OGC® DOCUMENT: 24-040R1

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t20-D013

Open Geospatial Consortium

OGC TESTBED 20: GIMI LESSONS LEARNED AND BEST PRACTICES REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2025-04-28
Approval Date: 2025-06-12
Publication Date: 2025-MM-DD
Editor: Joan Masó Pau, Núria Julià Selvas, Rob Smith

Note

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.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I OVERVIEW

The GIMI format is very versatile and it is based on HEIF that is structured in boxes that can contain both data and metadata. In the OGC Testbed 20 several best practices were defined and are included in this Report. A complementary approach to the `grid` that support larger images called `tili` was presented and proposed to the MPEG group for standardization. Affine transformations were proposed as a georeferencing mechanism that provides a strictly mathematical solution leaving no room for ambiguity about the axis order. The use of NaN was proposed to reduce risk of corruption caused by unchecked Sentinel's "no data" values. An in-depth analysis of "no data" options based on NaN and Sentinel was performed.

During OGC Testbed 20, several implementations were developed to support generating GeoHEIF output from raster data sources as well as to test and benchmark the reading of GeoHEIF files in a Java environment.

In addition, a motion imagery use cases for GIMI video design was analyzed. The prevalence of high-quality cameras in location-aware devices such as smartphones has made geotagged video affordable for consumers. Content creators are driving demand for formats like EXIF and WebVMT to share their data online in an accessible form. Proliferation of MISB Class 3 Imagery offers new resources from multiple sources with which to address ISR use cases through data aggregation and GEOINT analysis.

Despite the title of this document suggest that this document contain Best Practices, this document is an Engineering Report containing lessons learned and it is NOT an OGC Best Practice. This document does NOT represent the official position of the OGC but the opinions of the contributions only.

II EXECUTIVE SUMMARY

WARNING

Despite the title of this document suggest that this document contain Best Practices, this document is an Engineering Report containing lessons learned and it is NOT an OGC Best Practice. This document does NOT represent the official position of the OGC but the opinions of the contributors only.

This OGC Testbed 20 Report documents the work performed and the findings of the GEOINT Imagery Media for ISR (GIMI) Task. The primary goal of the OGC Testbed 20 GIMI task was to develop, implement, and validate content that will form the basis of a future GIMI Standard, addressing specific standardization issues within the context of payload optimization and metadata management. Also, the development of the OGC Testbed 20 GIMI Specification necessitates a reliable method to evaluate the performance and quality impacts of various design choices. This evaluation is critical to determine the most efficient design options

for the GIMI implementation, especially when comparing it to Cloud Optimized GeoTIFF (COG). Therefore, part of the GIMI task was to develop those benchmarks and propose their incorporation into the OGC Compliance and Interoperability Test Environment (CITE).

The discussions and lessons learned of the GIMI Task are summarized in this report. This Report presents a set of recommendations for the HEIF format as well as some of the experimentation performed during OGC Testbed 20.

Currently, the HEIF format includes an item type value called 'grid' that defines a derived image item whose reconstructed image is formed from one or more input images in a given grid order within a larger canvas. In practices this is a mechanism to tile images. However, this mechanism was designed for purposes other than the easy image pan and zoom application. The `grid` format may be acceptable for a small number of tiles but cannot support more than 256×256 tiles. This OGC Testbed 20 report proposes a complementary approach to support larger images that require more tiles. The item type `tili` is capable of supporting arbitrarily large resolutions, supporting tiled images in which some tiles are blank and not covered with image data, and supports saving tiles in an arbitrary order to permit gradually growing files. Multi-resolution pyramids with a mixture of `grid`, `tili`, and `unci` images is recommended as a best practice for partial compatibility with existing software without `tili` support.

This report includes considerations on how to structure the content in `mdat` box in image sequences when each frame requires specific metadata.

GeoHEIF (OGC 24-038), GeoTIFF (OGC 19-008r4), WorldFile and GML in JPEG 2000 (OGC 08-085r8) (among others) use affine transformations for mapping pixel coordinates to real world coordinates in georeferenced images. In the opinion of one of the authors of this document (Geomatys), affine transformations, when used correctly, are a strictly mathematical solution leaving no room for ambiguity in the axis order. The affine transformations recommendation discussed in this Report includes the consideration for image orientation and the CRS orientation and includes considerations for *"pixel center/corner"*. Considerations for the support in GDAL implementations are included.

NOTE 1: Most of the content in this section is not specific from the GeoHEIF GIMI format and is of general interest to avoid confusion in the axes order in geospatial grid coverages.

In most GIS raster formats, there is a way to flag some pixel values as missing. A value may be missing because the pixel was masked by a cloud, or because the pixel is located on a land while the raster was aimed to contain oceanographic data, or because the remote sensor did not pass over that area, *etc.* Missing values are typically represented by *Sentinel values*, such as -9999. In the opinion of one of the authors (Geomatys), in the case where the raster data are stored using the binary representation of floating point numbers, as defined by the IEEE 754 standard, NaN (Not-A-Number) values can be used instead of Sentinel's "no data" values. The IEEE 754 standard specifies behaviors that make NaNs not only convenient, but also (most important) safer and it is recommended as a best practice. The use of NaN values reduces the risk that corruption caused by unchecked Sentinel's "no data" values.

NOTE 2: Most of the content of the mentioned section is not specific from the GeoHEIF GIMI format and is of general interest to deal with no-data values in geospatial grid coverages. It expresses the opinion of a single author and its application can have risks in old version of C compilers without complete support to NaN.

During OGC Testbed 20, Ecere implemented support for GeoHEIF output in its GNOSIS Map Server implementations of OGC API — Maps and OGC API — Tiles with the help of the libheif library. Support for georeferencing information was added, in line with the latest draft GIMI specification developed during this initiative (D010 / OGC 24-038). The ability to encode image as tiles as well as optionally generating overviews was also developed, enabling support for multi-resolution tile pyramids in GeoHEIF outputs.

Geomatys tested and benchmarked the reading of GeoHEIF files in a Java environment using two different readers: A pure-Java implementation, and a binding to the C/C++ GDAL native library. The reader and the binding were developed in the Apache SIS project. In addition, George Mason University developed a Python-based converter to transform Earth observation datasets to Cloud Optimized GeoTIFF (COG) files.

In addition, Away Team analyzed motion imagery use cases for intelligence, surveillance and reconnaissance (ISR), and identified considerations for GIMI video design. The Motion Imagery Standards Board (MISB) Class 3 Imagery could significantly enhance the value of geospatial intelligence (GEOINT) data. The video use cases studied include aggregating geotagged video with AIS data for a maritime vessel, aligning dynamic camera orientation with a map in a web browser, monitoring roadside litter using dash cam footage from a vehicle fleet and identifying access time benchmarks for searching video metadata.

III KEYWORDS

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

hackathon, application-to-the-cloud, testbed, docker, web service, tiles

IV CONTRIBUTORS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Núria Julià Selvas	University Autonomous of Barcelona, Spain / CREAF	Editor
Joan Masó Pau	University Autonomous of Barcelona, Spain / CREAF	Editor
Rob Smith	Away Team	Editor

NAME	ORGANIZATION	ROLE
Dirk Farin	Dirk Farin Algorithmic Research e K	Contributor
Martin Desruisseaux	Geomatys	Contributor
Nicholas Kellerman	Helyx	Contributor
Eugene YU	George Mason University (GMU)	Contributor
Patrick Dion	Ecere	Contributor
Jérôme Jacovella-St-Louis	Ecere	Contributor

V FUTURE OUTLOOK

While the HEIF approach to tiles in HEIF was proposed, there was no time to carefully benchmark COHEIF (Cloud Optimized GeoHEIF) and contrast it with COG, both in the local drive and over the Internet using HTTP range use cases. This could be an OGC Testbed 21 activity.

While the affine transformations for georeferencing were implemented and tested, the approaches proposed to describe attributes (e.g., band names) in the HEIF file were not tested. Further, comparisons between the binary approach and an embedded JSON approaches should be performed.

The Annex E and Annex F annexes could be re-elaborated and released as independent documents in the future, as they are of a general interested and not limited to the GeoHEIF GIMI format.

Accessibility is key to data enrichment. Open formats could benefit Law Enforcement and Public Safety organizations by enabling aggregation with public data from consumer devices through crowdsourcing. GEOINT analysis could be revolutionized for these bodies using find-by-location queries to online search engines. It is recommended that the analysis of dash camera footage for litter monitoring should be proposed for OGC Testbed 21 to highlight how public safety can be improved using geospatial technology.

VI VALUE PROPOSITION

GIMI offers an opportunity for still imagery and model outputs of continuous variables. However, there is a need to embed the georeference information in the files to overlay and combine them with other GIS information and extend GIMI applicability. The description of the dimensions and cell properties based on URI enables for easily combining sensor data with imagery to scientific studies in big data use cases.

GIMI also provides scope to align modern video metadata design with a new generation of location-aware cameras including dash cams, drones and body-worn cameras. Mapping in-band SMPTE KLVs to out-of-band JSON-based formats such as WebVMT broadens accessibility to timed metadata in a form that is agnostic of video encoding. Data can be enriched by exporting MISB metadata to an out-of-band format that integrates seamlessly with web technologies. Examples include improved cognition of video by matching map data with camera geo-alignment using MapLibre and WebVMT, and GEOINT analysis to improve vessel tracking guided by accurate aggregation of maritime AIS and geotagged video with nautical chart data in a web browser.

1 INTRODUCTION

1 INTRODUCTION

The Next Generation ISR Imagery Standards (NGIIS) initiative is a forward-looking effort aimed at fundamentally transforming the standards for Intelligence, Surveillance, and Reconnaissance (ISR) imagery. Part of this initiative is the development of the GEOINT Imagery Media for ISR (GIMI), encapsulated within NGA Standard 0076 (NGA.STND.0076_1.0_GIMI). Pronounced "gimmie," GIMI is designed to revolutionize how still and motion imagery is managed, integrated, and utilized across various defense and intelligence platforms.

The GIMI format is very versatile as it is based in HEIF that is structured in boxes that can contain both data and metadata. In the OGC Testbed 20 some best practices on how to combine this boxes and propositions for new boxes were made. During the process some clarifications on the general use of the affine transformations as georeference mechanism was needed and exposed in the annex Annex E of this Report. The use of IEEE 754 standard, NaN (Not-A-Number) were exposed in the annex Annex F. While these two annexes were produced in the context of HEIF file they are a general applicability and the OGC may considered releasing them as independent Best Practices Reports.

NOTE: Most of the content of the mentioned two last sections section is not specific from the GeoHEIF GIMI format and is of general interest

In addition the GIMI format can be used to encoded motion imagery. Some considerations on how to address ISR use cases through data aggregation and GEOINT analysis was provided.

1.1. Aims

The aim of of the GIMI OGC Testbed 20 Task is to identify and document best practices on how to combine HEIF boxes and to propose for new boxes if necessary. Specifically, outline recommended content for future GIMI standards, addressing issues like prototype streaming of large images.

There is also the need for describe some implementations to read and present GeoHEIF files and extend the current open source libraries for HEIF and GIMI.

Full utilization of MISB Class 3 Imagery requires correct alignment of the GIMI model with other formats such as Society of Motion Picture and Television Engineers (SMPTE) Key Length Values (KLVs) and JSON-based WebVMT to ensure seamless interoperability. Identifying real-world use cases that could benefit from GEOINT analysis of geotagged video highlights how data can be enriched to address operational ISR requirements. Focus areas include data aggregation from multiple sources and techniques to improve cognition of still and motion imagery.

1.2. Objectives

The objectives of this Report are to:

- Identify best practices on how to combine HEIF boxes and propose a new box for tiles;
- Provide best practices on the use of the affine transformations as a georeference mechanism and to use NaN as nodata value;
- Describe some implementations to read, write, and present GeoHEIF files;
- Identify ISR use cases which can benefit from GEOINT analysis of geotagged video synchronized and aggregated with data from other sources; and
- Provide considerations on how to storage the information in `mdat` box for image sequence (frames and metadata).

2 TOPICS

2 TOPICS

2.1. Tiling for very high resolution images

When working with high-resolution images, it may not be feasible to keep the whole image in memory. Instead, accessing the image in units of small individual tiles is needed. When reading a tile, only the parts that are needed for decoding a tile must be read from the file and decoded. Similarly, when writing, each tile can be encoded separately and added to the image file.

The HEIF `grid` image format saves a tiled image as a collection of small images. Each tile is stored in the HEIF file as a separate image, which are then combined into a large grid image. However, the HEIF `grid` image item format is limited to images with less than 256×256 tiles. Moreover, storing tiled images as `grid` items has significant overhead because metadata has to be stored for each tile image.

To support larger images, it was decided during this OGC Testbed 20 to propose and implement a new image item type `tili` as an alternative to `grid`. This approach is discussed in detail in Annex B, by Dirk Farin Algorithmic Research e K.

The new `tili` image type covers the following characteristics:

- Support for arbitrarily large resolutions (over 1G x 1G pixels in a single image);
- Storing 2D images in a tiled file layout, supporting random access decoding of any chosen tile with a single byte range access to the file;
- Enable streaming the image content over the Internet with small initial setup delays, resulting in much less overhead than `grid`;
- Support tiled images in which some tiles are blank and not covered with image data;
- Saving tiles in arbitrary order to allow gradually growing files;
- Interleaved storage of multiple tiled images, e.g., for multi-resolution pyramids where storage of the lower resolution layer is interleaved with the higher-resolution layers; and
- Ability to build multi-resolution pyramids with a mixture of `grid`, `tili`, and `unci` images to have partial compatibility to software without `tili` support.

Additionally, a desirable feature is to extend the tiling scheme to data arranged as n-dimensional hyperrectangles.

The content for a `tili` image item consists of the concatenated compressed data for each image tile, and a table of pointers to the start of each tile in this data stream. The table is prepended to the coded tile data. Storing the table together with the compressed tile data in the `mdat` box

enables loading both the image tiles and also the pointers to the tile on demand (e.g., over a network connection).

The `tili` specification was submitted to MPEG for consideration to be included in the core HEIF standard. The proposal m70021:Tili proposal was written together with Joe Stufflebeam.

2.2. Georeferencing by affine transformations

GeoHEIF, GeoTIFF, WorldFile and GML in JPEG 2000 (among others) use affine transformations for mapping pixel coordinates to real world coordinates in georectified images. This approach is significantly different from another frequently used approach based on bounding boxes. In the opinion of one author of this document, the bounding boxes approach is often promoted as simpler than the affine transformations approach because it is easier to understand. However, the bounding boxes approach comes with many ambiguities that must be resolved using a stack of rules. By contrast, affine transformations, when used correctly, are a strictly mathematical solution leaving no room for ambiguity. The two approaches are discussed in Annex E, by Geomatys. In summary, georeferencing by bounding boxes defined in OGC API — Common — Part 2: Geospatial Data (draft) and in OGC Coverage Implementation Schema (CIS) have the following issues.

1. The georeferencing is affected by "pixel is point/area" considerations, forcing users to reverse engineer the metadata when they need to separate this measurement footprint from the "pixel center/corner" convention for calculation purposes (Annex E.1.1).

2. The need to reverse *y* axis direction between north and south is an implicit rule applied as an exception to the general formula pattern (Annex E.1.2). A CRS without north- or south-oriented axis may be an exception to the exception. Generally, it is not obvious whether an axis needs to be flipped.

3. The mapping from grid axes to CRS axes is unspecified. It could be specified by a `permutation` metadata, but a more widespread practice is to *implicitly* override the CRS definition with axis order and directions aligned with the grid. This reordering is based on heuristic rules covering only the most common cases, leaving other cases in gray areas (Annex E.1.2.2). This is not recommended, and OGC directive #14 requires that new OGC Standards choose a more reliable approach.

4. The bounding boxes and resolutions are partially redundant information. They are not fully redundant because the overlap between them enables inference of the information mentioned in item 1. Nevertheless, behavior in this situation is largely inconsistent and software dependent.

In the opinion of one of the authors of this document, georeferencing by affine transformations, when applied strictly mathematically, is unambiguous and non-redundant. This approach has some ambiguities in GeoTIFF because that format mixes the above-cited heuristic rules with the

affine transform mathematics (Annex E.2.1.5). But it is unambiguous in GeoHEIF, which takes the purely mathematical approach purged from the heuristic rules.

Georeferencing by bounding boxes is an oversimplification in some cases. Affine transformations can be seen as difficult to understand and a theoretical whim not solving any practical problem, but this is not the case. Understanding the value of affine transformations requires distinguishing some concepts that appear conflated in the OGC Common API, CIS, and GeoTIFF Standards.

1. Image orientation is conflated with CRS orientation for the convenience of formulas listed in Annex E.1.2. This conflation is the reason why CRS axis order is overridden in standards that rely on this simplification. This is because otherwise, a raster associated to a CRS having *(north, east)* axis order would have unpleasant default rendering (images would appear rotated). Understanding the value of affine transformations requires getting used to the idea that image orientation and the CRS orientation are independent. It does not mean that developers must complicate their code with axis order checks everywhere. Quite the opposite, when used correctly, affine transformations eliminate the vast majority of the needs to check axis order (Annex E.2.2.2).

2. The *"pixel is point/area"* metadata is conflated with *"pixel center/corner"* for the convenience of real-world bounding boxes usable for both georeferencing and data discovery (Figure E.1). A clarification gained by the use of affine transformations is that real-world bounding boxes are discovery metadata *derived* from georeferencing metadata rather than parts of the georeferencing definition. With this separation, the bounding boxes can be freely adjusted for measurement footprints without incidence on georeferencing. It follows that the "pixel is point/area" information is not georeferencing information neither but rather band (sample dimension) metadata (Annex E.1.1.1).

3. Conversion of pixel coordinates is conflated with an undefined third dimension in GeoTIFF's affine transformations (Annex E.2.1.5). It is not clear what the third dimension was intended to be, but it could be pixel values interpreted as *z* values in a digital elevation model. Such conflation of grid coordinates conversion with pixel values *transfer function* makes affine transformations look more complicated than they need to be. The discussion in this Testbed restricts the scope of affine transformations to grid coordinate conversions only, which implies that GeoTIFF affine transform matrix size should have been fixed to 3×3 instead of 4×4.

The use of affine transformations is a well-established practice in the computer graphics industry for decades. There is a direct correspondence between the affine transformations needed for georeferencing and some standard graphics API libraries (Listing E.7), despite those libraries being non-geospatial. Even advanced topics such as Jacobian matrices are more than 20 years old (Annex E.3.1). Nevertheless, the step-by-step guide in Annex E.2.1 is an attempt to address the concern of users being unfamiliar with affine transformations.

Another objection against affine transformations is that not all software is prepared to handle transformations with generic shears or rotations. But difficulties with rotations are generally a sign that affine transformations are not used correctly, i.e., that the software does not express all linear conversion steps as matrix multiplications (Annex E.2.2.2). If a standard nevertheless

wants to forbid shears and rotations, it is possible to define a simple profile with restrictions such as the examples given in Annex E.2.1.4.

2.2.1. Compatibility with existing ecosystems

GDAL 1 and 2 expected coordinates in *(longitude, latitude)* axis order. GDAL 3 introduced flexibility regarding axis order, but some parts of the GDAL ecosystem may not have been updated, or the updated parts may contain bugs that stay unnoticed because popular formats such as GeoTIFF overwrite the CRS axis order. The risk of improperly handled *(latitude, longitude)* axis order in GDAL ecosystem can be reduced by reordering axes according the traditional GDAL heuristic rules, but **inside the GDAL driver** instead of having that axis order specified in a standard. Annex E.2.1.6 shows how a GDAL driver can reorder axes. This approach is compatible with the goal of unambiguous axis order specification, because implementations are free to apply whatever heuristic rules they want as long as they update the affine transform accordingly. The heuristic rules no longer need to be specified in a standard but can instead be an implementation-specific (or software ecosystem-specific) black box without compromising the file format interoperability.

2.3. Misconceptions about NaN

There is often a need to flag some pixel values as missing. A value may be missing because the pixel was masked by a cloud, or because the pixel is located on a land while the raster was aimed to contain oceanographic data, or because the remote sensor did not pass over that area, *etc.* Sometimes there is a need to distinguish between different reasons why a value is missing (for example, *land* versus *cloud*). Missing values are typically represented by *Sentinel values*, such as -9999, considered as unrealistic for the phenomenon of interest.

> ***WARNING***
>
> *Sentinel values has been used in computer programming to denote either terminating values or missing values. Please note that this concept has nothing to do with the SENTINEL constellation of remote sensing satellites managed by the European Space Agency (ESA).*

More than one *sentinel value* may exist where each value is associated with a different reason for a missing value. However, the use of *sentinel values* has the following constraints:

- The geospatial library and the user must agree on which values are Sentinel's "no data" values; and
- Libraries and users must check for Sentinel's "no data" value in every calculation involving raster data.

The above constraints can pose a risk to the software industry. Some pieces of code may forget to check for *sentinel values* or may check for the wrong values (misconfiguration), or a developer

may wrongly assume that -9999 will never reach some code. It may be tempting to dismiss those risks by saying that developers will check carefully. But in the past the software industry where such optimism did not work: Buffer overflows caused by missing bound checks, which are still a major cause of bugs and security breaches despite decades of developers effort. These problems may not be noticed for an indefinite amount of time before they cause harm. Errors in the use of floating-point values should not be taken slightly since, for example, the first Ariane V rocket exploded because of a floating-point overflow.

NOTE 1: For example, compute the average of 1000 values chosen randomly between 10 and 50. The average value should be close to 30. But if one of the averaged values is the -9999 sentinel value, and if that sentinel value is not handled by the code (for example, by skipping that value), then the corrupted result is close to 20, an error of 10 compared to the expected result. Such corrupted result is well inside the [10 … 50] range of valid values, potentially making the error unnoticeable.

In the case where the raster data is stored using the binary representation of floating point numbers as defined by the IEEE 754 standard, NaN (Not-A-Number) values can be used instead of Sentinel's "no data" values. The IEEE 754 standard specifies behaviors that make NaNs not only convenient, but also (most important) safer. Annex F.2.1 gives examples of cases where the use of NaN values reduces the risk that corruption caused by unchecked Sentinel's "no data" values stay unnoticed. In the above example computing an average, if a NaN value is not handled by the program, then the result will be NaN. The desired result is lost, but this is a better situation than a corrupted result left unnoticed.

The use of NaN instead of *sentinel "no data" values* is discussed in Annex F, by Geomatys. In summary, NaNs cannot be used in all cases. They cannot be used with images that store data as integers or with lossy compression algorithms. They may have different representations with images that store data as texts (e.g., ASCII Grid format). However, when both the data and the relevant metadata are encoded as IEEE 754 binary floating-point values, the use of NaN values are worth consideration.

Sometimes there are objections against the use of NaN values in contexts where they would be well suited. But those objections are largely based on misconceptions. One of the most wide spread misconceptions is the idea that there is only one NaN value. This would be incompatible with the need to distinguish between different reasons why a value is missing. This is false: The single-precision `float` type offers more than 8 million distinct NaN values (Annex F.2.2). Furthermore, those values can be used as bitmasks allowing encoding 22 *combinable* reasons instead of 8 million distinct reasons (Annex F.3.3). For example, if an interpolation fails because one value is under a cloud and another value is above land, it is possible to encode that the interpolation result is missing because of clouds *and* lands. This information can help to decide whether it is worth to retry the calculation with another image.

Another objection is the idea that NaN values do not work well in the C/C++ language. However, multiple-NaN values are part of the C++ 11 standard (Annex F.4), and issues with NaN in arithmetic operations happen mostly when the developer uses explicitly compiler options that may be overly aggressive (Annex F.4.4). Issues may happen in other contexts such as `std::sort()` (Annex F.4.3), but they are issues that often need to be handled with Sentinel's "no data" values as well.

NOTE 2: All these considerations can be applied to any file format that supports floating point numbers such as HEIF, TIFF and NetCDF formats.

2.4. Encoding tiled and untiled HEIF files with pyramids

Ecere implemented support for GeoHEIF output in its GNOSIS Map Server implementations of OGC API — Maps and OGC API — Tiles (map tiles) with the help of the libheif library. Support for georeferencing information was added, including Coordinate Reference System (CRS) information and an affine transformation matrix, in line with the latest draft GIMI specification developed during this initiative (D010 / OGC 24-038).

The ability to encode image as tiles of a specific size as well as optionally generating overviews was also developed during the initiative, enabling support for multi-resolution tile pyramids in GeoHEIF outputs.

Annex Annex C, by Ecere, provides more details about the implementation, a potential alternative to an affine transformation matrix, experiments performed, challenges faced and future work.

2.5. GeoHEIF implementation experiment

Geomatys tested and benchmarked reading GeoHEIF files in a Java environment using two different readers: A pure-Java implementation, and a binding to the C/C++ GDAL native library. The reader and the binding were developed in the Apache SIS project and are presented in the OGC Testbed 20 GIMI Open Source Report OGC 24-042.

Annex Annex D, by Geomatys, contains some observations collected during those developments.

2.6. COG implementation and benchmarking

This section focuses on the implementation of Cloud-Optimized GeoTIFF (COG) files for benchmarking purposes. The implemented data was used to address key questions regarding the setup of a repeatable benchmark environment for testing cloud-optimized GEOINT imagery solutions (a.k.a. Cloud Optimized GeoHEIF, COHEIF) and compare COHEIF with COG.

This experiment involved developing a Python-based converter to transform Earth observation datasets to COG files. The primary goal was to facilitate efficient storage and access of geospatial data in cloud environments. The experiment focused on converting two specific datasets: MERRA-2, which provides atmospheric reanalysis data, and HSLS30, a high-resolution

land surface dataset. The findings from this experiment provide valuable insights into the best practices for converting Earth observation data into COG format, highlighting the trade-offs between different optimization strategies and their effects on performance and usability.

Annex Annex G, by George Mason University, contains the details of this implementation.

2.7. Aggregating data with geotagged video to improve cognition

GEOINT Imagery Media for ISR (GIMI), encapsulated within NGA Standard 0076 is designed to revolutionize management, integration and utilization of still and motion imagery in the defense and intelligence communities.

Identifying relevant use cases is important to ensure that the GIMI design is fit for purpose, especially for Intelligence, Surveillance and Reconnaissance (ISR) applications. Alignment with web-based technologies — and HTML in particular — is key to design interoperability that will integrate smoothly with other systems.

2.7.1. Introduction

By producing affordable high-quality digital cameras, the smartphone industry has helped to revolutionize photography for consumer markets in recent years. Integration of such devices with low-cost Global Navigation Satellite System (GNSS) technologies has spawned a new generation of mass-market location-aware cameras capable of geotagging still and motion imagery, including dash cams, drones, and body-worn cameras. Emergence of these markets has driven demand for formats such as the Exchangeable Image File Format (EXIF) and Web Video Map Tracks (WebVMT) that enable content creators to easily share their data online in an accessible format.

The prevalence of these devices has fueled proliferation of publicly available Motion Imagery Standards Board (MISB) Class 3 Imagery online. The ability to access and aggregate such data could significantly enhance the value of geospatial intelligence (GEOINT) data, particularly for Law Enforcement and Public Safety institutions who are already heavily invested in these devices. Adoption of open formats could enable these organizations to easily aggregate with data from external sources including government, industry, and crowdsourcing from the public.

2.7.2. Scope

In previous OGC Testbeds, Away Team demonstrated how WebVMT can make geotagged video more accessible, how moving objects can be tracked using GeoPose, and how data can be accurately synchronized from multiple sources including video. Successful data aggregation relies on accurate synchronization. This OGC Testbed 20 task builds on previous results to

demonstrate how aggregation of geotagged video with external data sources can improve geospatial cognition and add valuable insight for GEOINT use cases.

Video use cases studied include:

- **Maritime**: Analysis showing how on-board geotagged video from a sailing boat can be combined with Automated Identification System (AIS) data to improve vessel tracking data.
- **Camera Orientation**: A demonstration of how MISB Key-Length-Value (KLV) metadata can be exported to align a map view with camera orientation in a web browser to improve cognition.
- **Litter Monitoring**: A study of how roadside litter can be monitored using geotagged footage from a fleet of vehicles with dash cams.
- **Video Metadata Search**: Proposed tests with associated use cases to benchmark access times required to search video files for queries matching metadata patterns on the web.

Full details of these use cases are provided in the Away Team annex and a brief summary is presented below.

2.7.3. Maritime use case

In order to study the maritime use case, geotagged video footage was filmed on a smartphone aboard a yacht in the Solent near the Isle of Wight, UK. AIS data were aggregated with on-board video data to demonstrate how geospatial analysis can improve data cognition.

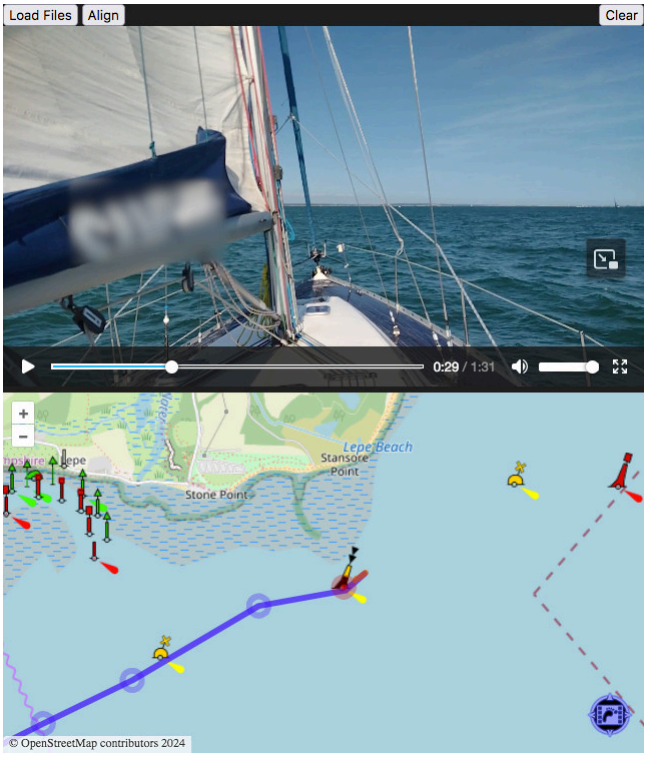


Figure 1 — Corrected AIS Data (Blue) Aggregated With On-Board Data (Orange) Synchronized With Video

Aggregated data were successfully integrated with other web technologies in a web browser to help guide GEOINT analysis and significantly enriched using simple techniques. Metrics show that the results obtained are accurate to within ten meters and synchronized to within two seconds.

