

In the case of airport and environmental light points, no model of a light fixture is provided (the MODL attribute is not allowed). Hence…

$$BSR = 0 \rightarrow BBH = 0$$

Currently, the light point datasets do not allow the HGT attribute, the client application may have to compute its value using the equation given previously…

$$HGT = BBH + Zr$$

where BBH is null.

$$HGT = Zr$$

And if the light point is positioned at an absolute height (AHGT is true), then…

$$HGT = Za - Gh$$

6.3.3. Recommendation

Refrain from using AHGT. There are several advantages to leave this flag set to false. First, it facilitates the creation of CDB datasets that are independent of each other. When the Z coordinate (altitude) of a feature is relative to the ground, the elevation dataset can be updated without the need to re-compute and update all features that have an absolute altitude.

Second, when a feature has an absolute altitude, it is possible that it will end up being *displayed* below the ground by a given client. How is this possible? Isn't it an error in the data store itself? No, this is not an error. It is perfectly possible to create content that is valid and – still – produce an incorrect result at the client level. Consider a feature that is positioned with an absolute height in a valley between two mountains of a high resolution terrain profile. At coarse LOD of terrain elevation, the valley and the mountains may (and will) be flattened producing a terrain skin that may no longer pass underneath the feature. Now imagine a client that uses that coarse LOD of elevation to create a terrain skin and then draw the feature at its absolute altitude, which happen to be underneath the terrain skin. The feature will not be visible or will be partially occluded by the terrain.

These reasons explain why the use of the AHGT flag should be avoided whenever possible.

6.3.4. When should AHGT be used?

Limit the use of AHGT to data whose source is inherently absolute. Such source data include geodetic marks or survey marks that provide a known position in terms of latitude, longitude, and altitude. Good examples of such markers are boundary markers between countries.h

6.4. Guideline: How to Model a Wind Turbine

Formerly Annex A.4 in the OGC CDB Best Practice, Volume 2.

This text proposes a way to create a 3D model representing an articulated wind turbine. The articulations of interest are the yaw control to orient the turbine in the direction of the wind, the

roll control to allow rotation of the rotor, and, optionally, the pitch control to change the orientation of the blades, if needed.

Beside is a typical Horizontal Axis Wind Turbine. The components of interest are the following:

- Turbine
- Rotor
- Blade

Looking at appendix F – CDB Model Components – we note that Turbine is not listed and, consequently, will be proposed as an extension to future version of the CDB standard.

The CDB metadata folder provides the proper code for a Wind Turbine, AD010-005 [3]. The code indicates the presence of a man-made point feature.

A = Culture

D = Power Generator

010 = Power Plant

005 = Wind

The hierarchy graph of the OpenFlight model could look like the one on the right. If individual control of the pitch of each blade is required, the Blades object (the lower right node) could be replaced with three (3) sub-trees each containing a Blade zone, a DOF node, and an object node.

With the proposed layout, a client device will detect the presence of a wind turbine through its feature attribute code (aka feature code), and recognize and control two articulations, the Turbine Yaw angle, and Rotor Roll angle.

A last note: to comply with the prescribed orientation of the CDB coordinate system as defined in section 6.3 Volume 6: OGC CDB Rules for Encoding Data using OpenFlight, the rotor *must* represent the front of the wind turbine (and not its right side).

Reference: http://en.wikipedia.org/wiki/Wind_turbine

6.5. Guideline: Handling of Model Interiors

Formerly Annex A.5 in the OGC CDB Best Practice, Volume 2.

CDB introduces the concept of the interior of a 3D model. The concept is developed in section 6.18, Model Interior, of the CDB Best Practice Volume 6: OGC CDB Rules for Encoding Data using OpenFlight. The following text serves as a complement to the standard to understand how the concept has been developed and how model interior is intended to be used.

6.5.1. Relationship between Model Shell and Model Interior

The ModelInteriorGeometry dataset is a subordinate dataset of the '*regular*' ModelGeometry dataset. It depends directly on it. This is best illustrated by an example.

LOD	**ModelGeometry (Shell)**	**ModelInteriorGeometry (Interior)**
…	-	-
0	-	-
1	-	-
2	Coarsest Shell	-
3	-	-
4	-	-
5	-	-
6	Medium Shell	Medium Interior
7	-	-
8	Fine Shell	Fine Interior
9	-	-
10	Finest Shell	Finest Interior
11	-	-
12	-	-
13	-	-
14	-	-
15	-	-
…	-	-

In the above table, the **Shell** column represents what is called the '*regular*' ModelGeometry dataset. In this example, the model appears at LOD 2, a better version exists at LOD 6, an even better at LOD 8, and finally, the most detailed shell is at LOD 10. The **Interior** column shows 3 different LODs of interiors. There cannot be more Interior LODs than Shell LODs. Also, once an interior is provided (here at LOD 6), it *must* be provided for all subsequent (finer) LODs of the shell (LOD 8 and 10). Which means… interior at LOD 8 and 10 *must* exist.

6.5.2. Detecting Presence of a Model Interior

It is expected that a client will first request the shell of the model, then discover that the model has an interior because of the presence of a CDB Zone whose name is Interior (see 6.18.2 Volume 6: OGC CDB Rules for Encoding Data using OpenFlight, Pseudo-Interior), and then decide if the pseudo interior is sufficient for the application or if the real interior is necessary.

6.5.3. Access of a Model Interior

Client applications that are interested in 3D models will typically perform the following sequence of actions:

1. Load the GS Features of a tile
2. Load the GS and GT Models referenced by the GS Features
3. For each model, traverse its graph and detect the presence of an optional Interior (Zone name = Interior)
4. Decide to load the corresponding Interior (or not)

Interior datasets exists for both geospecific and geotypical models. Hence, all features can be represented by a 3D model and all 3D models can have a separately modeled interior. Note the symmetry between the file names of shell and interior datasets. For geospecific models encoded as OpenFlight, the names of geometry files are…

- GeoCell_D30**0**_S001_T001_Lxx_Ux_Rx_FeatureCode_FSC_MODL.flt
- GeoCell_D30**5**_S001_T001_Lxx_Ux_Rx_FeatureCode_FSC_MODL.flt

For geotypical models encoded as OpenFlight, the file names become…

- D51**0**_S001_T001_Lxx_FeatureCode_FSC_MODL.flt
- D51**5**_S001_T001_Lxx_FeatureCode_FSC_MODL.flt

Note that in both cases, the only difference between the name of the shell and the name of the corresponding interior is the dataset code; and in both cases, a value of 5 is added to the '*regular*' ModelGeometry dataset code.

6.5.4. UHRB vs CDB Object Models

To help understand how CDB Model Interior maps to UHRB concepts, three (3) diagrams are provided below. The first two diagrams illustrate the UHRB Object Model [4] while the third diagram presents the corresponding CDB Object Model.

The first diagram is the UHRB Class Diagram presented in Figure A-4 below. The class diagram presents twelve classes of which eight are concrete classes that can be used to represent tangible objects. The UHRB_EDM_COMPLEX_FEATURE class implements an extension mechanism that is not required in the context of the CDB Specification. The remaining seven UHRB classes will be mapped to CDB zones.

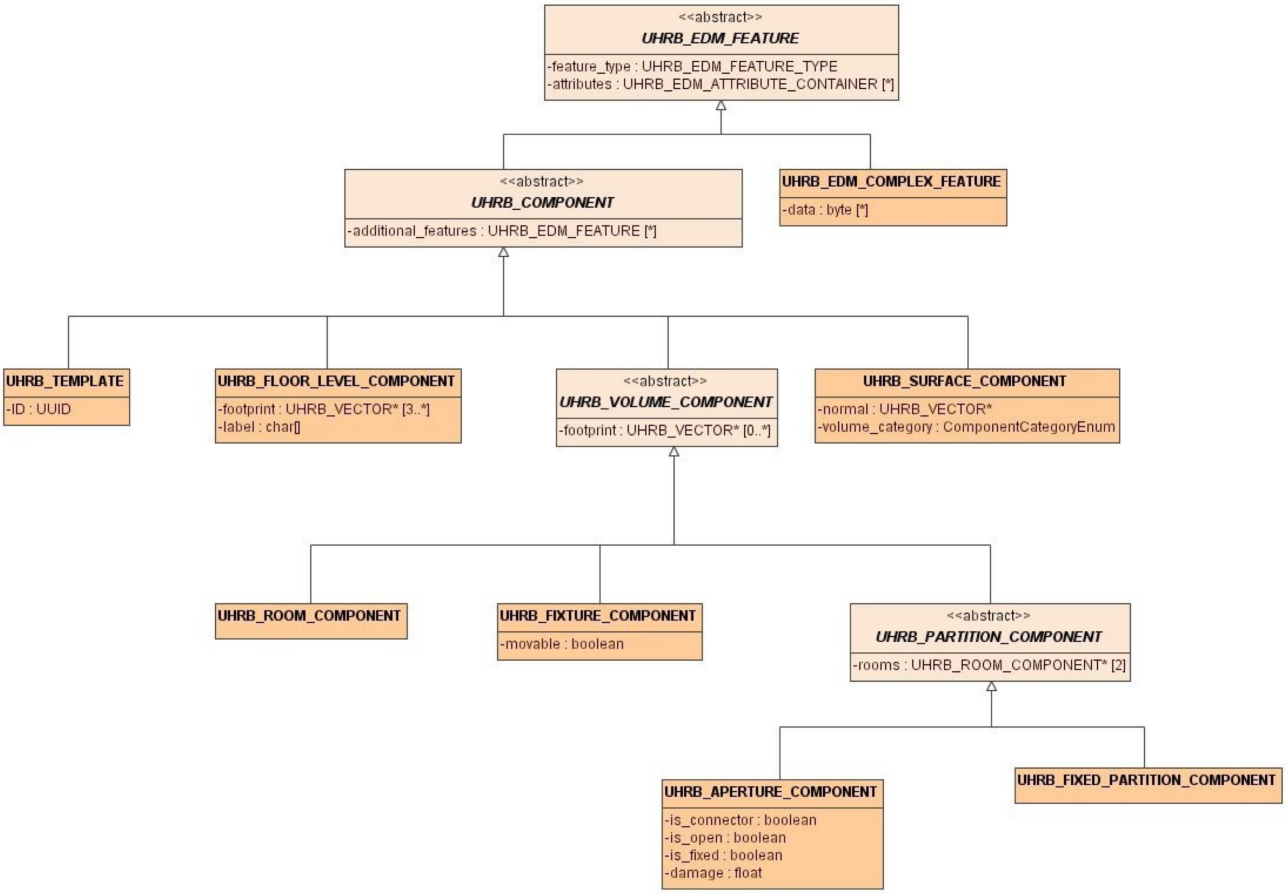


Figure A-4: UHRB Class Diagram

The second diagram is the UHRB Association Diagram of Figure A-5; it shows all permissible associations between the UHRB classes.

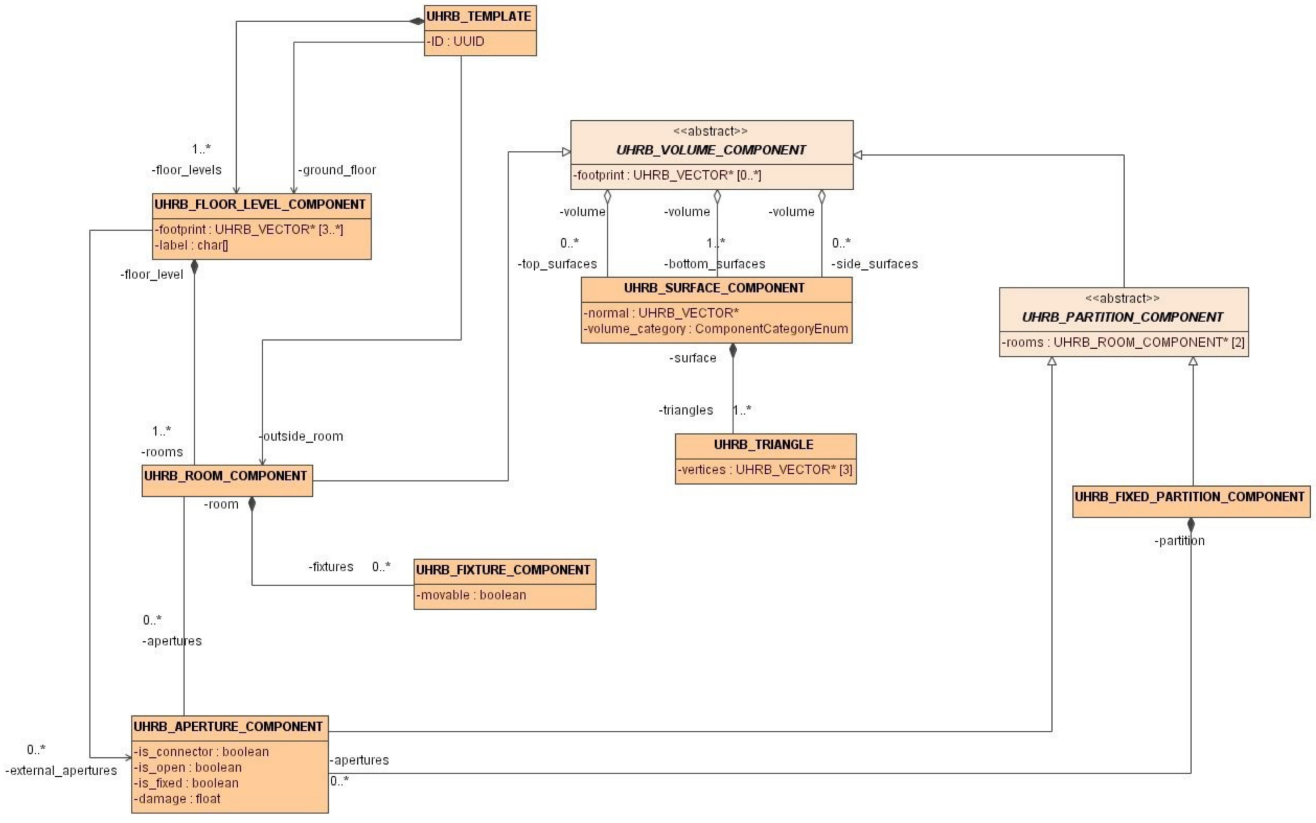


Figure A-5: UHRB Association Diagram

The third diagram, in Figure A-6 below, presents the Object Model proposed by CDB Model Interior objects. The UML diagram is both the class and association diagram of CDB zones listed in table 6-27 of section 6.18.5 of CDB 3.1.

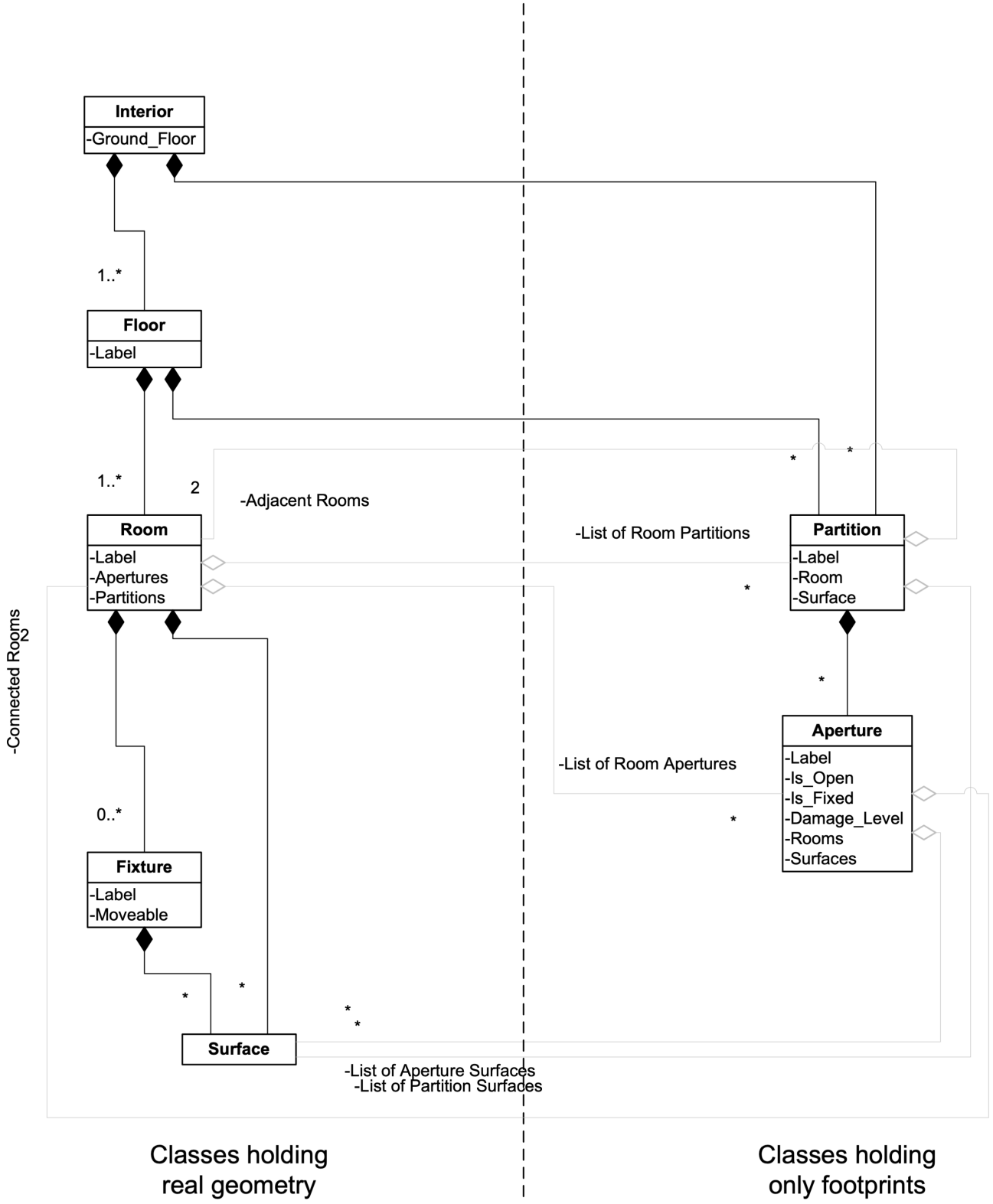


Figure A-6: CDB Model Interior Object Model

6.6. Guideline: Applying Constraints to Uniformly Gridded Terrain

Formerly Annex A.6 in the OGC CDB Best Practice, Volume 2.

The following sub-sections describe the handling of point, linear and polygon (polygon) constraint features into a Uniformly Gridded Terrain Elevation dataset (e.g. terrain x,y offset datasets are not available)

Note that the rendering outcome into the Elevation dataset may vary depending on the rendering order of overlapping points, lines or polygons (polygons). In order to achieve deterministic outcome by all types of client-devices, client-devices are required to sort features by their layer priority number LPN before using them to constrain the terrain elevation dataset.

The rendering of a point, a linear or polygon (polygon) features into the Uniformly Sampled Terrain Elevation dataset is performed into the same LOD as the LOD in which the vector feature appeared.

6.6.1. Constraint Points

This section describes the required client-device behavior for PointZ and MultiPointZ features used as terrain elevation constraint points (AHGT is true) into a uniformly sampled terrain elevation dataset.