Away Team has successfully demonstrated how geospatial data from multiple sources can be aggregated and accurately synchronized with video using WebVMT. Web browser integration with web technologies such as OpenSeaMap has made data more accessible, helped to guide GEOINT analysis and significantly enriched data using geospatial techniques.

Full details of this analysis can be found in the Away Team annex.

2.7.4. Camera orientation use case

Geotagged video data of a truck traveling along a highway, filmed from an aircraft flying over Wyoming, was used to demonstrate camera orientation support and accuracy in a web browser.

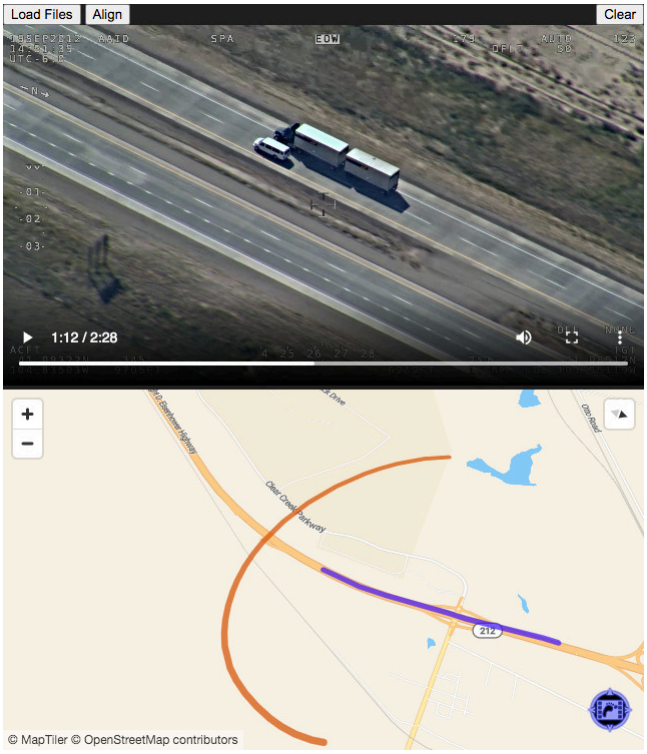


Figure 2 — Comparison Between Paths For Camera (Orange) & Frame Center (Blue)

Aligning map rotation with camera orientation improves geospatial cognition of geotagged images and simplifies GEOINT analysis. Vehicle satellite navigation systems and smartphone navigation applications have adopted the same proven technique to display geospatial information in a concise form that is aligned with the user's heading.

Combining camera orientation information with map and photogrammetry data can significantly improve cognition of imagery including features that are not normally marked on maps such as trees and road furniture. This demonstration highlights the importance of camera orientation data in the GIMI specification for still and motion imagery, and how GEOINT analysis can benefit from this information.

Full details of this analysis can be found in the Away Team annex.

2.7.5. Litter monitoring use case

Geotagged video observing target litter items was captured by front and rear dash cams at Ordnance Survey headquarters near Southampton, UK to demonstrate how roadside litter can be monitored using footage from a vehicle fleet.

Figure 3 — Front Video Segment With Litter Observations (Red) & Dash Cam Trajectory (Blue)

Video player requirements have been identified for geotagged video analysis including millisecond display, frame-level control, playlist support for segmented video, and zoom and pan to identify visual details within video frames. The mpv application offered the best support for these requirements.

Factors affecting observation visibility in video have been identified including field of view, obstruction by foreground and background objects, video resolution and compression.

Full details of this analysis can be found in the Away Team annex.

The litter monitoring dataset is ready for investigation but there was insufficient time in OGC Testbed 20 for further study. This analysis could be deferred to OGC Testbed 21 as described in Future Work.

2.7.6. Video metadata search use cases

Benchmark tests were identified for video metadata queries which provide structured metrics for accessing metadata in GIMI video files with associated use cases.

These tests are equally applicable to in-band and out-of-band metadata queries and serve to highlight the differences between them.

2.7.6.1. GIMI video metadata considerations

MISB Class 3 Imagery is an increasingly important resource due to proliferation of location-aware devices with cameras such as drones, dash cams, body-worn video and smartphones.

In-band and out-of-band metadata have strengths and weaknesses which depend on the individual use case. These two approaches are not mutually exclusive, and GIMI video metadata design should address both options.

JSON encoding of SMPTE KLVs (Key-Length-Value triplets that encode data into video feeds) can be facilitated by WebVMT which is well-integrated with HTML and CSS for access, sharing and searching online.

TAI (International Atomic Time) and UTC (Coordinated Universal Time) timestamps can be integrated with GIMI for in-band and out-of-band video metadata.

2.7.6.2. Video search benchmark analysis

Out-of-band metadata enables web crawlers to access video metadata on the web using sidecar files in a format that is agnostic of the video encoding. This enables online search engines to support video metadata queries.

Use of out-of-band metadata queries can improve privacy and security of personally identifiable information contained in video files, help to identify malign access and reduce file access bandwidth.

Benchmark tests have been designed to provide metrics for video metadata search access with identified use cases. These use cases include number plate recognition (ANPR), vehicle collision monitoring, metadata export and missing person search.

Full details of this analysis can be found in the Away Team annex.

No benchmark metrics were calculated in OGC Testbed 20 due to the lack of suitable GIMI video files for analysis. These tests are recommended for inclusion in OGC Testbed 21.

2.7.7. WebVMT enhancements

Analysis of these video use cases highlighted new requirements for the WebVMT design which have been implemented as prototypes in the engine and tools.

OpenLayers and MapLibre APIs have been integrated with the WebVMT JavaScript playback engine to support a wider range of map data and interface capabilities.

A new WebVMT feature has been prototyped to facilitate integration with maps that have customization requirements such as OpenSeaMap.

A new WebVMT utility has been created to calculate the heading of one path from another to support camera orientation.

WebVMT tools developed in OGC Testbeds 17, 19 and 20 have been updated into a single Perl command-line utility designed to create and manipulate WebVMT files to facilitate video metadata analysis.

The online community tool for importing GPX formatted data has been ported to a command-line utility and extended to read CSV files to help simplify data migration to WebVMT.

Negative cue times and unbounded cues are breaking changes to WebVTT and the `text/vtt` MIME type. A new IANA media type `text/vmt` should be considered to support these HTML features and improve timed video metadata integration with web browsers.

2.8. Image sequences metadata

During the OGC Testbed 20, support for HEIF image sequences with timed metadata was implemented and tested using libraries such as libheif.

The HEIF format uses media tracks that contain timed samples to store and organize image sequences, audio, and timed metadata within its container structure.

There are three ways to specify timed metadata applied to image sequences.

1. Metadata attached directly in the visual TrackBox (`trak`): Metadata samples can contain any arbitrary data, and the track header specifies what kind of data is contained through an URI-based naming scheme.

2. Timed metadata track: Metadata collected or captured as a timed sequence that is stored as a metadata track (`trak`). Relations between tracks are defined by track references. An example is using a `cdsc` (Content Description) reference from the metadata track to the visual track. Samples from different tracks do not necessarily come in pairs. Instead, they can be timed independently. For example: While the visual track may carry a 30 fps video stream, the metadata track may contain acceleration sensor data, sampled at a higher rate.

3. Sample Auxiliary Information (SAI): This approach supports carrying metadata for each sample in a timed track. The SAI can be included in visual and metadata tracks. SAI is a usually a small data package with auxiliary information that is directly attached to the sample.

The methods for specifying timed metadata can be combined with each other and encoded in the HEIF file in different ways depending on the type of metadata and the intended use of the image file.

1. For a file that uses minimal storage space, a visual track with a sequence of frames followed by a sequence of metadata could be used.

2. To stream images with maximum speed, a visual track with metadata and interlaced frames (still image + thin metadata + still image frame + thin metadata…) could be used.

More details about how to structure the content in the `mdat` box can be found in Annex B.7.1.

3 OUTLOOK

3 OUTLOOK

A number of topics are recommended for investigation in OGC Testbed 21.

3.1. Benchmarking

While the approach to using tiles in HEIF was proposed, there was no time to carefully benchmark COHEIF (Cloud Optimized GeoHEIF) and contrast it with COG, both in local drive and over the Internet using HTTP range. A careful study of the possible use of HEIF and GIMI as a Cloud Optimized format could be the focus of OGC Testbed 21. There is a need for both defined the order of the boxes in the COHEIF to ensure maximum performance and to implement the reading capacity as a JavaScript open source library that demonstrates fast pan and zoom to huge images (e.g., remote sensing satellite data).

3.2. Datacube descriptions

The approaches proposed to describe dimensions and properties of the images that are include in an HEIF file (e.g., band names) need to be implemented and tested. There are two competing approaches that should be benchmarked: A binary encoding describe in OGC 24-038 and a text approach based on JSON and similar to what the OGC API — Features and the OGC API — Coverages Standards are proposing.

3.3. Geotagged video use cases

Two motion imagery topics are recommended for future investigation.

3.3.1. Data analysis of litter monitoring use case

Analyze data gathered in Testbed-20 to demonstrate how roadside litter can be monitored using dash cam footage from a vehicle fleet.

1. Show how a database of litter observations can be accessed by a web search engine to demonstrate the video metadata search use case.
2. Demonstrate how an online database can be used to monitor litter accretion rates.

3. Quantify how data from rear-facing dash cams can add value to front dash cam footage for litter monitoring.

3.3.2. Benchmark metadata search use cases with GIMI video

Identify suitable files for video metadata benchmark testing and calculate metrics to compare in-band and out-of-band search request to:

1. Return all metadata for a video file;
2. Query video metadata that exactly matches a target value;
3. Query video metadata that exceeds a numerical target value; and
4. Query video metadata that is within a geospatial target region.

4

SECURITY, PRIVACY AND ETHICAL CONSIDERATIONS

4 SECURITY, PRIVACY AND ETHICAL CONSIDERATIONS

4.1. Georeference considerations

The use of incorrect georeferencing process can pose a risk to security as the incorrect determination of the location of events and the issue of incorrect instructions to remediate the effects of such events, may prevent saving lives and may create collateral damage. It is important to unambiguously georeference files using affine transformations.

4.2. Video use case privacy considerations

4.2.1. Out-of-band file permissions

Use of out-of-band video metadata can improve data privacy and security since the sidecar file has security permissions that are separate to the video file. This enables the two files to have different access permissions. More open permissions may be granted to access the sidecar file which enables search queries to determine whether the video file is of interest. This approach does not require direct access from the video content which may contain personally identifiable information such as images of faces or vehicle registrations.

Out-of-band metadata search queries only require access to the associated video file when a positive result is returned. Monitoring this file access pattern could identify malign access and help to curb web scraping activities employed by unscrupulous organizations abusing Big Data and Artificial Intelligence (AI) technologies.

4.2.2. WebVMT engine security

A new WebVMT engine feature was prototyped to generate an event after the map element has been initialized in the browser which enables the OpenSeaMap seamark layer to be properly integrated.

The prototype code utilizes the HTML Event interface (`CustomEvent`) to trigger execution of user code, instead of allowing that code to be called directly by the current thread. This design enables proper sandbox isolation that mitigates any threat from malicious code in the webpage and is consistent with HTML security guidelines.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Martin Daly: OGC 01-009, *OpenGIS Coordinate Transformation Service Implementation Specification*. Open Geospatial Consortium (2001). https://portal.ogc.org/files/?artifact_id=999.

[2] Carl Reed: OGC 19-014r3, *Topic 22 – Core Tiling Conceptual and Logical Models for 2D Euclidean Space*. Open Geospatial Consortium (2020). http://www.opengis.net/doc/AS/2D-tiles/1.0.

[3] Brittany Eaton: OGC 22-016r3, *Testbed-18: Moving Features Engineering Report*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/PER/T18-D020.

[4] Carl Stephen Smyth: OGC 21-056r11, *OGC GeoPose 1.0 Data Exchange Standard*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/IS/geopose/1.0.0.

[5] Emmanuel Devys, Ted Habermann, Chuck Heazel, Roger Lott, Even Rouault: OGC 19-008r4, *OGC GeoTIFF Standard*. Open Geospatial Consortium (2019). http://www.opengis.net/doc/IS/GeoTIFF/1.1.0.

[6] Emeric Beaufays, C.J. Stanbridge, Rob Smith: OGC 20-036, *OGC Testbed-16: Full Motion Video to Moving Features Engineering Report*. Open Geospatial Consortium (2021). http://www.opengis.net/doc/PER/t16-D021.

[7] Guy Schumann: OGC 21-036, *OGC Testbed-17: Moving Features ER*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/PER/t17-D020.

[8] Sina Taghavikish: OGC 23-042, *OGC Testbed-19 – Non-Terrestrial Geospatial Engineering Report*. Open Geospatial Consortium (2024). http://www.opengis.net/doc/PER/T19-D001.

[9] Lucio Colaiacomo, Joan Masó, Emmanuel Devys, Eric Hirschorn: OGC 08-085r8, *OGC® GML in JPEG 2000 (GMLJP2) Encoding Standard*. Open Geospatial Consortium (2018). http://www.opengis.net/doc/IS/GMLJP2/2.1.0.

[10] ISO/IEC: ISO/IEC 14496-12:2022, *Information technology – Coding of audio-visual objects – Part 12: ISO base media file format*. International Organization for Standardization, International Electrotechnical Commission, Geneva (2022). https://www.iso.org/standard/83102.html.

[11] ISO/IEC: ISO/IEC 15444-1:2024, *Information technology – JPEG 2000 image coding system – Part 1: Core coding system*. International Organization for Standardization, International Electrotechnical Commission, Geneva (2024). https://www.iso.org/standard/87632.html.

[12] ISO/IEC: ISO/IEC 15444-3:2007, *Information technology – JPEG 2000 image coding system: Motion JPEG 2000 – Part 3:*. International Organization for Standardization,

International Electrotechnical Commission, Geneva (2007). https://www.iso.org/standard/41570.html.

[13] ISO/IEC: ISO/IEC 15444-16:2021, *Information technology — JPEG 2000 image coding system — Part 16: Encapsulation of JPEG 2000 images into ISO/IEC 23008-12*. International Organization for Standardization, International Electrotechnical Commission, Geneva (2021). https://www.iso.org/standard/80620.html.

[14] ISO: ISO 19107:2019, *Geographic information — Spatial schema*. International Organization for Standardization, Geneva (2019). https://www.iso.org/standard/66175.html.

[15] ISO: ISO 19111:2019, *Geographic information — Referencing by coordinates*. International Organization for Standardization, Geneva (2019). https://www.iso.org/standard/74039.html.

[16] ISO: ISO 19123:2005, *Geographic information — Schema for coverage geometry and functions*. International Organization for Standardization, Geneva (2005). https://www.iso.org/standard/40121.html.

[17] ISO: ISO 19123-2:2018, *Geographic information — Schema for coverage geometry and functions — Part 2: Coverage implementation schema*. International Organization for Standardization, Geneva (2018). https://www.iso.org/standard/70948.html.

[18] ISO: ISO 19136:2007, *Geographic information — Geography Markup Language (GML)*. International Organization for Standardization, Geneva (2007). https://www.iso.org/standard/32554.html.

[19] ISO: ISO 19162:2019, *Geographic information — Well-known text representation of coordinate reference systems*. International Organization for Standardization, Geneva (2019). https://www.iso.org/standard/76496.html.

[20] ISO/IEC: ISO/IEC 23008-12:2022, *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 12: Image File Format*. International Organization for Standardization, International Electrotechnical Commission, Geneva (2022). https://www.iso.org/standard/83650.html.

[21] OGC Testbed 20 GEOINT Imagery Media for ISR (GIMI) Specification Report

[22] OGC Testbed 20 GIMI Benchmarking Report

[23] OGC Testbed 20 Open Source Report

[24] Carl Reed: OGC 08-038, *Revision to Axis Order Policy and Recommendations.* Open Geospatial Consortium (2017). https://portal.opengeospatial.org/files/?artifact_id=76024

[25] W3C: W3C webvmt, *WebVMT: The Web Video Map Tracks Format*. World Wide Web Consortium https://www.w3.org/TR/webvmt/.

[26] GEOINT Imagery Media for ISR (GIMI) Profile of ISOBMFF & HEIF

[27] Joe Stufflebeam, Dirk Farin: m70021: [HEIF] Tiled item type for very large images, 2024

A

ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

A ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

AI	Artificial Intelligence
AIS	Automatic Identification System
ANPR	Automatic Number Plate Recognition
API	Application Programming Interface
CG	Community Group
CIS	Coverage Implementation Schema
CITE	Compliance and Interoperability Test Environment
COG	Cloud Optimized GeoTIFF
CRS	Coordinate Reference System
CSS	Cascading Style Sheets
CSV	Comma Separated Values
EXIF	Exchangeable Image File Format
fps	Frames Per Second
FPU	Floating-Point Unit
GEOINT	Geospatial Intelligence
GIMI	GEOINT Motion Imagery for ISR
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit

GPX	GPS Exchange Format
HD	High Definition
HTML	Hypertext Markup Language
IANA	Internet Assigned Numbers Authority
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
ISR	Intelligence, Surveillance and Reconnaissance
JSON	JavaScript Object Notation
KLV	Key Length Value
MIME	Multipurpose Internet Mail Extensions
MISB	Motion Imagery Standards Board
MMSI	Maritime Mobile Service Identity
NAN	Not A Number
NGA	National Geospatial-Intelligence Agency
OS	Ordnance Survey
SCRA	Solent Cruising and Racing Association
SMPTE	Society of Motion Picture and Television Engineers
TAI	International Atomic Time
ULP	Unit in the Last Place
UTC	Coordinated Universal Time
WebVMT	Web Video Map Tracks
WebVTT	Web Video Text Tracks

B

ANNEX B (INFORMATIVE) TILING FOR VERY HIGH RESOLUTION IMAGES (AUTHORED BY DIRK FARIN)

B

ANNEX B (INFORMATIVE) TILING FOR VERY HIGH RESOLUTION IMAGES (AUTHORED BY DIRK FARIN)

When working with high-resolution images, keeping the whole image in memory may not be feasible. Instead, accessing the image as small individual tiles is needed. When reading a tile, only the parts of the file that are needed for decoding a tile must be read and decoded. Similarly, when writing, each tile can be encoded separately and added to the image file.

The HEIF `grid` image format saves a tiled image as a collection of small images. Each tile is stored in the HEIF file as a separate image, which are then combined into a large grid image. This format is used for example in mobile phone cameras that split the image usually into 512×512 tiles. However, the HEIF `grid` image item format is limited to images with less than 256×256 tiles because the number of tiles per row/column is stored in an 8 bit integer and because the number of references in `iref` is limited to 65535. Moreover, storing tiled images as `grid` items has significant overhead because metadata must be stored for each tile image. Each tile image has a copy of the metadata `iinf`, `ipma`, `iref`, `iloc` that sum to > 3.3 MB for a 256×255 tile image. This metadata overhead is significant because it is contained in the main `meta` box and has to be loaded completely before decoding of any tile of the image can start.

To support larger images, the Testbed participants decided to propose and implement a new image item type `tili` as an alternative to `grid`.

B.1. Requirements

The new `tili` image type should cover the following requirements:

- Support for arbitrarily large resolutions (over 1G x 1G pixels in a single image),
- Storing 2D images in a tiled file layout, supporting random access decoding of any chosen tile with a single byte range access to the file,
- Enable streaming the image content over the internet with small initial setup delays, i.e., much less overhead than `grid`,
- Support tiled images in which some tiles are blank and not covered with image data,
- Saving tiles in arbitrary order to allow gradually growing files,

- Interleaved storage of multiple tiled images, such as for multi-resolution pyramids where storage of the lower resolution layer is interleaved with the higher-resolution layers,
- Ability to build multi-resolution pyramids with a mixture of `grid`, `tili`, and `unci` images to have partial compatibility to software without `tili` support.

Additionally, a desirable feature is to extend the tiling scheme to data arranged as n-dimensional hyperrectangles. This covers several application areas, such as:

- Storing three-dimensional voxel images,
- Storing multispectral images, using the frequency band index as the third dimension,
- Providing a way to store complex numbers or vectors using lossy image codecs by using an additional dimension to differentiate between real and imaginary part, or to select the vector component,
- Providing a way to store an image series depending on one or several parameters, like exposure duration, capturing angle, a model parameter when recording the outcome of a simulation, or an experimental parameter, like an image series taken of a physical experiment at different pressures.

B.2. Proposed tili Image Item

This section describes the proposed data format for a `tili` image item type that covers the above requirements. The data of a `tili` image item consists of the concatenated compressed data for each image tile, and a table of pointers to the start of each tile in this data stream. The table is prepended to the coded tile data. Storing the table together with the compressed tile data in the `mdat` box allows to load both the image tiles and the pointers to the tile on demand (e.g., over a network connection).

The `tili` specification has been submitted to MPEG for consideration to be included in the core HEIF standard. The proposal m70021:Tili proposal was written together with Joe Stufflebeam. The proposal is an extended version of what is described in this Report and the interested reader is referred to that document.

B.3. Definition

- Box type: 'tilC'
- Container: ItemPropertyContainerBox
- Property type: Descriptive item property

- Mandatory (per item): Yes, for an image item of type 'tili'

The `TiledImageConfigurationBox` specifies the tile resolution, and the compression codec used to store the image tiles in an image of type `tili`. For N-dimensional (N>2) images, it also specifies the resolution of these extra dimensions.

B.3.1. Syntax

```
aligned(8) class TiledImageConfigurationBox
extends ItemFullProperty('tilC', version=0, flags) {

  unsigned int(32) tile_width;
  unsigned int(32) tile_height;

  unsigned int(32) tile_compression_type;

  unsigned int(8) number_of_extra_dimensions;
  for (int i=0; i<number_of_extra_dimensions; i++) {
    unsigned int(32) dimension_size[i];
  }

  unsigned int(8) number_of_tile_properties;
  for (int i=0; i<number_of_tile_properties; i++) {
    ItemProperty tile_image_property[i];
  }
}
```

Listing B.1 — Syntax for TiledImageConfigurationBox

B.3.2. Semantics

`tile_width`, `tile_height` is the size of a single tile. All tiles have the same size. Tiles at the right or bottom border may extend beyond the total image size. This is consistent with the definition of a tessellation type "square" in the OGC Abstract Specification Topic 22 — Core Tiling Conceptual and Logical Models for 2D Euclidean Space, OGC 19-014r3.

`tile_item_type` specifies the image item type used for all the individual tile images. `tile_item_type` is one of the possible four-character types of ordinary image items (e.g., `hvc1` for h265 compression or `j2k1` for JPEG2000).

`number_of_extra_dimensions` specifies the number of dimensions of the N-dimensional image as `number_of_extra_dimensions = N - 2`. A 2D image has `number_of_extra_dimensions=0`.

`dimension_size[i]` specifies the size of dimension `i+2` of the N-dimensional image. The size of the first two dimensions is obtained from the mandatory `ispe` item property.

`number_of_tile_properties` specifies the number of tile properties stored in the tilC.

`tile_image_property[]` are the image item properties used when decoding a tile image. This includes at least all mandatory item properties for an image item of type `tile_item_type` with the exception of ispe. If `tile_image_property[]` does not contain an `ispe` property box, the decoder shall synthesize an `ispe` property with `tile_width` and `tile_height` as the size.

`OffsetFieldLength = OFFS_LEN[flags & 0x03]` defines the number of bits used to store the offset to the image data of a specific tile. `OFFS_LEN[] = [ 32, 40, 48, 64 ]`

`SizeFieldLength = SIZE_LEN[(flags>>2) & 0x03]` defines the number of bits used to store the length of the image data of a specific tile. `SIZE_LEN[] = [ 0, 24, 32, 64 ]`

`(flags & 0x10)` is a hint to a decoder whether the compressed tile image data is stored consecutively in sequential order.

The four different offset pointer sizes defined the maximum `tili` image file sizes as shown in Table B.1

Table B.1 — Maximum image file size depending on pointer lengths.

POINTER LENGTH	MAXIMUM COMPRESSED IMAGE SIZE
32 bit	4 GB
40 bit	1 TB
48 bit	256 TB
64 bit	16 EB

The four different tile size field lengths define the maximum compressed tile sizes as shown in Table B.2

Table B.2 — Maximum compressed tile size depending on size field length.

SIZE FIELD LENGTH	MAXIMUM TILE SIZE
0	depending on pointer length
24 bit	16 MB
32 bit	4 GB
64 bit	16 EB

B.3.3. `tili` Item Data

The item data consists of an offset pointer table `TiledImageOffsetTable`, followed by the compressed image data.

The number of tile offsets stored in the table (`NumTiles`) is computed by:

```
  TileColumns = (ispe_width + tile_width -1)/tile_width;
  TileRows    = (ispe_height + tile_height -1)/tile_height;

  NumTiles = TileColumns * TileRows;
  for (i=0; i<number_of_extra_dimensions; i++) {
    NumTiles = NumTiles * dimension_size[i];
  }
```

Listing B.2

`ispe_width` and `ispe_height` is the total image size as specified in the mandatory `ispe` item property.

```
aligned(8) class TiledImageOffsetTable {

  for (int i=0; i < NumTiles ; i++) {
    unsigned int(OffsetFieldLength) tile_start_offset[i];
    unsigned int(SizeFieldLength) tile_size[i];         // note: not present if
SizeFieldLength==0
  }
}
// ... followed by compressed tile data ...
```

Listing B.3

B.3.4. Semantics

`tile_start_offset[i]` points to the start of the compressed data of the tile. The position is given relative to the start of the `TiledImageOffsetTable` data. Note that this is not a file offset, but an offset into the item's data that can potentially span several `iloc` extents. If a tile is not coded and the corresponding image area is undefined, the `tile_start_offset[i]` shall be 0. If a tile is not coded, but the displayed image should be taken from a lower-resolution layer (in a `pymd` stack), `tile_start_offset[i]` shall be 1.

NOTE: this can be used for maps where large areas contain not much detail, such as water areas.

`tile_size[i]` (if present) indicates the number of bytes of the coded tile bitstream.

The entries in the offset table are ordered in row-major sequence. In other words, for a 2D image, they are indexed as `[y][x]`, a three dimensional volumetric image (extra dimension `z`) would be indexed as `[z][y][x]`.

The compressed tiles data may be stored in the file in arbitrary order, i.e., the `tile_start_offset[]` are not necessarily in increasing order.

If the `tile_size[i]` variables are not present, the decoder has to infer them from the `tile_start_offset[]` values. For the case that the tiles are stored in sequential order (`flags & 0x10 == 0x10`), the `tile_size[i]` can be computed as `tile_start_offset[i+1] - tile_start_offset[i]` with the exception of the last tile, which extends until the end of the data. If the tiles are not stored in sequential order, the decoder first must sort the tile start offsets before it can again compute the size from the difference to the next tile start. Note that in this case, the decoder cannot read the offset table on-demand. Thus, storing the tile sizes is advised in this case if on-demand access of the tile offsets is desired. Multiple tiles can use the same

`tile_start_offset` to reference a similar image content. This case must be taken care of when computing the tile sizes.

The file structure of a `tili` image item is illustrated in Figure B.1.

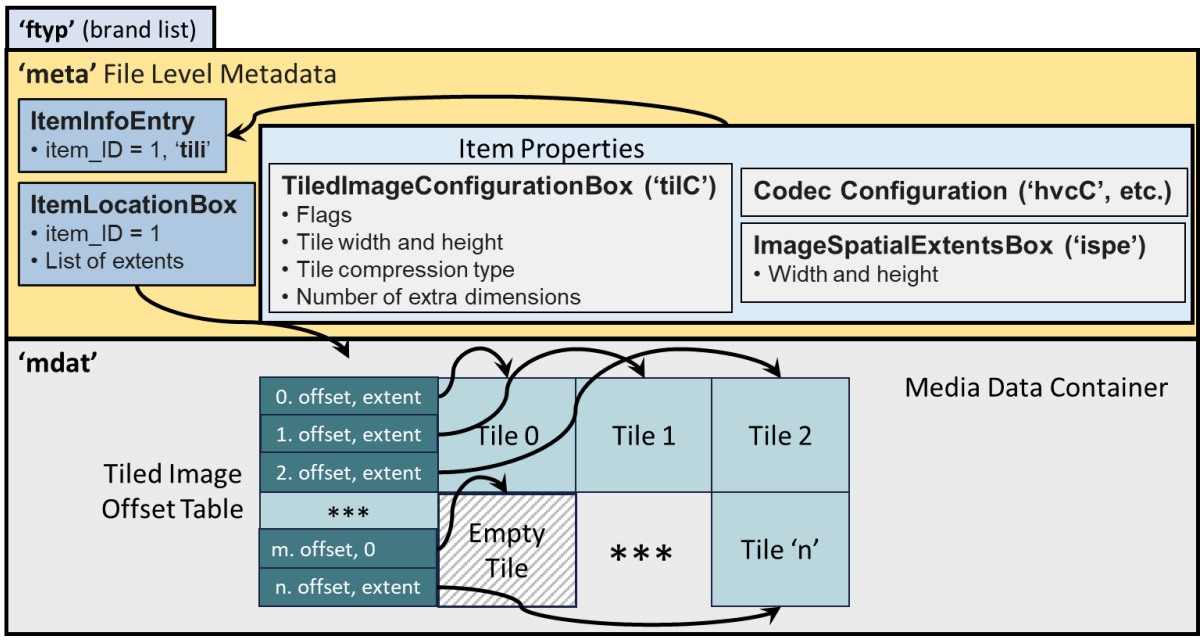


Figure B.1 — File structure of a `tili` image item. Picture taken from m70021:Tili proposal, drawn by Joe Stufflebeam.

B.4. Decoding Process of a Single Tile

Each tile in a `tili` image item shall be compressed with the same encoder and settings. The item properties for the tiles are not stored separately for each tile. Instead, only one set of properties is stored in the `tilC` box and implicitly assigned to each tile. All essential properties of the tile image shall be included in the `tile_image_property[]` array, except for the `ispe` property, which is synthesized by the decoder from the `tile_width` and `tile_height` variables in the `tilC` box if no `ispe` property is present in `tile_image_property[]`. Thus, a `tili` image with `tile_item_type=hvc1` shall have an associated `hvcC` box that describes the coded stream of each tile.

The `ispe` item associated with the `tili` defines the total of the `tili` image, not the size of a tile. If this total image size is not an integer multiple of the tile size, the image data of the tiles at the right and bottom border is cropped to the total image size.

Decoding of a single tile shall be done equivalently to the following steps:

- Create a virtual image item of type `tile_item_type`,
- Assign the item properties in `tile_image_property[]` to the virtual tile image item,

- If no `ispe` item is defined in `tile_image_property[]`, assign an `ispe` item property of size (`tile_width`, `tile_height`) to the virtual image item,
- Decode the virtual image item.

B.4.1. Multi-Dimensional Data

With the `extra_dimensions` variable in the `tilC` configuration header, it is possible to store not only 2D image data in a `tili` image item but also organize the tiles into datasets with additional dimensions.

For example, one extra dimension could be used to stack the 2D tiled images into a 3D voxel image volume.

Another example would be to use the extra dimension to address different frequency bands in a multispectral image. This is illustrated in Figure B.2.

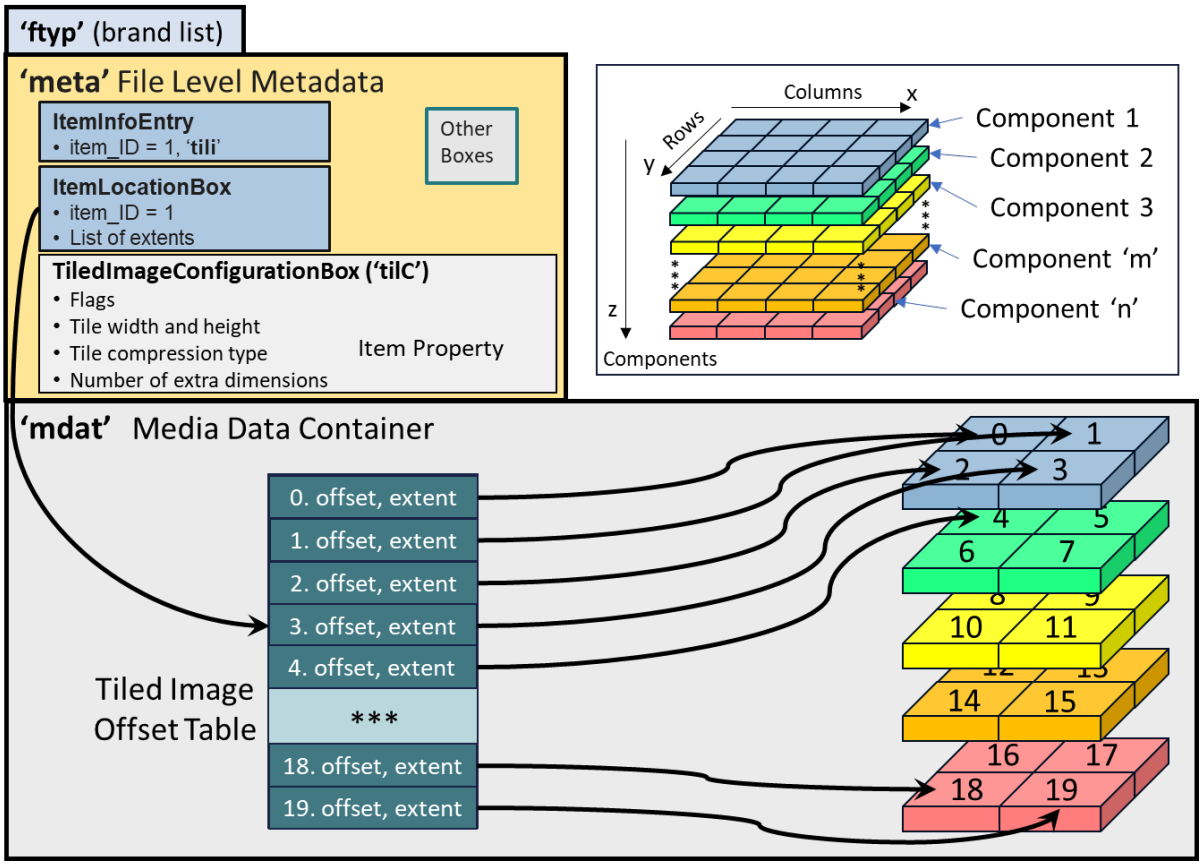


Figure B.2 — File structure of a `tili` image item. Picture taken from m70021:Tili proposal, drawn by Joe Stufflebeam.

This scheme can be extended to even more extra dimensions. As an example, when two extra dimensions `i` and `k` are used, the tiles in the `TiledImageOffsetTable` are indexed by `[k][i][y][x]`.

B.5. File structure

Even though the compressed tile data logically follows continuously after the metadata, we can still write the data interleaved into the file (e.g., intermixed with other `tili` resolution layers) by employing `iloc` extents.

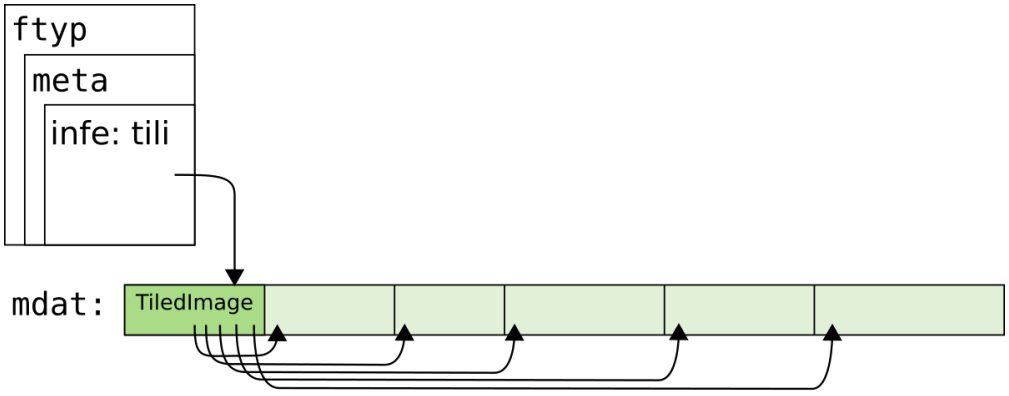


Figure B.3 — Simple file with single `tili` image

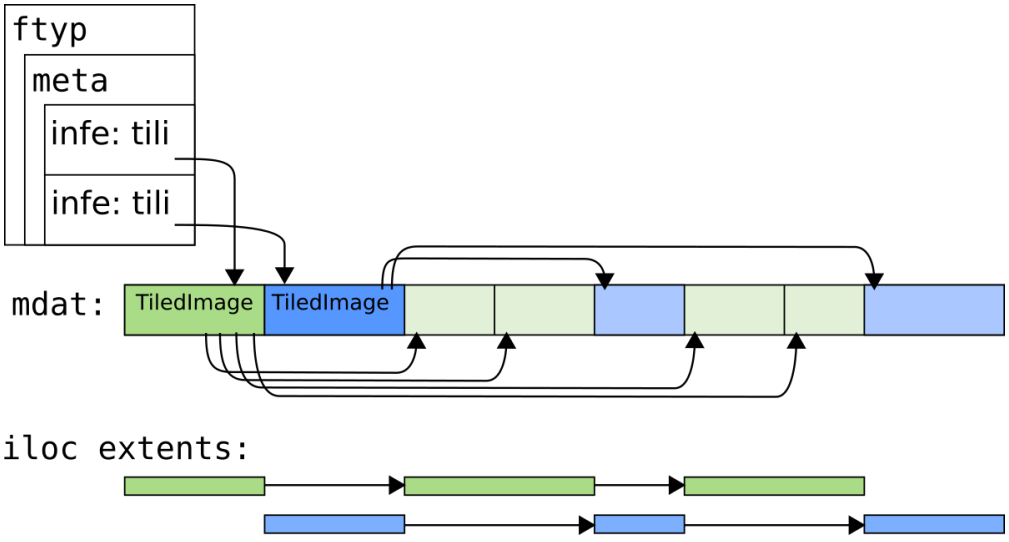


Figure B.4 — File with two interleaved `tili` images

B.6. tili, grid, and unci coexistence

When building a multi-resolution `pymd` pyramid, different image types can be used for each layer. For example, using `grid` images for the lower resolution layers is possible so that these can be read with software that does not understand `tili` image types. Software support for `tili` is only needed for the high-resolution layers.

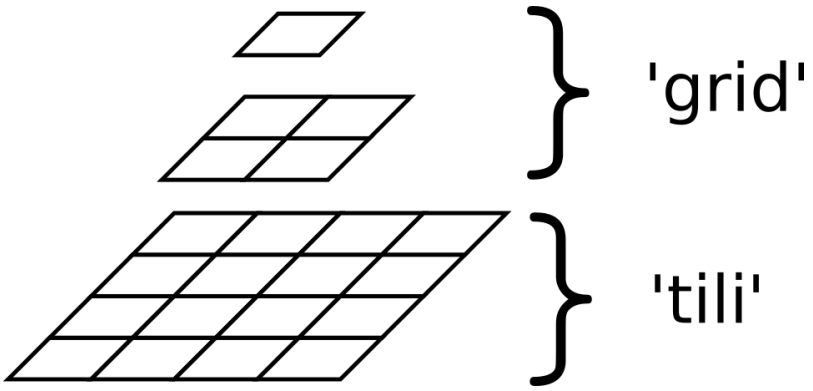


Figure B.5 — A `pymd` image pyramid with different tiling methods used for the resolution layers.

B.7. Image Sequence Implementation

The implementation of the image sequences builds upon the ISO Base Media File Format (ISOBMFF) Standard for the overall framework and the file structure used for video sequences. In addition, HEIF (ISO/IEC 23008-12:2022) defines images sequences as a variant of the [ISO14496-12] video sequences. The encapsulation of the various video codecs is described in various documents as detailed in Table B.3. Finally, GIMI specific metadata is implemented as specified in NGA.STND.0076_1.0_GIMI.

Table B.3 — Standards covering storage of coded image sequences

CODEC NAME	CODEC STANDARD	ISOBMFF (VIDEO) ENCAPSULATION	HEIF (IMAGE) ENCAPSULATION
H.265	ISO 23008-12/ITU-T H.265	ISO 14496-15:2010 DAM 2	ISO 23008-12
H.266	ISO 23090-3	ISO 14496-15:2019 DAM 2	ISO 23008-12:2017 DAM 2
AV1	https://aomediacodec.github.io/av1-spec/av1-spec.pdf	https://aomediacodec.github.io/av1-isobmff/	https://aomediacodec.github.io/av1-avif/
JPEG-2000	ISO 15444-1/ITU-T T.800	ISO 15444-3:2007/ITU-T T.802	ISO 15444-16/ITU-T T.815

CODEC NAME	CODEC STANDARD	ISOBMFF (VIDEO) ENCAPSULATION	HEIF (IMAGE) ENCAPSULATION
JPEG	ISO 10918-1/ITU-T T.81	ISO 23008-12	ISO 23008-12
uncompressed	ISO 23001-17	ISO 23001-17	ISO 23001-17

B.7.1. File Storage Layout Considerations

The ISOBMFF Standard does not define a specific order in which the compressed data packets are stored in the `mdat` container. Instead, it merely holds a list of pointers to the start of each sample's data and its respective size. These pointers are independent of the samples in each track and thus, the data of separate tracks can either be stored interleaved or contiguous for each track.

Similarly, the sample auxiliary information (SAI) data packets are also stored in the `mdat` with pointers to the packets and their sizes in the track definition. However, when all SAI packets are stored contiguously, a special mode can be used that supports omitting all pointers except the first to the beginning of the sequence of contiguous data packets, leading to a smaller file size.

```
This leads to a trade-off a writer of the file can consider: The
writer may either store all SAI packets of the whole sequence
contiguously, resulting in a smaller file size and making it easier
to read the whole SAI metadata without reading the video frames.
Alternatively, the writer may choose to store the SAI metadata
interleaved with each image that it belongs to. This has the advantage
that video and SAI metadata can be read together in one network
request since its data is stored contiguously.
```

Figure B.6

When using libheif, the stored file layout can be chosen with the configuration parameter `heif_track_info::write_aux_info_interleaved`.

For the user, this choice is completely transparent with the exception that the user may experience different performance depending on the application use case.

A similar choice can be considered whether multiple tracks should be stored interleaved or whether each track should be stored contiguously. In this case, the situation is a bit more complex since the samples in each track may have a different number of samples per second and thus there might not be a fixed interleaving pattern. The current implementation using libheif always store each track data contiguously.

C

ANNEX C (INFORMATIVE) GEOHEIF OUTPUT USING LIBHEIF (ECERE)

ANNEX C (INFORMATIVE) GEOHEIF OUTPUT USING LIBHEIF (ECERE)

Ecere implemented support for GeoHEIF output in its GNOSIS Map Server implementation of OGC API — Maps and OGC API — Tiles (map tiles) with the help of the libheif library.

Support for georeferencing information was added, including Coordinate Reference System (CRS) information and an affine transformation matrix, in line with the latest draft GIMI specification developed during this initiative (D010 / OGC 24-038).

The ability to encode image as tiles of a specific size as well as optionally generating overviews was also developed during the initiative, enabling support for multi-resolution tile pyramids in GeoHEIF outputs.

C.1. Georeferencing information

Encoded HEIF outputs include the following GeoHEIF aspects:

- inclusion of the `ogeo` compatible brand in `ftyp` box
- `mcrs` Coordinate Reference System Property, using the `curi` (CURIE) encoding
- `mtxf` Model Transformation Property for the affine transformation matrix

The `wkt2` (Well-known text representation of coordinate reference systems) encoding of the CRS was also temporarily used while waiting for GDAL to support the `curi` encoding.

Some difficulties were encountered with the CURIE encoding due to expectations of a safe CURIE (including square brackets), and a related bug reading the null-terminated `utf8string` existing at one point in the GDAL driver which was subsequently fixed.

Implementing and testing the affine transformation matrix correctly across a variety of CRSs proved challenging, in particular because of running into a bug with the new implementation which resulted in a classic axis-order confusion. Ecere collaborated with other participants to resolve this issue.

C.2. Potential alternative to affine transformation matrix

Based on long discussions with other participants, and practical development experience during this initiative, Ecere suggested that it might be preferable to encode the transformation information presented in the `mtxf` affine transformation matrix in a different way, while preserving the exact same information. Instead of a 3×3 matrix for a 2D image, a scale factor and translation offset could be associated with an integer axis number representing each of the image dimensions. For example:

- CRS CURIE: [ESPG:4326]
- First CRS axis (latitude): image axis 1 (i), scale 10, offset 20
- Second CRS axis (longitude): image axis 0 (j), scale 5, offset 30

would indicate that the first axis of the `EPSG:4326` CRS (latitude) corresponds to the vertical axis 1 (`j`) of the image, where indices along this vertical axis must be multiplied by a scale of 10 and an offset of 20 must then be added to obtain the latitude coordinates. Similarly, the second axis (longitude) corresponds to the horizontal axis 0 (`i`) of the image, where indices along this horizontal axis must be multipled by 5 and an offset of 30 must then be added to obtain the longitude coordinates.

This approach would not seem to imply that rotation or shearing can potentially be part of the transformation, and would be generally easier to understand and more likely to be implemented correctly. Describing an arbitrary number of dimensions (n) would also scale better, as only three numbers per dimension are required using this approach — 3n values, instead of requiring an additional row and column for each dimension with an affine transformation matrix — $(n+1)^2$ values. Although the space saving in the data cube header is not very significant compared to the size of the actual data, as an example, assuming unsigned byte axis numbers, double-precision (64-bit) real numbers for scales and offsets, and omitting the last row of the matrix as done for GeoHEIF, a 2D transformation would require 34 bytes using this alternative (`2×(1+2×8)`) instead of 48 bytes (`2×(2+1)×8`) using the matrix, and a 4D transformation would require 68 bytes (`4×(1+2×8)`) instead of 160 bytes (`4×(4+1)×8`). As Ecere has been developing a 3D graphics engine for almost three decades, including software rendering, OpenGL and Direct3D backends, this suggestion does not originate from a lack of familiarity with transformation matrices.

Even if the `mtxf` affine transformation matrix approach is adopted for GeoHEIF, this alternative could be considered for other future OGC raster data cube encodings, such as a GeoTIFF 2.0 revision.

C.3. Work on OGC API — Coverages output

At the end of the initiative, HEIF encoding in the GNOSIS Map Server was limited to encoding 8-bit RGBA imagery data or maps for OGC API — Maps or OGC API — Tiles output. Support for

OGC API — Coverages HEIF output was not yet completed, as it was dependent on capabilities not yet implemented in libheif.

Associated with output of non-visual channels, work was done on implementing support for describing the logical schema and associated semantic of the different bands included in a GeoHEIF file. This was described at one point in the D010 report using the `cdsc` (content description) and `bndi` (band information) properties, with the ability to include a logical schema in JSON Schema, following the approach for OGC API — Features — Part 5: Schemas as well as a proposed equivalent OGC API — Common part which are returned by OGC APIs at `/collections/{collectionId}/schema`.

Since support for OGC API — Coverages and tiled coverage data through OGC API — Tiles was not yet deployed and this functionality was not kept in the D010 specification, these capabilities were not yet integrated and deployed to the demonstration GNOSIS Map Server endpoint.

C.4. Overviews, Tiles and Compression

Support for tiled image output was implemented with the `unci` encoding. This encoding was selected to avoid the excess overhead of metadata stored for each tile, and for lossless compression capability.

Provisional query parameters were added to the GNOSIS Map Server to control tiling and overview options. These parameters, which could potentially be considered for future extensions of OGC APIs, are:

- `overviews`: whether to include overviews, `true` or `false` (defaults to `false`)
- `tile-size`: the tile-size (an integer), implying tiled output (defaults to not tiled)
- `compression`: the lossless compression algorithm to apply, `deflate`, `zlib`, `brotli` or `none` (defaults to `none`)

Including the tile size in the request to the GNOSIS server API generates a tiled image output, for example:

https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?f=heif&tile-size=256

As `unci` encoding strictly requires that image size be an integer multiple of the tile size, Ecere implemented support for tile requests not being such multiples by using transparent pixel padding for the portion of tiles falling outside of the image, for example:

https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?f=heif&width=1000&height=500&tile-size=256

Support for lossless compression was also implemented, independently of whether overviews or tiles are used. Example request:

https://maps.gnosis.earth/ogcapi/collections/blueMarble/map?f=heif&overviews=true&compression=brotli

When overviews are enabled using the `overviews` parameters, a pyramid of overviews are generated for lower resolution versions of the output, and then each overview image is encoded, using `unci` tiles if tiling is enabled. Finally, a `pymd` pyramid entity group is created referencing the identifiers of each overview.

C.5. Experiments

Ecere implemented a unit test for exporting raster data to GeoHEIF, and verified output results using `heif-info` (from libheif), `gdalinfo` and `gdaltransform` tools (from GDAL), as well as QGIS using the latest development builds of GDAL.

```
MIME type: unknown
main brand: mif2
compatible brands: mif1, miaf, ogeo
Box: ftyp -----
size: 28   (header size: 8)
major brand: mif2
minor version: 0
compatible brands: mif1,miaf,ogeo

...

| Box: iinf -----
| size: 203   (header size: 12)
| | Box: infe -----
| | size: 21   (header size: 12)
| | item_ID: 1
| | item_protection_index: 0
| | item_type: unci
| | item_name:
| | hidden item: false
| |

...

| | | index: 23
| | | Box: mcrs -----
| | | size: 28   (header size: 8)
data: 0000: 00 00 00 00 63 75 72 69  5b 45 50 53 47 3a 34 33
      0010: 32 36 5d 00
| | |
| | | index: 24
| | | Box: mtxf -----
| | | size: 60   (header size: 8)
data: 0000: 00 00 00 01 00 00 00 00  00 00 00 00 bf c6 80 00
      0010: 00 00 00 00 40 56 80 00  00 00 00 00 3f c6 80 00
      0020: 00 00 00 00 00 00 00 00  00 00 00 00 c0 66 80 00
      0030: 00 00 00 00

...
```

```
| Box: grpl -----
| size: 122   (header size: 8)
| | Box: pymd -----
| | size: 114   (header size: 12)
| | group id: 10
| | entity IDs: 9 8 7 6 5 4 3 2 1
| | tile size: 256x256
| | layer 0:
| | | binning: 256
| | | tiles: 4x8
| | layer 1:
| | | binning: 128
| | | tiles: 4x8
...
| | layer 8:
| | | binning: 1
| | | tiles: 4x8
...
```

Listing C.1 — Extracts of boxes from GNOSIS Map Server HEIF output inspected with `heif-info --dump-boxes`

```
Driver: HEIF/ISO/IEC 23008-12:2017 High Efficiency Image File Format
Files: bm2048-256-brotli-ov.heif
Size is 2048, 1024
Coordinate System is:
GEOGCRS["WGS 84",
    ENSEMBLE["World Geodetic System 1984 ensemble",
        MEMBER["World Geodetic System 1984 (Transit)"],
        MEMBER["World Geodetic System 1984 (G730)"],
        MEMBER["World Geodetic System 1984 (G873)"],
        MEMBER["World Geodetic System 1984 (G1150)"],
        MEMBER["World Geodetic System 1984 (G1674)"],
        MEMBER["World Geodetic System 1984 (G1762)"],
        MEMBER["World Geodetic System 1984 (G2139)"],
        MEMBER["World Geodetic System 1984 (G2296)"],
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]],
        ENSEMBLEACCURACY[2.0]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["Horizontal component of 3D system."],
        AREA["World."],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
Data axis to CRS axis mapping: 2,1
Origin = (-180.000000000000000,90.000000000000000)
Pixel Size = (0.175781250000000,-0.175781250000000)
Subdatasets:
  SUBDATASET_1_NAME=HEIF:1:bm2048-256-brotli-ov.heif
  SUBDATASET_1_DESC=Subdataset 1
  SUBDATASET_2_NAME=HEIF:2:bm2048-256-brotli-ov.heif
  SUBDATASET_2_DESC=Subdataset 2
```

```
  SUBDATASET_3_NAME=HEIF:3:bm2048-256-brotli-ov.heif
  SUBDATASET_3_DESC=Subdataset 3
  SUBDATASET_4_NAME=HEIF:4:bm2048-256-brotli-ov.heif
  SUBDATASET_4_DESC=Subdataset 4
  SUBDATASET_5_NAME=HEIF:5:bm2048-256-brotli-ov.heif
  SUBDATASET_5_DESC=Subdataset 5
  SUBDATASET_6_NAME=HEIF:6:bm2048-256-brotli-ov.heif
  SUBDATASET_6_DESC=Subdataset 6
  SUBDATASET_7_NAME=HEIF:7:bm2048-256-brotli-ov.heif
  SUBDATASET_7_DESC=Subdataset 7
  SUBDATASET_8_NAME=HEIF:8:bm2048-256-brotli-ov.heif
  SUBDATASET_8_DESC=Subdataset 8
  SUBDATASET_9_NAME=HEIF:9:bm2048-256-brotli-ov.heif
  SUBDATASET_9_DESC=Subdataset 9
Corner Coordinates:
Upper Left  (-180.0000000,  90.0000000) (180d 0' 0.00"W, 90d 0' 0.00"N)
Lower Left  (-180.0000000, -90.0000000) (180d 0' 0.00"W, 90d 0' 0.00"S)
Upper Right ( 180.0000000,  90.0000000) (180d 0' 0.00"E, 90d 0' 0.00"N)
Lower Right ( 180.0000000, -90.0000000) (180d 0' 0.00"E, 90d 0' 0.00"S)
Center      (   0.0000000,   0.0000000) (  0d 0' 0.01"E,  0d 0' 0.01"N)
Band 1 Block=256x256 Type=Byte, ColorInterp=Red
Band 2 Block=256x256 Type=Byte, ColorInterp=Green
Band 3 Block=256x256 Type=Byte, ColorInterp=Blue
```

Listing C.2 — Information Extracts of boxes from GNOSIS Map Server HEIF output inspected with GDAL's `gdalinfo`

Ecere also tested the tile pyramids outputs of the GNOSIS Map Server with the Tiled Image Viewer created by Dirk Farin, and compared outputs with pre-generated test HEIF files made available by participants.

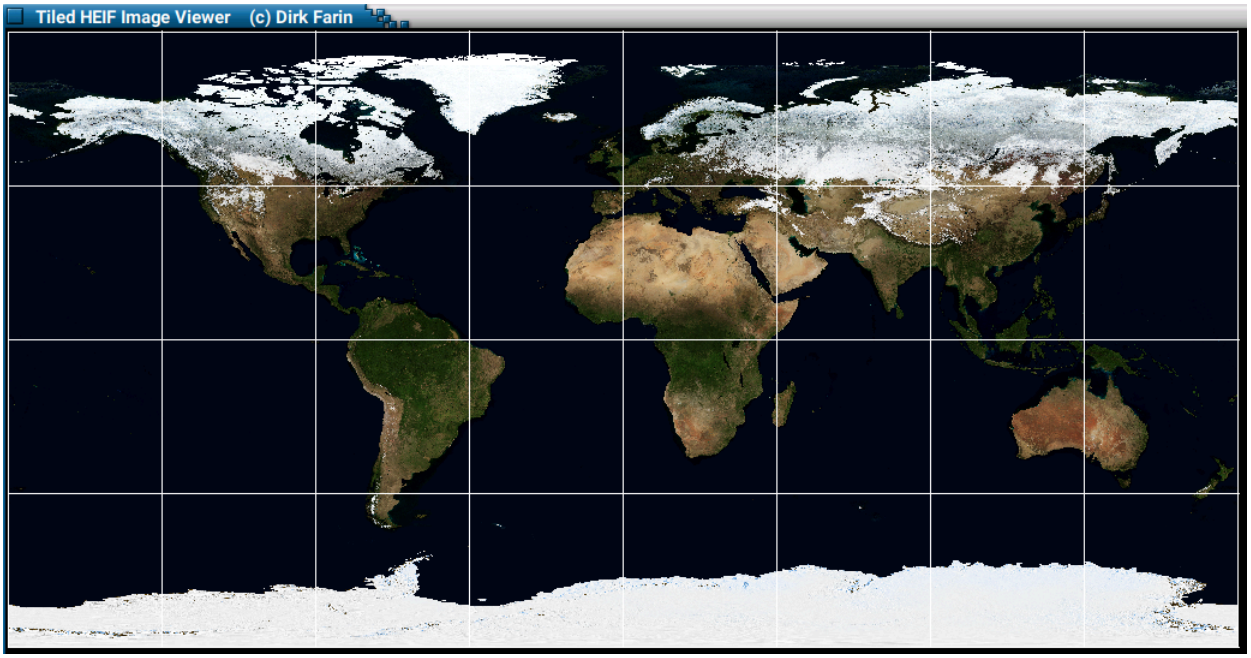


Figure C.1 — Visualizing HEIF output from GNOSIS Map Server for NASA Earth Observatory's *Blue Marble Next Generation* in Dirk Farin's Tiled Image Viewer

C.6. Challenges

Ecere faced some challenges related to the use of relatively recent C++ features (from C++ 20, which was still considered experimental in GCC at the time of the initiative) in libheif which required upgrading development toolchains across a cross-platform Continuous Integration / Continuous Delivery/Deployment (CI/CD) build environment producing binaries targeting a large variety of runtime platorms.

In particular, an obscure compiler bug in the latest TDM-GCC compiler was difficult to track down as the cause of odd crashes.

Ecere also contributed fixes allowing to include libheif in C and eC source files without errors or warnings and work-arounds allowing to build libheif with GCC 8 which supports the necessary C++ 20 features using the `-std=c++2a` compilation flag (issues #1398 as well as pull requests #1400 and #1429).

Not being deeply familiar with or having access to the MPEG / ISO HEIF standards further complicated the development. For bytes to be written in the correct order, endianness had to be accounted for when encoding a raw property into a HEIF context.

Later clarifications and improvements to the D010 GIMI Specification eased these difficulties by relying less on the assumptions of familiarity of the reader with the ISO HEIF standards.

Example usage in `heif-enc`, on the libheif Wiki page for tiled images, as well as guidance and help from other testbed participants greatly facilitated completing the implementation of HEIF output with tile pyramids and georeferencing.

C.7. Future work

Ecere plans to finalize multiband coverage support once this capability is available using the libheif API.

Client-side support for visualizing and importing GeoHEIF data is also planned.

Ecere also plans to implement support for the `compression`, `tile-size` and `overviews` parameter for GeoTIFF output, therefore enabling support for Cloud Optimized GeoTIFF (COG). Additionally, Ecere still plans to implement support for JPEG-XL output for multiple OGC API standards: OGC API — Maps, Coverages, Tiles and DGGS.