The application of a constraint point P is very much like drawing an anti-aliased rectangle centered on P into the uniform terrain elevation grid. The rectangle shape is defined by feature attributes BBL, BBH and AO1. Consider a terrain grid element A in the immediate vicinity of a constraint point P. After applying the constraint P to terrain grid element A, the new elevation E_A is:

$$E_A = E_P * \mathrm{Ain}_{PA} + E_A * \mathrm{Aout}_{PA}$$

where…

E_A is elevation of grid element A

E_P is elevation of constraint point P

Ain_{PA} is the percentage overlap of constraint point P onto grid element A

$$\mathrm{Aout}_{PA} = (1 - \mathrm{Ain}_{PA})$$

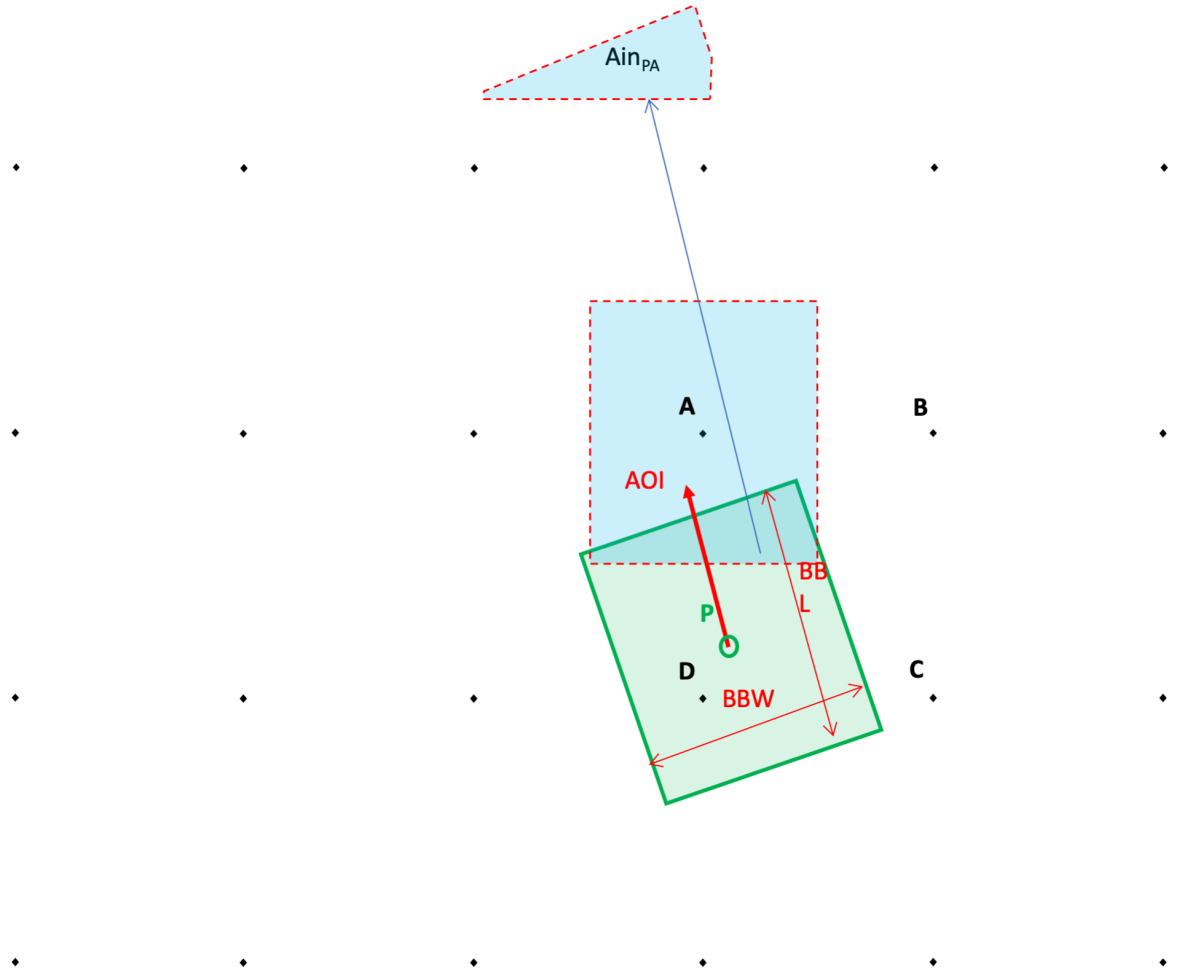


Figure A-7: Application of Constraint Point - Uniformly-Sampled Terrain

6.6.2. Constraint Linear Features

This section describes the required client-device behavior for PolyLineZ features used as terrain elevation constraint linear feature (AHGT is true) into a uniformly sampled terrain elevation dataset.

First, the PolyLineZ feature is broken into a series of constraint lines. The application of each constraint line L is very much like drawing an anti-aliased line centered on L into the uniform terrain elevation grid. The width of the line is defined by feature attribute WGP. Consider a terrain grid element A in the immediate vicinity of a constraint line L, defined by vertices V1 and V2. After applying the constraint line L to terrain grid element A, the new elevation E_A is:

$$E_A = E_{\mathrm{LA}} * \mathrm{Ain}_{\mathrm{LA}} + E_A * \mathrm{Aout}_{\mathrm{LA}}$$

where…

E_{A} is elevation of grid element A

E_{LA} is interpolated elevation of constraint line L at grid element A

$\mathrm{Ain}_{\mathrm{LA}}$ is the percentage overlap of constraint line L onto grid element A

$$Aout_{LA} = (1 - Ain_{LA})$$

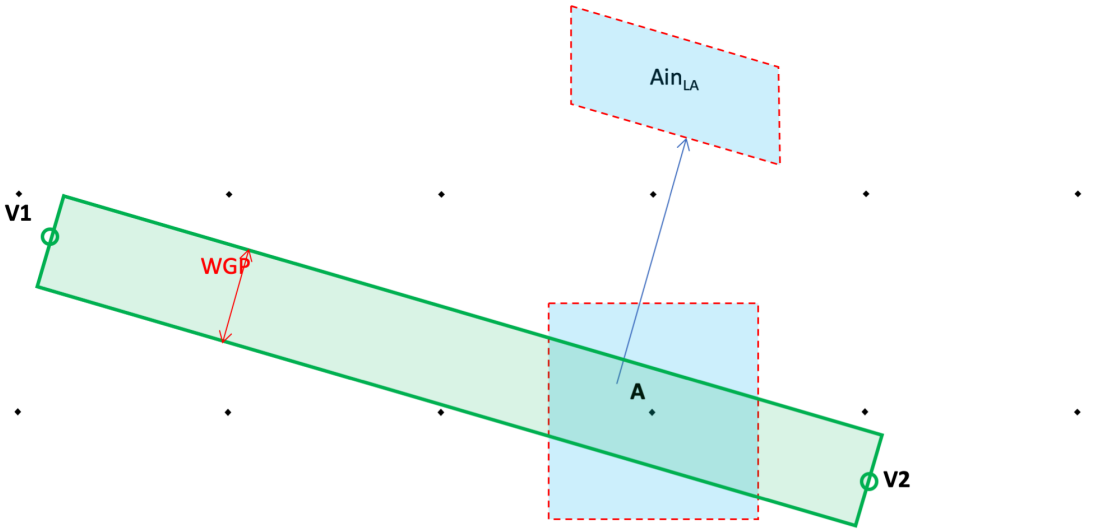
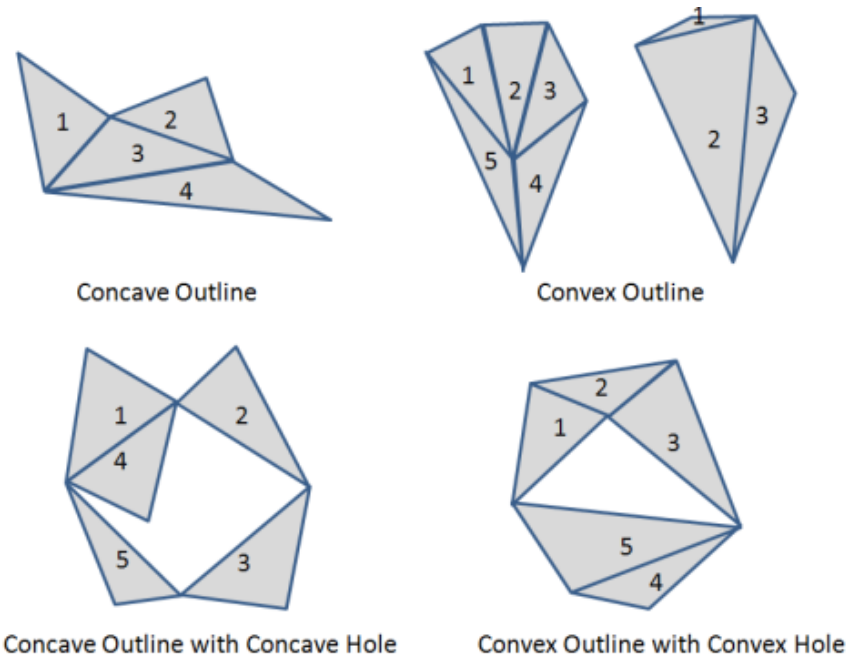


Figure A-8: Application of Constraint Line - Uniformly-Sampled Terrain

6.6.3. Constraint Polygons



This section describes the required client-device behavior of PolygonZ and MultiPatch features

used as terrain elevation constraint points (AHGT is true) into a uniformly sampled terrain elevation dataset.

Each vector PolygonZ feature consists of a number of rings (or parts). Each ring is a closed (the first vertex is same as the last vertex), non-self-intersecting loop. A PolygonZ feature may contain multiple outer rings. A sequence of rings can describe a convex or concave feature outline. In the CDB standard, rings can only be made up of triangles.

Each vector MultiPatch feature consists of a number of rings (or parts). Each ring is a closed (the first vertex is same as the last vertex), non-self-intersecting loop. A sequence of rings can describe a convex or concave feature outline. While the vector MultiPatch feature permits multiple inner rings (aka parts), this capability is dis-allowed in CDB. Furthermore, rings can only be made up of triangles.

The rendering of the vector feature is handled as a series of constraint triangles applied in the order in which they appear within the vector PolygonZ record. The application of each constraint triangle T is very much like drawing an anti-aliased triangle into the uniform terrain elevation grid. Consider a terrain grid element A in the immediate vicinity of a constraint triangle T, defined by vertices V1, V2 and V3. After applying the constraint triangle T to terrain grid element A, the new elevation E_A is:

$$E_A = E_{TA} * \text{Ain}_{TA} + E_A * \text{Aout}_{TA}$$

where…

E_A is elevation of grid element A

E_{TA} is interpolated elevation of constraint triangle T at grid element A

Ain_{TA} is the percentage overlap of constraint line T onto grid element A

$$\text{Aout}_{PA} = (1 - \text{Ain}_{TA})$$

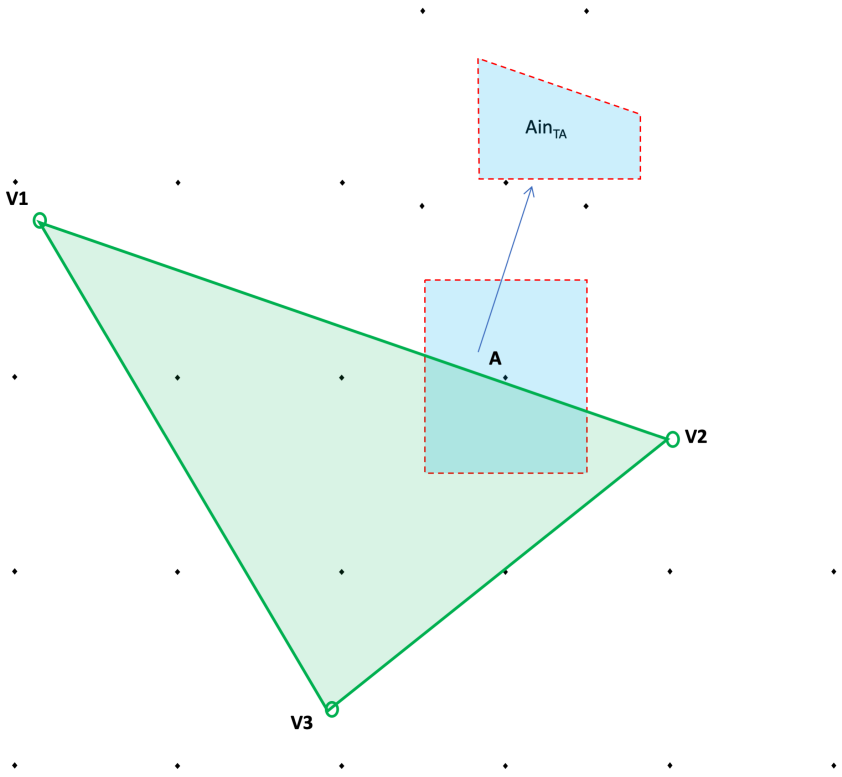


Figure A-9: Constraint Polygons

6.7. Guideline: Applying Constraints to Non-Uniform Gridded Terrain (A.7)

Formerly Annex A.7 in the OGC CDB Best Practice, Volume 2.

The following sub-sections describe the rendering of point, line and polygon (polygons) into a Non-Uniformly Gridded Terrain Elevation dataset described in addendum “CDB Standard Addendum – Non-Uniform Sampled Terrain Elevation”

Note that the rendering outcome into the Elevation dataset may vary depending on the rendering order of overlapping points, lines or polygons. The **L**ayer **P**riority **N**umber (LPN) attribute is used to achieve deterministic outcome by all types of client-devices. When ECP is supplied, client-devices are required to sort overlapping constraint points, lines and polygons in low-to-high order and then render them in that order. Value of LPN can range from 0-32767.

The rendering of a point, a line or polygon features into the Non-uniformly Sampled Terrain Elevation dataset is performed into the same LOD as the LOD in which the vector feature appeared.

6.7.1. Constraint Points

This section describes the required client-device behavior for PointZ and MultiPointZ features used as terrain elevation constraint points (AHGT is true) into a non-uniformly sampled terrain elevation dataset.

The application of a constraint point P is applied as follows.

1. The x,y address of the affected terrain grid element is computed by truncating the lat-long coordinates of point P; note that the truncation operation varies in accordance to LOD of the terrain; however, it always yields grid element addresses in the range of 0-1023.
2. The x,y offset of the affected terrain grid element is computed by performing a MOD of the lat-long coordinates of point P in accordance to its LOD.

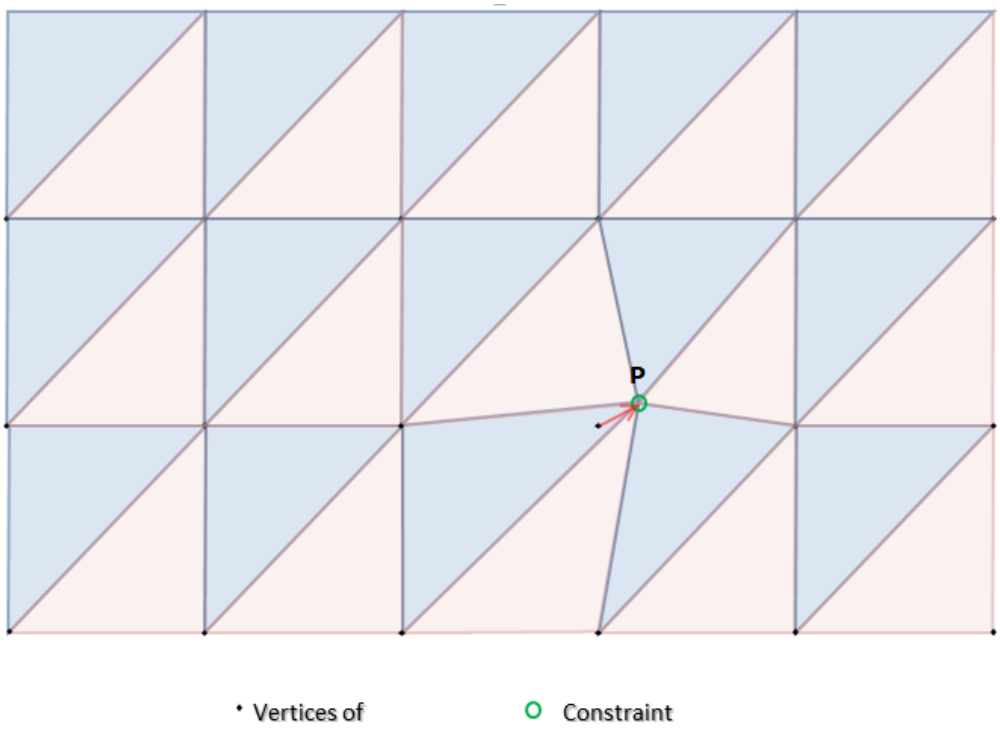


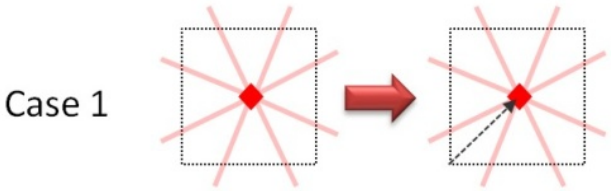
Figure A-10: Application of Constraint Point – Non-uniform Grid

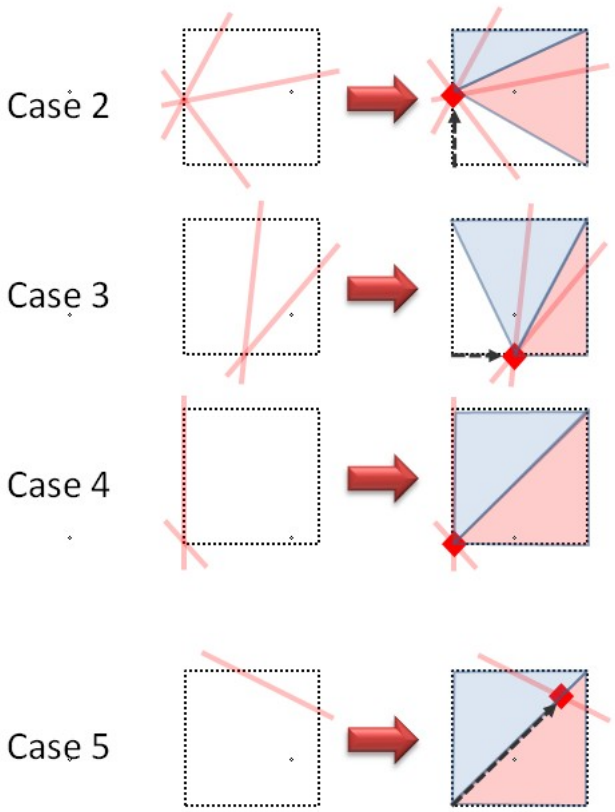
6.7.2. Constraint Linear Features

This section describes the required client-device behavior for PolyLineZ features used as terrain elevation constraint line (AHGT is true) into a non-uniformly sampled terrain elevation dataset.

First, the PolyLineZ feature consisting of n vertices is broken-down into (n-1) line segments defined by successive pairs of vertices.

The application of a constraint line segment L is applied as follows.