D

ANNEX D (INFORMATIVE) GEOHEIF IMPLEMENTATION EXPERIMENT (GEOMATYS)

ANNEX D (INFORMATIVE) GEOHEIF IMPLEMENTATION EXPERIMENT (GEOMATYS)

Geomatys tested and benchmarked the reading of GeoHEIF files in a Java environment using two different readers: A pure-Java implementation, and a binding to the C/C++ GDAL native library. The reader and the binding were developed in the Apache SIS project and are presented in the OGC Testbed-20 GIMI Open Source Report OGC 24-042. This annex contains some observations collected during those developments.

The two following observations are discussed in separated annexes because they were subject to debates. These observations are applicable to many raster formats, not only GeoHEIF:

- Annex E about georeferencing using affine transforms; and
- Annex F about IEEE 754 Not-A-Number (NaN) values.

The following list contains minor observations about the GeoHEIF file format. Those observations do not necessarily require changes in the GeoHEIF draft specification, as they may be handled by documentation or as future works.

- ISOBMFF boxes support extending the format with ease. Future enhancements should be possible without breaking the core parsing.
- The different box definitions are located in many different specifications such as ISO-14496, ISO-23001, ISO-23008 and NGA.STND.0076_1.0_GIMI. A complete implementation would need several more standards to achieve full support. Finding and understanding the relationship between some boxes require an effort that may be an impediment for open-source projects with minimal development resources.
- For object-oriented programming languages, representing each ISOBMFF box by a class is a strategy that works well.
- ISOBMFF provides `ComponentDefinition` and `PixelInformation` boxes for describing bands (coverage sample dimensions). However, the provided information does not include a transfer function or multiple no-data values.
- `PrimaryItem` can only reference a single item. This restriction avoids the need to scan all entries (i.e., helps to exclude the tiles and pyramid levels), but it is unclear whether it supports an efficient storage of multiple resources (rasters) in a single file.

The following sub-sections contain observations about libraries and Java or C/C++ APIs.

D.1. Development effort

D.1.1. Two approaches for modular development

GDAL and Apache SIS have different development strategies. GDAL is a translation layer between a unique API and various independent libraries. This approach allows GDAL to support a large number of formats and allows a large part of the work to be delegated to other teams such as the `libheif` developers. By contrast, Apache SIS supports a smaller number of formats but often implements these formats itself rather than delegating to an external library. This approach looks like a much larger effort, duplication of work and risk of bugs. But actually, GeoTIFF, GeoHEIF and many other chunked formats have many fundamental principles in common. The formulas for translating areas of interest and subsampling into tile indices, scanline strides, pixel strides and band indices are the same, even if the way to use this information is different. Decompression algorithms such as “deflate” and JPEG operate on arbitrary streams of bytes, and therefore are either already format-independent, or can be made format-independent in the cases of algorithms originally developed specifically for GeoTIFF. The algorithms for caching tiles and assembling them in larger images are also the same.

D.2. Mapping user requests

GDAL and Apache SIS also differ in how they respond to requests. GDAL tries to return exactly the image subregion requested by the user, which may imply cropping and resampling the image if necessary. By contrast, Apache SIS takes the liberty of returning more data than requested, or with a different subsampling. For example, if an area of interest begins and ends in the middle of a tile:

- GDAL will crop the returned image to the exact requested area; and
- Apache SIS may return a larger image in order to contain only complete tiles (except if the image is a single huge tile, in which case SIS will crop like GDAL).

The rational for Apache SIS’s strategy is that it supports directly reusing the cached tiles, without copy operations. The difference between what the users requested and what they got is specified by the pixel coordinates of the upper-left corner. As a rule, the pixel coordinates (0,0) always specify the beginning of a request. If the result of the request contains more data than requested, then, the image upper-left corner will have negative coordinates. Conversely, the coordinates can also be positive if the user’s request is partially out of bounds, in which case Apache SIS returns less data than requested. This strategy is transparent when using the standard Java2D API as in Listing E.7, because Java2D automatically takes in account the upper-

left pixel coordinates both at rendering time and for analytic work using `Raster.getPixel(…)` methods.

D.3. Error handling

Apache SIS registers an error handler with `CPLSetErrorHandler(…)` for collecting GDAL warnings and non-fatal errors. Those warnings are redirected to the listeners registered by users in Java if any, or to Java loggers otherwise. It works for all categories of GDAL error except `CE_Fatal`. The latter can be handled in a particular way, through C++ exceptions or `longjmp(…)`, but these two technics are not accessible with the binding framework used in this Testbed.

E

ANNEX E (INFORMATIVE) GEOREFERENCING BY AFFINE TRANSFORMATIONS (GEOMATYS)

E ANNEX E (INFORMATIVE) GEOREFERENCING BY AFFINE TRANSFORMATIONS (GEOMATYS)

NOTE 1: Most of the content of this section is not specific from the GeoHEIF GIMI format and is of general interest to avoid confusion in the axes order in geospatial grid coverages.

GeoHEIF, GeoTIFF, WorldFile and GML in JPEG 2000 (among others) use affine transforms for mapping pixel coordinates to real world coordinates in georectified images. This approach is significantly different from another frequently used approach based on bounding boxes. The bounding boxes approach is often promoted as simpler than the affine transforms approach because easier to understand. However, this simplification comes at the cost of less information. For example, a two-dimensional bounding box is described by 4 independent numbers instead of 6 numbers for a two-dimensional affine transform. In a rectangle described by 4 numbers, the only room left for information such as axis order and directions is 2 bits (one bit per dimension). Those 2 bits come from the possibility to store a "lower" coordinate value which is greater than the "upper" coordinate value, but this is rarely seen in practice except for wraparound axes (the latter is not discussed in this section).

Since axis order and directions cannot be stored in a bounding box alone (if we consider 2 bits as insufficient), and if the information is not stored in additional data structures such as a permutation array, then the missing information needs to be "hard-coded" by arbitrary rules in the specification. For example, the specification may force the axis order to (east, north). However, the latter is ambiguous with any CRS having axis directions that cannot be mapped to one of the cardinal directions.

In contrast, affine transforms, when used correctly, are a strictly mathematical solution leaving no room for ambiguity. Sometime, affine transforms are perceived ambiguous because of a confusion about which step the transform is describing (Annex E.2.2). This annex aims to compare the two approaches, explains the ambiguities of the bounding boxes approach, and proposes a step-by-step guide for unambiguous georeferencing by affine transforms.

NOTE 2: Georectified image or raster is a grid with regularly spaced cells in a geographic (i.e., latitude/longitude) or map coordinate system defined in a CRS so that any cell in the grid can be geolocated given its grid coordinate and the grid origin, cell spacing, and orientation.

Georeferencable image or raster is a grid with irregularly spaced cells in any given geographic/map projection coordinate system, whose individual cells can be geolocated using geolocation information supplied with the data but cannot be geolocated from the grid properties alone.

E.1. Referencing by bounding boxes

In OGC API — Common — Part 2: Geospatial Data (draft) and in OGC Coverage Implementation Schema (CIS) models, the mapping from pixel coordinates to real world coordinates is specified by the following information:

- the Coordinate Reference System (CRS),
- the *real world* bounding box in CRS units,
- the *grid* bounding box in pixel units (integers), and
- sometime the resolution (partially redundant with the above).

In OGC API — Common, the grid bounding box starts at (0,0) and only the number of pixels is encoded. Another variant is to replace the real world bounding box by a single corner and make the resolution mandatory. While this annex focuses on the above set of properties, the discussion is applicable to such variants as well.

The resolution is a partially redundant information because it can be derived from the bounding boxes. But it is not fully redundant because, in OGC API — Common and in Coverage Implementation Schema (CIS) models, the resolution depends on whether the "real world" bounding box contains the whole area of all pixels, or only the pixel centers. This is illustrated in the figure below. On the left side, the bounding box (in yellow) contains the full pixel areas, while on the right side, the bounding box contains only the pixel centers.

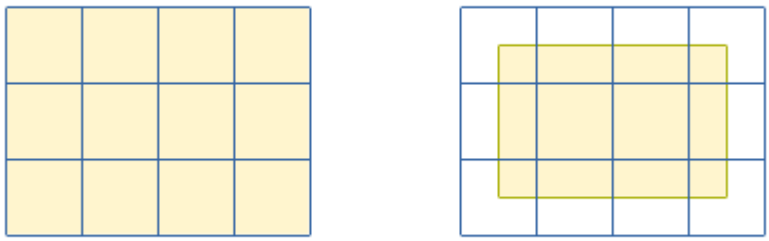


Figure E.1 — Bounding box containing pixel areas (left) or pixel centers (right)

In addition to whether the *real world* bounding box (in CRS units) contains pixel areas or pixel centers, it is necessary to specify whether the *grid* bounding box (in pixel units) has inclusive or exclusive bounds. In the CIS model, both the lower and upper bounds are inclusive. In such case:

$$cells\ count = (upper_{grid} - lower_{grid}) + 1$$

NOTE: Another frequently used convention is to declare the lower bounds as inclusive and the upper bounds as *exclusive*. With that convention, *cells count* = $upper_{grid} - lower_{grid}$ (without +1). In the common case where $lower_{grid}$ is zero, the $upper_{grid}$ value

become directly the number of cells. That convention is sometime useful, as seen in Figure E.2. This is the convention used by OGC API — Common (draft), contrarily to CIS.

With the CIS convention that all bounds are inclusive, the resolution is computed as below in the "pixel areas" case:

$$resolution = (upper_{world} - lower_{world}) / (upper_{grid} - lower_{grid} + 1)$$

and as below in the "pixel centers" case (right side of above Figure E.1):

$$resolution = (upper_{world} - lower_{world}) / (upper_{grid} - lower_{grid})$$

Those differences matter when using OGC API — Common (draft) or OGC Coverage Implementation Schema (CIS), because both "pixel is point" (center) and "pixel is area" interpretations are used. Which interpretation is implied can be detected by checking which one of above formulas gives a result that match the values declared in CIS metadata. Client applications sometime need to infer this information for reasons given in Annex E.1.1. If the CIS metadata match none of above formulas, it can be interpreted as if each value is representative of an area somewhere between the point and the full pixel area, with a span between 0 and 1 pixel. The "pixel is point/area" interpretation may be different for each dimension.

The following JSON snippets describe the same georeferencing of the same raster, but using two different OGC encoding, both based on bounding boxes. The fist snippet below is derived from OGC API — Common — Part 2 (draft). This snippet uses the "pixel is point" (center) interpretation because $(upper_{world} - lower_{world}) / resolution = cells\ count - 1$:

```
"extent" : {
      "spatial" : {
         "bbox" : [ [ -180, -90, 180, 90 ] ],
         "grid" : [
            {
               "cellsCount" : 65537,
               "resolution" : 0.0054931640625
            },
            {
               "cellsCount" : 32769,
               "resolution" : 0.0054931640625
            }
         ]
      }
   }
```

Listing E.1 — Raster georeferencing example using OGC API — Common — Part 2 (draft)

The following snippet describes the same thing using OGC Coverage Implementation Schema (CIS), except that the CRS is explicitly specified as `EPSG:4326` instead of implicitly assumed `OGC:CRS84`. As a reminder, note that `upperBound` is inclusive:

```
{
   "type" : "DomainSet",
   "generalGrid" : {
      "type" : "GeneralGridCoverage",
      "srsName" : "http://www.opengis.net/def/crs/EPSG/0/4326",
      "axisLabels" : [
         "Lat",
         "Lon"
      ],
```

```
      "axis" : [
         {
            "type" : "RegularAxis",
            "axisLabel" : "Lat",
            "lowerBound" : -90.0,
            "upperBound" : 90.0,
            "uomLabel" : "deg",
            "resolution" : 0.0054931640625
         },
         {
            "type" : "RegularAxis",
            "axisLabel" : "Lon",
            "lowerBound" : -180.0,
            "upperBound" : 180.0,
            "uomLabel" : "deg",
            "resolution" : 0.0054931640625
         }
      ],
      "gridLimits" : {
         "type" : "GridLimits",
         "srsName" : "http://www.opengis.net/def/crs/OGC/0/Index2D",
         "axisLabels" : [
            "i",
            "j"
         ],
         "axis" : [
            {
               "type" : "IndexAxis",
               "axisLabel" : "i",
               "lowerBound" : 0,
               "upperBound" : 32768
            },
            {
               "type" : "IndexAxis",
               "axisLabel" : "j",
               "lowerBound" : 0,
               "upperBound" : 65536
            }
         ]
      }
   }
}
```

Listing E.2 — Raster georeferencing example using OGC Coverage Implementation Schema (CIS)

In those two snippets, the *"pixel is point/area"* interpretation is not encoded explicitly and must be inferred indirectly if that information is needed (Annex E.1.1). In order to separate some assumptions behind the "pixel is point/area" interpretations, it is useful to note that the formulas relating the *resolution* to the *bounding boxes* (Annex E.1) can be reformulated in terms of whether $upper_{grid}$ is inclusive or exclusive. Figure E.2 below is the same as Figure E.1, but with pixels in lower/upper bounds colored in red.

1. When using the *"pixel is area"* interpretation (left side), the real world bounding box can be computed by considering that the linear relationship from *pixel coordinates* (expressed as integer values) to *CRS coordinates* maps pixel *upper-left corners*, and that the grid upper bounds are *exclusive* (the corresponding pixels are outside the yellow box).

2. When using the *"pixel is point"* interpretation (right side), the real world bounding box can be computed by considering that the linear relationship from *pixel coordinates* (expressed as integer values) to *CRS coordinates* maps pixel *centers*, and that the grid upper bounds are *inclusive*.

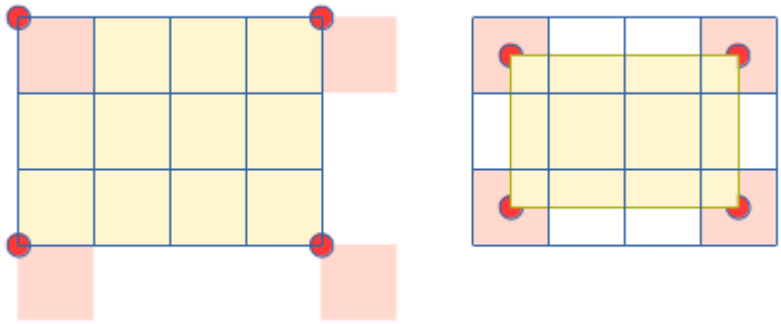


Figure E.2 — Pixel areas (left) versus pixel centers (right) reinterpreted as bounds inclusiveness

The key benefit of this reformulation is to separate georeferencing aspects from measurement aspects, as the reformulation replaced "pixel is point/area" by "pixel center/corner" + bounds inclusiveness. The *"pixel is point/area"* concept fundamentally describes whether the measurement is representative of the whole pixel area (for example, infrared radiance) or of a smaller region close to pixel center (for example, time needed for a radar pulse to travel out and back). In other words, "pixel is point/area" gives an information about the measurement **footprint**. By contrast, the *"pixel is center/corner"* concept is only about whether there is an offset of 0.5 pixel in the formulas that georeference the **grid**, independently of the footprint of the measurement inside each grid cell.

The real world bounding box given in OGC API — Common (Listing E.1) and in CIS (Listing E.2) is conceptually the smallest box containing all measurement footprints. This design choice makes testing whether the raster's data intersects the bounding box of a query easier, but complicates other operations as explained in Annex E.1.1. The "pixel is point/area" concept has the *appearance* of being a georeferencing information because, in the "pixel is point" case, the bounding box enclosing all measurement footprints is conveniently computed using the "pixel center" convention as illustrated on the right side of Figure E.2. Therefore, it is tempting to consider "pixel is point" as synonymous with "pixel center", but they are not. The next subsection explains how those two concepts are sometime in agreement and sometime have opposite values.

E.1.1. Why "pixel is point/area" are not synonymous of "pixel center/corner"

Consider the process of resampling a source raster in one CRS into a destination raster in another CRS. Let *T* be the coordinate operation for the whole chain of transforms, from pixel coordinates in the source raster to pixel coordinates they should have in the destination raster, with a map projection in the middle if desired. The resampling process will iterate over all pixels

to fill in the *destination* raster and, for each destination pixel, use the *inverse* of the coordinate operation (T^{-1}) for computing the pixel coordinates where to look for a value in the source raster. The destination pixel coordinates will typically be integers (represented in the figure below by circles in pixel centers), since the process iterates over these pixels. But the corresponding source pixel coordinates (represented in the figure below by arrows) can be anywhere. They will usually be non-integers, thus requiring interpolations between two or more source pixel values.

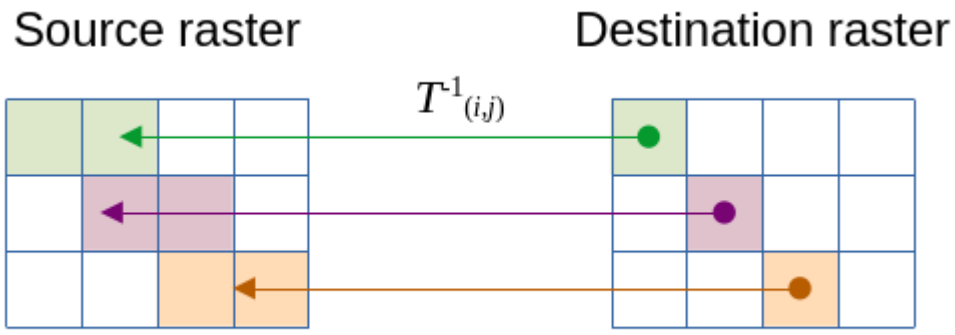


Figure E.3 — Resampling a raster with a coordinate operation *T* from source pixels to destination pixels

Because bilinear or bicubic interpolations are computed between points, the resampling process needs to choose which point in cell will be used. Should $T_{(i,j)}$ transforms from pixel center to pixel center, or from pixel corner to pixel corner? For the "pixel is point" footprint, mapping pixel centers seems a natural choice. However, **also** for the "pixel is area" footprint, mapping pixel centers is desired. If a measurement is the average value of the whole pixel area, then when the area of a destination pixel covers the areas of two source pixels, the destination pixel should be the average of the two source pixels. This is illustrated on the left side of the figure below, with the yellow color representing the average of the green and red colors.

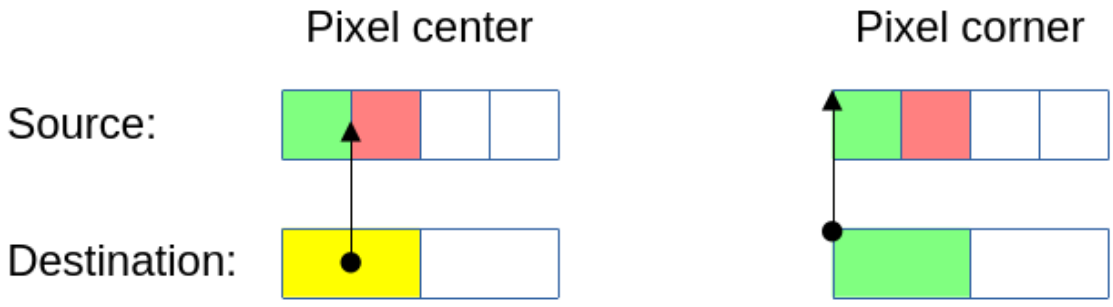


Figure E.4 — Linear interpolation of pixel values with different "pixel center/corner" conventions

Computing the average of two values is mathematically equivalent to a linear interpolation evaluated midway between these two values. Therefore, mapping pixel centers produces the desired effect with linear interpolation for both "pixel is point" and "pixel is area" measurement footprints. If the transform was mapping pixel corners instead, as illustrated on the right side

of above figure, the interpolation result would be the value of the leftmost pixel instead of the average value.

NOTE: The case illustrated on the right side of Figure E.4 could be “fixed” by adding an offset after the source pixel coordinates have been computed. But the offset to add is not trivial to compute, especially if the transform is non-linear. In the latter case, the offset would be different for each pixel. It is much easier to use the “pixel center” convention from the beginning.

A resampling process may also need to estimate the real world bounding box of the result. For the sake of simplicity, consider pixels with a resolution of 1° in longitude. If the image width is 4 pixels, the real world width may be 4° or 3° of longitude depending on which side of Figure E.1 is applicable. Lets assume a coordinate operation scaling the resolution to 2° of longitude, as illustrated in Figure E.4. When using the “pixel corner” convention (i.e., the strategy illustrated on the left side of Figure E.2), the green box on the left side of the figure below is transformed to the green box on the right side. The result is 2 pixels in width and is aligned with the new grid. But when using the “pixel center” convention, the yellow box on the left side become the yellow box on the right side. The result has a width of 1.5 pixels and is not aligned with the grid, i.e., the bounding box starts at -0.25 pixel relative to the new pixel centers (red circles).

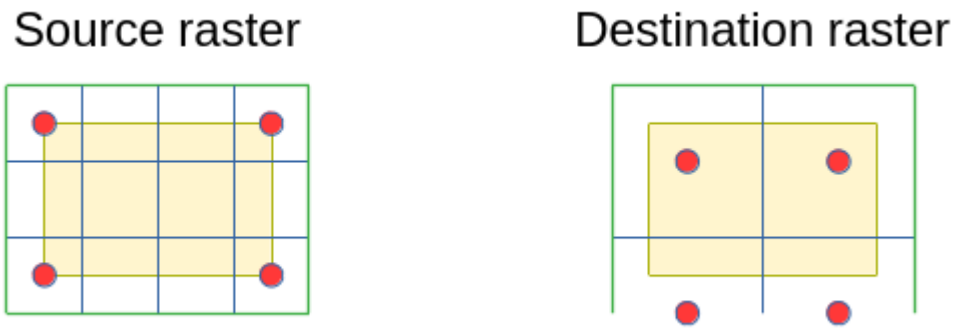


Figure E.5 — Bounding box transformation using "pixel corner" (green) and "pixel center" (yellow) conventions

It may be argued that for a “pixel is point” footprint, the yellow bounding boxes are physically more correct. However, for some purposes such as building an image pyramid, the green bounding boxes have more convenient properties. Therefore, the use of a “pixel corner” convention can be legitimate even with a “pixel is point” footprint.

E.1.1.1. Summary

The “pixel is point/area” concept is an information about the **measurement footprint**, which should be independent of the georeferencing of the **grid**. When mapping pixel coordinates to real world coordinates, the choice of whether to map pixel centers or pixel corners depends on the **calculation** to be performed, not necessarily on the measurement footprints. For example, a raster resampling process will typically want to map pixel corners for computing the destination bounding box (Figure E.5), then pixel centers for interpolating the pixel values (Figure E.4), regardless the measurement footprints. On the contrary, a process calculating the data bounding box (the smallest bounding box containing all footprints) may want to use a “pixel center/corner”

convention that corresponds to the “pixel is point/area” footprint (Figure E.2). But the latter equivalence is a special case.

The real world bounding box given in OGC API — Common (Listing E.1) and in CIS (Listing E.2) metadata is the above-cited data bounding box. The “pixel is point/area” footprint is embedded in the calculation of that bounding box for search convenience. But the same bounding box is also used for defining the linear relationship from pixel coordinates to real world coordinates, through the $lower_{world}$ and $upper_{world}$ terms in the equations of Annex E.1.2, thus establishing an implicit equivalence between “pixel is point/area” footprint and “pixel center/corner” convention. This equivalence can be convenient for calculations involving data bounding boxes, but not for all calculations. For example, a raster resampling process may be forced to “reverse engineer” the OGC API — Common or CIS metadata in order to retrieve which “pixel center/corner” convention is implied, for allowing the process to switch to the convention needed for its calculation.

E.1.1.2. Alternative

The problem described in this section does not exist when referencing by the affine transform approach (Annex E.2). This is because that approach does not use a “real world” bounding box. Such bounding boxes are nevertheless useful for search operations, but they are considered as *discovery metadata* clearly separated from georeferencing metadata, Therefore, users are free to compute the box in whatever way is best for reflecting the measurement footprints, with no side-effect on the raster georeferencing. The footprints could be arbitrary geometric shapes and could be different for each pixel. This flexibility was allowed in the old ISO 19123:2005 Standard with `CV_FootPrint`.

E.1.2. Ambiguities in axis order and directions

In the OGC API — Common (draft) and CIS models, the linear relationship from pixel coordinates *(i, j, k, m)* to real world coordinates *(x, y, z, t)* can be as below ($lower_{grid}$ is zero in the OGC API case):

$$x = lower_{world}[0] + (i - lower_{grid}[0]) \times resolution[0]$$
$$y = upper_{world}[1] - (j - lower_{grid}[1]) \times resolution[1]$$ (assuming inclusive upper bound)
$$z = lower_{world}[2] + (k - lower_{grid}[2]) \times resolution[2]$$
$$t = lower_{world}[3] + (m - lower_{grid}[3]) \times resolution[3]$$

Typically, only the two first dimensions are used in above list. This approach is generalizable to any number of dimensions, ignoring the ambiguity about whether other special cases like *y* may exist. However, this approach assumes that the real world coordinates are in the same order as the grid coordinates. This assumption is problematic and has been a great source of confusion for decades. This is because screen coordinates and geographic coordinates have inherited different conventions from their history. Most raster producers and users want the *grid* (image) axes organized in such a way that the image appears with a familiar orientation (e.g., north toward up) when shown in a web browser or other non-geospatial software. Since screen coordinate systems traditionally use *(right, bottom)* axis orientations in that order, the

corresponding CRS axis orientations would be *(east*, *south*). However, many geographic CRS use *(north*, *east)* axis orientations instead (there is also some west-orientated and south-orientated geographic CRS). Not only the axis order differs, but the north-south orientation also differs. The "georeferencing by bounding boxes" approach usually tries to address this mismatch with hard-coded rules.

E.1.2.1. Axis orientation

A specification may required to reverse the direction of the *y* axis in order to have north on top. However, nothing in Listing E.1 or Listing E.2 tells users that a direction needs to be flipped. For example, axis flips could have been specified explicitly by negative scale factors, but all values are positive in the closest OGC API Common or CIS equivalent concept (the resolutions). Therefore, an axis reversal rule is implicitly handled as an exception to the general pattern of above formulas (the *y* case). What to do if no CRS axis is oriented toward north or south is unclear and may be interpreted as an exception to the exception.

E.1.2.2. Axis order

When the CRS is specified by an authority code, the axis order is defined by the authority. If that CRS axis order does not correspond directly to the visualization axis order expected by the grid, then the formulas listed in Annex E.1.2 cannot be applied. Some standards such as GeoTIFF addresses this problem by explicitly overriding the authority's definition with a requirement saying that CRS axis directions shall be *(east*, *north)* in that order, no matter what the authority said. **OGC directive #14 does not recommend this practice**, because it is ambiguous when axes have other directions. For examples, what an application should do with the *north-north-west* direction? Or with projections at the North pole where all axes are oriented toward south? Or with engineering CRS having *(forward*, *starboard)* axis directions?

OGC 08-038 (OGC directive #14) identifies 6 ways to handle axis order. The directive acknowledges that overriding CRS definitions is a common practice in legacy OGC standards (case 4 in the OGC 08-038 document) or in community standards defined outside OGC (case 6). However, the policy requires that future versions of OGC standards (case 5) SHALL, as far as possible, be made compliant with one of the recommended ways to handle axis order. The recommended ways listed in OGC 08-038 are cases 1, 2 and 3.

1. *Axis Order is defined by the CRS specified in the interface, protocol, or payload.* It can be, for example, a full CRS definition using an encoding such as Well-Known Text (WKT) or Geographic Markup Language (GML). Since these formats contain explicit axis declarations, rasters can declare the axis order of their choice.

2. *Payload encoding explicitly overrides the Axis Order as specified in the metadata.* For example, the ISO 19107:2019 standard defines metadata with a `permutation` attribute which specifies explicitly how axes shall be reordered. This is the approach used implicitly (with no need for a permutation attribute) by affine transforms.

3. *Payload includes metadata for a derived CRS or a local CS transformation setting Axis Order.* This is based on the `DerivedCRS` construct of OGC Topic 2 / ISO 19111:2019. This approach is not used in this annex.

Georeferencing by bounding boxes tends to encourage CRS overriding and to do that in a way that conflate image orientation with CRS orientation, for the convenience of the formulas listed in Annex E.1.2. The overriding could be made unambiguous with an explicit `permutation` attribute similar to what ISO 19107:2019 does (case 2), but this practice is not yet widespread in OGC standards. CRS definitions in WKT or GML could in principle resolve the problem (case 1), but users sometime want to keep the CRS identifier (`ID[authority, code]` in WKT), which requires staying in compliance with the authority definition. Unfortunately, not only the ambiguous overriding of CRS definitions are still widespread (ambiguous because applied implicitly according heuristic rules that do not cover all cases), but new standards such as GeoPackage perpetuate this ambiguity on the argument that they are following *de facto* standards set by previous specifications, without mentioning the directive #14's recommendations.

E.1.2.3. Alternatives

For geometry data, the easiest way to resolve axis order ambiguity may be to require an explicit `permutation` metadata inspired from ISO 19107:2019. For raster data, an explicit `permutation` attribute could also be added in a revision of the OGC API or CIS Standards. However, this would add complexity to already not-so-simple standards. By contrast, georeferencing by affine transforms have axis permutations embedded in their mathematics (Annex E.2.1). The affine transform approach resolves the axis order problem with no additional complexity.

E.2. Referencing by affine transforms

Affine transforms are extensively used in the computer graphics industry and in formats such as PostScript and PDF. They are also used by ISO/IEC 14496-12:2022 (the format used as a container for GeoHEIF) for the transformation of video images. Affine transforms are core to standard libraries such as OpenGL and Java2D for drawing on a screen, and have hardware support in the GPUs. They are one of the first topics taught in computer graphics. For example, the Introduction to computer graphics online book (retrieved in October 30, 2024) introduces those transforms in §2.1 (immediately after the introduction chapter) without calling them "affine", then introduces them more formally in §2.3. Another example is *Principles of interactive computer graphics* (Newman and Sproull) published in 1979. For an example of how closely affine transforms are integrated in standard graphics libraries, see Listing E.7.

Affine transforms are also used extensively for geospatial information, but not always in a form that makes users realize that they are handling affine transforms. This is similar to §2.1 in the above-cited introductory book on graphics, which talks about affine transforms without calling them by their name. Some examples of affine transform usages are as follow.

- Apache SIS way to georeference rasters:

The `GridGeometry.getGridToCRS(PixelInCell)` method usually returns an affine transform (not always, because Apache SIS supports also non-linear georeferencing) with an arbitrary number of dimensions. The transform can map "pixel center" or "pixel corner" at user's choice (specified in argument), because this choice depends on the calculation rather than the data (Annex E.1.1.1).

- GDAL way to georeference rasters:
  The `GDALDataset::GetGeoTransform()` function always returns a two-dimensional affine transform mapping pixel corners.

- WorldFile format:
  The content of the `.tfw` file is an affine transform. The coefficient values in these files can be used verbatim by above-cited Apache SIS and GDAL APIs.

- EPSG:9624 — Affine parametric transformation:
  Not used for rasters, but this is the same mathematical operation.

- OGC Geodetic data Grid eXchange Format (GGXF) (v1.0):
  This format uses an affine transform for georeferencing the grid, as described in GGXF §5.6.1.

- ISO 19123-2:2018 — Schema for coverage geometry and functions — Part 2: Coverage implementation schema:
  The `<gml:origin>` and `<gml:offsetVector>` elements of `RectifiedGrid` are an affine transform in which the translation column is stored in a separated element (the origin).

- OGC GML in JPEG 2000:
  Same as above (Annex E.2.1.4).

- OGC GeoTIFF:
  The `ModelTransformationTag` value is an affine transform with some particular rules (Annex E.2.1.5).

- • Coordinate Transformation Services — OLE/COM (OGC 01-009) introduces Jacobian matrices, but without calling them "Jacobian". Jacobian matrices are closely related to affine transforms in contexts where the coordinate operation is non-linear, such as a map projection. This OGC standard was published in 2001, therefore demonstrating that this concept is not new.

An affine transform can be used for the precise location of a **georectified** raster. It can also be used as an approximation of the location of a **georeferenceable** raster. For example, the affine transform coefficients may be the best fit (e.g., calculated by the *least square* method) of a collection of tie-points. In the OGC Topic 2 / ISO 19111 model, such "best fit" coordinate operations are called **transformations**, while coordinate operations that are exact by definition are called **conversions**. ISO 19111:2019 actually defines those concepts in terms of datum changes, but this is closely related to whether stochastic errors exist. Therefore, this annex adopts the following terminology.

- Affine transforms used for georectified rasters are called *affine conversions*.

- Affine transforms used for georeferenceable rasters are called *affine transformations*.

- *Affine transforms* is used for the generic case covering both conversions and transformations.

Above terminology is specific to this annex and not widely adopted. In a GeoHEIF file, those two cases can be distinguished as below.

- Rasters with a `ModelTransformationProperty` only are georectified.
- Rasters with a `ModelTiepointProperty` are georeferenceable.

In the latter case, finding an affine approximation (if desired) is left to the implementation. Alternatively, if both tie-points and affine transform are specified, the latter could be considered a precomputed approximation of the former, in which case the raster would still be considered georeferenceable. This annex covers only the case of georectified rasters.

E.2.1. Mapping grid axes to CRS axes

This section provides a step-by-step guide for defining an affine conversion in replacement of the "georeferencing by bounding boxes" approach. For the sake of simplicity, the example used in this section is a two-dimensional raster written in the GeoHEIF format with *(latitude, longitude)* CRS axes and with the (0,0) pixel coordinates located in the upper-left corner. However, the steps described here are generalizable to any number of dimensions, any grid corner location, any axis order and orientation, and any raster formats capable to store affine transforms.

The simple case shown in this section goes as follows.

- In GeoHEIF's `mcrs` box, `crsEncoding` is set to `curi`.
- In GeoHEIF's `mcrs` box, `crs` is set to "EPSG:4326".
- In GeoHEIF's `mcrs` box, `epoch` is unset (bit 1 of `flags` is zero).
- "Pixel is point/area" is out of scope of the referencing process (Annex E.1.1).
- Georeferencing uses "pixel corner" convention as required by the `mtxf` box specification.
- The numbers of pixels in each dimension are the image size.

The affine conversion can be built in a two-steps process as follow.

E.2.1.1. Step 1: conversion between aligned axes

The first step is to write the matrix as if the CRS axes were aligned with the image axes. For the example in this section, it means that the geographic coordinates and the resolutions are in *(longitude, latitude)* order in this step despite the authoritative EPSG:4326 axis order. Then:

- Start with an identity matrix of size 3×3;
- Put the geographic coordinates of the upper-left image corner in the last column;
- Put the resolutions on the diagonal; and
- Make the resolution of φ (or y) negative, because the image axis and CRS axis are in opposite directions.

NOTE:The above sub-steps can be reformulated in a more generic way applicable to any number of dimensions and any grid corner location.

1. Start with an identity matrix of size (n+1)×(n+1) where *n* is the number of dimensions.
2. Put the CRS coordinates of the grid origin — i.e., the cell at grid coordinates (0,0,…,0) — in the last column. The CRS coordinates are those of the corner in the direction of lowest **grid** coordinate values. Note that this is not necessarily the corner with lowest CRS coordinate values.
3. Put the resolutions on the diagonal.
4. If any grid axis has a direction opposite to the CRS axis, make the resolution (scale factor) negative on that row and add the image span (in CRS units) to the translation column in the same row.

In the case where the grid lower bound is zero, the value in the translation column is either $lower_{world}$ when the scale factor on the diagonal is positive, or $upper_{world}$ (inclusive) when the scale factor is negative.

For an image covering the world with a resolution of $\Delta\lambda$ degrees in longitude and $\Delta\varphi$ degrees in latitude, the result is the square matrix shown in the middle of the equation below. The column matrices on the left and right sides are shown only for providing context but are not encoded in GeoHEIF or other raster formats. Note that the upper-left corner is at +90°N rather than −90° because of the inversion of axis direction.

$$\begin{bmatrix}\lambda\\ \phi\\ 1\end{bmatrix}=\begin{bmatrix}\Delta\lambda & 0 & -180\\ 0 & -\Delta\phi & +90\\ 0 & 0 & 1\end{bmatrix}\times\begin{bmatrix}i\\ j\\ 1\end{bmatrix} \tag{E.1}$$

E.2.1.2. Step 2: axis swapping

The next step is to reorder axes according the authoritative CRS axis order. Because EPSG:4326 has *(latitude, longitude)* axis order, those two axes must be swapped. The rule is very simple: Just move the matrix rows as needed for having them in the same order as the CRS axes, without changing any value. The result is the square matrix shown in the middle of the equation below. The column matrices on both sides are again shown only for providing context. Note that the column matrix on the left side shall have the same row reordering as the middle matrix (thus resulting in CRS axis order), while the column matrix on the right side shall stay unchanged.

$$\begin{bmatrix}\phi\\ \lambda\\ 1\end{bmatrix}=\begin{bmatrix}0 & -\Delta\phi & +90\\ \Delta\lambda & 0 & -180\\ 0 & 0 & 1\end{bmatrix}\times\begin{bmatrix}i\\ j\\ 1\end{bmatrix} \tag{E.2}$$

Verification: the above matrix multiplication can be expanded as below:

```
#  =   0 × i  +  -Δϕ × j  +   90
λ  =  Δλ × i  +   0 × j  +  -180
```

Which can be simplified as below to a form similar to the equations in Annex E.1.2. Note that the $lower_{grid}$ subtractions do not appear explicitly but are embedded in the translation terms if the grid indices do not start at zero.

```
λ  =  -180  +  i × Δλ
#  =   +90  -  j × Δϕ
```

In a GeoHEIF file, the coefficients of the middle matrix are written in the `m00`, `m01`, *etc.* fields of the `mtxf` box in a row-major fashion (from left to right then from top to bottom). The last matrix row is omitted.

The fact that affine transforms can represent rotations is a bonus, but not the main reason why they are desirable. The most important reason is their unambiguity. No heuristic rules were involved in above steps. All information about axis swapping and direction reversal are embedded in the matrix, in the form of swapped rows and negative scale factors respectively. Some values are typically zeros in absence of rotation, but they are not meaningless: the positions of those zeros in the matrix carry an information about axis swapping.

E.2.1.3. From “pixel corner” to “pixel center” convention

The above-described steps define an affine transform which maps pixel corners. If a mapping to pixel centers is desired instead (left side of Figure E.4), this change of convention can be done simply by adding an offset of 0.5 pixel *before* to apply the above affine transform. Any translation can be concatenated to an affine transform as below.

- Start with an identity matrix of the same size.
- Put the desired (T_x, T_y, …) translation in the last column.
- Multiply the original matrix by above translation matrix.

In matrix multiplications, the order of operands matters. For applying the translation *before* the “Grid to CRS” transform (i.e., in *pixel* units), the translation matrix shall be on the *right* side. For applying the translation *after* the “Grid to CRS” transform (i.e., in *CRS* units), the translation matrix shall be on the *left* side. In other words, when a chain of affine transforms is represented by matrix multiplications, the transforms are applied from right to left. For the simple example used in this section, a translation of (T_φ, T_λ) in degrees of latitude and longitude would be as below:

$$\begin{bmatrix} 1 & 0 & T_\phi \\ 0 & 1 & T_\lambda \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & -\Delta\phi & +90 \\ \Delta\lambda & 0 & -180 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -\Delta\phi & +90+T_\phi \\ \Delta\lambda & 0 & -180+T_\lambda \\ 0 & 0 & 1 \end{bmatrix} \tag{E.3}$$

And a translation of (T_i, T_j) in raster's pixel units would be as below:

$$\begin{bmatrix} 0 & -\Delta\phi & +90 \\ \Delta\lambda & 0 & -180 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & T_i \\ 0 & 1 & T_j \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -\Delta\phi & +90-(\Delta\phi)T_j \\ \Delta\lambda & 0 & -180+(\Delta\lambda)T_i \\ 0 & 0 & 1 \end{bmatrix} \tag{E.4}$$

The above rule of thumb regarding the order of the operands in matrix multiplications works for all kinds of operation (scale, shear, rotation, flip, axis swapping) in any number of dimensions. For converting the simple example from "pixel corner" to "pixel center" convention, simply use the above equation with (T_i, T_j) set to (0.5, 0.5) pixel. With standard graphics libraries such as Java2D, this multiplication can be as simple as below:

```
AffineTransform gridToCRS = ...;     // Using "pixel corner" convention.
gridToCRS.translate(0.5, 0.5);       // Now using "pixel center" convention.
```

Listing E.3 — From "pixel corner" to "pixel center" with Java2D standard library

The above call to Java2D `translate` method is equivalent to building a translation matrix and using it as shown in Formula (E.4). Note that different graphics libraries use different conventions regarding whether the translation matrix will be the left or right multiplication operand. Libraries often provide the choice between the two possibilities, but different libraries may use different (sometime contradicting) naming conventions.

E.2.1.4. GML in JPEG2000 case

GML in JPEG2000 (derived from ISO 19123-2) splits the affine transform into two components: the offset vectors and the origin. These two components can easily be assembled into the matrix form used by this annex:

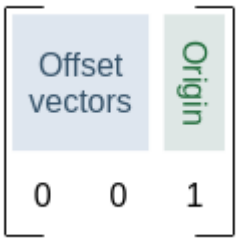


Figure E.6 — Offset vectors and origin in an affine transform

That "offset vectors + origin" way to specify the conversion is as good as the matrix form used in this annex. The separation between offset vectors and origin is sometime convenient in explanatory texts. For example, for saying that there is a strict correspondence between grid

axes and CRS axes (no shear, rotation or axis swapping, but potentially axis direction reversal), the two following sentences are equivalent.

- The matrix is diagonal, ignoring the last column.
- The offset vectors part of the matrix is diagonal.

For specifying that axis swapping are authorized but not shears or rotations (except by multiple of 90°), the two following sentences are equivalent.

- Each matrix row and each matrix column shall contain exactly one non-zero value, ignoring the last column.
- Each offset vector shall contain exactly one non-zero value, and the index of that non-zero value shall be different for all offset vectors.

With the above rule, the interpretation of the *offset vectors* part of the matrix is as follows. For any row *j* having exactly one non-zero value in column *i*, that value is the scale factor no matter in which column it appears. Index *j* is the CRS axis index and *i* is the grid axis index. The scale factor is multiplied by the grid coordinate at index *i* and the result is stored in the CRS coordinate at index *j*. When the scale factor is on the diagonal, the grid and CRS axes are at the same position *(j = i)*, i.e., these axes are aligned.

E.2.1.5. GeoTIFF case

The GeoTIFF's `ModelTransformationTag` value is an affine conversion with some heuristic rules. GeoTIFF fixes the matrix size to 4×4 (the size of a three-dimensional operation) even in the two-dimensional case. This encoding rule may be a source of confusion, as it gives the impression that affine transforms are a concept specific to 3D graphics and unnecessarily complicated for the 2D case. When using the `ModelTransformationTag` matrix with a two-dimensional image, the encoded matrix should be reduced to a 3×3 matrix by removing the third row and the third column.

Another particularity is that the GeoTIFF specification overwrites the CRS definition with unconditional *(longitude*, *latitude)* axis order. This is not recommended as per OGC directive #14, but cannot be changed for historical reason. Therefore, the affine conversion encoded in `ModelTransformationTag` usually does not contain the axis swapping described in Annex E.2.1.2.

Additionally, the `ModelTransformationTag` affine conversion may apply either to the "pixel corner" or "pixel center" convention, depending on the `GTRasterTypeGeoKey` value. In principle, that key is for specifying the "pixel is point/area" footprint but is commonly interpreted as also synonymous with "pixel center/corner". This merge of two distinct concepts is not recommended (Annex E.1.1), but cannot be changed for compatibility reasons.

Alternatively, the combination of `ModelPixelScaleTag` and `ModelTiepointTag` is a way to specify a subset of affine conversion coefficients (only the translation column and the diagonal). However, the GeoTIFF scale factors also have a particularity in that the sign of the *y* scale factor is reversed (i.e., scale is positive when the raster's axis is oriented downward). Furthermore, the

tie point does not give directly the translation column if the *(i,j)* pixel coordinates of the tie point are not zeros.

E.2.1.5.1. Comparison with GeoHEIF

GeoHEIF differs from GeoTIFF in that it also uses an affine conversion, but without the above heuristic rules. In GeoHEIF, the matrix size matches the number of dimensions + 1, axis swapping is explicitly specified with the positions of matrix elements, and the affine conversion unconditionally maps pixel corners. An example of the consequences of above differences is as follow: If the same raster is encoded using EPSG:4326 and using OGC:CRS84, then:

- In GeoTIFF files, the affine transforms will be the exact same matrix because the GeoTIFF specification overwrites the CRS; and
- In GeoHEIF files, the affine transforms will have different matrices because the rows shall be swapped in the EPSG:4326 case.

E.2.1.5.2. Conversion from GeoHEIF to GeoTIFF

When converting a GeoHEIF file to a GeoTIFF file, the matrix of the affine transform should not be copied verbatim, because GeoHEIF and GeoTIFF interpret that transform differently. Because GeoTIFF said that the CRS definition shall be overwritten by a CRS with *(longitude, latitude)* axis order, it has a concept of *"effective CRS"* which differs from the *"real CRS"* (note that this complication does not exist in GeoHEIF). The process for converting a GeoHEIF matrix to a GeoTIFF matrix should be as shown below.

- Find the axis order of the GeoTIFF's "effective CRS" according to GeoTIFF heuristic rules. For EPSG:4326, the "effective CRS" is CRS:84.
- Because the transform in a GeoTIFF file shall map pixel coordinates to the "effective CRS" rather than the real CRS, a translator needs to reorder the matrix rows for matching the axis order of the GeoTIFF's *effective* CRS. A translator can process the image by finding which axes are swapped when going from the real CRS to the effective CRS and swapping the same rows in the matrix.

The following pseudo-code is one possible algorithm for axis reordering. For most geographic CRS, `matrixInGeoTIFF` should result to a diagonal matrix (ignoring the translation column).

```
int dimension = 2;          // Could be anything, this algorithm is generic.
double[][] matrixInGeoHEIF = ...;
double[][] matrixInGeoTIFF = new double[dimension][];
for (int inEffectiveCRS = 0; inEffectiveCRS < dimension; inEffectiveCRS++) {
    int inRealCRS = ...;    // Find the index in the real CRS having the same
axis direction.
    matrixInGeoTIFF[inEffectiveCRS] = matrixInGeoHEIF[inRealCRS];
}
```

Listing E.4 — Reordering axes from GeoHEIF to GeoTIFF

E.2.1.6. Note for implementers of GDAL drivers

GDAL versions older than GDAL version 3 assumed *(longitude, latitude)* axis order. Starting with GDAL version 3.0, some functions were added for managing different axis orders. The following functions are of particular interest for developers of GDAL drivers.

- `OGRSpatialReference::setAxisMappingStrategy(…)` for specifying whether to apply heuristic rules.
- `OGRSpatialReference::GetDataAxisToSRSAxisMapping()` for fetching the result of GDAL heuristic rules.

> ***WARNING***
>
> *The above functions address GDAL version 3 and later compatibility issues with previous versions. Geospatial libraries newer than GDAL should avoid introducing such functions in their APIs and/or API definitions. For example, Apache SIS relies entirely on the mathematics of affine transforms and does not add a "data axis to SRS axis mapping" on top of that.*

The default axis mapping strategy is `AUTHORITY_COMPLIANT`, which means that axis order is as defined by the authority. In the GeoHEIF case, it means that the axis order for EPSG:4326 shall be *(latitude, longitude)* because GeoHEIF does not overwrite the authoritative CRS definition (contrarily to GeoTIFF). For historical reasons, there is a risk that not every parts of the GDAL ecosystem handles such axis order issues correctly. This is because GDAL version 1 and version 2 existed many years before the introduction of above functions, and because overwriting the authoritative CRS definition with *(longitude, latitude)* axis order is a common practice in GeoTIFF and other formats. One way to detect a possible issue with axis order is to execute `gdalinfo` before and after a process (for example, `gdal_translate`) on a raster referenced to EPSG:4326. Before the process, the `gdalinfo` output may contain lines like below:

```
Data axis to CRS axis mapping: 1,2
GeoTransform =
  90, 0, -0.3515625
  -180, 0.3515625, 0
```

Listing E.5 — Example of GDAL information for a raster having axis swapping

If, after the process, the *"Data axis to CRS axis mapping"* become `2,1` without changes in the `GeoTransform` matrix, then some code somewhere applied heuristic rules in addition to what was intended to be pure mathematics. The consequence is an unexpected 90° rotation in the image result.

New GDAL drivers for "no heuristic" formats such as GeoHEIF should consider applying GDAL heuristics at reading time as below (replace `GeoHEIF` by the actual GDAL driver class name).

- Invoke `OGRSpatialReference::setAxisMappingStrategy(OAMS_TRADITIONAL_GIS_ORDER)` at CRS parsing time.

- Make sure that above method is invoked before `GeoHEIF::GetSpatialRef()` and `GeoHEIF::GetGeoTransform(…)`.
- Invoke `OGRSpatialReference::GetDataAxisToSRSAxisMapping()` for determinig how to modify the matrix returned by `GeoHEIF::GetGeoTransform(…)`. The *"Data axis to CRS axis mapping"* should give unambiguous instructions for matching whatever heuristic rules have been applied.

The following pseudo-code shows how the matrix can be modified. Null checks, bound checks and defensive copies are omitted for simplicity. This pseudo-code handles the multi-dimensional case as a matter of principle, but it can be simplified in the context of `GeoHEIF::GetGeoTransform(…)` because that transform is always two-dimensional.

```
GeoHEIF::InitializeCRS() {
    OGRSpatialReference crs = ...;          // Parse the CRS from the GeoHEIF
file.
    crs.setAxisMappingStrategy(OAMS_TRADITIONAL_GIS_ORDER);
}

GeoHEIF::GetGeoTransform(double *padfTransform) {
    OGRSpatialReference crs  = ...;         // Above CRS with OAMS_TRADITIONAL_
GIS_ORDER.
    int[]      dataAxisToSRS = crs.GetDataAxisToSRSAxisMapping()
    int[]      rasterSize    = ...;         // Raster width and height in pixels.
    double[][] matrixInHEIF  = ...;         // The matrix read from GeoHEIF.
    double[][] matrixInGDAL  = new double[dimension][];
    for (int j=0; j<dimension; j++) {
        int dataIndex = abs(dataAxisToSRS[j]) - 1;
        matrixInGDAL[dataIndex] = matrixInHEIF[j];      // Take a matrix row.
        /*
         * By GDAL convention, a negative value means that the axis direction is
reversed.
         * This implies that the scale factor shall be negated and the
translation changed
         * from minimum CRS value to maximum CRS value (if the scale factor was
positive).
         */
        if (dataAxisToSRS[j] < 0) {
            double span = 0;
            for (int i=0; i<dimension; i++) {
                double coefficient = matrixInGDAL[dataIndex][i];
                matrixInGDAL[dataIndex][i] = -coefficient;
                span += coefficient * rasterSize[i];
            }
            matrixInGDAL[dataIndex][dimension] += span;
        }
    }
    // Copy values to `padfTransform`, keeping in mind that GDAL puts
translation first.
}
```

Listing E.6 — Pseudo-code for applying GDAL heuristic rules on a referenced raster

Applying such heuristic rules at the level of the GDAL driver is okay, contrarily to embedding these rules in a standard specification such as GeoHEIF. This is because for whatever magic is executed behind `setAxisMappingStrategy(…)`, the driver can unambiguously adjust the affine transform matrix thanks to the information provided by `GetDataAxisToSRSAxisMapping()`, with no need to know on which criteria GDAL based its decision. By contrast, when the

heuristic rules are embedded in a specification such as GeoTIFF, the specification needs to describe the rules in detail, usually leaving gray areas for cases not well covered (e.g., CRS with uncommon axis directions). Any deviation from the standardized heuristic rules (e.g., for above-cited gray areas) introduces interoperability problems. When the heuristic rules and affine transform adjustments are applied together as in Listing E.6, the rules for axis order can be left at implementer's choice without causing interoperability problem.

> ***CAUTION***
>
> *The assertions about the GDAL libraries and technologies have not been independently verified by GDAL developers.*

E.2.2. Transforms chain from raster to screen

The affine conversion built in Annex E.2.1 is describing the conversion from raster coordinates to **CRS** coordinates, not to screen coordinates. In this section, the above-cited conversion is called the *Grid to CRS conversion*. For visualization, another step from CRS coordinates to screen coordinates is required. That second step can also be expressed as an affine transform and is called the *CRS to Display conversion* in this section. Those two conversions have clearly separated tasks.

- **Grid to CRS** depends only on the **data**. It is specified by the data producer and does not depend on the display device. This conversion is static: A user's actions such as zoom or pan have no effect on this conversion.
- **CRS to Display** depends only on the **display device** and **user's actions**. This conversion is chosen by the client application (viewer) and does not depend on the data, except for choosing the initial view area if the application wishes to do so. This conversion is dynamic: it is modified every time that the user zooms or pans.

Note that the *CRS to Display* conversion is the same for all data to be rendered on screen: The other rasters, the geometries, and all feature as long as they are using the same CRS. Note also that this conversion is always needed no matter how the data are georectified (by bounding boxes or by affine transforms). This is not a complication introduced by the "georeferencing by affine transforms" approach. The only difference is that this section will describe that conversion more rigorously using affine transforms.

For the first part of this section, the "CRS" term in "Grid to CRS" and in "CRS to Display" must refer to the exact same CRS, axis order included. If that CRS uses *(latitude, longitude)* axis order, then the "CRS to Display" conversion also needs to swap axes to the *(x, y)* order. The case where the CRS are not the same will be discussed in Annex E.2.2.3.

E.2.2.1. Mapping CRS axes to visualization axes

The “CRS to Display” transform can be constructed by a three-steps process very similar to the “Grid to CRS” case (Annex E.2.1). First, define a “Display to CRS” (note the reverse direction) matrix as if the CRS axes were aligned with the axes of the display device.

- Start with an identity matrix of size 3×3.
- Put in the last column the geographic coordinates of the upper-left corner of the region to visualize.
- Put in the diagonal the desired spans (in CRS units) of display device’s pixels.
- Make negative the Δy value on the diagonal because the display axis and the CRS axis have opposite directions.

Above sub-steps can be generalized to any number of dimensions in the same way as described for the “Grid to CRS” case. The case of projecting three-dimensional data on a two-dimensional screen is described in Annex E.2.2.4. In the simple case of the two-dimensional example used by this annex, if the client application wants to visualize a world-wide image in a window of size 800×600 pixels, then the initial matrix (before the users zooms and pans) is as below. Again, the column matrices on both sides are only for providing context.

$$\begin{bmatrix}\lambda\\ \phi\\ 1\end{bmatrix} = \begin{bmatrix}360/800 & 0 & -180\\ 0 & -180/600 & +90\\ 0 & 0 & 1\end{bmatrix} \times \begin{bmatrix}x\\ y\\ 1\end{bmatrix} \tag{E.5}$$

Next, the client application can check the directions of the CRS axes. From the EPSG:4326 definition, the client application can see that the first axis is oriented toward north. If the application wants the north-oriented axis to be second, then it can decide to swap the two first axes. Note that such axis flip or swap decisions are fully up to the viewers, which can decide whatever orientation they like. Axis reordering is done in exactly the same way as Annex E.2.1.2. The result for above example is as below:

$$\begin{bmatrix}\phi\\ \lambda\\ 1\end{bmatrix} = \begin{bmatrix}0 & -180/600 & +90\\ 360/800 & 0 & -180\\ 0 & 0 & 1\end{bmatrix} \times \begin{bmatrix}x\\ y\\ 1\end{bmatrix} \tag{E.6}$$

Finally, the above “Display to CRS” conversion needs to become a “CRS to Display” conversion. The is done simply by inverting the matrix (any graphics libraries do that very well). The result is as below:

$$\begin{bmatrix}x\\ y\\ 1\end{bmatrix} = \begin{bmatrix}0 & 800/360 & 400\\ -600/180 & 0 & 300\\ 0 & 0 & 1\end{bmatrix} \times \begin{bmatrix}\phi\\ \lambda\\ 1\end{bmatrix} \tag{E.7}$$

The full chain of operations from raster grid indices *(i,j)* to display’s pixel coordinates *(x,y)* is the result of the following matrix multiplication (reminder: the order of matrix multiplication operands is the **reverse** of operations order — see Annex E.2.1.3). Note that the axis swapping

in “Grid to CRS” is cancelled by the axis swapping in “CRS to Display”, resulting in a transform where raster axes are aligned with display axes.

$$\begin{bmatrix} 0 & 800/360 & 400 \\ -600/180 & 0 & 300 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & -\Delta\phi & +90 \\ \Delta\lambda & 0 & -180 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \Delta\lambda \times (800/360) & 0 & 0 \\ 0 & \Delta\phi \times (600/180) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{E.8}$$

Very often, the above multiplication does **not** need to be computed explicitly since it is done automatically by graphics libraries. For example, in Java2D (the standard graphics library on the Java platform), The `AffineTranform` instances used by Listing E.7 correspond directly to the affine conversions introduced in this annex:

- In `Graphics2D.drawRenderedImage(RenderedImage, AffineTransform)`, the given transform is “Grid to CRS”.
- In `Graphics2D.transform(AffineTransform)`, the given transform is “CRS to Display”.

The following code snippet shows how those transforms are used. Note that Java2D was released in 1998 and took inspiration from PostScript. Therefore, this snippet follows practices established for decades in graphics libraries.

```
RenderedImage   raster       = ...;     // Read from GeoHEIF file.
AffineTransform gridToCRS    = ...;     // Read from GeoHEIF metadata.
AffineTransform crsToDisplay = ...;     // Computed by the viewer.
Graphics2D      output       = ...;     // Typically provided by widget.

output.transform(crsToDisplay);
output.drawRenderedImage(gridToCRS, raster);    // Java2D does matrix
multiplication for us.

// Example of a geometry in geographic coordinates (EPSG:4326).
var shape = new GeneralPath();
shape.moveTo( 30, 50);      // 30°N, 50°E
shape.lineTo(-20, 60);      // 20°S, 60°E
shape.lineTo(-10, -5);      // 10°S,  5°W
output.draw(shape);         // The "CRS to Display" transform applies also to
this shape.
```

Listing E.7 — Example of affine transform usage with Java2D standard library

NOTE: Java2D automatically handles not only the affine transforms, but also the potentially non-zero pixel coordinates of the upper-left corner of the raster. Apache SIS uses this feature as a degree of liberty when responding to request. It has performance advantages discussed in Annex D.2.

E.2.2.2. Mapping CRS to visualization axes without knowing the CRS

A viewer may want to display the raster without bothering about its CRS. For example, a viewer may not want to care about whether the raster uses EPSG:4326 or OGC:CRS84. One possible approach is to, at first, ignore this whole Annex E and simply scale the image by *(display size) / (raster size)* factors at rendering time. Doing so does not prevent the use of the “Grid to CRS” transform for, example, getting the geographic coordinates of the mouse cursor.

A more elaborated approach is to still use the "Grid to CRS" followed by "CRS to Display" chain of operations, but with a "CRS to Display" transform adjusted automatically to whatever axis order the CRS is using. Assuming that the raster is ready to show (i.e., that grid axes are aligned with visualization axes), the first step is like the previous paragraph: Map raster coordinates directly to display coordinates, ignoring the CRS and this whole annex. Then, express that mapping as an affine transform. This usually requires nothing more than putting the *(display size) / (raster size)* scale factors on the diagonal. For the last step, use the following symbols.