1. The x,y offsets of the grid elements of each vertex are computed. (see application of constraint points into non-uniformly sampled terrain (case 1).
2. The offsets of all of the other grid elements that are intersected by the line segment are handled in accordance to the illustration shown here. (Case 2 to Case 5)

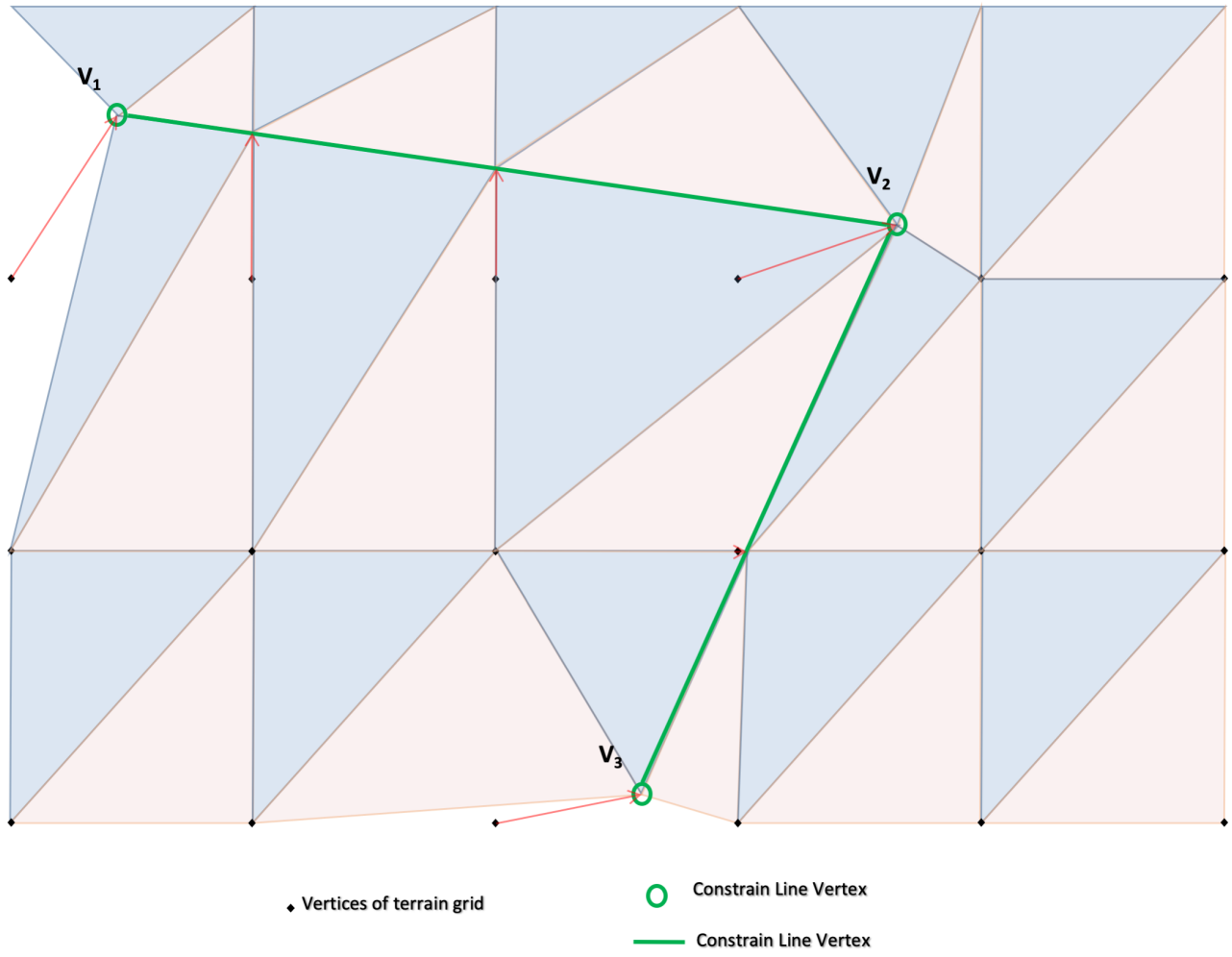
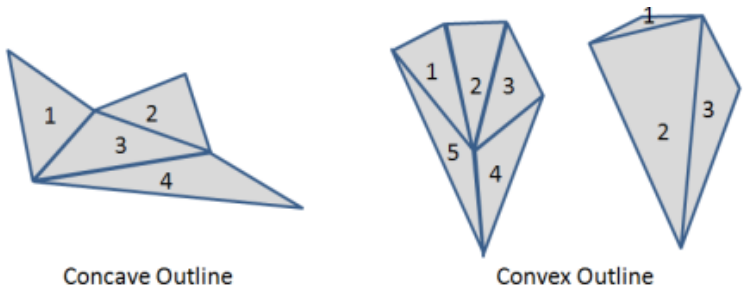


Figure A-11: Application of Constraint Line – Non-uniform Grid

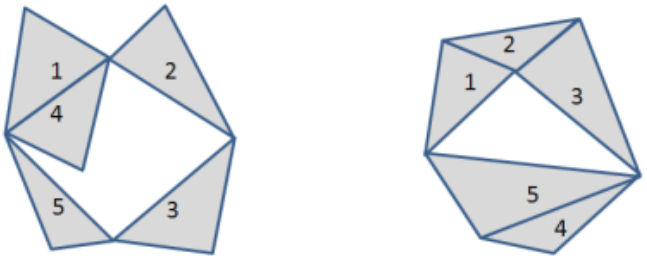
6.7.3. Constraint Polygons

This section describes the required client-device behavior of PolygonZ and MultiPatch features used as terrain elevation constraint points (AHGT is true) into a non-uniformly sampled terrain elevation dataset.

Each vector PolygonZ feature consists of a number of rings (or parts). Each ring is a closed (the first vertex is same as the last vertex), non-self-intersecting loop. A PolygonZ feature may contain multiple outer rings. A sequence of rings can describe a convex or concave feature outline. In the CDB standard, rings can only be made up of triangles.



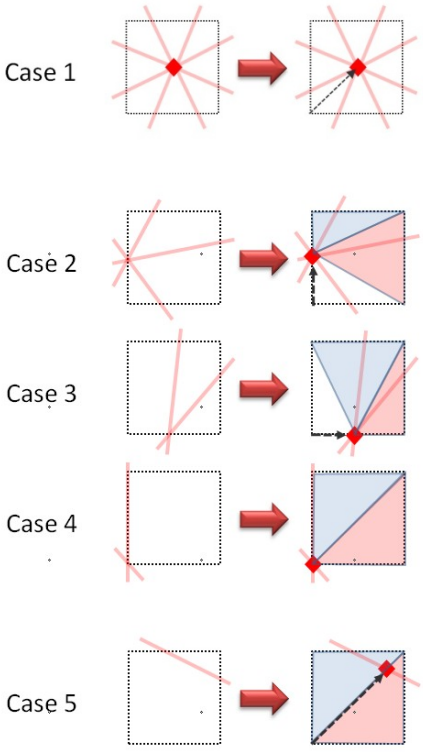
Concave Outline Convex Outline



Concave Outline with Concave Hole Convex Outline with Convex Hole

Each vector MultiPatch feature consists of a number of rings (or parts). Each ring is a closed (the first vertex is same as the last vertex), non-self-intersecting loop. A sequence of rings can describe a convex or concave feature outline. While the vector MultiPatch feature permits multiple inner rings (aka parts), this capability is dis-allowed in CDB. Furthermore, rings can only be made up of triangles.

The application of a constraint triangle T is applied as follows.



1. The x,y offsets of the grid elements of each vertex are computed. (see application of constraint points into non-uniformly sampled terrain (Case 1).
2. The x,y offsets of all the other grid elements that are intersected by the line segments are handled in accordance to the illustration shown here. (Case 2 to Case 5)

3. The x,y offsets of all the other grid elements elevation are set to 0 and the elevation at that lat-long is interpolated using the elevation at the triangle's vertices.

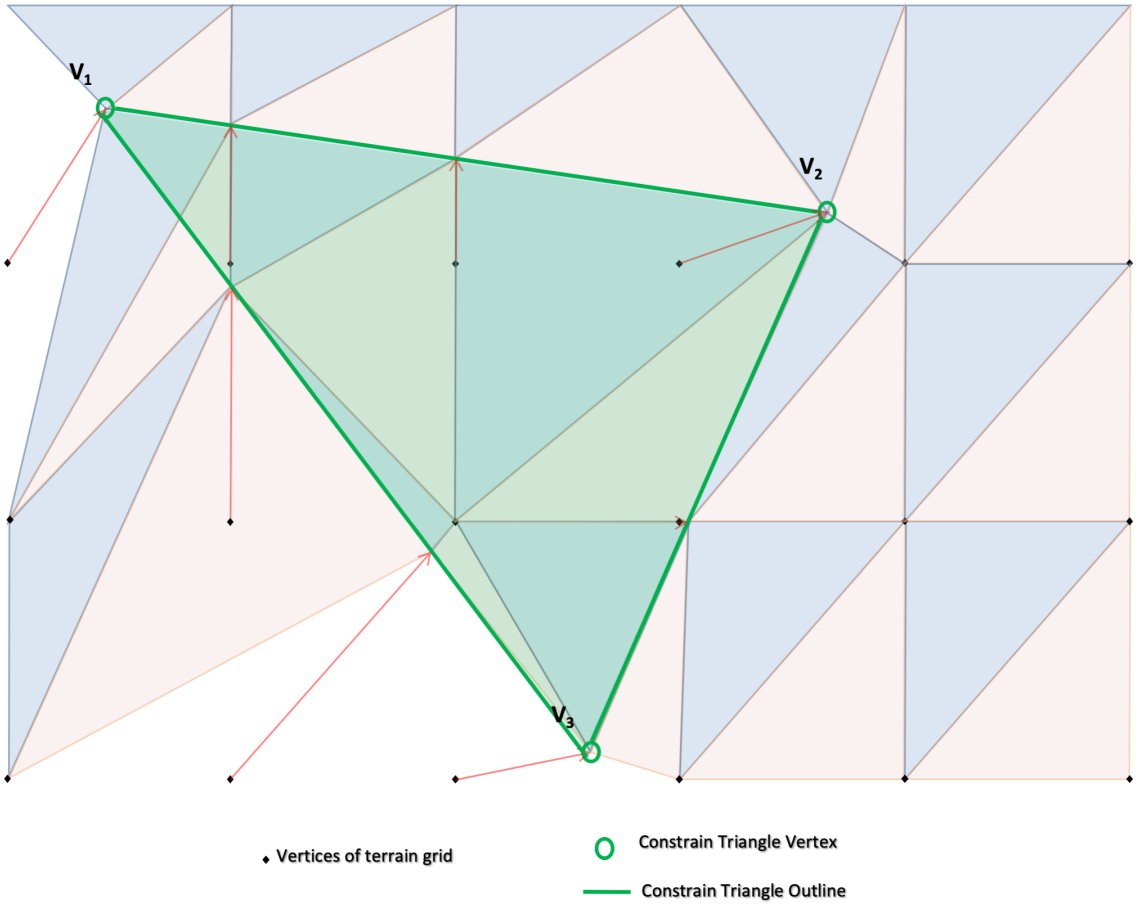


Figure A-12: Application of Constraint Polygon – Non-uniform Grid

6.8. Guideline: LOD Read Behavior of Subordinate Datasets (A.8)

Formerly Annex A.8 in the OGC CDB Best Practice, Volume 2.

In the CDB Standard, LOD read behavior of subordinated datasets was mentioned only briefly in…

- Section 5.2.1.2.3 Subordinate Terrain Elevation Components (Volume 1: OGC CDB Core Standard: Model and Physical Database Structure) which stated "The CDB standard does not permit the use of subordinate Terrain Elevation component when the primary Terrain Elevation component is not generated."
- Section 5.2.1.3.4 Default Read Value: which stated "Simulator client-devices should assume … if the data values are not available (files associated with the Subordinate Terrain Elevation component for the area covered by a tile, at a given LOD or coarser, are either missing or cannot be accessed).
- Section 5.2.1.6 Subordinate Bathymetry Component: which stated "Furthermore, since the Bathymetry values are relative to Terrain Elevation component, each value in the Bathymetry component *must* be matched to the finest available LOD elevation values of the Terrain Elevation component".

- Section 5.2.1.7.3 Default Read Value: which stated “Simulator client-devices should assume … if the data values are not available (files associated with the Subordinate Terrain Elevation component for the area covered by a tile, at a given LOD or coarser, are either missing or cannot be accessed).
- Section 5.2.2.3.2 Default Read Value: which stated “Simulator client-devices should assume … if the data values are not available (files associated with the Subordinate Terrain Elevation component for the area covered by a tile, at a given LOD or coarser, are either missing or cannot be accessed).

This guideline provides clarification on the client-device LOD read behavior of subordinated datasets; it describes the mandated behavior of a simulator client-device when reading a LOD of a Primary Elevation Component and combining it with another LOD of a Subordinate Terrain Elevation Component

Consider the case where a simulator client-device is attempting to read CDB data for a given region of the CDB at LOD = *p*. The CDB region has a Primary Elevation Component populated with data ranging from LOD = -10 to LOD = *m,* and a Subordinate Elevation Component populated with data ranging from LOD = -10 to LOD = *n*.

The required client-device read behavior is illustrated in Figure A-13 below, and can be summarized as follows.

- For $-10 \le p \le m$, the client-device accesses the primary elevation data at LOD = *p*.
- For $p > m \ge -10$, the client-device accesses the primary elevation data at LOD = *m*.
- For $-10 \le p \le n$, the client-device accesses the subordinate elevation data at LOD = *p*.
- For $p > n \ge -10$, the client-device accesses the primary elevation data at LOD = *n*.
- For $p > m$ and $p < n$ and $m < n$, the client-device interpolates the primary elevation data from LOD = *m* to LOD = *p* before combining it with the subordinate elevation data of LOD = *p*.
- For $p > m$ and $p > n$ and $m < n$, the client-device interpolates the primary elevation data from LOD = *m* to LOD = *n* before combining it with the subordinate elevation data of LOD = *n*.
- For $p < m$ and $p > n$ and $m > n$, the client-device interpolates the subordinate elevation data from LOD = *n* to LOD = *p* before combining it with the primary elevation data of LOD = *p*.
- For $p > m$ and $p > n$ and $m > n$, the client-device interpolates the subordinate elevation data from LOD = *n* to LOD = *m* before combining it with the primary elevation data of LOD = *m*.
- For $n = \varphi$ *(unavailable)* and $p > m \ge -10$, the client-device accesses the default value in Defaults.xml for the subordinate elevation data.
- The combination of ($m = \varphi$ *(unavailable)* and $n \ge -10$), is not permitted, i.e., the generation of Subordinate Terrain Elevation LODs is not permitted if the Primary Terrain Elevation component have not been generated.
- If the default value for the Primary Elevation dataset is unavailable in Defaults.xml, or if Defaults.xml file is missing, then the client-device *must* revert to the client-device’s internal default value for this dataset.
- If the default value for the Subordinate Elevation dataset is unavailable in Defaults.xml, or if Defaults.xml file is missing, then the client-device *must* revert to the client-device’s internal

default value for this dataset.

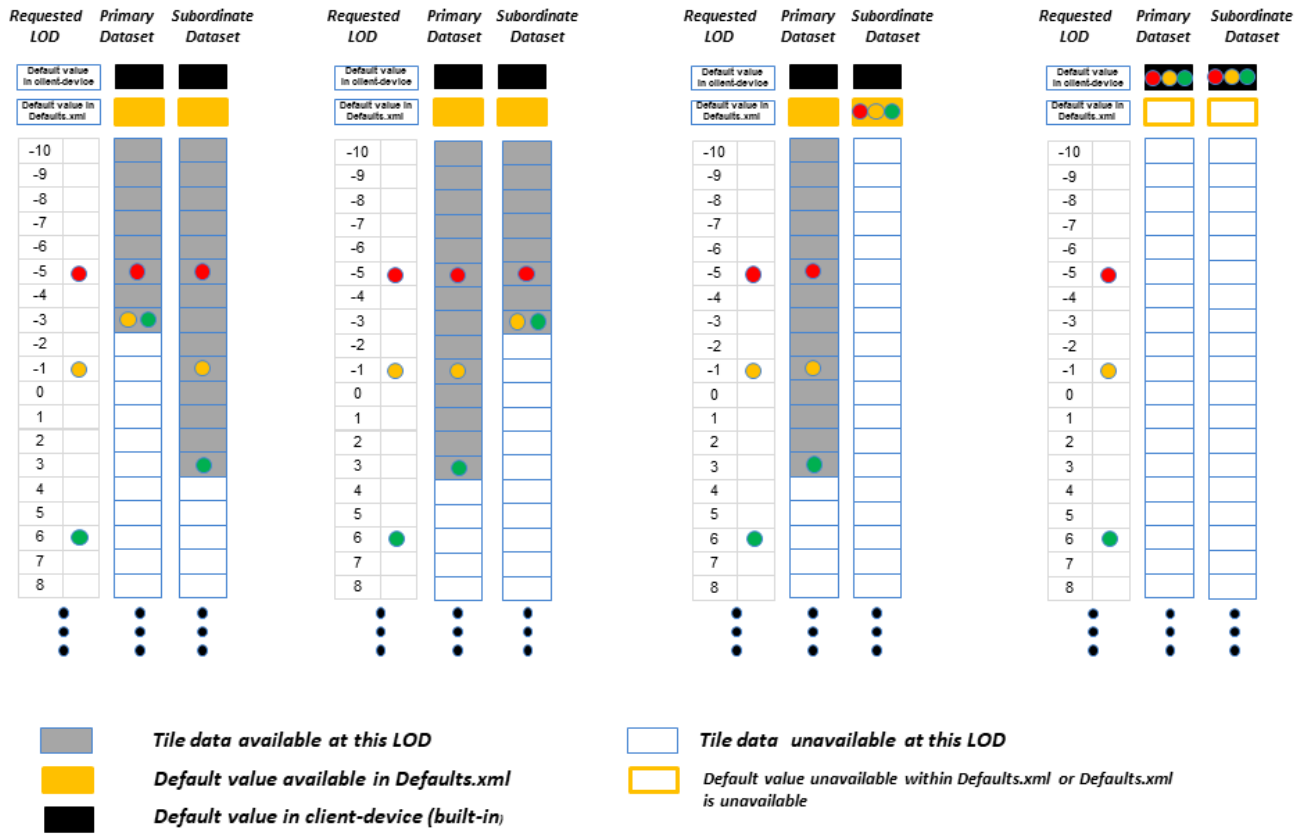


Figure A-13: Client-device Read Behavior

The default value for the Primary Terrain Elevation component is the constant Primary_Elevation, which can be found in \CDB\Metadata\Defaults.xml. The CDB standard recommends that the value for Primary_Elevation = 0. In the case where the default value cannot be found within the Defaults.xml file, or that the Defaults.xml file cannot be found, the CDB standard recommends that client-devices internally generate a default value of Primary_Elevation = 0.

The default values for the Subordinate Terrain Elevation layer "*n*" (where "*n*" is the subordinate elevation layer number, e.g., a value from 2 to 99) is the constant Subordinate_Elevation-*n*, which can be found in \CDB\Metadata\Defaults.xml. The CDB standard recommends that the value for Subordinate_Elevation-*n* = 0. In the case where the default value cannot be found within the Defaults.xml file, or that the Defaults.xml file cannot be found, the CDB standard recommends that client-devices internally generate a default value of Subordinate_Elevation-*n* = 0.

The CDB standard does not permit the use of Subordinate Terrain Elevation components when the Primary Terrain Elevation component is not generated.

6.9. Information: Tide Simulation Modeling Alternatives (Was A15)

Formerly Annex A.15 in the OGC CDB Best Practice, Volume 2.

The availability of a Tide component permits realistic simulation of tides with a minimal computational overhead by the simulation application. Furthermore, the Tide component also permits simulation of tides whose amplitude varies differently with location. In order to determine the shoreline profile at a given location, the simulator client-devices *must* first determine the height of (say) the ocean in the immediate vicinity of that location. The sophistication of this calculation can vary greatly with simulation fidelity.