- **T** the above-cited transform from raster coordinates to display coordinates.
- **G** the "Grid to CRS" transform specified with the raster.
- **D** the "CRS to Display" transform, which is unknown.

In Formula (E.8), **T** was obtained by first applying the **G** transform, followed by the **D** transform. In matrix form, this is written as below (reminder: operations are applied from right to left in matrix multiplications):

$$\mathbf{T} = \mathbf{D} \times \mathbf{G}$$

T and **G** are known, and the transform to find is **D**. Both sides can be multiplied by $\mathbf{G}^{-1}$, keeping in mind that the operand must be on the same side (which is the right side in the equation below) of each multiplication because matrix multiplications are not commutative.

$$\mathbf{T} \times \mathbf{G}^{-1} = \mathbf{D} \times (\mathbf{G} \times \mathbf{G}^{-1})$$

The insertion of parenthesis is okay because matrix multiplications are associative. Because ($\mathbf{G} \times \mathbf{G}^{-1}$) = **I** (the identity matrix), the result is:

$$\mathbf{D} = \mathbf{T} \times \mathbf{G}^{-1}$$

In other words: Viewers can setup their "raster to display" transform directly, multiply it by the inverse of the raster's "Grid to CRS" transform, and the result is the initial "CRS to Display" transform to use. This result can be obtained without any CRS knowledge. If any axis swapping or any reversal of axis direction were required, they will be "magically" handled by above mathematics. This approach works even in presence of shear and rotation.

E.2.2.2.1. General recommendation

The approach discussed in this sub-section is a general recommendation when working with affine transforms: Applications should try to express all their coordinate operations in terms of matrix multiplications and inversions. Any coordinate operation done "manually" (without matrix) should be reformulated as matrix operations. Most user's actions such as pans and zooms should also be expressed as affine transforms concatenated (by matrix multiplications) to the "CRS to Display" transform, for reasons explained in Annex E.3. Algebras like above (for finding the **D** matrix) can be applied to most problems for finding where to insert a matrix multiplication in order to obtain the desired effect.

While rotation support is not the most important reason for using affine transforms (the main reason advocated in this annex is unambiguous handling of axis swapping and flipping), rotations

are nevertheless an excellent test. If an application has any issue with rotations, it is probably not using affine transforms correctly. A proof of concept is shown by a JavaFX application described in Annex E.3.

E.2.2.3. Transforms chain with change of CRS

The previous sections assumed that the display CRS is the same as the raster data CRS, axis order included. It is also possible to dissociate the two CRSs. The chain of operations becomes:

(raster to display) = *(grid to source CRS)* ⟶ **(source CRS to target CRS)** ⟶ *(target CRS to display)*

The difference compared to previous sections is the insertion of a *source CRS to target CRS* step. This step may be a non-linear coordinate operation such as a map projection. But it can also be simply an axis swapping if the *source CRS* is EPSG:4326 and the *target CRS* is OGC:CRS84. Automatically finding the coordinate operation between a pair of CRS is the task of referencing libraries such as Apache SIS and PROJ. This is standardized in OGC Topic 2 / ISO 19111:2019 with the `findCoordinateOperations(sourceCRS, targetCRS)` method in `RegisterOperations`.

NOTE: Many coordinate operations may exist between the same pair of CRS with different domains of validity and different positional accuracies. Client applications should choose a coordinate operation with a domain of validity that contains the raster's extent. Applications may also choose an operation with an accuracy just sufficient for the raster data's resolution, using the argument that less accurate operations are sometime faster. For example, a less accurate operation may use an Helmert transformation instead of datum shift grids.

A client application can choose to setup a *CRS to display* transform as if the CRS was OGC:CRS84 (i.e., without axis swapping), and relies on Apache SIS, PROJ or other referencing libraries for inserting an axis swapping step in the *source CRS to target CRS* transform.

E.2.2.4. Visualizing three-dimensional data on a two-dimensional screen

All previous sections on this Annex assume that the numbers of source and target dimensions are the same, in which case all matrices are square. It is also possible, while more complex, to use affine transforms when the number of dimensions differ. For example, a chain of operations can be constructed between three-dimensional coordinates. Then, the result can be made two-dimensional by dropping a matrix row while keeping all columns. For example, for a three-dimensional transform as below (the dots represent arbitrary values that are ignored for this discussion):

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \Delta x & . & . & T_x \\ . & \Delta y & . & T_y \\ . & . & \Delta z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} \tag{E.9}$$

The result can be projected on the *(x, y)* plane simply by dropping the *z* row. The result is a non-square matrix, with more columns than rows.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \Delta x & . & . & T_x \\ . & \Delta y & . & T_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} \tag{E.10}$$

Dropping a row can be expressed by a multiplication with a now-square "dimension selector" matrix. The advantage of this approach is that the slice does not need to be in the *(x, y)*, *(x, z)* or *(y, z)* plane. By putting the right coefficients in the "dimension selector" matrix, the result can be any inclined plane.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \Delta x & . & . & T_x \\ . & \Delta y & . & T_y \\ . & . & \Delta z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \Delta x & . & . & T_x \\ . & \Delta y & . & T_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{E.11}$$

An inconvenient of this approach is that non-square matrices are more difficult to invert. For example, the inverse of dropping a dimension is to insert a new dimension. Such matrices have more rows than columns (the converse of dropping a dimension). However, because in the above example the *z* value has been lost, it is difficult to do better than setting the re-inserted coordinate value to a constant. This insertion of a constant coordinate value can be done by the following matrix, where the constant value is C_k:

$$\begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} = \begin{bmatrix} \Delta i & . & T_i \\ . & \Delta j & T_j \\ 0 & 0 & C_k \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{E.12}$$

E.3. Proof of concept

Apache SIS does all its linear georeferencing with affine transforms. One of the very first things that Apache SIS does with all raster data is to replace "georeferencing by bounding boxes" with affine transforms. Another thing that Apache SIS does almost immediately after reading each tile is to replace all Sentinel's "no data" values by NaN (Annex F). User's action such as zooms and pans (translations) are expressed by affine transforms, then concatenated to the *"raster's grid to display's coordinates"* chain of transforms. This approach opens new possibilities. For example, with some matrix algebras such as Annex E.2.2.2, a user's action can be converted from pixel units to CRS units and conversely. Therefore, a user's action in one window can be propagated to other windows even if the windows are showing raster data of different resolutions at different zoom levels in different geographic areas. This can be done with the following chain of operations, where the arrows represent a "CRS to Display" transform (Annex E.2.2.1) or its inverse:

(pan in pixels of windows 1) ⟶ *(pan in meters of common CRS)* ⟶ *(pan in pixels of windows 2)*

NOTE 1: The above arrows do not represent conversions of coordinates, but rather conversions of deltas. The latter is also an operation commonly supported by standard graphics libraries. For example, this is done with `AffineTransform.deltaTransform(…)` in Java2D.

NOTE 2: The above chain can work even when the two windows use different map projections. But the generalization to non-linear transforms requires the introduction of Jacobian matrices, which are out of scope of this annex.

Apache SIS is a library for numerous geospatial applications but provides a Java FX graphical user interface for demonstration purposes. Apache SIS extensive usage of affine transforms can be seen in many, sometime subtle, places. All examples given in the following sub-sections work even in presence of rotations (keyboard: [Alt] + [left arrow] or [right arrow]).

E.3.1. Transition between map projections

When reprojecting the raster data to another CRS, Apache SIS updates the "CRS to display" transform for the change of CRS in such a way that the visual change is imperceptible at the location of the mouse cursor. To demonstrate:

- Open a raster dataset, ideally a few tens of degrees in size;
- Choose an easily identifiable feature on the displayed image and right-click on it; and
- Select "Reference System", then "Centered projections", then "Universal Transverse Mercator" and Apache SIS will automatically select the UTM zone for the mouse cursor's position.

Note that the feature where the right-click occurred seems unchanged. If the raster data covers a small area, it may look like nothing happened at all. On raster datasets large enough, the change become more and more apparent as the distance from the right-click position increases.

While the current version of the JavaFX application does not yet show the following statement, Apache SIS is capable of following the displacement of a vehicle with automatic changes of UTM zone. This could also show automatic changes from UTM to stereographic projection when the vehicle enters polar regions, and keep the changes visually unnoticeable by users — at least in the vehicle's vicinity. This feature requires the support of Jacobian matrices (not discussed in this annex), which can be seen as a complement to affine transforms for non-linear cases. The above scenario works for any map projection in which Jacobian matrix support has been implemented. Jacobian matrices were introduced (under a different name) more than twenty years ago in OGC 01-009.

E.3.2. Positional accuracy shown in coordinate values

The bottom right corner of the image below shows the geographic or projected coordinates of the mouse cursor's position. Each coordinate value uses, independently of other values in the coordinate tuple, the minimal number of digits needed for distinguishing two screen pixels at the

cursor's position. This number of digits varies not only with the zoom level, but also simply by moving the mouse to different regions of the displayed raster dataset. To demonstrate:

- Open a world-wide raster which uses OGC:CRS84 or equivalent;
- Right-click on the raster, select "Reference System", then "WGS 84 / World Mercator" for reprojecting the raster data;
- Right-click on the bottom-right corner, select "Reference System", then "WGS 84" for displaying geographic coordinates; and
- Note that when moving the mouse cursor towards a pole, the number of digits shown in the latitude coordinate increases while the number of digits shown in the longitude coordinate stay unchanged.

This is done by inspecting the scale factors (a little bit more than only the diagonal) of the "CRS to display" matrix. If the chain of operations involves a datum change, Apache SIS also compares the scale factors with the operation positional accuracy and appends a text such as "± 100 metres" when the pixel size become smaller than the operation accuracy.

Figure E.7 — Demo application showing positional accuracy

E.3.3. Synchronized user's action between windows

Another affine transform usage demonstrated by the Apache SIS viewer is the synchronization of the geographic areas shown in different windows. When many windows are showing the same geographic area of different rasters, and when the user changes the viewed area in one window, it is easy to propagate that change to all windows if the same "real world" bounding box and the same orientation (rotation) is applied everywhere. However, when the windows are showing data in different geographic areas (for example, one window showing an area at the south of another window), potentially using different orientations or even map projections, synchronizing the windows become more challenging. It is possible to perform those more advanced synchronizations by expressing every user's action, such as zoom or pan, in one window as an affine transform. That affine transform can be converted from pixel units of one window to real world units, then to pixel units of another window. The result can be applied to the other window even if the two windows are showing different data at different resolutions with different zoom levels, geographic areas, rotations or map projections. To demonstrate:

- Open one or two raster datasets;
- With one raster dataset visualized, click on the "Windows" tab, then "New window;"

- Optionally visualize the other raster in the main window (or keep the same raster);
- Apply different zooms, pans (translations) and/or map projections in the two windows;
- In the “Windows” tab of the window that is more zoomed-in, select (by checkbox) the other window; and
- Pan the window that is more zoomed-in (right side in the figure below). The same pan, after direction and amplitude adjustments, is applied in all windows that are selected by checkbox (left side in the figure below).

This synchronization is done by expressing all user’s actions as affine transforms, converting these transforms with matrix operations (inversions and multiplications), then concatenating the converted actions to the affine transform of another window.

NOTE: The process of converting the user’s action may be non-linear (e.g., may involve a map projection) if the two windows use different CRSs. In such cases, the action to concatenate would vary for each point of the target window. Since a single converted action must be chosen for the whole target window, Apache SIS uses the location of the mouse cursor. Said otherwise, it is not possible to describe a non-linear map projection by an affine transform, but it is possible to use an affine transform as an approximation in the vicinity of a given point. This is the purpose of Jacobian matrices mentioned in Annex E.3.1.

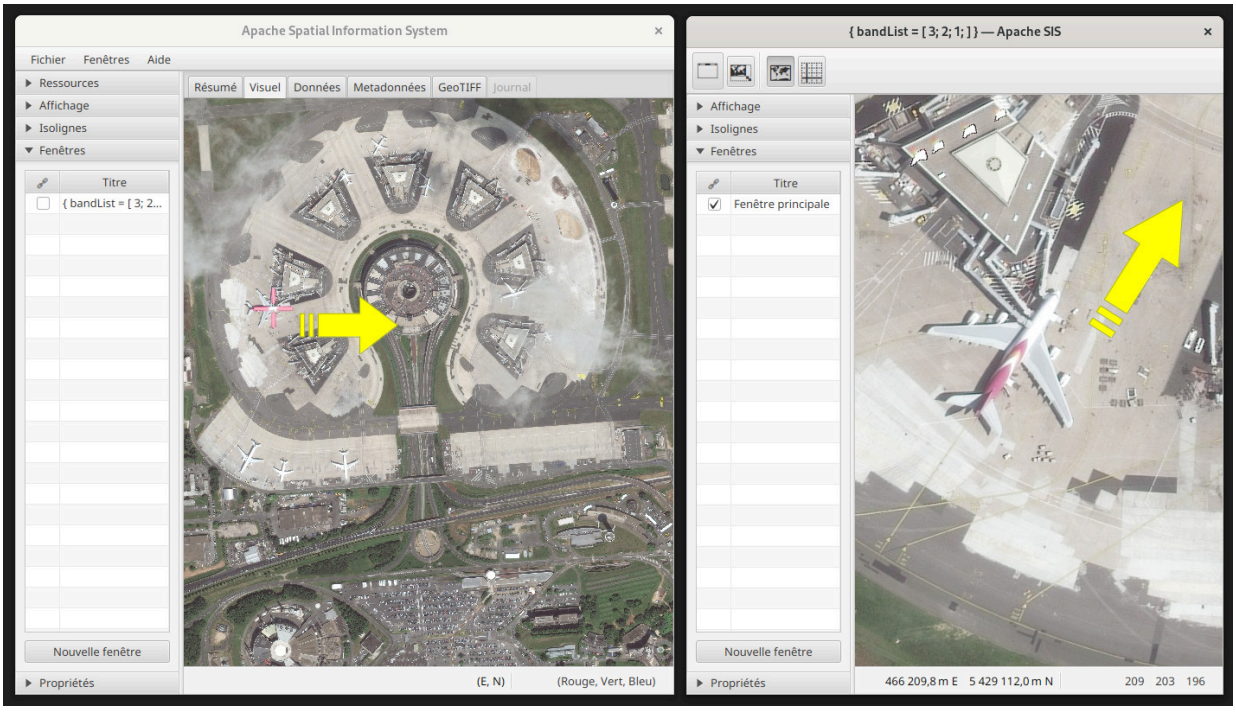


Figure E.8 — User’s action propagated to different windows

F

ANNEX F (INFORMATIVE) MISCONCEPTIONS ABOUT NAN (GEOMATYS)

ANNEX F (INFORMATIVE) MISCONCEPTIONS ABOUT NAN (GEOMATYS)

NOTE: Most of the content of this section is not specific from the GeoHEIF GIMI format and is of general interest to deal with no-data values in geospatial grid coverages. It expresses the opinion of a single author and its application can have risks in old version of C compilers without complete support to NaN.

This annex explains the rational behind a provision of the `cell-property-type` requirement in the GEOINT Imagery Media for ISR (GIMI) Specification Report. Note 2 of the `cell-property-type` requirement allows, but does not mandate, the use of IEEE 754 NaN values in cells where the value was not obtained or is known invalid. This is different than some other specifications which implicitly or explicitly forbid the use of NaN. This annex presents arguments for allowing them in GeoHEIF, keeping in mind that their use is optional.

F.1. Context

Geospatial rasters are often stored as integer values with a linear relationship from the integer values to the real values in a target unit of measurement such as °C. The relationship is typically expressed by a scale and an offset. In the ISO 19115 metadata standard, this relationship is called the *transfer function*.

There is often a need to flag some values as missing, for example because the pixel was masked by a cloud, or because the remote sensor did not pass over that area. Those missing values are represented by *Sentinel values*, such as -9999, considered as unrealistic for the phenomenon of interest.

> ***WARNING***
>
> *Sentinel values has been used in computer programming to denote either terminating values or missing values. Please note that this concept has nothing to do with the SENTINEL constellation of remote sensing satellites managed by the European Space Agency (ESA).*

Taking in account those Sentinel's "no data" values can be considered as part of the transfer function. Therefore, the full transfer function from integer values to real values may be implemented as below:

```
class TransferFunction {
```

```
    double scale  = ...;
    double offset = ...;
    int    noData = ...;

    double apply(int rasterValue) {
        if (rasterValue == noData) {
            return Double.NaN;    // Or whatever the application chooses to use
for missing values.
        } else {
            return rasterValue * scale + offset;
        }
    }
}
```

Listing F.1 — Example of a transfer function

However, when the raster data are stored as floating-point numbers, the real values can be stored directly with no need for the scale factor. The offset may still be useful for improving the precision, but the following discussion will ignore that details.

NOTE: For an example of how offset(s) can improve precision, consider sea water density. The values in the oceans are typically between 1020 and 1030 kg/m³. If those values are stored as single-precision floating-point numbers, the precision (ULP) around the value 1030 is approximately 1.2×10^{-4}. This is insufficient when five (5) digits precision is desired. However, oceanographers commonly use *sigma-t density* instead, which is the density minus 1000 kg/m³. Those sigma-*t* values are typically between 20 and 30 kg/m³, and the precision (ULP) of single-precision floating-point numbers around the value 30 become approximately 1.9×10^{-6}. Therefore, *shifting* the numbers closer to zero frees some digits that can be used for better precision. Note, however, that *scaling* the numbers has no significant effect on the precision of IEEE 754 floating-point numbers. In practice, most data have a range of valid values close enough to zero (relatively to the range's span) for making the use of offset unnecessary.

When values are stored as floating-point numbers, there is usually no scale or offset to apply. But very often, there is still a need to take in account the Sentinel's "no data" values. Therefore, users are not completely dispensed from the need to apply a transfer function.

Some (but not all) binary file formats, such as GIMI and netCDF (but not GeoTIFF), can encode the Sentinel's "no data" values using the IEEE 754 binary representations of floating-point values. Those binary representations are commonly named `float` for the 32 bit version, and `double` for the 64 bit version. When those `float` or `double` types are used both in the metadata and in the raster data, it is possible to use the *Not-a-Number* (NaN) feature defined by the IEEE 754 standard. This standard is widely supported in modern hardware, NaN included. Nevertheless, despite NaN values being well suited to the representations of missing values, NaNs are not used very often. Quite the opposite: It is common to see recommendations to *avoid* NaNs. But a majority of those objections are based on misconceptions.

F.2. Common misconceptions

In order to refute the idea that NaN values should be avoided in geospatial applications, an executable demo program was created in Java, C/C++ and Python (Annex F.5). That program

demonstrates how NaNs can be used instead of Sentinel's "no data" values in raster data. The main characteristics of the program are as follow.

- Test data is a raster dataset of size 800 × 600 pixels in 1 band filled with:
  - random floating-point values between −100 and +100,
  - approximately 12.5% of random missing values.
- Raster data are duplicated in 4 RAW files storing the same values in IEEE 754 `float` binary format, but:
  - one file using Sentinel's "no data" values and big-endian byte order,
  - one file using Sentinel's "no data" values and little-endian byte order,
  - one file using NaN values and big-endian byte order, and
  - one file using NaN values and little-endian byte order.
- For each above rasters, bilinear interpolations are performed at 20,000 pre-determined random positions.
- Above bilinear interpolations are repeated 10 times, with the interpolation points translated in each iteration.
  - This is intentionally chaotic, with errors that amplify quickly after a few iterations.
  - The intention is to observe the effect of C/C++ compiler optimization options such as `-ffast-math` (Annex F.4.4).
- The 200,000 expected interpolation results are precomputed with unlimited precision arithmetic and saved for comparison with the actual values computed by each test.
- Tests are executable in Java, C/C++ and Python languages (Annex F.5).

In the following sub-sections, the paragraphs labeled "Verification" give more details about how the demonstration was done to address for specific misconceptions.

F.2.1. "NaNs are toxic"

NaNs change everything they touch to NaN. But this is exactly the desired feature. For example, in `(x + y) / 2`, if `y` is unknown, then the result is unknown. Without NaNs, the classical approach is to use Sentinel's "no data" values such as -9999 like below:

```
double average(double x, double y) {
    if (x == -9999 || y == -9999) {
        return -9999;
    } else {
        return (x + y) / 2;
    }
```

```
}
```

Listing F.2 — Example of code handling a Sentinel's "no data" value

NaNs make the above verifications unnecessary. This is not only a convenience, but also (most important) a safety. In complex applications, it is very likely that some checks will be forgotten, or that the developer will wrongly assume that -9999 will never reach some point. This risk can be compared with buffer overflows caused by missing bound checks, which are still a major cause of bugs and security breaches despite decades of developers effort. Those problems can go unnoticed for some time before they cause harm.

Example 1: When computing the average value of {23, 33, -9999, 30, 34} where -9999 is a Sentinel's "no data" value that the developer forgot to handle, the result is -1975.8. At this point, the developer may suspect that something is wrong, but the software will probably not. However, if the average is computed over 1000 random values in the [10 … 50] range, then the corrupted result is close to 20 (by comparison, the normal value is close to 30). This corrupted result is well inside the range of valid values, potentially making the error unnoticeable even for the developer.

Example 2 (could be a demonstration): The demo programs check for missing values in raster data. However, it does not check for missing values in the coordinates of the points where to interpolate. If those coordinates were latitudes and longitudes, and if the calculation involved a map projection where all latitude values were given to `sin`, `cos` or `tan` trigonometric functions (this is a common scenario in map projections), then -9999° would be fully equivalent to 81°. It can make corrupted values totally indistinguishable from valid values.

Such floating-point errors should not be taken lightly since, for example, the first Ariane V rocket exploded because of a floating-point overflow. The use of NaN values reduces at least the risk that corruption caused by unchecked "no data" values stay unnoticed. Even if a library performs very carefully all "no data" checks, if the library and the caller using the library do not agree on the same Sentinel's "no data" values (for example because of a misconfiguration), the result is the same as if the library performed no check at all.

F.2.2. "There is only one NaN, while we need many no-data values"

This idea is false. The IEEE 754 `float` type has 8,388,608 distinct quiet NaN values (ignoring signaling NaNs, see below for notes on the difference) while the `double` type has over 10^{15} distinct quiet NaNs. When the bit pattern of a `float` is interpreted as an `int`, all values in the range `0x7fc00000` to `0x7fffffff` inclusive are NaN values. This is not the only range of NaN values, but this is the most convenient range, and the one used in the OGC Testbed 20 demonstration. See Annex F.3.1 for an example of definition of NaN values in that range.

NOTE: 16 million of distinct NaN values for the single-precision floating point type may seem a waste of space that would be better spent in precision. However, NaN values are not defined at the expense of precision. They are defined at the expense of one exponent value (in base 2), thus limiting the maximal value to about 3.4×10^{38} instead of 6.8×10^{38}. Considering that the maximal exponent value is also reserved for representing infinity, NaN values are using space that would be lost anyway.

Verification: for demonstrating the availability of many distinct NaN values, the OGC Testbed 20 demonstration (Annex F.5) used four of values:

- `UNKNOWN` for any value missing for an unknown reason.
- `CLOUD` for a value missing because of a cloud.
- `LAND` for a value missing because the pixel is on a land (assuming that the data are for some oceanographic phenomenon).
- `NO_PASS` for a value missing because the remote sensor did not pass over that specific area.

To demonstrate that NaN values can be analyzed, the bilinear interpolation arbitrarily applies the following rule: If exactly one value needed for the interpolation is missing, the interpolation result is missing for the same reason. If two or more needed values are missing, the interpolation result is missing for the reason having the highest priority in the following order (highest priority first): `NO_PASS`, `LAND`, `CLOUD` and `UNKNOWN`. Executions and comparisons against the precomputed expected values show that the different NaN values are correctly distinguished in all tested languages.

F.2.3. "Not the right semantic (we want `null`, `unknown`, ...)"

NaN means "Not A Number". There is no semantic associated to why a value is NaN. It can be any reason of the user's choice. With more than 4 millions distinct NaN values available, there is plenty of room for assigning whatever semantic the user wants to different NaN values. Assigning a "no data" meaning to, for example, -9999, which *is* a number contrary to NaN, is not semantically better.

NaN cannot represent directly a JavaScript `null` or `unknown` value (neither can real values such as -9999). However, `null` and `unknown` semantics are a layer on top of the IEEE 754 values. For example, in the Java and C/C++ languages, `null` values cannot be assigned to primitive types. In these languages, `null` values can only be assigned to wrappers (Java) or pointers (C/C++). For example, a `java.lang.Float` (in Java) or a `float*` (in C/C++) can be null, but a `float` cannot. In JavaScript, the fact that `null` and `unknown` apply to all kinds of objects is an indication that they are handled at a level other than a primitive type.

F.2.4. "The payloads that distinguish NaN values may be lost"

This argument is based on the fact that IEEE 754 recommends, but does not mandate, NaN's *payload propagation*. This issue is discussed in Annex F.2.5, but does not apply to loading or saving values from/to files. When reading an array of `float` values, the operating system is reading **bytes**. If data are compressed with a lossless algorithm, decompression algorithms such as GZIP usually operate on bytes. Next, the conversion from big-endian byte order to little-endian or conversely is applied on bytes. Only after all these steps have been completed is the final result interpreted as an array of floating-point values. Casting a `byte*` pointer to a `float*` pointer in C/C++ does not rewrite the array, it only changes the way that the computer interprets the bit patterns that are stored in the array. Even after the cast, if a `float` array needs

to be copied, the `memcpy` (C/C++) or `System.arraycopy` (Java) functions copy bytes without doing any interpretation or conversion of the bit patterns. Nowhere does the Floating-Point Unit (FPU) need to be involved. Therefore, there is no way that the NaN's payload can be lost with the above steps.

Verification: The demonstration (Annex F.5) provides the same raster data in two versions:

- one file using big-endian byte order,
- another file using little-endian byte order.

The demonstration reads the binary files, changes the byte order if needed, then interprets the result as floating-point values. See for example the `loadRaster()` method in the `TestCase.cpp` file, which ends with `return reinterpret_cast<float*>(bytes)`. In the Java case, the read operation involves a copy from the `byte[]` array to a newly allocated `float[]` array, but execution shows that no payload is lost.

NOTE: Some compression algorithms may work on floating-point values instead of bytes. For example, LERC — Limited Error Raster Compression has this option. In such cases, how the algorithm handles NaN values is an algorithm's choice. For example, LERC replaces NaNs by masks or Sentinel's "no data" values. For this annex, such algorithms are considered lossy because they choose (intentionally in LERC case) to sacrifice some information (the NaN payload), even if the algorithm is lossless for other values.

F.2.5. "Payload propagation is implementation-dependent"

When a quiet NaN value is used in an arithmetic operation, IEEE 754 *recommends* that the result is the same NaN as the operand. However, this is not mandatory and the actual behavior may vary depending on the platform, language, or applied optimizations. Furthermore, the situation is even more uncertain when two or more operands are different NaN values. Which one takes precedence? Empirical tests suggest that, at least with Java 22 on Intel® Core™ i7-8750H, the leftmost operand takes precedence in additions and multiplications, while the rightmost operand takes precedence in subtractions and divisions. However, despite this uncertainty, users still have the guarantee that the result will be *some* NaN value. The fact that users do not know for sure *which* NaN values they got is not worse than the approach using Sentinel's "no data" values. For example, when computing the average value of {2, 8, 9999, 3} where 9999 represents a missing value, if developers forget to check for missing values, they will get 2503. Subsequent operations consuming that value are likely to fail to recognize 2503 as a missing value, leading to unpredictable consequences. By contrast, when using NaNs, developers get some NaN result even if they forgot to check for missing values. Even if the users are uncertain about *which* NaN value was obtained, this situation is still better than an accidental value indistinguishable from real values. The preceding example only computed the average of 4 values, but the more values to average there are, the more difficult it becomes to notice Sentinel's "no data" values polluting the computation.

Verification: The demonstration (Annex F.5) using Sentinel's "no data" values needs to check for missing values *before* using them in calculations. The same is true (in a more lenient way) for applications using NaN values if they wish to distinguish between the different NaN payloads.

However, users of NaN values have two additional alternatives that users of Sentinel's "no data" values do not have:

- If an application does not care about the reason why a value is missing, it can skip the check completely; and
- An application can check for NaN results afterward instead of before doing the calculation and go back to the original inputs only if it decides that it wants to know more about the reasons.

The `TestNodata` demonstration had to check for Sentinel values before any computation. This was no choice. The `TestNaN` demonstration could have done the same check but instead arbitrarily chooses to check for NaN payloads *after* calculations. This is an arbitrary choice, `TestNaN` could have been kept in a form almost identical to `TestNodata`. This is done for illustrating the flexibility mentioned in previously made points. This strategy can provide a performance gain when the majority of the values are not missing.

NOTE: All of the discussion in this section and all demonstrations apply to *quiet* NaNs. Another type of NaN exists, named *signaling* NaNs. The two types of NaN are differentiated by bit #22, which shall be 1 for quiet NaNs. Signaling NaNs are out of scope for this discussion. Signaling NaNs may be silently converted to quiet NaNs, or may cause an interrupt. They are used for different purposes (e.g., lazy initialization) than the "no data" purpose discussed here.

F.3. Analysis of code differences

The `TestNaN` and `TestNodata` classes used in the demonstration are very similar, especially when ignoring the arbitrary choice of testing NaN after the calculation instead of testing it before. Using NaN values in Java and C/C++ does not bring significant complexity compared to using Sentinel's "no data" values. NaN values can be as reliable as "no data" values: The payload cannot be lost during read, copy and write operations manipulating bytes. On the other hand, using NaNs provides a level of safety impossible to achieve with Sentinel's "no data" values, as developers are protected against missing values accidentally corrupting value computations. The main differences between the `TestNodata` and `TestNaN` classes are described below.

F.3.1. "No data" values declaration

The `TestNodata` class uses the following Sentinel values for missing data:

```
/**
 * Sentinel value for missing data. A value may be missing for different reasons,
 * which are identified by different Sentinel values.
 */
static final float UNKNOWN = 10000,
                   CLOUD   = 10001,
                   LAND    = 10002,
                   NO_PASS = 10003;

/**
```

```
 * The threshold used for deciding if a value should be considered as a missing
value.
 * Any value greater than this threshold will be considered a missing value.
 */
static final float MISSING_VALUE_THRESHOLD = UNKNOWN;
```

Listing F.3 — Example of distinct Sentinel's "no data" value definitions

Note that the `MISSING_VALUE_THRESHOLD` value is arbitrary and data-dependent, as it must be a value not used by real values. Changing the threshold can require changes in the code, for example if the threshold becomes a value smaller than all the valid values instead of greater than them. By contrast, an approach based on NaN values does not need such an arbitrary choice. The `TestNaN` class uses the following NaN values for missing data:

```
/**
 * Value of the first positive quiet NaN.
 */
private static final int FIRST_QUIET_NAN = 0x7FC00000;

/**
 * NaN bit pattern for missing data. A value may be missing for different
reasons,
 * which are identified by each of the different NaN values.
 */
static final int UNKNOWN = FIRST_QUIET_NAN,      // This is the default NaN
value in Java.
                 CLOUD   = FIRST_QUIET_NAN + 1,
                 LAND    = FIRST_QUIET_NAN + 2,
                 NO_PASS = FIRST_QUIET_NAN + 3;
```

Listing F.4 — Example of distinct NaN value definitions

Alternatively, C/C++ developers can use the `std::nanf(char*)` function instead. This function is defined by the C++ 11 standard and an usage example is given in Annex F.4.

F.3.2. "No data" values checks

The `TestNodata` class uses the following code for checking for missing values. In this case, this check must be done before using the values in formulas. The use of `max` is an optimization based on the fact that, in this test, the reasons why a value is considered missing are sorted in order of precedence (see Annex F.5.2 for more details). Production codes may be more complex if they cannot rely on this assumption.

```
float v00, v01, v10, v11 = ...;   // The raster data to interpolate.
float missingValueReason = max(
        max(v00, v01),
        max(v10, v11));

double result;
if (missingValueReason >= MISSING_VALUE_THRESHOLD) {
    // At least one value is missing.
    result = missingValueReason;
} else {
    // All values are valid. Interpolate now.
    result = ...;
```

```
}
```

Listing F.5 — Example of checking Sentinel's "no data" values before interpolation

The `TestNaN` class uses the following code for checking for missing values. In this case, contrary to `TestNodata`, developers are free to check before or after any interpolation is computed. This example arbitrarily performs the check after the interpolation. The use of `max` below is the same trick as the one used above. Production codes may be more complex for the same reasons but have no reason to be more complex than the code using "no data" Sentinel values.

```
float v00, v01, v10, v11 = ...;   // The raster data to interpolate.
double result = ...;              // Interpolate regardless if values are valid.
if (isNaN(result)) {
    // At least one value is missing.
    // This block could be completely omitted if the developer
    // does not care about the reason why a value is missing.
    int missingValueReason = max(
            max(floatToRawIntBits(v00), floatToRawIntBits(v01)),
            max(floatToRawIntBits(v10), floatToRawIntBits(v11)));

    // Result was already NaN, but replace with another NaN for the selected
reason.
    result = intBitsToFloat(missingValueReason);
}
```

Listing F.6 — Example of checking NaN values after interpolation

F.3.3. Multiple missing value reasons for the same pixel

Handling NaN values as bit patterns allows a more advanced use case which is not possible (or at least not as efficiently) with "no data" Sentinel values. Instead of associating the missing value reasons to different NaN values (over 2 million possibilities), users can associate these reasons to different **bits**. The NaN payload has 21 bits (ignoring the signaling NaN bit and the sign bit), which gives room for 21 different reasons (or 22 if the sign bit is also used). The `TestNaN` code declaring NaN values can be replaced by the following:

```
static final int UNKNOWN = FIRST_QUIET_NAN,     // No bit set. This is the
default NaN value in Java.
                 CLOUD   = FIRST_QUIET_NAN | 1,
                 LAND    = FIRST_QUIET_NAN | 2,
                 NO_PASS = FIRST_QUIET_NAN | 4,
                 OTHER   = FIRST_QUIET_NAN | 8;
```

Listing F.7 — Example of NaN definitions allowing multiple reasons to be packed in a single value

And the code computing the missing value reasons (previously using `max`) become the following:

```
if (isNaN(result)) {  // Optional, this check is also done by `if
(missingValueReasons != 0)`.
    int missingValueReasons = 0;
    if (isNaN(v00)) missingValueReasons |= floatToRawIntBits(v00);
    if (isNaN(v01)) missingValueReasons |= floatToRawIntBits(v01);
    if (isNaN(v10)) missingValueReasons |= floatToRawIntBits(v10);
    if (isNaN(v11)) missingValueReasons |= floatToRawIntBits(v11);
    if (missingValueReasons != 0) {
        // At least one value is NaN.
    }
```

```
}
```

Listing F.8 — Example of code packing multiple reasons in a single NaN value

With this approach, declaring that a value is missing for multiple reasons is possible. For example, this is because of `LAND` *and* `NO_PASS`. A use case for this feature is: Consider a calculation involving many steps (resampling followed by convolution, *etc.*), each step using many inputs. If each step encodes for each output value *all* the reasons why the value is missing, and if a too large number of values appear to be missing at the end of the process chain, then users can check what is the main reason why values are missing by checking which NaN bit is most frequently set. For example, are final results missing mostly because of clouds? Or mostly because the remote sensor did not pass over the area? In the latter case, using a different remote sensing product may resolve the problem, while in the former case it may not unless the acquisition times are different enough.

F.3.4. Code patterns leveraging NaNs

When codes can rely on NaN values behaving as specified by the IEEE 754 standard, some coding habits may be worth changing. Developers should keep in mind that any comparison (`<`, `>`, `<=`, `>=` and `==`) involving at least one NaN value will always be evaluated to false. This include `x == x`, which is false if *x* is a NaN value. Therefore, the following code:

```
void compare(double x, double y) {
    if (x != -9999 && y != -9999 && x < y) {
        // Do something if x<y AND both x and y are real values (not NaN).
    }
    if (x == -9999 || y == -9999 || x < y) {
        // Do something if x<y OR at least one of x and y is NaN.
    }
}
```

Listing F.9 — Comparisons using a Sentinel's "no data" value

Can be replaced by the following code. Because the use of `!(x >= y)` may seem an unnecessary complication compared to `(x < y)` for developers who are not familiar with IEEE 754, a good precaution is to add a comment saying that the use of the negative form is intentional.

```
void compare(double x, double y) {
    if (x < y) {
        // Do something if x<y AND both x and y are real values (not NaN).
    }
    if (!(x >= y)) {    // Use `!` for entering in the block when a value is NaN.
        // Do something if x<y OR at least one of x and y is NaN.
    }
}
```

Listing F.10 — Comparisons relying on IEEE 754 behavior

F.4. Notes on C/C++ implementation

The C/C++ standard library supports NaN values. Not only does the standard define an `NAN` constant and an `std::isnan(…)` function (among others), but the existence of multiple distinct NaN values has been explicitly supported since the introduction of the C++ 11 standard.

- The `std::nanf(char*)` function generates a quiet NaN that can be used by library implementations to distinguish different NaN values (sources: cplusplus.com, cppreference.com).
- The `std::strtof(char*)` function can parse character strings such as `NAN("cloud")` (source: cppreference.com).

In our tests with GCC 14.2.1 20240912 (Red Hat 14.2.1-3), the `strtof` function returns `7fc00000` for character strings that do not represent a number. For character strings that represent a decimal number, the value seems to be simply added to `7fc00000`. Therefore, code such as:

```
const int32_t UNKNOWN = FIRST_QUIET_NAN,
              CLOUD   = FIRST_QUIET_NAN + 1,
              LAND    = FIRST_QUIET_NAN + 2,
              NO_PASS = FIRST_QUIET_NAN + 3;
```

Listing F.11 — Example of distinct NaN value definitions using bit patterns

Could be replaced by:

```
const float UNKNOWN = std::nanf("0"),
            CLOUD   = std::nanf("1"),
            LAND    = std::nanf("2"),
            NO_PASS = std::nanf("3");
```

Listing F.12 — Example of distinct NaN value definitions using C\++ 11 standard

Developers may wish to keep the former approach for better control. However, the latter approach shows that the existence of multiple distinct NaN values is recognized by the C++ 11 standard. The documentation of `std::nanf(char*)` also says:

> If the implementation supports IEEE floating-point arithmetic (IEC 60559), it also supports quiet NaNs.

F.4.1. Getting the bits of a float

The following function is equivalent to `java.lang.Float.floatToRawIntBits(float)`. Note that NaN values have a sign bit (it is possible to have “negative” NaN values), so codes really need the signed type shown below. The choice of signed or unsigned type changes the way that the `max` function will behave when checking which NaN has precedence in this test.

```
inline int32_t floatToRawIntBits(float value) {
    return std::bit_cast<int32_t>(value);
```

```
}
```

Listing F.13 — Float to raw int bits using the C++ 20 standard

Alternatively, bit patterns can be read directly from the data array without transiting by a floating-point type. This approach ensures that no Floating Point Unit (FPU) operation is involved, not even a copy (the truly paranoiacs could go further by applying `reinterpret_cast` on the array of bytes read from the file instead of the array of `float` values):

```
void foo(float* rasterData, int x, int y) {
    int32_t* dataAsInts = reinterpret_cast<int32_t*>(rasterData);
    int      offset     = y * rasterWidth + x;
    int32_t  bitPatern  = dataAsInts[offset];
    float    value      = rasterData[offset];
    // Do some stuff.
}
```

Listing F.14 — Float to raw int bits for a whole array

F.4.2. Detecting NaNs without floating-point support

If the `std::isnan(float)` function is not available (for example, because of compiler options discussed in Annex F.4.4), quiet NaNs can still be detected from an array of IEEE 754 values interpreted as integers (e.g., using `reinterpret_cast<int32_t*>` as in Annex F.4.1) as below:

```
#define FIRST_QUIET_NAN 0x7FC00000

void foo(int32_t* dataAsInts) {
    int offset = ...;
    if ((dataAsInts[offset] & FIRST_QUIET_NAN) == FIRST_QUIET_NAN) {
        // The value is a quiet NaN.
    }
}
```

Listing F.15 — NaN check without floating point operation

For detecting both quiet and signaling NaNs, the mask would be slightly different, and another condition would need to be added for differentiating NaN from infinite values.

F.4.3. Stability of standard functions

Some functions provided by standard libraries may have undefined behavior with NaN inputs. For example, the result of `std::sort()` on a `vector<float>` is only partially sorted when the vector contains NaN values. This issue is language dependent. For example, `Arrays.sort(float[])` is fully defined in Java (NaN values are sorted last and −0 is sorted before +0), while Python behaves like C/C++. For languages with undefined behavior, users may need to filter the NaN values before invoking the standard function. Alternatively, users can sometime provide a custom comparison function. However, the filtering step or custom comparator is often needed anyway even with Sentinel's "no data" values. For example, both NaNs and Sentinel's "no data" values may need to be ignored when searching for the minimum or maximum values in an array because:

- The standard `min/max` functions may return NaN if any value is NaN, either by function's contract (e.g., this behavior is explicitly specified in the Java standard library) or as a result of unspecified behavior; and
- Sentinel's "no data" values may have precedence over valid values in the common case where the former are smaller or greater than all of the latter.

It may be unnecessary to filter the Sentinel's "no data" values before sorting if the caller knows that these Sentinel's values are smaller or greater than all valid values, but this is a special case. Filtering is also unnecessary in languages where the `sort` function is fully defined by default, including the NaN case.

F.4.4. Effect of compiler options

By default, `gcc` and probably most C/C++ compilers produce code compliant with IEEE / ISO rules. The code generated with these default options passes the C/C++ test described in this section. However, the `gcc` compiler provides a `-ffast-math` option with the following warning:

> This option is not turned on by any `-O` option besides `-Ofast` since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications.

The `-ffast-math` compiler option is a shortcut for the options listed in the following table. For verifying the effect of those options, the demo program has been compiled multiple times with the options added in the order listed below. The column on the right side reports whether adding an option made a difference compared with the previous compilation.

Table F.1 — Effect of `-ffast-math` compiler options

OPTION	CHANGE COMPARED TO THE EFFECT OF ALL PREVIOUS ROWS
`-fno-signaling-nans`	None (this option is set by default).
`-fno-rounding-math`	None (this option is set by default).
`-fexcess-precision=fast`	None (this option is usually effective by default).
`-fcx-limited-range`	None (this option applies to complex numbers).
`-fno-math-errno`	Reduction between 2 and 10% of the file size.
`-fno-trapping-math`	Executable file of same size but different content.
`-fno-signed-zeros`	Executable file of same size but different content.

OPTION	CHANGE COMPARED TO THE EFFECT OF ALL PREVIOUS ROWS
`-fassociative-math`	None.
`-freciprocal-math`	Executable file of same size but different content.
`-ffinite-math-only`	Executable file of same size but different content.

The tests pass with identical results for all options except `-ffinite-math-only`. With the latter option, the only behavioral change that impacted the demo program is in the `std::isnan` function. The `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`, `floor`, `ceil`, `trunc`, `pow`, `sqrt`, `hypot` and `nanf` functions continue to work as expected with NaN values. For working around the non-availability of the `std::isnan` function when the `-ffinite-math-only` option is used, the following check:

```
void foo(float value) {
    if (std::isnan(value)) {
        // Do some stuff.
    }
}
```

Listing F.16 — NaN check using std::isnan(...)

can be trivially replaced by the following, providing that `missingValueReason` is a *signed* integer and that all used NaN values are quiet NaNs and are "positive" (sign bit unset):

```
void foo(float value) {
    int missingValueReason = floatToRawIntBits(value);
    if (missingValueReason >= FIRST_QUIET_NAN) {
        // Do some stuff.
    }
}
```

Listing F.17 — NaN check without std::isnan(...) (limited applicability)

The above trick is possible in the test case (see the C/C++ source code) because of the way that `missingValueReason` is computed. This test uses `max` operations, together with the fact that "positive" quiet NaNs are greater than all other IEEE 754 values when compared as signed integers. Annex F.4.2 provides another way to identify the NaN values.

F.4.4.1. Questioning the pertinence of `-ffinite-math-only`

Developers should consider whether they really need the `-ffinite-math-only` option. This option and the `-fno-signed-zeros` option optimize trivial codes such as `x + 0.0` where `0.0` is a *compile-time constant* (the compiler cannot omit `+ 0.0` when the sign of zero matter, because `-0.0 + 0.0 = +0.0`). There is no performance gain expected in arithmetic expressions such as `x + y` if *x* and *y* are known only at runtime, even when *y* is zero. Indeed, the execution of above-cited tests do not show any change in the way that arithmetic operations and mathematical functions handle NaN values, except for the `std::isnan(…)` function which is itself a highly controversial exception.

Verification: The following code was compiled with and without the `-ffinite-math-only` and `-fno-signed-zeros` options. The executable files produced with this test differ only because of the `+ 0.0` code fragment. If that code fragment is removed (removing or not the `* 1.0` code fragment makes no difference), then the result become identical: Arithmetic expressions on *x* and *y*, the comparison between *z* and *w*, and calls to trigonometric functions are compiled in the same way no matter if the `-ffinite-math-only` option is specified or not.

```
int main() {
    double x = std::sin(rand());    // The tested compiler options have no
effect here.
    double y = std::cos(rand());
    double z = x + y + 0.0;         // Compilation result depends on -fno-signed-
zeros.
    double w = z * y * 1.0;         // Multiplication by 1 is removed in all
tested cases.
    printf("%f %f\n", z, w);
    if (z < w) {                    // Tests show no effect, unless < is
replaced by ==.
        printf("z < w\n");
    }
    return 0;
}
```

Listing F.18 — Dummy code for testing the effect of compiler options

However, if `(z < w)` is replaced by `(z == w)`, then the compilation results differ. For a more extensive verification, the full bilinear interpolations test was modified by removing the call to `std::isnan`, then was compiled with and without the `-ffinite-math-only` option. The assembler codes emitted by the compiler in both cases were compared. The main differences appear to be in the comparisons of floating point numbers with the `==` operator. The assembly codes as below:

```
ucomisd (%rdx), %xmm0   # Accept quiet NaNs ('u' stands for "unordered").
jp .L340           # Jump if unordered (i.e., if any operand is NaN).
jne .L340           # Jump short if not equal.
```

Listing F.19 — Assembler code without -ffinite-math-only

becomes as below:

```
comisd (%rdx), %xmm0   # Signal an invalid operation for any kind of NaN.
jne .L340           # Jump short if not equal.
```

Listing F.20 — Assembler code with -ffinite-math-only

The difference between `ucomisd` and `comisd` is that the latter signals an invalid operation when a source operand is any kind of NaN (including quiet NaNs), while the former signals an invalid operation only for signaling NaNs. The `jp` instruction has the effect of jumping if the result is unordered (the `u` in `ucomisd`), i.e., if one of the inputs is NaN. This is in conformance with the IEEE 754 requirement that `x == y` is always false if one of the operands is NaN, even if both are the same NaN. Therefore, the addition of a `jp` instruction in every `==` comparison between floating point numbers seems to be the only runtime performance penalty of the `-ffinite-math-only` option on arithmetic operations in this test. However, this apparent penalty may actually be a performance gain if the code leverages IEEE 754 rules with code patterns similar to Annex F.3.4, as it replaces two comparisons by a single one.

For verifying the performance impact of that compiler option, the bilinear interpolation tests were benchmarked with 200 iterations. Since those tests do not use macros or templates, `-ffinite-math-only` should have few effects. The results are below:

Table F.2 — Average execution times in milliseconds

TEST	FINITE MATH ONLY		STANDARD SUPPORT OF NAN	
	Time (ms)	Std. dev	Time (ms)	Std. dev
Sentinel's "no data", big-endian	8.215	0.338	8.122	0.425
Sentinel's "no data", little-endian	8.214	0.242	8.076	0.229
NaN values, big-endian	8.969	0.278	8.827	0.294
NaN values, little-endian	8.993	0.335	8.876	0.370

NOTE: The tests with NaN values intentionally interpolate more points than the tests with "no data" values. Therefore, it is normal that the former appear slower than the latter.

Time differences are indistinguishable from noise. The above benchmark suggests that the code is *slower* with the `-ffinite-math-only` option, but this is really only noise. Which test is faster varies from run to run.

F.4.4.1.1. Conclusion

The `-ffinite-math-only` and `-fno-signed-zeros` options are useful for cases when macros or templates are used, because their expansions may reveal *constants* that the compiler can omit in expressions such as `+ 0.0`, or reveal `if (std::isnan(x)) {…}` expressions where the compiler can prune the entire `{…}` branch. It may have a marginal effect on floating point comparisons with the `==` operator, but that effect may be unnoticeable and even counterbalanced by the `(x == mySentinelNoData)` checks that the developers must add in the code. In conclusion, the `-ffinite-math-only` and `-fno-signed-zeros` compiler options can be sources of problems for performance gains that may not exist, depending on the content of the macros or templates that the code may use.

F.5. Running the tests

A demo program was created in a GitHub repository, with a copy in the OGC GitLab repository. The same test is available in the following languages: Java, C/C++ and Python. All commands listed below should be executed in a Unix shell with the project root directory (the directory containing this `README.md` file) as the current directory. The Java test should be run at least

once before any of the tests in other languages because the test data are generated by the Java version of the tests. The test requires Java 17 or later.

Java: The following command will build, generate test files, and execute the tests.

```
mvn package

# (Optional) Rerun the tests without Maven
java --module-path target/NaN-1.0-SNAPSHOT.jar --module com.geomatys.nan/com.
geomatys.nan.Runner
```

Listing F.21 — Build and execution of NaN tests in Java

C/C++: The following commands create the binary in the `target` directory. Note that the above Java code must have been built at least once before the following commands can be executed.

```
mkdir target/cpp
cd target/cpp
cmake ../../src/main/cpp
cmake --build .

# Execution
./NaN-test
```

Listing F.22 — Build and execution of NaN tests in C/C++

Python: The Java code must have been built at least once before this command can be executed.

```
python src/main/python/TestCase.py
```

Listing F.23 — Execution of NaN tests in Python

F.5.1. Expected results

The tests implemented in Java and C/C++ produce the same result, shown below. Those implementations use `Math.fma(…)` and `std::fma` respectively for "fused multiply-add" in bilinear interpolations. The drifts observed during the last iterations are caused by the intentionally chaotic nature of the test, which makes the last iterations very sensitive to small errors in previous iterations. This sensitivity is used for observing how large those drifts become when implementation details are changed or when C/C++ compiler options are added.

```
Errors in the use of raster data with NaN values in big endian byte order:
   Count     Minimum     Average     Maximum   Number of "missing value"
mismatches
   11732      0,0000      0,0000      0,0000      0
   10771      0,0000      0,0000      0,0000      0
   10809      0,0000      0,0000      0,0000      0
   10762      0,0000      0,0000      0,0000      0
   10851      0,0000      0,0000      0,0000      0
   10814      0,0000      0,0000      0,0016      0
   10826      0,0000      0,0000      0,0871      0
   10918      0,0000      0,0042     18,1068      0
   10734      0,0000      0,0208     65,3695      9
   10792      0,0000      0,1692    127,6496     35
```

Listing F.24 — Drifts of test using double precision and FMA

The test implemented in Python differs from the Java and C/C++ implementations in that it does not use the FMA operation. Instead, the bilinear interpolation formulas are implemented in a more traditional way (except for `+ 0.0`) as below:

```
v0 = (v01 - (v00 + 0.0)) * xf + v00
v1 = (v11 - (v10 + 0.0)) * xf + v10
```

Listing F.25 — Fragment of interpolation code in double precision (Python)

The reason for `+ 0.0` is because `v00`, `v01`, `v10` and `v11` are single precision floating-point numbers. The single precision nature of those numbers is explicit in Java and C/C++ code, but implicit and hard to see in the Python code. If the values are not cast to `double`, then `(v01 - v00)` and `(v11 - v10)` are computed with single precision. The precision lost causes a faster drift from the expected results. To force a cast to double precision, the Java and C/C++ code use the following:

```
double v0 = (v01 - (double) v00) * xf + v00;
double v1 = (v11 - (double) v10) * xf + v10;
```

Listing F.26 — Fragment of interpolation code in double precision (Java and C/C++)

In Python, adding 0.0 is a trick that appears to work. The result of the test in Python is shown below. The tests in Java and C/C++ produces an identical result if their code using FMA is replaced by the above code. This result shows a small loss of accuracy compared to the code using FMA. Note that on the machine used for the test (Intel® Core™ i7-8750H) and the `gcc` compiler used (14.2.1 20240912 (Red Hat 14.2.1-3)), the output is the same even with the `-ffast-math` option.

```
Errors in the use of raster data with NaN values in big endian byte order:
   Count     Minimum     Average     Maximum   Number of "missing value"
mismatches
   11732      0,0000      0,0000      0,0000      0
   10771      0,0000      0,0000      0,0000      0
   10809      0,0000      0,0000      0,0000      0
   10762      0,0000      0,0000      0,0000      0
   10851      0,0000      0,0000      0,0000      0
   10814      0,0000      0,0000      0,0009      0
   10826      0,0000      0,0001      0,0871      0
   10918      0,0000      0,0047     18,1068      0
   10735      0,0000      0,0383     94,2899     10
   10789      0,0000      0,2025    127,6496     42
```

Listing F.27 — Drifts of test using double precision

If the single precision numbers are not forced to double precision before calculation, i.e., if the Python code is as below (as above but with `+ 0.0` removed):

```
v0 = (v01 - v00) * xf + v00
v1 = (v11 - v10) * xf + v10
```

Listing F.28 — Fragment of interpolation code in single precision

Then a larger drift is observed:

```
Errors in the use of raster data with NaN values in big endian byte order:
   Count     Minimum     Average     Maximum   Number of "missing value"
mismatches
   11732      0,0000      0,0000      0,0000      0
   10771      0,0000      0,0000      0,0011      0
```

```
10809      0,0000      0,0006      0,1469      0
10760      0,0000      0,0264     12,2387      6
10814      0,0000      0,3692    130,8832     90
10670      0,0000      1,1468    138,7692    379
10522      0,0000      2,2290    144,7353    799
10379      0,0000      3,5549    158,2099   1383
 9938      0,0000      5,3378    161,3066   2001
 9783      0,0000      7,2493    171,4665   2595
```

Listing F.29 — Drifts of test using single precision

F.5.2. Notes on an optimization strategy (optional)

`TestNaN` and `TestNodata` both use the same optimization strategy for selecting a missing reason in `NO_PASS`, `LAND`, `CLOUD` and `UNKNOWN` precedence order. This demo exploits the facts that:

- All "no data" values used in this demo are greater than all valid values; and
- Missing reasons having highest priority are assigned the highest "no data" values.

The combination of these two facts allows the code to simply check for the maximum value, no matter if the raster has a mix of "no data" and real values. However, this trick would not work anymore if the code did not know in advance that all "no data" values are greater than all of the valid values. If they were less than the valid values, some `>=` operators would need to be replaced by `<=` in the `TestNodata` code (in addition of the `max` function being not applicable anymore). `TestNodata` would be even more complex if the raster had a mix of "no data" values that were both less than and greater than the real values (the use of `abs` could reduce this complexity but would require positive and negative "no data" values in the same range as the absolute values). By contrast, optimizations can be done more easily with NaN because the condition equivalent to #1 is more reliable.

F.6. Additional notes on checking for NaN with GCC (Ecere)

In past initiatives, notably Testbed 19 / GeoDataCubes, Ecere ran into difficulties checking for NaN values in input data in C code compiled with GCC with `-ffast-math` optimizations, even when attempting to explicitly disable *finite math only* (`-fno-finite-math-only` compiler flag) at the function level using `__attribute__((__optimize__("no-finite-math-only")))`.

This might have been caused by a bug in earlier GCC versions, although the GCC compilers used at the time were almost certainly more recent than the version 5.1 where this bug is said to have been fixed. Another possibility is that the string inside `__optimize__()` should have been `-fno-finite-math-only` rather than `no-finite-math-only`, although both seem to have the same effect with GCC 14.2.1, and neither generate a *bad option* warning.

The following C code does seem to correctly detect a NaN value even when compiled with the `-O2 -ffast-math` compiler flags using GCC 14.2.1 on Linux.

Ecere plans to test this same code on additional GCC compiler versions and platforms such as the MinGW-w64 environment used to produce Windows binaries to further validate this approach.

```c
#include <stdio.h>
#include <math.h>

__attribute__((__optimize__("-fno-finite-math-only")))
int isNan(float n)
{
   return isnanf(n);
}

int main()
{
   float n = nan("0");
   printf("isNan(%f) returns %s", n, isNan(n) ? "true" : "false");
   return 0;
}
```

Listing F.30 — Sample C code detecting a NaN value with *finite math only* explicitly disabled at the function level

G

ANNEX G (INFORMATIVE) COG IMPLEMENTATION AND BENCHMARKING (GEORGE MASON UNIVERSITY)

ANNEX G (INFORMATIVE) COG IMPLEMENTATION AND BENCHMARKING (GEORGE MASON UNIVERSITY)

G.1. Introduction

This section outlines the work conducted by George Mason University under OGC Testbed 20 task D122. This task focused on the implementation of Cloud-Optimized GeoTIFF (COG) files for benchmarking purposes. The primary objective was to prepare data for subsequent benchmarking activities, which were carried out under Testbed 20 task D124. The implemented data were used to address key questions regarding the setup of a repeatable benchmark environment for testing cloud-optimized GEOINT imagery solutions. These questions include identifying necessary systems, software, and capabilities, structuring performance tests to measure execution times, error rates, and cloud service usage, and determining optimal tiling and interleaving schemes for accessing imagery content in cloud environments. The results from these performance tests will contribute to the development of GIMI standards and guide future implementors in selecting the most effective imaging solutions.

G.2. Implementation

This task (experiment) involved developing a Python-based converter to transform Earth observation datasets into the Cloud Optimized GeoTIFF (COG) format. The primary goal was to facilitate efficient storage and access of geospatial data in cloud environments. The experiment focused on converting two specific datasets: MERRA-2, which provides atmospheric reanalysis data, and HSLS30, a high-resolution land surface dataset.

During the conversion process, various optimization parameters were tested to evaluate their impact on the performance and usability of the resulting COG files. These parameters included different tiling schemes, compression methods, and interleaving options. The performance metrics assessed included conversion speed, file size, and access efficiency in cloud storage

environments. Usability was evaluated based on the ease of integration with existing geospatial tools and the accuracy of the data representation.

The findings from this experiment provide valuable insights into the best practices for converting Earth observation data into COG format, highlighting the trade-offs between different optimization strategies and their effects on performance and usability. These insights are crucial for improving the efficiency of geospatial data management and accessibility in cloud-based systems.

G.2.1. Experiment Datasets

Two types of data were selected for experiments of COG implementations. They are as follow.

1. MERRA-2 (Modern-Era Retrospective Analysis for Research and Applications, Version 2): MERRA-2 is NASA's comprehensive atmospheric reanalysis dataset, designed to provide a detailed and consistent record of the Earth's atmosphere. This dataset integrates a wide range of observational data, including satellite and ground-based measurements, to produce high-resolution atmospheric variables. The original MERRA-2 data format is NetCDF (Network Common Data Form) and HDF (Hierarchical Data Format), which are widely used for storing large and complex scientific data. These formats facilitate efficient data storage and access, making MERRA-2 a valuable resource for climate research, weather forecasting, and atmospheric studies.