Figure A-23: Examples of Ocean Tide Simulation Fidelity in Simulator, illustrates examples of how tide simulation might be handled. At the low-end of the fidelity spectrum, the tide level (expressed as a value between –100% (average low tide) and 100% (average high tide) could be provided directly at the simulator's control console. In a high-end simulation, one could develop a simulation of the earth's oceans that takes into account Bathymetry profile of the oceans and the ephemeris model (particularly moon and sun) as a function of time and date. Regardless of simulation fidelity, the CDB internal representation facilitates the work of simulation client devices that are interested in obtaining the shoreline profile and ocean heights.

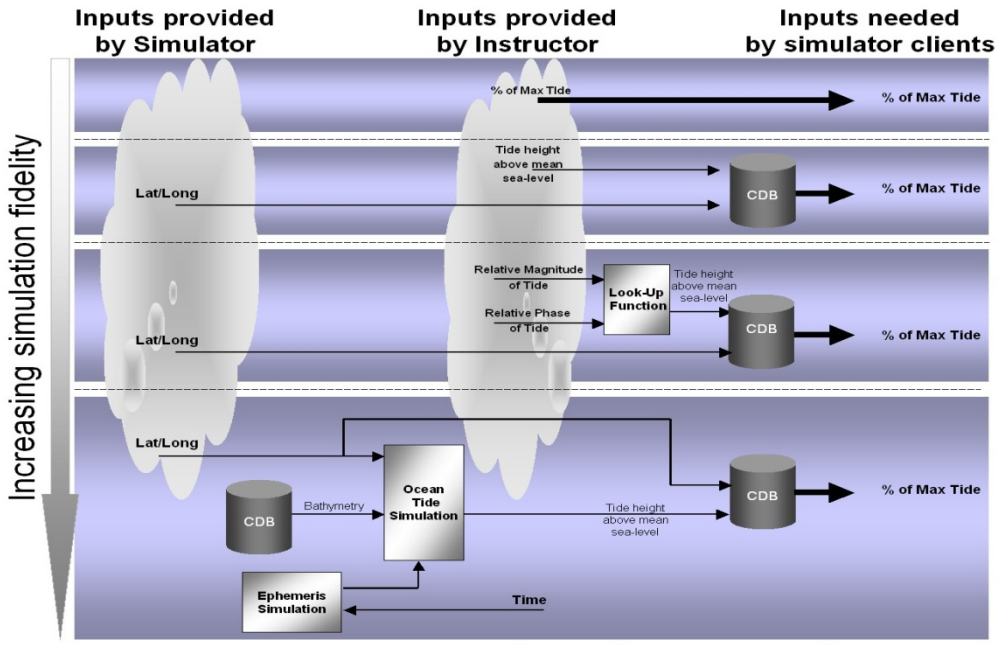


Figure A-23: Examples of Ocean Tide Simulation Fidelity in Simulator

6.10. CDB and FalconView (Was A.16)

Formerly Annex A.16 in the OGC CDB Best Practice, Volume 2.

While the CDB file naming convention and its directory structure are somewhat different from that used in FalconView [5], it is possible to find equivalent files between the two.

The FalconView directory structure contains some metadata describing its content and area

coverage; it has a three-level directory structure. The first level “rpf” is a raster product format: the second level being the dataset such as “gnc” (global navigation chart): and the third level relates to the zones; all files are under the third level. The file name is eight characters long followed by a three-character file extension, and the file name portion uses a radix 34 numbering notation that is based on the position of the frame in the zone as well as revision info and the producer ID key. The file extension is based on the dataset and the zone. Note that frames are equivalent to CDB tiles.

From information such as a given lat/lon, a given resolution such as one-meter pixel size and the dataset such as global navigation chart, it is possible to generate the corresponding FalconView file name and its path. Similarly, given a lat/lon, an LOD and a dataset it is possible to generate a CDB file name and its path. Though not identical in coverage and resolution these two files should be similar in content for the same dataset.

Note that when given a CDB file name, it is possible to extract the tile position in lat/lon, the dataset it belongs to, and the LOD, even its full path name, i.e. the file name is unique for the entire CDB. This is not the case for FalconView. Since the resolution is not implicit in the name, the file itself *must* be read to extract this information; the dataset and zone info can be extracted from the file extension. Also note that directories in FalconView can potentially be very large since all files in a zone reside in the same directory; this is especially true for fine resolutions.

The FalconView directory structure follows the guidelines and conventions specified by MIL-STD-2411.

The algorithms used to find file name are given by examples within the MIL-C-89041 Controlled Image Base (CIB) document; in that document, zones are shown as overlapping. Note that this may not reflect the manner in which FalconView was implemented; nonetheless this does not affect the methodology provided in this section.

6.10.1. FalconView Directory structure

In FalconView, a top-level directory contains files that are metadata containing information about the various datasets and files in the directories.

The FalconView directory structure is as follows:

Falconviewmaps

+---covdata Coverage data

| cgnc.cov Global Navigation charts

| cjga.cov Joint Operation Graphics Air

| cjnc.cov Jet Navigation Chart

| conc.cov Operational Navigation Chart

| ctpc.cov Tactical Pilotage Chart

| mm100.cov 1:100,000 maps

| mm250.cov 1:250,000 maps

| sigfile.sig

| trs_8km.cov Township Range Section

|

+---rpf Raster Product Format

| +---cgnc Global Navigation Map

| | +---1 Zone

| | | 00023023.GN1 File Name

6.10.2. FalconView Zones definition

MIL-STD-2411 divides the world into 18 zones, nine in the northern hemisphere and nine in the southern hemisphere. The first eight zones in both hemispheres are divided into frames, which in turn are divided into sub-frames. Frames are made of pixels with 1536 x 1536 pixels in a frame; there are 36, 6x6 sub-frames per frame. Between each zone, there is an overlap of one frame; this implies that the size of zones will vary slightly depending on the resolution that is used. Table A-5 Zones Range No Overlap gives the approximate range of each zones; 1 – 9 in the north, A - J in the south. The two extreme zones, which cover the north and south poles, use a different scheme and are not discussed here.

Table A-5: Zones Range No Overlap

Zone	**Zone Extent** **No overlap (deg)**	**Zone extent** **No overlap (deg)**
1, A	0	32
2, B	32	48
3, C	48	56
4, D	56	64
5, E	64	68
6, F	68	72
7, G	72	76
8, H	76	80

9, J	80	90

6.10.3. FalconView Zone resolution

Along lines of constant longitude, the pixel constant used to determine the size of frames is a function of the resolution but is independent of the zone. Along lines of constant latitude the constant is a function of both resolution and zone and is based on the mid latitude of the zone. Table A-6 Example Resolution east-west pixel constants that is extracted from MIL-C-89041 enumerates the factors for three resolutions.

Table A-6: Example Resolution east-west pixel constants

Zone	**Pixel constant (10 meter product)**	**Pixel constant (5 meter product)**	**Pixel constant (1 meter product)**
1,A	3,696,640	7,393,280	36,966,400
2,B	3,025,920	6,051,840	30,259,200
3,C	2,457,600	4,915,200	24,576,000
4,D	1,991,680	3,983,360	19,916,800
5,E	1,633,280	3,266,560	16,332,800
6,F	1,372,160	2,744,320	13,721,600
7,G	1,100,800	2,201,600	11,008,000
8,H	824,320	1,648,640	8,243,200
Lat	1,000,960	2,001,920	10,009,600

The north-south or latitudinal pixel constant is the number of pixels from the equator to the pole (90°). The east-west pixel constant is the number of pixels longitudinally from the 180° west longitude meridian going 360° in an easterly direction along the zone midpoint.

6.10.4. FalconView Zone extension based on resolution

To illustrate, we will use as an example a resolution of 10 meters. To calculate the exact latitudinal zone extent for a given resolution, first calculate the number of pixels in a degree of latitude for the resolution

The number of frames needed to reach the nominal zone boundary is the number of pixels per

degree of latitude multiplied by the nominal zone boundary (in degrees), divided by 1536, the number of pixels rows in a frame, and rounded up to the nearest integer. For example in the first zone the number of frames is

The extent of the zone is then

In order to find the extent of the next zone we use the following method, which applies to all zones from 2 to 8 or B to H.

Since there is an overlap of one frame the start point of the zone 2 will be the number of frames required to reach the next zone which nominally is at 48 is: and the extent is

The number of longitudinal frames and subframes is computed by determining the number of subframes to reach around the earth along a parallel at the zone midpoint. The east-west pixel constant is divided by 256 pixels to determine the number of subframes. The results are divided by 6 and rounded up to obtain the number of frame columns.

For example, longitudinally in the first zone we get subframes and frames. Table A-7 Frame/Subframe Sizes for Source Image GSD of 10 Meters, shows the complete set for a resolution of 10 meters.

Table A-7: Frame/Subframe Sizes for Source Image GSD of 10 Meters

Zone Number	Subframes in Zone (Rows) Latitudinal	Frame Rows in Zone Latitudinal	Equator-ward Zone Extent with Overlap	Pole-ward Zone Extent with Overlap
1,A	1,392	232	0°	32.0409207
2,B	702	117	31.9028133	48.0613811
3,C	354	59	47.9232737	56.0716113
4,D	348	58	55.9335038	64.0818414
5,E	180	30	63.9437340	68.0869565
6,F	180	30	67.9488491	72.0920716
7,G	180	30	71.9539642	76.0971867
8,H	180	30	75.9590793	80.1023018
9,J	———	———	varies	90°

Zone Number	Subframes (Columns) Longitudinal	Frames (Columns) Longitudinal	E-W Pixel Constant
1,A	14,440	2,407	3,696,640
2,B	11,820	1,970	3,025,920
3,C	9,600	1,600	2,457,600

4,D	7,780	1,297	1,991,680
5,E	6,380	1,064	1,633,280
6,F	5,360	894	1,372,160
7,G	4,300	717	1,100,800
8,H	3,220	537	824,320

6.10.5. FalconView Frame Position

MIL-C-89041 states that “the origin for counting nonpolar frame rows and columns is the southernmost latitude of the zone and 180° west longitude, with columns counted in an easterly direction from that origin, as opposed to frames and subframes where “the origin for the subframe and pixel numbering within frames and subframes shall be from the upper left corner”.

For a given latitude and longitude the row and column for the frame where that geographic position is situated can be computed. The determination of the zone is derived from the latitude except at the border of zones where an overlap exists and the zone *must* be picked.

The row is given by where is the bottom southern-most latitude of the zone at resolution *r* and is the number of pixels per degrees of latitude at resolution *r*. Similarly, the column corresponding to the longitude is given by where is the number of pixel per longitudinal degrees in zone *z* at resolution *r*, ranges from –180 to 180.

As an example, for latitude of 36 degrees and longitude of –88 degrees we would get for a resolution of 10 meters

6.10.6. FalconView File Naming Convention

MIL-C-89041 for Controlled Image Base (CIB) states that:

“The naming convention for all resolutions of images registered in MIL-STD-2411-1, where it is intended for producers to provide contiguous [frame file] coverage, shall conform to MIL-STD-2411. In addition, the CIB [frame file] names are further restricted to conform to the form “ffffffvp.ccz.” The “ffffff” portion of the name shall be a radix 34 value that encodes the unique cumulative frame number within a zone in base 34, with the right-most digit being the least significant position. The radix 34 value incorporates the numbers 0 through 9 and letters A through Z exclusive of the letters “I” and “O” as they are easily confused with the numbers “1” and “0”. For example, the “ffffff” portion of the names would start with “000000,” proceed through “000009,” “00000Z,” “000010,” and so forth until “ZZZZZZ.” This allows 1,544,804,416 unique [frame file] names; a contiguous grid of frame names down to a resolution of 0.2 meters (approximately 8 inches) can be defined. The “v” portion of the name shall be a radix 34 value that encodes the successive version number. The “p” portion of the name shall be a radix 34 value that designates the producer code ID, as defined in MIL-STD-2411-1. The “cc” and “z” portions of the name extension shall encode the data series code and the zone, respectively, as defined in MIL-STD-2411-1. The CIB producers are responsible to ensure that [frame files] for all image resolutions, zones, and revisions, have unique names.”

In our case:

…

In the example of a lat of 36 and lon –88 with a resolution of 10 meters we get:

ffffff = 503+29x1970=57633 or $001FV3_{(34)}$

… where 1970 is number columns in zone 2 as given in Table A-7 Frame/Subframe Sizes for Source Image GSD of 10 Meters, and in RADIX 34 we get ffffff = 001FV3 ; for a global navigation chart dataset a version level 0, a manufacturer code of 3 and zone 2 the file name would be equal to "001FV303.GN2."

Note that nothing in the file name defines the resolution for the data; this information is part of the [coverage section] in the file itself (see section 3.12.3 in MIL-C-89041). Also note that the file name is unique only to the zone at a given resolution.

On the other hand a similar file for imagery (VSTI, Visible Spectrum Terrain Imagery) in the CDB convention for an LOD of 04 which has a resolution of approximately 8 meters; at position lat 36 and lon –88 we would get for the file name:

\CDB\Tiles\N36\W088\004_Imagery\L04\U0\
N36W088_D004_S001_T001_L04_U0_R0.jp2

Note that the file name itself is unique worldwide and that from it we can derive the directory path to which it belongs.

6.11. Managing CDB Data Store Versions (Was A.18)

Formerly Annex A.18 in the OGC CDB Best Practice, Volume 2.

The incremental versioning mechanism of a CDB data store provides a fast method of creating versions of the CDB data store changes since all the data changes are located under a single root directory. The creation and the managing of the (incremental) data files are however under the application control.

A CDB data store can simultaneously hold multiple incremental versions of the data. As a result, it is possible to select any of the versions without transferring or copying files. Consider the case where a data store generation facility, a data store quality assurance facility, a simulator mission planning facility, a mission rehearsal facility and a mission debrief are all operating concurrently on distinct versions of the CDB. This is illustrated in Figure 3 2: Concurrent Usage of Versions of the CDB data store. By the fourth day, there are four versions of the CDB data store, say the active default CDB (v1) and three incremental versions (v2, v3, v4). Any of these four versions can be instantly invoked (without copying or transferring files) by the simulator operator at the Mission Rehearsal facility, or by an instructor at the Mission Debrief facility.

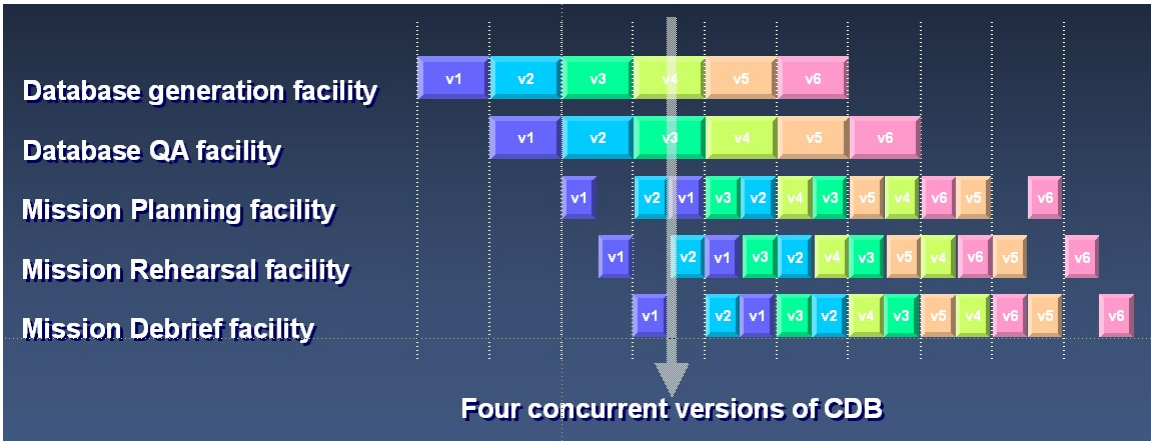


Figure A-24: Concurrent Usage of CDB Versions

The underlying CDB versioning mechanism is a fine-grained file-level mechanism, i.e., only the affected files of the geographic areas of the CDB data store need to be versioned, leaving the rest of the CDB data store intact. This approach is invaluable in mission rehearsal applications where the target areas of the CDB data store require frequent updates based on the latest intelligence data.

The approach can also be applied to the handling of classified secure data. In this case, a CDB version can be used to hold the portion of the CDB data store that contains the classified information. The incremental versioning mechanism would be used to segregate the classified portion of the CDB data store onto a separate storage medium. Since the classified portion of the CDB data store is embedded within the overall CDB structure, it is possible for the runtime publishers to instantly switch back and forth between the classified and non-classified versions of the data store.

6.12. Guideline: Handling of GS and T2D Models (Was A.19)

Formerly Annex A.19 in the OGC CDB Best Practice, Volume 2.

6.12.1. GSModels

GSModel Levels-of-detail

The insertion of a 3DModel-LOD into the LOD hierarchy of the GSModel Dataset is solely dependent on its Location, its Significant Size and on its Storage Size.

The location and Significant Size of a 3DModel-LOD determines where it is nominally inserted into the GSModel Dataset hierarchy. This approach ensures that the modeled content is organized in files that contain co-located objects of similar size. *This approach provides client-device with an optimal means of accessing and filtering modeled content (by location and by size).*

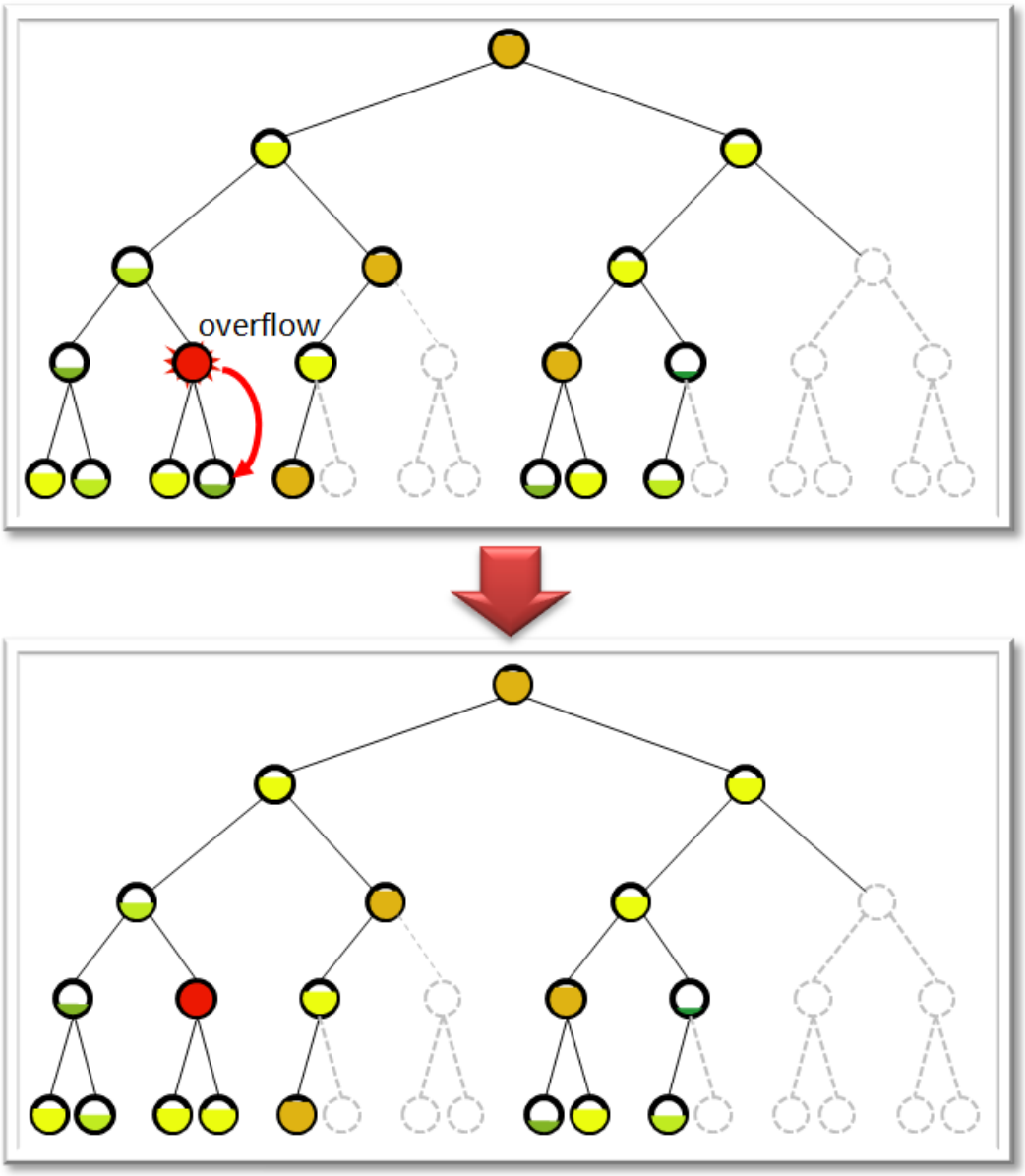


Figure A-25: Handling Tile-LOD Overflows in GSModel Dataset

3DModel-LODs are accumulated into the Tile-LODs of the GSModel hierarchy. The size of these Tile-LODs is capped to *GSModelFileSize*. In the event that a group of 3DModel-LODs nominally assigned to a Tile-LOD causes this limit to be exceeded, the 3DModel-LODs that are deemed to have the lowest contribution to the Tile-LOD are moved to finer (children) Tile-LODs until the Tile-LOD is once again within its size limit (illustrated in Figure A-25: Handling Tile-LOD Overflows in GSModel Dataset). In the event that a 3DModel-LOD is itself larger than *GSModelFileSize*, the 3DModel-LOD is moved to the 4 finer Tile-LODs of the GSModel Dataset hierarchy. *This approach ensures that the modeled content is accessible in chunks that are bounded; this improves the allocation and management of memory allocation in the client-devices.*

Note: The CDB Specification defines the value of *GSModelFileSize* to 4 MB

NOTE	The Significant Size of a 3DModel-LOD determines where it is nominally inserted into the 3DModel LOD hierarchy. In this nominal case, each Tile-LOD of the 3DModel Dataset holds a group of 3DModels-LODs that have similar Significant Sizes. This enables the client-devices to determine the range at which the 3DModel-LOD can be optimally blended-in to the scene (so that the model falls within a specified angular error criterion).

The bounding criterion of 3DModel Tiles can lead to LOD migration, thus breaking the relationship between the Significant Size of a 3DModel-LOD and the nominal CDB LOD it belongs to. As a result, client-devices can no longer guarantee the range at which the 3DModel-LOD will be blended-in to the scene. In effect, each time the 3DModel-LOD is migrated by one LOD, the client-device will likely shorten the range at which it is blended into the scene by a factor of 2X, leading to potentially distracting artifacts. The severity of the artifacts is proportional to the amount of content that has migrated to finer LODs and to the number of LODs by which the content has moved.

While the CDB standard allows the migration of 3DModel-LODs to finer LODs when Tile-LODs overflows are encountered, it is understood that this may lead to rendering artifacts that might be considered unsatisfactory. *Consequently, it is strongly recommended that tools (that generate the CDB hierarchy) be designed to optionally disallow the migration of 3DModel-LODs to finer LODs upon overflows, and instead flag the overflow condition and then abort.* Upon such cases, modelers can then re-assess which 3DModels should be discarded or remodeled in order to simultaneously satisfy the CDB bounding criteria and the application requirements.

Each of the 3DModel-LODs is nominally configured as exchange-LODs. The exchange-LOD mechanism assumes that client-devices gradually substitute a coarser 3DModel-LOD located in a coarser Tile-LOD with a finer 3DModel-LOD located in a finer Tile-LOD.

While this exchange-LOD mechanism is simple, it can lead to inefficiencies when extremely fine features cause the GSModel Dataset hierarchy to be extended by several LODs. Consider the case of a 1 meter road sign located next to a large building (30m wide x 30m long x 10 m high). As we will see in the following section, the road sign would nominally be inserted at LOD 9 of the GSModel Dataset hierarchy. Conversely, the large modeled building would nominally be inserted at LOD 4. The road sign forces the GSModel Dataset hierarchy to be extended by 5 additional LODs.

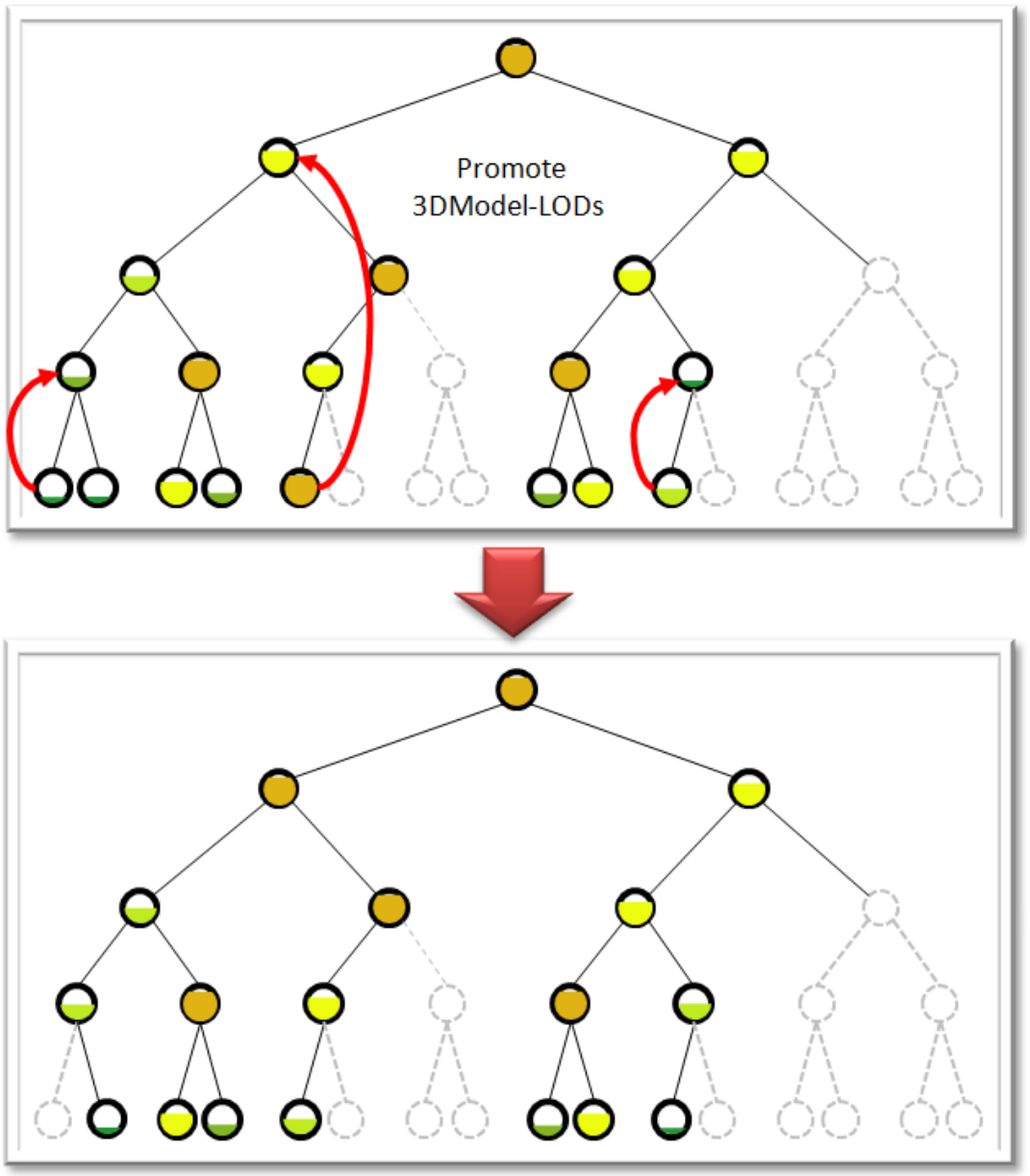


Figure A-26: Compacting the GSModel Dataset

In order to reduce the depth of the LOD hierarchy, the GSModel Dataset is post-processed and subjected to a "compaction" process, starting from the finest LOD (e.g. LOD_{max}) and progressing to the coarser levels. The compaction process takes finer 3DModel-LODs and appends them to the corresponding 3DModels in coarser Tile-LODs of the GSModel Dataset. The appended (finer) 3DModel-LOD *must* have an explicit OpenFlight LOD node with the Significant Size of the 3DModel-LOD; this provides the necessary information for the client-device to control the range at which the 3DModel-LOD will be introduced into the rendered scene. The process is recursively applied to the coarser LODs until the parent LOD is packed to capacity. *This approach ensures that the modeled content is accessible in similarly-sized chunks of processing; this provides the means to improve internal parallelism and pipelining (i.e. improves client-device determinism).*

The access and selection of 3DModel-LODs is done through the GSFeature Dataset. Each of the Tile-LODs of the GSFeature Dataset contains a list of Features; each Feature in turn points to a 3DModel-LOD at the appropriate LOD. In effect, the appearance of a Feature (along with its modeled representation) and the evolution of its modeled representation are entirely controlled by the GSFeature Dataset. As a result, the 3DModel-LODs of a 3DModel need not be located in consecutive

LODs of GSModel Dataset hierarchy.

CDB LOD versus GSModel Significant Size

Section 6.8.3 of CDB Standard Volume 6: OGC CDB Rules for Encoding Data using OpenFlight provides a set of guidelines to establish the values for Significant Size SS_c and SS_{LOD} for GSModels.

Table 3 1: CDB LOD vs. Model Resolution shows the nominal position of a GSModel within the LOD hierarchy of the GSModel Dataset. Note all of the GSModel-LODs of a GSModel normally fall within a range of 8 levels-of-detail (i.e. the smallest tile size the GSModel can sit on). However, it is possible to extend this range by breaking up a GSModel-LOD into several OpenFlight files.

Here is a summary of the rules required by the CDB standard in order to ensure deterministic operation from client-devices.

1. Each feature may have multiple modeled representations at progressively coarser levels of detail. Each of the modeled representations is referred to as a GSModel-LOD. In absence of pre-modeled coarser LOD representations, the tools may automatically generate coarser modeled levels-of-detail.
2. A GSModel-LOD consists of a group of polygons that represent a feature at a specific level-of-detail; this group of polygons shares a unique Model Identifier derived from the Feature Attribute Code a Feature Sub-Code (FSC), a Model Name (MODL or MMDC), the GSModel-LOD's Significant Size SS'_{LOD}.
3. Each GSModel has a distinct Significant Size *SS'* value based on its dimensions. In turn, each GSModel-LOD of a same GSModel has a distinct Significant Size value SS'_{LOD} based on its modeled accuracy.
4. Insertion of a GSModel-LOD into the GSModel Dataset hierarchy proceeds as follows. Starting with LOD_{max} (LOD_{max} is a variable set by the user that sets the maximum depth of the LOD hierarchy) and progressing to coarser LODs…
    1. For each Tile-LOD, create a Model_List that is constructed from the GSModel-LODs that straddle the Tile-LOD.

        (i) If the GSModel-LOD is not the coarsest LOD and its Significant Size is in accordance to Table 3 1: CDB LOD vs. Model Resolution, then add it to the Tile-LOD. Only the coarser GSModel-LODs of this GSModel are available for future insertion into the GSModel LOD hierarchy.

        (ii) If the GSModel-LOD is the coarsest LOD of the GSModel and its Significant Size is in accordance to Table 3 1: CDB LOD vs. Model Resolution, then insert it at this LOD of the hierarchy. If the GSModel-LOD matches the Tile-LOD, remove it from the list for the processing of the coarser Tile-LOD.

    2. If the Model_List is less than *GSModelFileSize*, no further processing is required.

        NOTE The Storage Size of (statically-positioned) MModels is assumed to be zero.

    3. The Model_List of each Tile-LOD is sorted in decreasing order of Diff, where Diff is the difference between the Significant Size SS of the Model and the Significant Size as specified

in Table 2.

4. If the size of the Model_List is greater than *GSModelFileSize*, then (starting with the first entry in the sorted Model_List), Models are simplified one-by-one until the size of the Model_List is less than *GSModelFileSize*. When a simplification occurs, the Model_List is re-sorted using the Diff value.
5. If a) the Model_List is deemed non-reducible and b) the Model_List is still greater than *GSModelFileSize* …

   i. If LOD < LOD_{max}, then…

   (1) a Temp_Model_List is created and initialized with the contents of the Model_list. Starting from the end of the Model_List, Models are removed one-by-one from the Model_list (starting with the first Model in the Model_List) and are copied into the Temp_Model_List until the Model_List reaches *GSModelFileSize*.

   (2) The Temp_Model_List is merged to the children Tile-LODs and the children are re-processed using steps 4a to 4e. The process is iterative, i.e., the “overflow” is propagated into the finer LODs of the GSModel hierarchy.

   ii. Else…

   (1) Models are removed one-by-one, starting with the first Model in the Model_List, until the Model_List is less than *GSModelFileSize*. The corresponding GSModels are removed from the CDB and a warning is issued stating that content was removed

NOTE It is strongly recommended that GSModels be modeled using several GSModel-LODs, spanning a wide range of fidelity. The availability of many LODs ensures suitability of the resulting CDB for real-time use with a minimum degradation in fidelity. Conversely, a low number of LODs can lead to unacceptably large steps in fidelity.

NOTE It is strongly recommended that the coarsest modeled LOD of GSModels have no more than 128 vertices; this reduces the likelihood that the coarsest modeled LOD need be propagated to a finer LOD of the hierarchy.

NOTE This algorithm preserves the highest available modeled content while ensuring that the runtime constraint file size limits are respected. While the CDB data model allows for infinitely-sized GSModel-LODs, a client-device may refuse to render the GSModel-LOD if it has insufficient memory to load all of the OpenFlight files that make-up the GSModel-LODs.

5. Each GSModel-LOD is subject to an OpenFlight file size limit of *GSModelFileSize*, i.e. several OpenFlight files, each within the *GSModelFileSize* limit, can be used to represent a very complex GSModel-LOD. Each of OpenFlight files that form the GSModel-LOD share the same GSModel-LOD Identifier (see rule 2) and GSModel-LOD origin. Client-devices *must* render the GSModel-LOD in its entirety, even if it is allocated to several OpenFlight files.

NOTE	While the CDB data model allows for infinitely-sized GSModel-LODs, a client-device may refuse to render the GSModel-LOD if it has insufficient memory to load all of the OpenFlight files that make-up the GSModel-LODs.

1. Each Tile-LOD is subject to a file size limit of *GSModelFileSize*.
2. All of the GSModel-LODs in a GSModel OpenFlight file are nominally exchange-LODs (see exception in next rule).
3. The depth of the GSModel LOD hierarchy should be reduced by folding-in the finer GSModel-LOD located in a finer Tile-LOD to the next coarser Tile-LOD of the hierarchy. Failure to perform this "compaction step" may result in significantly deeper GSModel LOD hierarchy when the finest GSModel-LODs consist of small features or small details on the same features (e.g., small posts next to a terminal building or fine window details on a large building).
4. The finer modeled representation of a GSModel (i.e. a GSModel-LOD with a smaller Significant Size) always appears in finer LODs of the GSModel Dataset LOD hierarchy than a coarser GSModel-LOD.
5. A Tile-LOD cannot contain more than one GSModel-LOD of the same GSModel.
6. Once inserted into the GSModel Dataset LOD hierarchy, there is no mandatory requirement to clip the contents of a GSModel Tile-LOD against its Tile-LOD boundaries. However, the contents of the GSModel Tile-LOD cannot protrude Tile-LODs by more than ½ the dimension of the Tile-LOD.
7. There is no mandatory requirement to have consecutive GSModel-LODs in consecutive LODs of Tile-LOD hierarchy; it is permissible to have gaps within the Tile-LOD hierarchy.
8. Gaps in the LOD file hierarchy of the GSFeature Dataset are not permitted. This may result in Tile-LODs that are empty (e.g. without any GSFeatures). The presence of an empty Tile-LOD file for the GSFeature Dataset indicates the availability of modeled content invoked by finer LODs of the GSFeature hierarchy.

Example – Insertion of a GSModel with 3 LODs into the CDB Hierarchy

Consider an industrial building 200m wide x 200m length x 10m high. The modeler has not supplied any values for its Significant Size, nor has he provided a value for RTAI. It is modeled in three distinct levels of detail as follows:

1. Coarsest level: 5 polygons
2. Mid level: 60 polygons
3. Finest level: 300 polygons

Based on this information, we can derive Significant Size values for each of the modeled representation as follows and determine where within the hierarchy each of the GSModel-LODs should be inserted:

a. **Coarsest level-of-detail:**

1. Compute the model's Significant Size …

$$SS = \sqrt{\frac{(10 \times .96) \times 200) + \ (200 \times 200 \times .259\ )}{\pi}}$$

$$SS = \ 62.5m$$

2. Since the model is opaque and has no assigned value for RTAI, the final value for *SS* is 62.5m.
3. Table 3 1: CDB LOD vs. Model Resolution, tells us that the (coarsest LOD) of the model should be nominally inserted at LOD 3 of the Tile-LOD (assuming its file size limit is not exceeded)

2. **Mid level-of-detail:**

1. Compute the ratio of vertices

$$R = \frac{V_{\text{LOD}}}{V_{\text{coarsest}}} = \frac{60}{5} = 12$$

2. Compute the Significant Size of the GSModel-LOD…

$$SS'_{\text{LOD}} = \frac{SS'_{\text{coarsest}}}{\sqrt{12}} = 18.04m$$

3. Since the model is opaque and has no assigned value for RTAI, the final value for SS'_{LOD} is 18.04m.
4. Table 3 1: CDB LOD vs. Model Resolution, tells us that the (mid- LOD) of the model should be nominally inserted at LOD = 5 of the Tile-LOD (assuming its file size limit is not exceeded)

3. **Finest level-of-detail:**

1. Compute the ratio of vertices

$$R = \frac{V_{\text{LOD}}}{V_{\text{coarsest}}} = \frac{300}{60} = 5$$

2. Compute the Significant Size of the GSModel-LOD…

$$SS'_{\text{LOD}} = \frac{SS'_{c}}{\sqrt{5}} = 8.07m$$

3. Since the model is opaque and has no assigned value for RTAI, the final value for SS'_{LOD} is 8.07.
4. Table 3 1: CDB LOD vs. Model Resolution, tells us that the (finest-LOD) of the model should be

nominally inserted at LOD = 6 of the Tile-LOD (assuming its file size limit is not exceeded)

6.12.2. T2DModels

The T2DModels are stored in the OpenFlight format. The CDB conventions described herein are designed to facilitate the integration of such models onto the terrain tile, hence the name "**T**iled **2D Models**". Each 2DModel can have one or more modeled representation (called a 2DModel-LOD) that represents the feature to a certain level of fidelity. 2DModel-LODs are re-grouped into T2DModel Tile-LODs; this re-grouping approach is designed to reduce the overheads associated with the access of 2DModel-LODs. Furthermore, T2DModel-LODs can be accessed without a prior reference to a corresponding feature in the GSFeature dataset.

The integration of T2DModels to the underlying terrain skin is performed by the client-devices at runtime. Historically, this integration has always been performed by the tools and was "baked-in" into the SE terrain skin during the offline data store generation process. Many client-specific considerations went into the mechanisms required to support this integration and as a result, the resulting synthetic environments were very client-specific and did not scale easily to higher resolutions.

In line with CDB principles, the T2DModel Dataset defers this integration and imposes it on the consumers (not the producers) of synthetic environments. As a result, client-devices can independently access, manage and control each dataset, i.e., the Primary Elevation, the VSTI Imagery, the T2DModel, etc. This layered approach to synthetic environment production and consumption provides a much greater level of abstraction between the SE data model and the data models internal to each client-device. It is understood, that the deferral of the integration process imposes added functionality and computational requirements on the part of the CDB client-devices.

While it would be possible, in theory, to use the T2DModel Dataset for the modeling of the terrain skin, this use-case is specifically forbidden because the T2DModel Dataset does not provide a guarantee of full tile coverage. As a result, the Primary Elevation Dataset is always required regardless of whether a corresponding Tile-LOD of the T2DModel is present or not. Furthermore, since CDB forbids the duplication of information, the terrain skin cannot be duplicated by the T2DModel Dataset.

Client-devices *must* always access the Primary Elevation prior to any other raster datasets. Once a Tile-LOD of the Primary Elevation is loaded, a client-device can then access the T2DModel Dataset at an "appropriate" LOD [6]. Following this, the client-device *must* integrate the models found within the T2DModel Tile-LOD with the terrain found in the Primary Elevation dataset.

T2DModel Levels-of-detail

As with 3D features, 2D features can have modeled representations at varying levels of detail. Each of these modeled-representations is referred to as a 2DModel-LOD. A 2DModel-LOD consists of a group of polygons that represent a 2D feature at a specific level-of-detail.

Once a 2DModel-LOD is inserted into the T2DModel Dataset hierarchy, it is then referred to as a T2DModel-LOD. The insertion of a 2DModel-LOD into the LOD hierarchy of the T2DModel Dataset is solely dependent on its Location, its Significant Size and on its Storage Size. 2DModel-LODs are regrouped into files called T2DModel Tile-LODs. Note that when a 2DModel is clipped to the

T2DModel's Tile-LOD boundaries, each of the clipped model fragments will appear in distinct OpenFlight files of the T2DModel Dataset. The T2DModel Tile-LODs are assembled into a hierarchy of Tile-LODs called the T2DModel Dataset.

The organization of the modeled content into files that contain co-located objects of similar size greatly improves runtime performance. The location and Significant Size of a 2DModel-LOD determines where it is nominally inserted into the T2DModel LOD hierarchy. This approach ensures that the modeled content is organized in files that contain co-located objects of similarly size. *This approach provides client-device with an optimal means of accessing and filtering modeled content (by location and by size).*

2DModel-LODs are accumulated into Tiles for each LOD of the T2DModel hierarchy. The size of these T2DModel Tiles is capped to *T2DModelFileSize* [7]. The current value for *T2DModelFileSize* is 4 megabytes. In the event that the insertion of a 2DModel-LOD causes this limit to be exceeded, the 2DModel-LODs that are deemed to have the lowest contribution to the Tile are moved to finer Tiles of the T2DModel hierarchy until the Tile is once again within its size limit. In the event that the 2DModel-LOD is larger than *T2DModelFileSize*, the 2DModel-LOD can be moved to the 4 finer Tiles of the T2DModel hierarchy and clipped against the Tile boundaries as illustrated in Figure A-27: Handling Tile-LOD Overflows within the T2DModel Dataset Hierarchy. *This approach ensures that the modeled content is accessible in chunks that are bounded; this is critical to the effective allocation and management of memory in the client-devices as well as improving client-device performance and determinism.*

NOTE	The CDB Specification defines the value of T2DModelFileSize to 4 MB
NOTE	The Significant Size of a 2DModel-LOD determines where it is nominally inserted into the T2DModel LOD hierarchy. In this nominal case, each Tile-LOD of the T2DModel Dataset holds a group of 2DModel-LODs that have similar Significant Sizes. This enables the client-devices to determine the range at which the T2DModel-LOD can be optimally blended into the scene so that the model falls within a specified angular error criterion.

The bounding criterion of T2DModel Tiles can lead to LOD migration, thus breaking the relationship between the Significant Size of a 2DModel-LOD and the nominal CDB LOD it belongs to. As a result, client-devices can no longer guarantee the range at which the 2DModel-LOD will be blended into the scene. In effect, each time the 2DModel-LOD is migrated by one LOD, the client-device will likely shorten the range at which it is blended into the scene by a factor of 2, leading to potentially distracting artifacts. The severity of the artifacts is proportional to the amount of content that has migrated to finer LODs and to the number of LODs by which the content has moved.

While the CDB Standard allows the migration of 2DModel-LODs to finer LODs when Tile-LODs overflows are encountered, it is understood that this may lead to rendering artifacts that might be considered unsatisfactory. *Consequently, it is strongly recommended that tools (that generate the CDB hierarchy) be designed to optionally disallow the migration of T2DModel-LODs to finer LODs upon overflows, and instead flag the overflow condition and then abort.* Upon such cases, modelers can then re-assess which T2DModels should be discarded or remodeled in order to simultaneously satisfy the CDB bounding criteria and the application requirements.

Each of the Tile-LODs of the T2DModel Dataset is nominally configured as exchange-LODs (aka substitution-LODs) as defined in chapter 6.

The exchange-LOD mechanism assumes that client-devices gradually substitute a coarser Tile-LOD with a four finer Tile-LODs.

While this exchange-LOD mechanism is simple, it can lead to inefficiencies when extremely fine features cause the T2DModel Dataset hierarchy to be extended by several LODs. Consider the case of 13m road lines overlaid with 6 cm stripe lines. As we will see in the following section, insertion of the **Stripe** line would nominally occur at LOD=7 of the T2DModel hierarchy while the **Road** line would occur at LOD=-1. The Stripe lines force the T2DModel Dataset hierarchy to be extended (and clipped) to 8 additional LODs. In effect, the Road lines are repeated [8] in LODs 0 through 7 leading to important storage inefficiencies and greater computational burden by the client-devices.

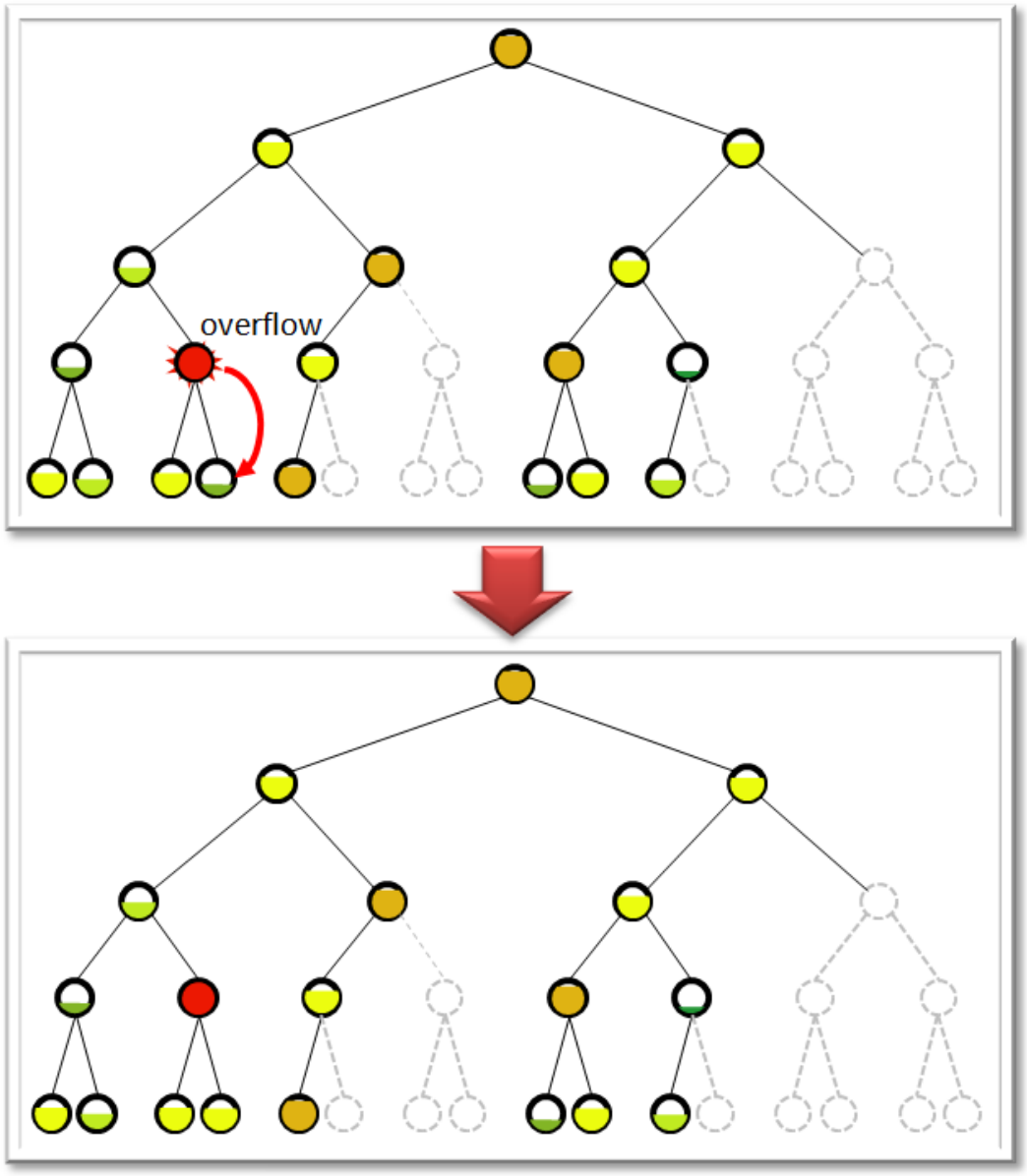


Figure A-27: Handling Tile-LOD Overflows within the T2DModel Dataset Hierarchy

In order to resolve this use-case, the T2DModel Dataset is post-processed and subjected to a “compaction” process, starting from the finest LOD (e.g. LOD_{max}) and progressing to the coarser levels. The compaction process takes the content of the Tile-LODs located at LOD_{max} and packs them

as an additive LODs of the parent Tile-LOD at (LOD_{max} – 1) of the parent Tile-LOD. The process is recursively applied to the coarser LODs until the parent LOD is packed to capacity. *This approach ensures that the modeled content is accessible in similarly-sized chunks of processing; this provides the means to improve internal parallelism and pipelining (ie. improves client-device determinism).* The result is a LOD hierarchy which is less deep, and with content which is more uniformly distributed; both of these characteristics improve runtime performance and determinism.

The T2DModel LOD structure is continuous i.e. there is no gap in the LOD hierarchy. This means that once a 2DModel-LOD is inserted into a finer level of the T2DModel hierarchy, the same 2DModel-LOD is propagated to coarser LODs until a coarser 2DModel-LOD is available.

Note that some client-devices may be sensitive to the precision of clipped vertices; some client-devices may demand that the clipped vertices be shared at the tile boundary between two tiles of the same LODs. This can be done as follows.

- The X coordinate (longitude) of clipped vertices along the top or bottom edges of the tile can be used to uniquely identify the matching coordinate in an adjacent tile.
- The Y coordinate (latitude) of clipped vertices along the right or left edges of the tiles can be used to uniquely identify the matching coordinate in an adjacent tile.

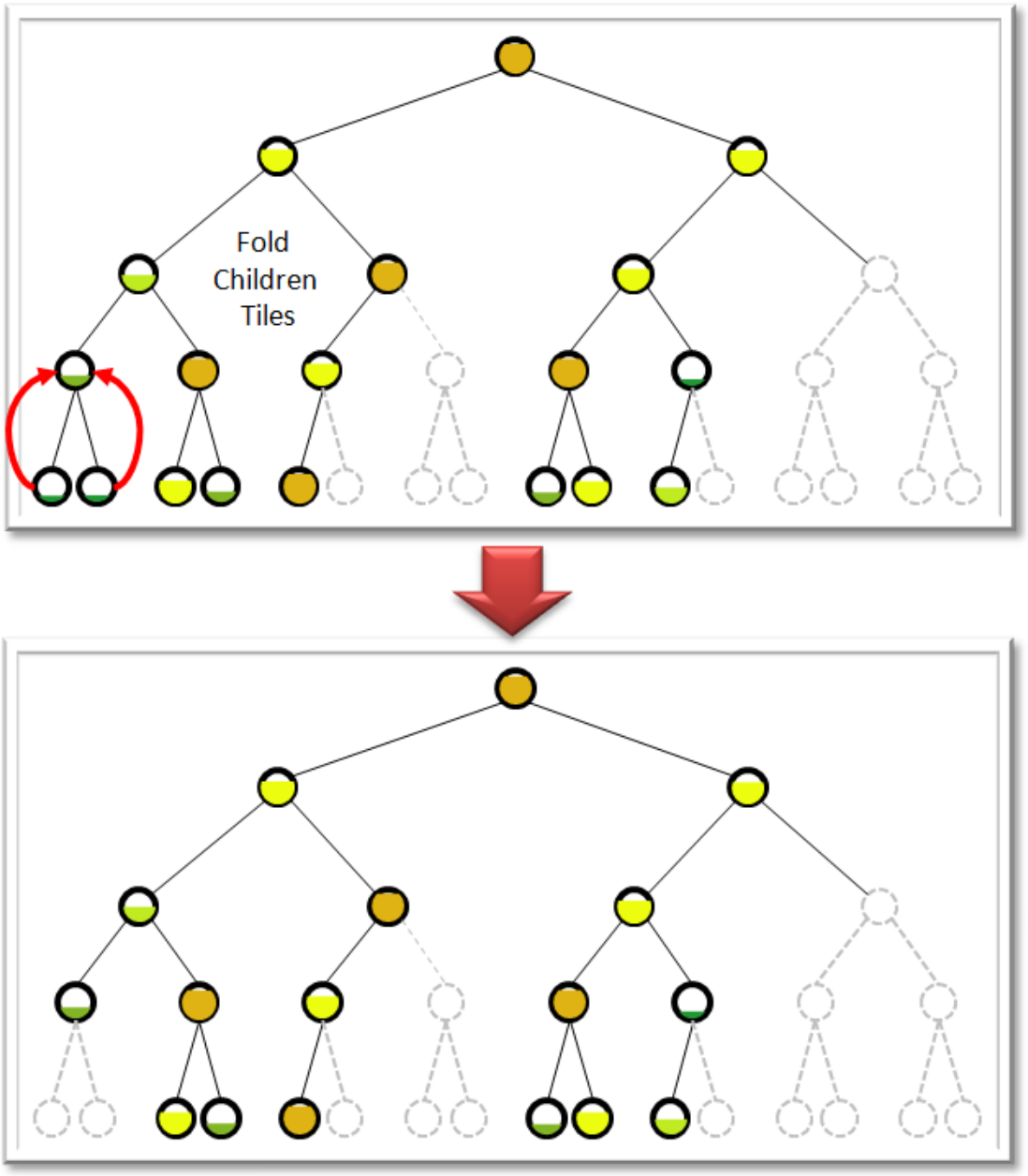


Figure A-28: Compacting the T2DModels Dataset Hierarchy

CDB LOD versus T2DModel Significant Size

Section 6.8.3 of the CDB Standard Volume 6: OGC CDB Rules for Encoding Data using OpenFlight provides a set of guidelines to establish the values for Significant Size SS_c and SS_{LOD} for T2D Models (for both lines and polygons).

Table 3 32: T2DModel LOD versus Significant Size, shows us the relationship between SS_c and SS_{LOD}. They are offset by 3 LODs. The implication of this statement is in the case of a model with two LOD, the finer 2DModel-LOD *must* have sufficient detail to justify its existence.

NOTE	Each of the 2DModel-LODs of a 2DModel *must* differ by at least one CDB LOD. Some 2DModel-LODs will be discarded if this relationship is not respected.

Consider for example a 12m line road feature with two modeled representations. The nominal CDB LOD for the coarsest 2DModel-LOD is LOD=3 in accordance to the table below. The Significant Size

of the finer 2DModel-LOD is obtained by “walking” around its outline; we determine that the largest value of *d* for successive vertex triplets is 3m, hence SS_{LOD} = 3m. Table A-9: T2DModel LOD versus Significant Size, tells us that the 2DModel-LOD should also be nominally inserted at CDB LOD = 3. Since both 2DModel-LODs have the same nominal CDB LOD, only one of them is retained (preferably the more detailed of the two).

Table A-9: T2DModel LOD versus Significant Size

T2DModel CDB Level	Significant Size SS~c ~(Coarsest Model-LOD)	Significant Size SS_{LOD} (Other Model-LODs) OTHER Interp. Max Error with respect to finest	Tile-LOD Size
-10	56 km < SS < 110 km	SS < 14 km	110 km
-9	28 km < SS ≤ 56 km	SS < 6.9 km	110 km
-8	14 km < SS ≤ 28 km	SS < 3.5 km	110 km
-7	6.9 km < SS ≤ 14 km	SS < 1.7 km	110 km
-6	3.4 km < SS ≤ 6.9 km	SS < 870 m	110 km
-5	1.7 km < SS ≤ 3.4 km	SS < 430 m	110 km
-4	860 m < SS ≤ 1.7 km	SS < 220 m	110 km
-3	430 m < SS ≤ 860 m	SS < 110 m	110 km
-2	220 m < SS ≤ 430 m	SS < 54 m	56 km
-1	110 m < SS ≤ 220 m	SS < 27 m	28 km
0	54 m < SS ≤ 110 m	SS < 13 m	14 km
1	27 m < SS ≤ 54 m	SS < 6.8 m	6.9 km
2	13 m < SS ≤ 27 m	SS < 3.4 m	3.4 km
3	6.7 m < SS ≤ 13 m	SS < 1.7 m	1.7 km
4	3.4 m < SS ≤ 6.7 m	SS < 840 mm	860 m
5	1.7 m < SS ≤ 3.4 m	SS < 420 mm	430 m

T2DModel CDB Level	Significant Size SS~c ~(Coarsest Model-LOD)	Significant Size SS_{LOD} (Other Model-LODs) OTHER Interp. Max Error with respect to finest	Tile-LOD Size
6	840 mm < SS ≤ 1.7 m	SS < 210 mm	220 m
7	420 mm < SS ≤ 840 mm	SS < 110 mm	110 m
8	210 mm < SS ≤ 420 mm	SS < 52 mm	54 m
9	110 mm < SS ≤ 210 mm	SS < 26 mm	27 m
10	52 mm < SS ≤ 110 mm	SS < 13 mm	13 m
11	26 mm < SS ≤ 52 mm	SS < 6.6 mm	6.7 m
12	13 mm < SS ≤ 26 mm	SS < 3.3 mm	3.4 m
13	6.7 mm < SS ≤ 13 mm	SS < 1.6 mm	1.7 m
14	3.4 mm < SS ≤ 6.7 mm	SS < 820 um	840 mm
15	1.7 mm < SS ≤ 3.4 mm	SS < 410 um	420 mm
16	820 um < SS ≤ 1.7 mm	SS < 210 um	210 mm
17	410 um < SS ≤ 820 um	SS < 100 um	110 mm
18	210 um < SS ≤ 410 um	SS < 51 um	52 mm
19	110 um < SS ≤ 210 um	SS < 26 um	26 mm
20	52 um < SS ≤ 110 um	SS < 13 um	13 mm
21	26 um < SS ≤ 52 um	SS < 6.7 um	6.7 mm
22	13 um < SS ≤ 26 um	SS < 3.4 um	3.4 mm
23	SS ≤ 13 um	SS < 1.7 um	1.7 mm

Rules Governing T2DModel LOD Hierarchy

Here is a summary of the rules required by the standard in order to ensure deterministic operation

from client-devices.

1. Each feature may have multiple modeled representations at progressively coarser levels of detail. Each of the modeled representations is referred to as a 2DModel-LOD. In absence of pre-modeled coarser LOD representations, the tools may automatically generate coarser modeled levels-of-detail.
2. A 2DModel-LOD consists of a group of polygons that represent a feature at a specific level-of-detail; this group of polygons shares a common Feature Attribute Code, a Feature Sub-Code (FSC), a Model Name (MODL) and 2DModel-LOD's Significant Size SS'_{LOD}.
3. Each 2DModel has a distinct Significant Size value *SS'* based on its dimensions. In turn, each of the 2DModel-LODs of a 2DModel has a distinct Significant Size value SS'_{LOD} based on its modeled accuracy.
4. Insertion of a 2DModel-LOD into the T2DModel Dataset hierarchy proceeds as follows. Starting with LOD_{max} (LOD_{max} is a variable set by the user that sets the maximum depth of the LOD hierarchy) and progressing to coarser LODs…
    1. For each Tile-LOD, create a Model_List that is constructed from the 2DModel-LODs that straddle the Tile-LOD.
        i. If the 2DModel-LOD is not the coarsest LOD and its Significant Size is in accordance to Table A-9: T2DModel LOD versus Significant Size, then iteratively simplify the 2DModel-LOD (iterate until its Significant Size is no longer in accordance to Table A-9: T2DModel LOD versus Significant Size and keep results of previous iteration) and add it to the Tile-LOD. Only the coarser 2DModel-LODs of this 2DModel are available for future insertion into the T2DModel hierarchy.
        ii. If the 2DModel-LOD is the coarsest LOD of the 2DModel and its Significant Size is in accordance to Table A-9: T2DModel LOD versus Significant Size, insert it at this LOD of the hierarchy. If the 2DModel-LOD matches the Tile-LOD, remove it from the list for the processing of the coarser Tile-LOD.
    2. If the Model_List is less than *T2DModelFileSize*, no further processing is required.
    3. The Model_List of each Tile-LOD is sorted in decreasing order of Diff, where Diff is the difference between the Significant Size *SS* of the Model and the Significant Size as specified in Table 3.
    4. If the Model_List is greater than *T2DModelFileSize*, then (starting with the first entry in the sorted Model_List), Models are simplified one-by-one until the size of the Model_List is less than *T2DModelFileSize*. When a simplification occurs, the Model_List is re-sorted using the Diff value.
    5. If a) the Model_List is deemed non-reducible and b) the Model_List is still greater than *T2DModelFileSize* …
        i. If LOD < LODmax, then…

        (1) a Temp_Model_List is created and initialized with the contents of the Model_list. Starting from the end of the Model_List, Models are removed one-by-one from the Model_list (starting with the first Model in the Model_List) and are copied into the Temp_Model_List until the Model_List reaches *T2DModelFileSize*.

(2) The Temp_Model_List is merged to the children Tile-LODs and the children are re-processed using steps 4a to 4e. The process is iterative, i.e., the “overflow” is propagated into the finer LODs of the T2DModel hierarchy.

ii. Else…

(1) Models are removed one-by-one, starting with the first Model in the Model_List, until the Model_List is less than *T2DModelFileSize*. The corresponding T2DModels are removed from the CDB and a warning is issued stating that content was removed.

NOTE The algorithm preserves the highest available modeled content while ensuring that the runtime constraint file size limits are respected. While the CDB data model allows for infinitely-sized 2DModel-LODs, a client-device may refuse to render the 2DModel-LOD if it has insufficient memory to load all of the OpenFlight files that make-up the 2DModel-LOD.

5. Each T2DModel Tile-LOD is subject to an OpenFlight file size limit of *T2DModelFileSize*, i.e., several OpenFlight files, each within the *T2DModelFileSize* file size limit, can be used to represent a very complex T2DModel Tile-LOD. Each of T2DModel-LODs of an T2DModel Tile-LOD share the same T2DModel-LOD Identifier (see rule 2)
6. Each Tile-LOD is subject to a file size limit of *T2DModelFileSize*.
7. All of the 2DModel-LODs in a T2DModel Tile-LOD are nominally exchange-LODs (see exception in next rule).
8. The depth of the T2DModel LOD hierarchy should be reduced by folding-in the Tile-Models_List of finer Tile-LODs as an additive LOD to the Tile-Model_List of a coarser Tile-LOD. Failure to perform this “compaction step” may result in significantly deeper T2DModel LOD hierarchy when the finest 2DModel-LODs consist of small details (e.g., thin stripes and markings on roads), and reduce the paging performance of client-devices.
9. The finer modeled representation of a T2DModel (i.e., a 2DModel-LOD with a smaller Significant Size) always appears in finer LODs of the Tile-LOD hierarchy than a coarser 2DModel-LOD.
10. A Tile-LOD cannot contain more than one 2DModel-LOD of the same T2DModel.
11. All T2DModels are clipped against the Tile-LOD boundaries.
12. Gaps in the LOD file hierarchy of the T2DModel Dataset are not permitted. This may result in Tile-LODs that are empty (e.g., without any T2DModels). The presence of an empty Tile-LOD file indicates the availability of content in T2DModel files located in finer LODs of the T2DModel hierarchy.

6.13. Guideline: Examples of Vector Dataset Usages (Was A.20)

Formerly Annex A.20 in the OGC CDB Best Practice, Volume 2.

6.13.1. Linear Feature Radar Simulation Example

The following diagram represents a typical usage of a linear model in a CDB data store for a typical radar client-device.

The radar application first extracts the line features from the CDB data stores and constructs an object. The constructed object contains the necessary information for the radar to compute the equivalent radar image using the radar cross-section (RCS) of the line features with material attributes and directivity, etc.

NOTE	With the introduction of version 3.2 of the CDB Specification (prior to OGC submission), it is recommended that the terrain-conformal features be modeled using T2DModels and that radar client-devices use this modeled representation instead of the vector line and polygon features.

Figure A-31: Example of Line Features, illustrates three line features stored in the tile in in a vector data set. The junction nodes of each line feature represents the start and end junctions of the line feature. In this example, there is only one chain per line feature.

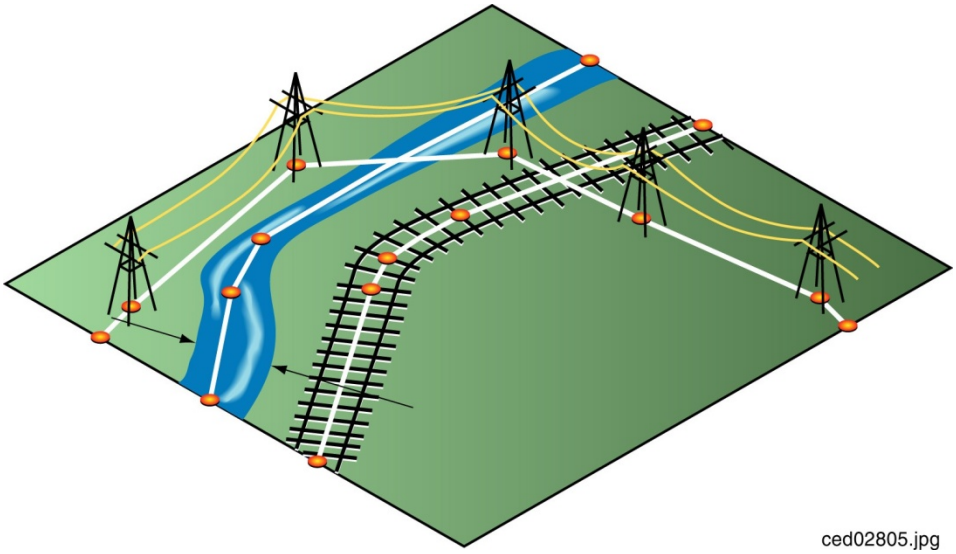


Figure A-31: Example of Line Features

The radar uses the position of the lineal coordinates to construct a line representation of the radar image. It extracts the line feature information from the chains to construct an internal local representation. The necessary information needed by radar is…

1. **Network Datasets:**
   The datasets along with the Feature Attribution Code indicate if the feature is a road, a highway, or river for example. In the above illustration, we have a river, a powerline and a railway. The CDB Standard represents this in the *.dbf file of the Shapefile representation.
2. **Composite Material IndeX (CMIX):**
   The Composite Material IndeX attribute points to the Composite Material Table and provides the Radar the types of Base Materials that the feature is made of. This information is used, in addition to the geometry of the line feature or a generic RCS, to provide a radar signature of the target, which is proportional to the reflection value of the various materials. The intensity of the radar image represents the interaction of the simulated Radar Beam with the features in the

synthetic environment. Each line contains a reference to a composite material which in turn is mapped to a reflectivity factor value in the radar simulation.

3. **Width (WGP):**
   The width of the line features is also taken into consideration. This information is part of the vector data used to construct a 2D radar image of the terrain. The width information is encoded as an attribute of the line feature.

4. **Height (HGT):**
   The height of the line feature is used to indicate the height of each point/line with respect to the terrain height.

NOTE	The height value is a delta height above the terrain and is only provided for objects that require it such as the powerlines or the train tracks in this example. The height property is especially valuable to radar client-devices because erect objects in the data store produce significant returns and occultation areas in the displayed radar image. The height property can be assigned to the train tracks, long fences and the powerlines each with average altitudes.

5. **Position:**
   Currently, this information is contained in the line *.shp files [9]. The x and y coordinate of each point is extracted from those objects.

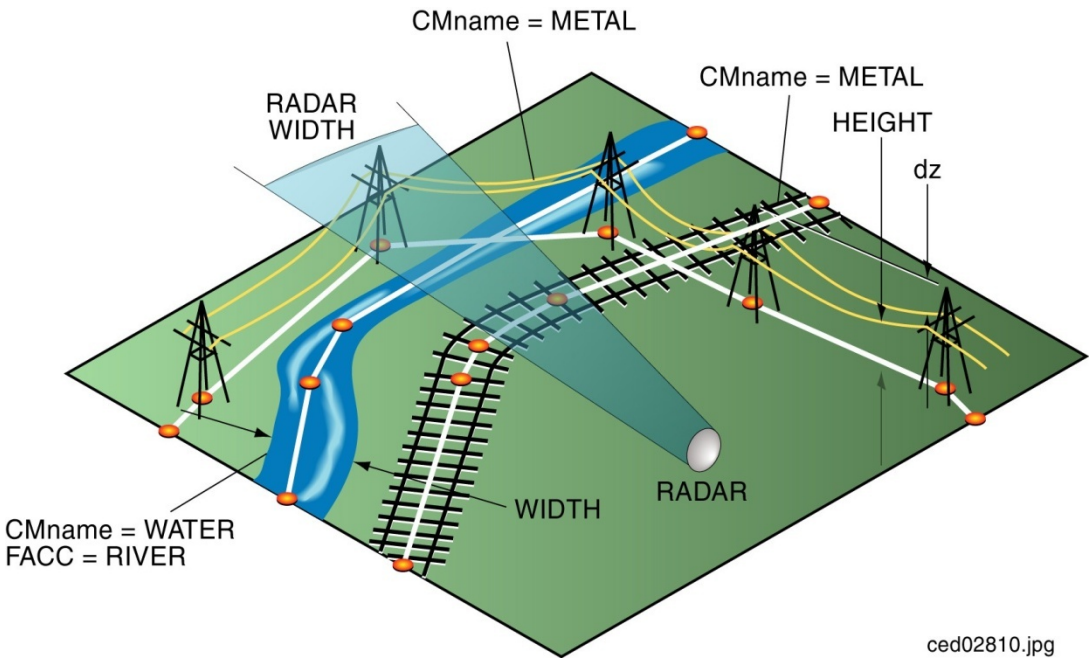


Figure A-32: Radar Beam Simulation

The radar then uses a beam simulation to process the above information and construct an image representing the content of each small beam sections. The intersection of the beam pie slice is compared with the line feature's position and converted into an image whose intensity is based on the computed RCS of the line features. As mentioned above, the RCS value (which is modeled internally in the radar simulation) takes into account the properties (which are derived from the attributes) of each line feature.

6.13.2. Road Following Example

Figure A-33: Network Dataset Used to Describe a Navigable Network, illustrates how line features can be used to describe a navigable network; the example could represent a network of roads. First, the application reads a vector line file describing the chains, then the vector point file describing the junction nodes. For each junction nodes the application makes up a list of attached chains ending up with a network as illustrated in Figure A-33: Network Dataset Used to Describe a Navigable Network, where there are six chains (labeled in this example as CSLID1 to CSLID6) that are joined through intersection nodes (labeled in this example as CSZID1 to CSZID6). The small black dots represent points forming the segments of a chain; they are essentially used to describe the deviation from a straight line between nodes.

In Figure A-33: Network Dataset Used to Describe a Navigable Network, the green line shows an example of what a shortest path algorithm could determine if asked to find the shortest route between CSZID1 and CSZID5 based on the lengths of the chains. First, the algorithm would move to CSZID2 via the chain CSLID1; when at CSZID2 it has two alternatives, either take CSLID4 or use CSLID3 and CSLID6. In our example it would have determined that the latter alternative is the shortest path; the entity would then follow the path given by the green line going through all the segments in the chains.

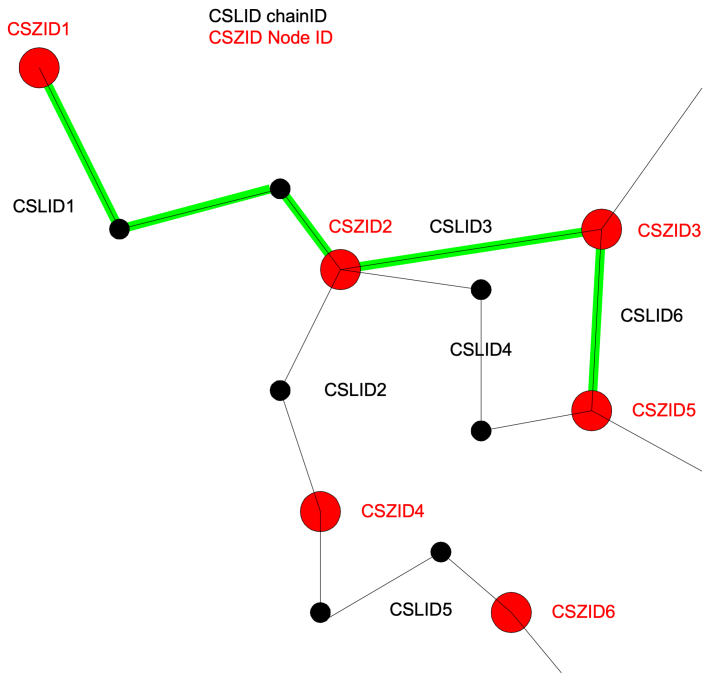


Figure A-33: Network Dataset Used to Describe a Navigable Network

6.13.3. Point Feature Radar Simulation Example

The following diagram represents a typical usage of a point feature as modeled in the CDB data store for a typical radar client-device. The radar client-device extracts the point feature from the data store using the format described in the Standard and constructs an object. The constructed object will contain the necessary information for the radar to compute the equivalent radar image using the radar cross-section (RCS) of the object over the terrain; the RCS is derived from the point features characteristics.

NOTE	The example below illustrates the use of point-feature data by a radar client-device. However, we recommend that the radar client-devices use the modeled representation of the feature rather than the feature location, type and attribution data.

In Figure A-34: Objects Represented on a Terrain Tile, a series of different objects are represented on a terrain tile. The objects are modeled as single point of zero dimensions in radar. The radar will position the different points according to their geographic position and altitude. The height data corresponds to a height of the feature with respect to the terrain.

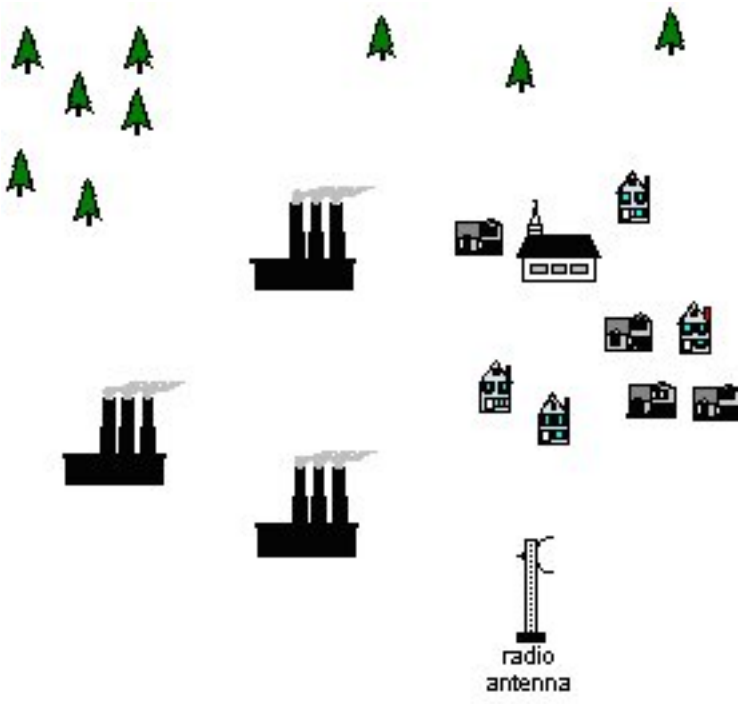


Figure A-34: Objects Represented on a Terrain Tile

The necessary information needed by radar is typically…

1. **Feature Code:** This information indicates if the feature is a tree, a pylon, or a church for example. In the above drawing this would mean a tree, an industry, a house or a radio station antenna.
2. **Composite Material IndeX (CMIX):** The Composite Material IndeX attribute points to the Composite Material Table and provides the Radar client-device with the type of material that the feature is made of. This information is used, in addition to the width of the point feature, to provide a generic RCS of the target, which is proportional to the reflection value of the various materials. The RCS is then used by radar to determine the intensity of the radar image representing the point feature, based on the aspect and grazing angles to the Radar Beam. Each point contains a reference to a Composite Material.
3. **Bounding Sphere Radius (BSR):** The radius of the point is also taken into consideration. This information is part of the vector data used to construct a 2D radar image of the terrain. The width is part of the point object attributes.
4. **Height (HGT):** Since the radar sees the terrain with a perspective angle that can be computed using the radar altitude and the feature distance, the height of objects on the terrain becomes

important to create the radar display image. This attribute of the point is used to indicate the differential height of each point with respect to the terrain height. In the example above, the trees all have the same average altitude. The other point features have different height.

5. **Position:** The object point location in the CDB data store. The x and y coordinate of each point is extracted from those objects. The position when combined with the delta heights will create a pseudo-3D point feature object.
6. **Orientation (AO1):** The radar needs the orientation of each point feature. This is needed because radar has a series of RCS tables, one for each of the Feature Identification Code. Those RCS tables give the RCS value for each incident angle of the radar beam. This angle is computed by taking into account the radar beam angle and the point feature orientation. For example, in the above drawing, the radio antenna has an orientation of 90 degrees. This means that if a radar beam comes from the right and points to the antenna at 270 degrees, the RCS value will be at maximum. The radar simulation would use a RCS table that represents the RCS with respect to the incident angle as follows:

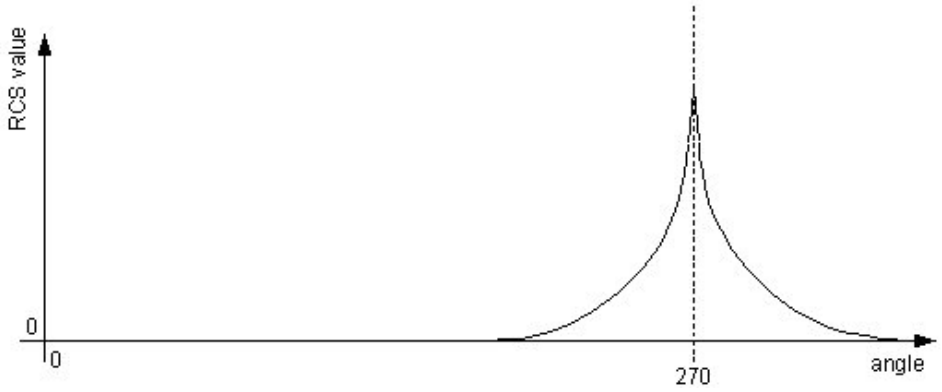


Figure A-35: Incident Angle

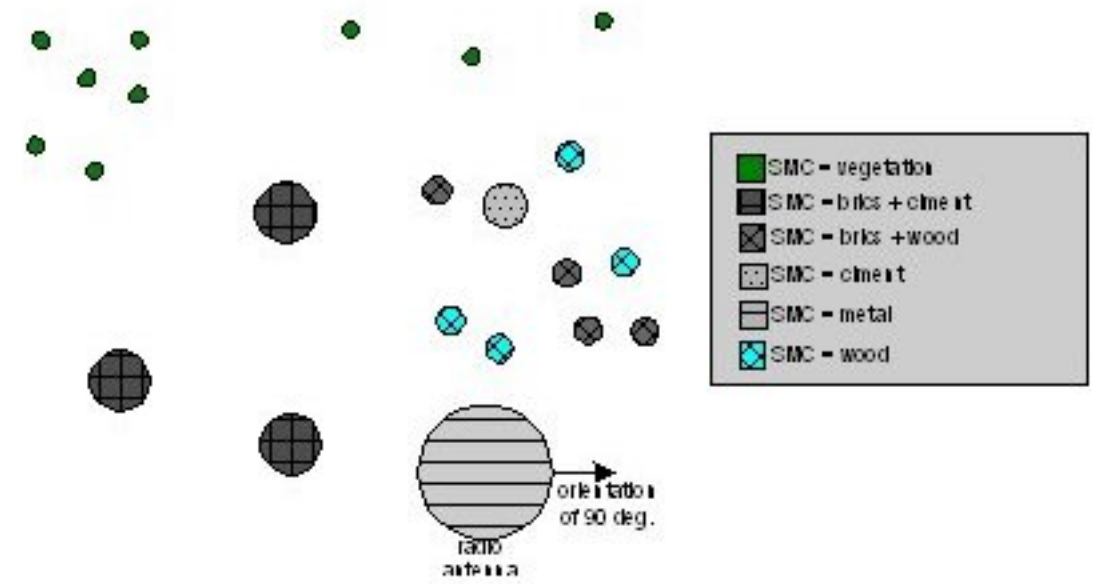


Figure A-36: Beam Simulation

The radar then uses a beam simulation to process the above information and construct an image representing the content of each small beam sections. The intersection of the beam pie slice is

compared with the point's position and converted into an image whose intensity is based on the computed RCS of the points. As mentioned above, this RCS takes into account the attributes of each point feature.

If the size of the object referred to by the point feature is much larger than a specific threshold, the simulation could in addition use the MODL field of that point feature to extract a more precise geometrical 3D model from the CDB to increase the simulation fidelity.

6.13.4. Polygon Feature Radar Simulation Example

The following diagram represents a typical usage of a polygon feature as modeled in the CDB for a radar client-device.

NOTE	With the introduction of version 3.2 of the CDB Standard (prior to OGC submission), we recommend that the terrain-conformal features be modeled using T2DModels and that radar client-devices use this modeled representation instead of the vector linear and polygon features.

Currently, in a manner similar to the line example, the Radar extracts this polygon from the data store using the ShapeFile (*.shp) file [10] and constructs a tile in its memory. The constructed tile will contain the necessary information for the radar to compute the equivalent radar image using the radar cross-section (RCS) of the represented surface polygon over the terrain intersecting the Radar beam.

In Figure A-37: Four Polygon Features Stored in the Tile, four polygon features are stored in the tile with their surface material and feature classification attributes. Each of the features points to an array of segments.

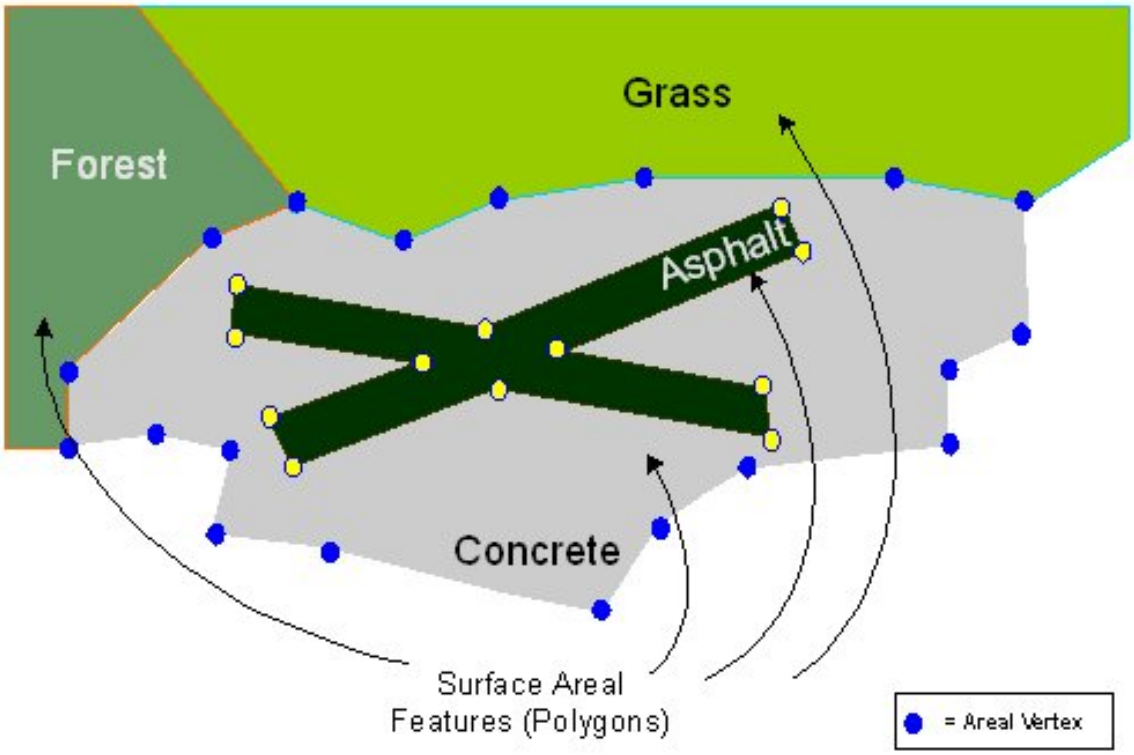


Figure A-37: Four Polygon Features Stored in the Tile

The radar simulation uses the segment coordinates to construct a polygonal representation on the radar image. It extracts the polygon feature attribute information from the segments and constructs its tile data in memory. The necessary information needed by the radar simulation is typically…

- **Feature Code:**
  This information is the identification of the surface feature. It indicates if the polygon feature represents a forest, a lake, or an airport runway for example. In the above drawing, this would translate to a forested area, a grassy area, a concrete section and a dual runway intersection.
- **Composite Material IndeX (CMIX):**
  The Composite Material IndeX Surface Material Code attribute points to the Composite Material Table and provides to the Radar with the type of material that the feature is made of. This information is used, in addition to the shape of the polygon feature, to provide a generic RCS of the simulated area. This RCS is proportional to the reflection value of the various materials constituting the polygon or the simulated texture of its components (e.g., an industrial area made up of metallic roofs). The RCS is then used by radar to determine the intensity of the radar image representing the polygon feature, based on the aspect and grazing angles to the Radar Beam. Each chain will contain a reference to a material namespace object in the CDB.
- **Height (HGT):**
  Normally, the radar simulation sees the terrain with a perspective angle that can be computed using the radar altitude and the feature distance. Because of this angle, the height of objects on the terrain becomes important to create the image that the radar “sees”. This attribute of the chain is used to indicate the average height of each polygon object. In the example above, the forested area could be elevated to roughly 25 or 30 feet to produce a forest ”canopy” which will look elevated to the radar.

The following image shows how a typical radar beam would intersect the different parts of polygon features that are part of the terrain represented previously.

Figure A-38: Radar Beam Simulation, shows the radar using a beam simulation to extract the above information and construct an image representing the content of each small beam sections (aka bins). The intersection of the beam pie slice is performed against the polygon feature polygons. Then the material of the polygon falling in the beam bin is converted into image intensity, which is relative to the computed RCS of that polygon’s material. As mentioned above, this RCS takes into account the attributes of the segment of each polygon feature.

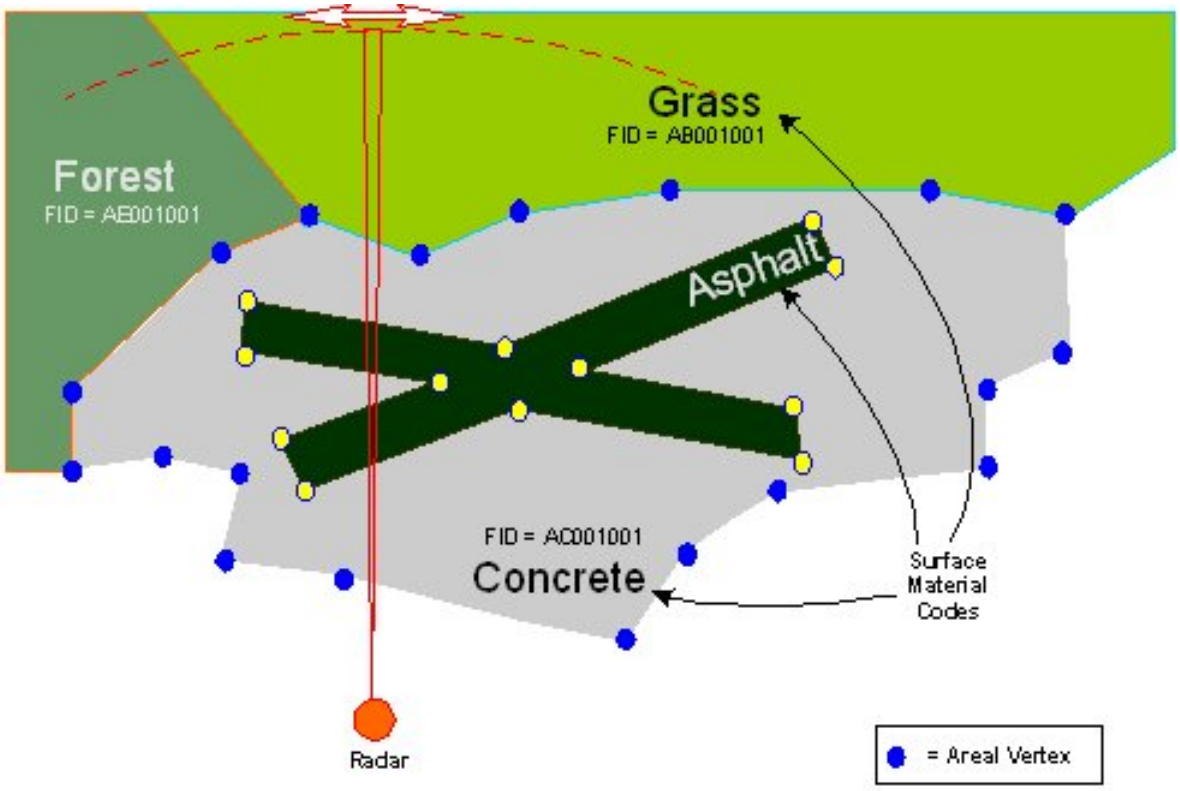


Figure A-38: Radar Beam Simulation

6.14. Guideline: Vector Priority Tile-LOD Generation (Was A-21)

This section describes how to produce the finest Tile-LOD of linear and areal vector datasets, and how to recursively generate the coarser Tile-LODs of these datasets.

There are two new terms that will be used in the lineal LOD generation process below: interior vertex, and similar feature.

An interior vertex is any vertex that is not the first or last vertex of a network lineal feature. Since the first and last vertices are usually junction vertices, modifying them changes the connections of the feature, and thus the network topology. This LOD scheme will seek to preserve network topology by only modifying interior vertices.

Similar features are connected features that describe a single conceptual linear, such as a particular highway that has nearly identical attribution. Certain attributes, like the length of the lineal (LENL), may not be identical. This LOD scheme merges such similar features when possible to attain a better simplification.

6.14.1. Creation of the Finest Tile-LOD

The finest Tile-LOD for lineal and areal vector datasets can be determined and generated using the following steps, starting with the full resolution dataset for a CDB Tile at the coarsest Tile-LOD:

1. For each Tile-LOD, if the number of vertices is greater than the vertex limit specified in Table 3-27, do the following:

1. If the current Tile-LOD is negative, go to the next finest LOD. If the current Tile-LOD is non-negative, subdivide the tile into four finer Tile-LODs.
2. For network lineal features, introduce Junction IDs at the boundaries of tiles when features are subdivided into finer Tile-LODs.
3. Repeat step 1 on each subdivided tile.

It is recommended to stop when the vertex limit is satisfied so as to limit the number of even finer Tile-LODs that would otherwise be necessary to simply satisfy the spatial significance criterion alone.

6.14.2. Network Lineal Tile-LOD Generation

This section describes how to produce coarser Tile-LODs of lineal networks from finer Tile-LODs, taking into account each vector's priority, as defined in Section 5.7.1.6.4, Network Vector Priority.

1. Begin with the set of finest Tile-LODs for this dataset determined in section A.21.1 above.
2. Create the coarser Tile-LODs:
    1. Merge up to four finer Tile-LODs by removing those junction IDs at the tile boundary that are no longer necessary, by combining those features that were split during the creation of the finest Tile-LOD.
    2. Remove any Junction IDs introduced to connect this feature to a figure point or areal feature if both of the following criteria are satisfied:
        i. The figure point or areal feature at the same LOD is no longer present because of LOD generation for those datasets.
        ii. There are two similar linear features that share this Junction ID and have nearly identical attribution (only differing in LENL, for example).
    3. Remove interior vertices on all features such that the CDB Spatial Significance criteria, described in Section 5.7.1.6.5, is satisfied.
    4. If the vertex count limit is still exceeded, do the following:
        i. Begin removing complete features in the lowest remaining priority class if and only if they are not part of a longer connection (i.e., one endpoint junction ID is not connected to any other feature including connecting to another tile).
        ii. If there are no unconnected features in the lowest remaining priority class, remove all features at that priority class.
        iii. As lower priority class features are removed, merge similar higher priority class features whose junctions were present only to topologically connect to the removed features by converting the junction to an interior vertex in the merged feature and adjusting attribution like LENL to reflect the combined features.
        iv. Repeat step d as necessary.

In the case that all features have been removed in step d, create empty Tile-LODs at this and every coarser Tile-LOD to indicate the presence of content at finer LODs of this dataset.

6.14.3. Non-Networked Lineal Tile-LOD Generation

Creating coarser Tile-LODs of non-network lineals from finer Tile-LODs is a process similar to the one for networked lineals, but without the junction ID and topology preserving steps.

1. Begin with the set of finest Tile-LODs for this dataset.
2. Create the coarser Tile-LODs:
    1. Merge up to four finer Tile-LODs by combining features that were split during the creation of the finest Tile-LOD.
    2. Remove interior vertices on all features such that the CDB LOD based spatial significance criteria from section 5.7.1.9.2 is satisfied.
    3. If the vertex limit is still exceeded, begin removing features that have the smallest significant size (as defined by section 5.7.1.9.1), until the number of vertices remaining is below the vertex count limit.

In the case that all features have been removed in step c, create empty Tile-LODs at this and every coarser Tile-LOD to indicate the presence of content at finer LODs of this dataset.

6.14.4. Areal Tile-LOD Generation

This section describes how to produce coarser Tile-LODs of areal vector data from finer Tile-LODs. For network areal data, there is a single Junction ID for the entire feature. Therefore, the network only requires the presence of the feature, rather than a specific vertex, to maintain topological completeness. As such, network and non-network areals can be treated similarly, where the important characteristics are whether a feature is present in a particular Tile-LOD, and how precisely its shape has been preserved.

This process is similar to the one for lineal vectors.

1. Begin with the set of finest Tile-LODs for this dataset.
2. Create the coarser Tile-LODs:
    1. Merge up to four finer Tile-LODs by combining features that were split during the creation of the finest Tile-LOD.
    2. Remove vertices on all areal features such that the CDB LOD based spatial significance criteria from section 5.7.1.9.2 is satisfied.
        i. For areal features that share a vertex, remove that vertex if and only if the spatial significance criteria would remove it in each individual areal vector. This will help maintain feature topology by preserving shared edges between areal features.
    3. If the vertex limit is exceeded, begin removing features that have the smallest significant size (as defined by section 5.7.1.9.1), until the number of vertices remaining is below the vertex limit.
    4. In the case that all features have been removed in step c, create empty Tile-LODs at this and every coarser Tile-LOD to indicate the presence of content at finer LODs of this dataset.

[1] See the OGC CDB Best Practice. That document is version 3.x of the CDB specification as orignally submitted to the OGC and is the basis for the formal, OGC Member approved CDB Standard and Best Practices starting at OGC Version 1.0

[2] Need correct cross reference.

[3] As of pre OGC CDB Specification 3.2, the CDB FDD is no longer provided in the documents to avoid the risk of miscorrelation between the appendix and the metadata. The FDD is now exclusively found in the Metadata folder.

[4] The two UHRB diagrams presented here come from the document entitled UHRB_2_Object_Model.pdf available on the OneSAF web site: www.onesaf.net.

[5] FalconView® is a multi-platform mapping and mission planning application developed by the Georgia Tech Research Institute for the United States Department of Defense. With a 20-year history of active development the software has become a de facto standard within the US DoD and is also used by various Federal Agencies and Allied Countries. https://www.falconview.org/trac/FalconView

[6] In this context, “appropriate” means a LOD that falls within the capabilities of the client-device.

[7] The *T2DModelFileSize* storage size limit for T2DModel Tile-LODs is critical in achieving runtime determinism.

[8] Since the nominal LOD mechanism is the exchange-LOD, and that gaps are not permitted in the LOD hierarchy

[9] Future versions of this Best Practice may include guidance on using other vector encodings such as GeoJSON, GML, or GeoPackage.

[10] Future versions of this Best Practice may include guidance on using other vector encodings such as GeoJSON, GML, or GeoPackage.

Annex A: Revision History

Date	Release	Editor	Primary clauses modified	Description
2016-11-23	1.0	C. Reed	Various	Prepare for publication.
2017-11-20	1.1	C. Reed	Various	Minor edits for version 1.1. Also added new clauses to make this document consistent with late 3.2 changes as approved by the user community.
2019-11-11	1.2	C. Reed	Various	Update for version 1.2

Annex B: Bibliography