2. HLSL30 (Harmonized Landsat Sentinel-2 Land Surface): HLSL30 is a harmonized surface reflectance product that combines data from the Landsat and Sentinel-2 satellite missions. This dataset provides consistent and high-quality surface reflectance measurements at a 30-meter spatial resolution, making it ideal for land surface monitoring, vegetation analysis, and environmental studies. The original format of HLSL30 data is GeoTIFF, a widely used format for geospatial data that supports metadata and allows for efficient storage and manipulation of raster images. The harmonization of Landsat and Sentinel-2 data ensures that users can seamlessly integrate observations from both missions, enhancing the temporal and spatial coverage of land surface monitoring efforts.

These datasets were selected for their relevance and utility in common geospatial applications, particularly in atmospheric and land surface studies. The objective of converting these datasets into Cloud Optimized GeoTIFF (COG) format is to enhance their accessibility and performance in cloud-based environments, enabling more efficient data analysis and application. In OGC Testbed 20, various schemes for implementing COG were experimented with to benchmark these typical data types in geospatial applications.

G.2.2. COG Experiments

To optimize the conversion of Earth observation datasets into Cloud Optimized GeoTIFF (COG) format, various experimental parameters were tested. These parameters were chosen

to evaluate their impact on performance, storage efficiency, and usability in cloud-based environments.

Block Sizes Tested:

- **128 x 128 pixels**: This smaller block size was tested to determine its effect on data retrieval speed and storage efficiency, particularly for applications requiring high-resolution data access.
- **256 x 256 pixels**: As a mid-range block size, this configuration aimed to balance performance and storage efficiency, providing a compromise between smaller and larger block sizes.
- **512 x 512 pixels**: The largest block size tested, intended to evaluate its impact on reducing the number of read operations required for large-scale data processing tasks.

Compression Methods: The primary focus was on LZW (Lempel-Ziv-Welch) compression due to its advantages in geospatial data applications:

- **Lossless Compression**: Ensures that no data is lost during the compression process, maintaining the integrity and accuracy of the original datasets.
- **Well-Supported Across GIS Platforms**: LZW compression is widely supported by various GIS software, ensuring compatibility and ease of use across different platforms.
- **Efficient for Both Sparse and Dense Data**: LZW compression is effective in reducing file sizes for both sparse datasets (with many zero or near-zero values) and dense datasets (with more uniform data distribution), making it versatile for different types of geospatial data.

G.2.3. Implementation Details

The conversion of Earth observation datasets into Cloud Optimized GeoTIFF (COG) format was carried out with careful consideration of several key implementation details to ensure consistency, accuracy, and efficiency.

- **GDAL for Conversion Process**: The Geospatial Data Abstraction Library (GDAL) was utilized for the conversion process. GDAL is a robust and widely used open-source library for reading and writing raster and vector geospatial data formats. Its comprehensive functionality and support for various formats made it an ideal choice for converting datasets into COG format.
- **Consistent Global Bounds**: To maintain uniformity across different datasets, consistent global bounds were applied. This ensured that all datasets aligned spatially, facilitating seamless integration and comparison during analysis.
- **Preserved Original CRS (EPSG:4326)**: The original CRS of the datasets, EPSG:4326 (WGS 84), was preserved throughout the conversion process. This coordinate systems is a

standard for many global datasets, providing a common reference system that supports interoperability and ease of use in various geospatial applications.

- **Google Maps Compatible Tiling Scheme**: A Google Maps compatible tiling scheme was implemented to enhance the usability of the COG files in web mapping applications. This tiling scheme divides the data into manageable tiles, optimizing it for efficient retrieval and display in web-based GIS platforms.
- **Handling NoData Values**: Special attention was given to preserving NoData values during the conversion process. NoData values represent missing or undefined data points, and their accurate preservation is crucial for maintaining the integrity of the datasets. For more details about de the NoData values see Annex F.
- **Overview Generation**: Proper overview generation was included to improve the performance of the COG files. Overviews are reduced-resolution versions of the dataset that allow for faster data access and visualization, especially when zooming out or viewing large areas.
- **Standardized Naming Convention**: A standardized naming convention was adopted to ensure clarity and consistency in file management. The naming convention followed the format {*experiment_name*}_{*base_name*}_{*date_str*}_{*resolution*}_COG.tif, where:
  - *experiment_name* refers to the specific experiment or project;
  - *base_name* is the original name of the dataset;
  - *date_str* indicates the date of the dataset or conversion; and
  - *resolution* specifies the spatial resolution of the dataset.

G.2.3.1. Python Scripts

Python was used as the core source code for this task. The following source list describes the core Python code.

```
None selected

Skip to content
Using Gmail with screen readers
Enable desktop notifications for Gmail.
   OK  No thanks

Conversations
5.98 GB of 15 GB used
Terms · Privacy · Program Policies
Last account activity: 7 hours ago
Details
import os
from osgeo import gdal

def convert_to_cog(input_path, output_path):
    dataset = gdal.Open(input_path)
    if not dataset:
```

```
            print(f"Failed to open dataset: {input_path}")
            return

        translate_options = gdal.TranslateOptions(
            format='COG',
            creationOptions=[
                "COMPRESS=LZW",
                "BLOCKSIZE=512"
            ]
        )
        gdal.Translate(output_path, dataset, options=translate_options)
        dataset = None  # Close the dataset

    def convert_nc4_to_cog(input_file, subdatasets, base_output_dir, date_str):
        output_dir = os.path.join(base_output_dir, "MERRA2", "MERRA2-512M-LZW")
        os.makedirs(output_dir, exist_ok=True)  # Ensure output directory exists

        for subdataset in subdatasets:
            subdataset_path = f'HDF5:"{input_file}":{subdataset}'
            output_file = generate_nc4_output_filename(output_dir, subdataset, date_
    str)
            try:
                convert_to_cog(subdataset_path, output_file)
                print(f"Converted {subdataset} to COG: {output_file}")
            except Exception as e:
                print(f"Failed to convert {subdataset}: {e}")

    def generate_nc4_output_filename(base_dir, subdataset, date_str):
        variable_name = subdataset.strip("//")
        filename = f"MERRA2_{variable_name}_{date_str}_512m_COG.tif"
        return os.path.join(base_dir, filename)

    def generate_tif_output_filename(base_dir, filename):
        base_name = os.path.splitext(filename)[0]
        output_filename = f"{base_name}_512m_LZW_COG.tif"
        return os.path.join(base_dir, output_filename)

    def process_tiff_files(input_folder, base_output_dir):
        output_dir = os.path.join(base_output_dir, "HLS.L30", "HLSL30-512M-LZW")
        os.makedirs(output_dir, exist_ok=True)  # Ensure output directory exists

        print(f"Processing TIFF files in folder: {input_folder}")
        for filename in os.listdir(input_folder):
            if filename.endswith(".tif") and not filename.startswith("._"):
                input_path = os.path.join(input_folder, filename)
                output_path = generate_tif_output_filename(output_dir, filename)
                print(f"Processing TIFF: {filename}")
                try:
                    convert_to_cog(input_path, output_path)
                    print(f"Converted {filename} to COG: {output_path}")
                except Exception as e:
                    print(f"Failed to convert {filename}: {e}")

    if __name__ == "__main__":
        # Input paths
        nc4_file = "/Volumes/T7 Shield/CSISS/OGC_Testbed_20/Data/MERRA2_400.tavg1_2d_
    slv_Nx.20140101.nc4"
        input_folder = "/Volumes/T7 Shield/CSISS/OGC_Testbed_20/Data/Landsat/HLSL30_
    2.0-20240912_144107"
        base_output_dir = "/Volumes/T7 Shield/CSISS/OGC_Testbed_20/Data/COG_
    converted"

        # Ensure base output directory exists
```

```
    if not os.path.exists(base_output_dir):
        os.makedirs(base_output_dir)

    # Subdatasets for NC4 files
    subdatasets = [
        "//CLDPRS", "//CLDTMP", "//DISPH", "//H1000", "//H250", "//H500", "//
H850",
        "//OMEGA500", "//PBLTOP", "//PS", "//Q250", "//Q500", "//Q850", "//
QV10M",
        "//QV2M", "//SLP", "//T10M", "//T250", "//T2M", "//T2MDEW", "//T2MWET",
        "//T500", "//T850", "//TO3", "//TOX", "//TQI", "//TQL", "//TQV", "//
TROPPB",
        "//TROPPT", "//TROPPV", "//TROPQ", "//TROPT", "//TS", "//U10M", "//U250",
        "//U2M", "//U500", "//U50M", "//U850", "//V10M", "//V250", "//V2M", "//
V500",
        "//V50M", "//V850", "//ZLCL"
    ]

    # Process the NC4 file
    print(f"Processing NC4 file: {nc4_file}")
    date_str = nc4_file.split('.')[-2]  # Extract date from filename
    convert_nc4_to_cog(nc4_file, subdatasets, base_output_dir, date_str)

    # Process all TIFF files in the input folder
    process_tiff_files(input_folder, base_output_dir)
```

Listing G.1

G.2.3.2. Quality Control

Ensuring the quality and integrity of the converted Cloud Optimized GeoTIFF (COG) files is crucial for their effective use in geospatial analysis. The following quality control measures were implemented to maintain high standards throughout the conversion process.

- **Consistent Geospatial Information Across Conversions**: Ensuring that all converted datasets maintain consistent geospatial information is vital for accurate spatial analysis. This includes verifying that the CRS, coordinate bounds, and geotransform parameters are correctly applied and preserved across all conversions. Consistency in geospatial information ensures that the datasets can be seamlessly integrated and compared.
- **Error Handling for Missing Geotransform or Projection Information**: A geotransform, a mathematical transformation consisting of six coefficients that converts pixel coordinates to georeferenced coordinates, is necessary to define the geospatial information. Together with Coordinate Reference Information, such as those described in WKT format, it provides the complete information for defining and providing the geospatial referencing information of an image in GeoTIFF. Robust error handling mechanisms were put in place to address any issues related to missing geotransform or projection information. If such information was found to be missing, the system flagged the error and either prompted for user input or applied predefined default values. This approach prevents the creation of incomplete or inaccurate COG files.
- **Default Values Implementation When Necessary**: In cases where certain metadata or geospatial parameters were missing or incomplete, default values were implemented to ensure the continuity of the conversion process. These default values were carefully

chosen based on standard practices and the specific requirements of the datasets, ensuring that the resulting COG files remained functional and accurate.

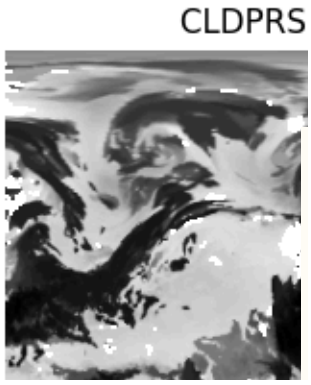
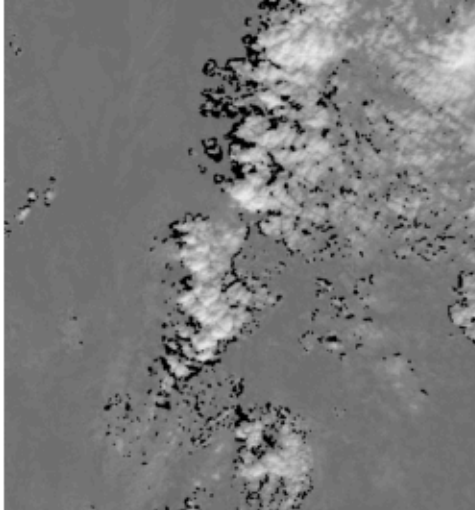
- **Verification of Proper NoData Value Preservation**: Proper preservation of NoData values is essential for maintaining the integrity of the datasets. Quality control checks were conducted to verify that NoData values were correctly identified and preserved during the conversion process. This ensures that areas with missing or undefined data are accurately represented in the COG files, preventing any misinterpretation of the data. Different data formats encode NoData differently. During the process of converting from native data formats to COG data formats, the NoData value may be lost. For example, in NetCDF, a "missing_value" attribute may be used to indicate the NoData. Conversion from these sources data to COG could result in the loss of such NoData information. In this experiment, the missing value tag (i.e., TIFFTAG_GDAL_NODATA) would need to be forcefully created by explicitly declaring the target NoData value, since GDAL is used as the primary driver for creating COG files.
- **Exclusion of System-Generated Hidden Files**: To maintain the cleanliness and organization of the dataset directories, system-generated hidden files (such as macOS "._" files) were excluded from the conversion process. These files can clutter the dataset and potentially interfere with data processing and analysis. By excluding them, the focus remained on the relevant geospatial data, ensuring a streamlined and efficient workflow.

G.3. Results and Discussions

G.3.1. Resulted COG Datasets

The resulting COG formatted datasets are shown in the following table — Table G.1. Two types of datasets were experimented with, including one from the MERRA-2 dataset and another from the Landsat 9 HLS sensor. Three types of block sizes were applied to both datasets: 128 by 128, 256 by 256, and 512 by 512.

Table G.1 — Out

Original			
128 x 128			
256 x 256			
512 x 512			

G.3.2. File Sizes

Table G.2 — Sizes of File

DATASET	BLOCK SIZE	FILE SIZE (MB)
MERRA2	128×128	10.02 MB
MERRA2	256×256	10.00 MB
MERRA2	512×512	9.00 MB

DATASET	BLOCK SIZE	FILE SIZE (MB)
HLS.L30	128×128	2.60 MB
HLS.L30	256×256	2.50 MB
HLS.L30	512×512	2.40 MB

G.3.3. Read Experiments

The following table (Table G.3) shows the results of reading experiments.

Table G.3 — Reading of COG Files

DATASET	BLOCK SIZE	READ TIME (LARGE BLOCK)	READ TIME (SMALL SUBSET)
MERRA2	128×128	0.1536 ms	2.17 ms
MERRA2	256×256	0.0792 ms	6.25 ms
MERRA2	512×512	0.0970 ms	13.87 ms
HLS.L30	128×128	0.1046 ms	0.06 ms
HLS.L30	256×256	0.0005 ms	0.14 ms
HLS.L30	512×512	0.0004 ms	0.40 ms

Observations from the Reading Experiment:

- Smaller block sizes (128×128): These performed better for random read operations, as they allow accessing smaller chunks of data more efficiently; and
- Larger block sizes (512×512): These were faster for full file reads or when accessing large contiguous regions, as fewer tiles need to be read, reducing overhead.

G.3.4. Tiling and HTTP Range Request Support

The tiling and HTTP range request capabilities of the COG files were inspected using gdalinfo. Below is an example output for the 256×256 block size configuration:

```
gdalinfo MERRA2_example.tif | grep "Block"
```

```
Band 1 Block=256x256 Type=Float32, ColorInterp=Gray
```

Listing G.2

Metadata Summary:

- HTTP Range Requests Supported: Yes
- Tile Structure: Verified that all COG files were correctly partitioned into tiles based on the specified block size.
- Overviews: Enabled for better zoom performance.

G.4. Summary of Analysis

- Impact of Block Size: The experiment demonstrated that block size influences access speed:
  - Smaller block sizes (128×128) improved performance for random access.
  - Larger block sizes (512×512) were more efficient for large area reads.
- Impact on File Size: The file size tends to be smaller when the block size is larger, although there is no significant difference on file sizes has been observed in this set of experiments.
- Parallel Reads and Subsetting: COG's internal tiling structure and support for HTTP range requests supports efficient parallel reading and subsetting. The tiled structure ensured that only the necessary tiles were loaded during read operations, enhancing performance.

H

ANNEX H (INFORMATIVE) AGGREGATING DATA WITH GEOTAGGED VIDEO TO IMPROVE COGNITION (AWAY TEAM)

H ANNEX H (INFORMATIVE) AGGREGATING DATA WITH GEOTAGGED VIDEO TO IMPROVE COGNITION (AWAY TEAM)

H.1. Introduction

In recent years, the smartphone industry has helped to revolutionize photography for consumer markets by producing affordable high-quality electronic cameras. Integration of such devices with low-cost Global Navigation Satellite System (GNSS) technologies has spawned a new generation of mass-market location-aware cameras capable of geotagging still and motion imagery, including dash cams, drones and body-worn cameras. Emergence of these markets has driven demand for formats such as EXIF (Exchangeable Image File Format) and WebVMT (Web Video Map Tracks format) that enable content creators to easily share their data online in an accessible format.

The prevalence of these devices has fueled proliferation of publicly available Motion Imagery Standards Board (MISB) Class 3 Imagery online. The ability to access and aggregate such data could significantly enhance the value of geospatial intelligence (GEOINT) data, particularly for Law Enforcement and Public Safety institutions who are already heavily invested in these devices. Adoption of open formats could enable these organizations to easily aggregate with data from external sources including government, industry and crowdsourcing from the public.

H.2. Scope

In previous OGC Testbeds, Away Team demonstrated how WebVMT can make geotagged video more accessible, how moving objects can be tracked using GeoPose, and how data can be accurately synchronized from multiple sources including video. Successful data aggregation relies on accurate synchronization. The Tesbed 20 study builds on previous results to demonstrate how aggregation of geotagged video with external data sources can improve geospatial cognition and add valuable insight for GEOINT use cases.

Video use cases studied include the following.

- **Maritime**: Analysis showing how on-board geotagged video from a sailing boat can be combined with Automated Identification System (AIS) data to improve vessel tracking data.
- **Camera Orientation**: A demonstration of how MISB Key-Length-Value (KLV) metadata can be exported to align a map view with camera orientation in a web browser to improve cognition.
- **Litter Monitoring**: A study of how roadside litter can be monitored using geotagged footage from a fleet of vehicles with dash cams.
- **Video Metadata Search**: Proposed tests with associated use cases to benchmark access times required to search video files for queries matching metadata patterns on the web.

H.3. Maritime Use Case

Geotagged video footage was filmed on a smartphone aboard a yacht in the Solent near the Isle of Wight, UK. This footage was used for study the maritime use case. AIS data were aggregated with on-board video data to demonstrate how geospatial analysis can improve data cognition.

H.3.1. Analysis

In August 2024, a short, geotagged video was filmed aboard a yacht under sail using an Android device running Away Team's TrkdCam application. The data were automatically timestamped using the WebVMT media start time in Coordinated Universal Time (UTC). Previous synchronization work in OGC Testbed 19 (OGC 23-042) quantified typical timing errors to be within 2 seconds due to device latencies.

H.3.1.1. Data Aggregation

AIS data were downloaded from a vessel tracking website using the Maritime Mobile Service Identity (MMSI) of the yacht. These data included the vessel's location, heading and speed at intervals varying between 3 and 10 minutes over a period of around 6 hours. Satellite-AIS data were not available, so the yacht's trajectory was divided into two segments that correspond to the outward and return legs of the trip.

Open source data from OpenSeaMap were integrated with the WebVMT JavaScript playback engine to display nautical chart data including seamarks.

A new utility was developed from the GPX Importer community tool to import AIS data in a Comma Separated Value (CSV) file into WebVMT format. WebVMT code developed in OGC Testbeds 17 and 19 was used to synchronize and aggregate imported AIS data with on-board video metadata using the clip and merge functions.

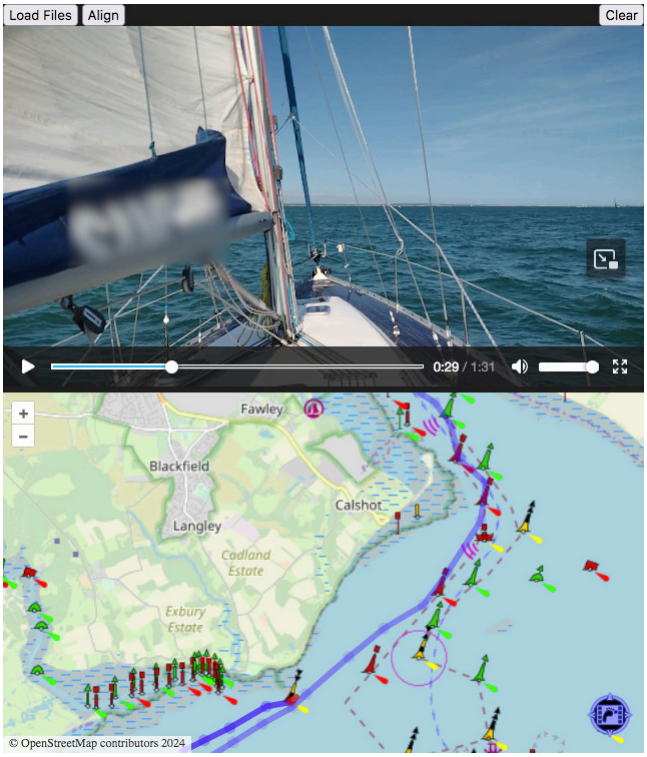


Figure H.1 — AIS Data Synchronized and Aggregated With On-Board Video

The AIS trajectory in Figure H.1 is shown as a faint blue line for the outward journey and a stronger blue line for the return trip. AIS data locations are marked along the line to highlight their sparse temporal and spatial nature. On-board data is shown in orange in Figure H.2.

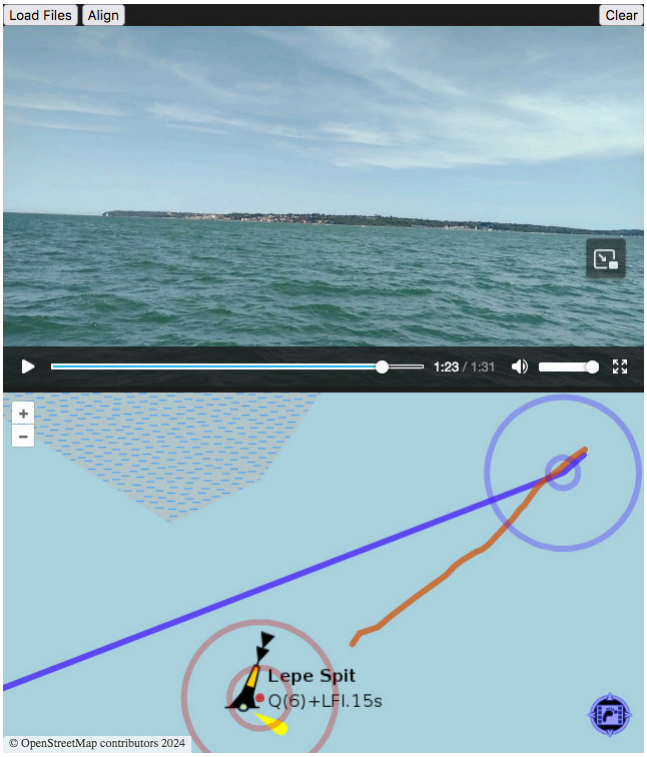


Figure H.2 — Comparison of AIS Trajectory (Blue) With On-Board Trajectory (Orange)

The location of the Lepe Spit mark — shown in red in Figure H.2 — was added from the 2024 GPX data published on the Solent Cruising and Racing Association (SCRA) website and circular zones were added with 20m and 50m radii for scale. Circles were also added to the AIS data location with 10m and 50m radii to help gauge distance.

There is a discrepancy between the mark's 2024 location and its OpenSeaMap position, though some margin of error should be allowed for tidal effects and the locations are still within 20 meters. AIS and on-board trajectories are within 10 meters once they pass the AIS data point in Figure H.2. The tracks diverge prior to this data point so further analysis was required.

H.3.1.2. GEOINT Analysis

At the start of the footage, the Lepe Spit mark is less than 100m away and is clearly visible in the video imagery. This mark closely resembles the symbol indicating its location on the nautical chart. Information published on the UK Trinity House website confirmed that this is a South Cardinal mark designed to warn of a maritime hazard to its north, and advises that "mariners will be safe if they pass…south of a south mark." An area of shallows is marked to the north of its location on the OpenSeaMap chart and therefore it is reasonable to assume that the vessel passed to the south of this mark.

The interpolated AIS track passes north of this mark in Figure H.2, so a single additional data point was added to the trajectory to correct this error. This location was estimated to be 20m from the Lepe Spit mark location to allow a margin of error for safe navigation. The trajectory

time was estimated from linear interpolation of the two adjacent AIS data points assuming movement over ground as a first approximation.

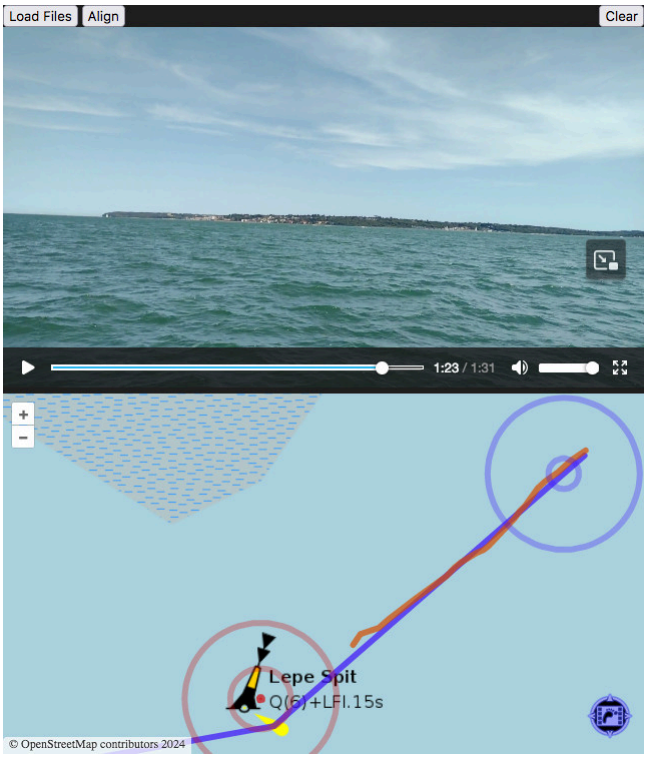


Figure H.3 — Comparison of Corrected AIS Trajectory (Blue) With On-Board Trajectory (Orange)

The corrected AIS trajectory shown in blue in Figure H.3 closely matches the on-board data shown in orange — both in terms of location and time. This estimate could be refined to model contributions from tide, engine, wind and sail, but the simple approximation used here is consistently accurate to within 10m and serves to demonstrate how basic aggregation can significantly enrich geospatial data.

H.3.2. Results

Geospatial data from multiple sources were aggregated and accurately synchronized with video using WebVMT. These data were successfully integrated with other web technologies in a web browser to help guide GEOINT analysis and were significantly enriched using simple techniques. Metrics show that the results obtained are accurate to within ten meters and synchronized to within two seconds.

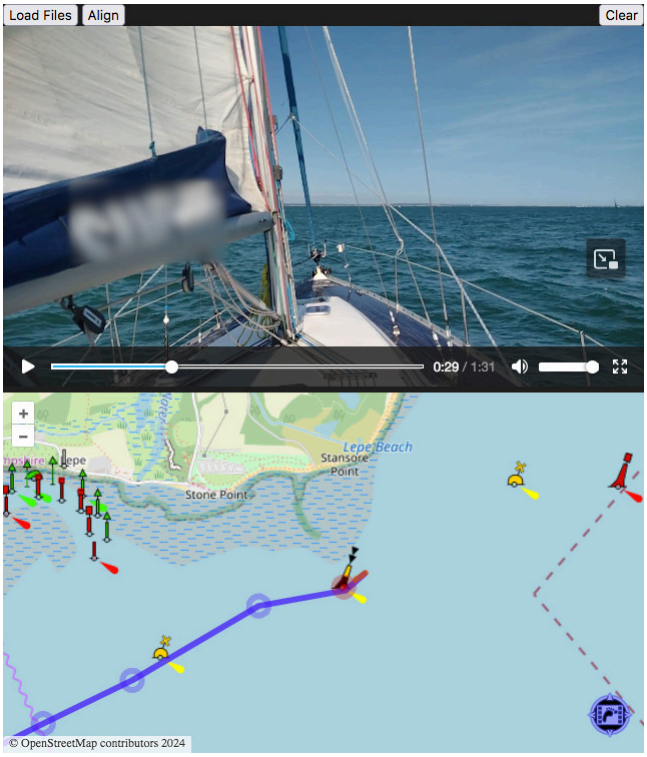


Figure H.4 — Corrected AIS Data (Blue) Aggregated With On-Board Data (Orange) Synchronized With Video

WebVMT tools developed in OGC Testbeds 17 and 19 were reused and updated to support negative cue times for non-destructive synchronization. A new utility was created to import GPX data — based on the online community tool for importing GPX data — and extended to read CSV files to facilitate data migration to WebVMT.

OpenLayers API was integrated with the WebVMT JavaScript playback engine to support OpenSeaMap nautical chart data. A new engine feature was prototyped to generate an event after the map element has been initialized in the browser which enables the OpenSeaMap seamark layer to be properly integrated.

The prototype code utilizes the HTML Event interface (`CustomEvent`) to trigger execution of user code, instead of allowing that code to be called directly by the current thread. This design enables proper sandbox isolation that mitigates any threat from malicious code in the webpage and is consistent with HTML security guidelines.

H.4. Camera Orientation Use Case

Geotagged video data of a truck traveling along a highway, filmed from an aircraft flying over Wyoming, was used to demonstrate camera orientation support and accuracy in a web browser.

H.4.1. Analysis

A video file with in-band metadata encoded in MISB ST 0601 was identified as a good candidate with which to investigate camera orientation issues. The footage was filmed from an aircraft that was circling over Wyoming, USA with a video camera trained on a truck that was traveling along a highway below. The relative motions of the aircraft and truck result in a complex sequence of camera angles that are constantly changing.

A C++ utility developed in OGC Testbed 16 was modified to export the SMPTE Key Length Value (KLV) sensor location (MISB ST 0601 tags 13, 14 & 15) and frame center (MISB ST 0601 tags 23, 24 & 25) video metadata to generate two WebVMT paths. These trajectories represent the tracks of the camera and its subject location respectively.

H.4.1.1. Camera Heading

Map rotation support is included in several publicly accessible Web Map APIs including OpenLayers and MapLibre. The WebVMT JavaScript playback engine was updated to include the MapLibre API in order to prototype this feature.

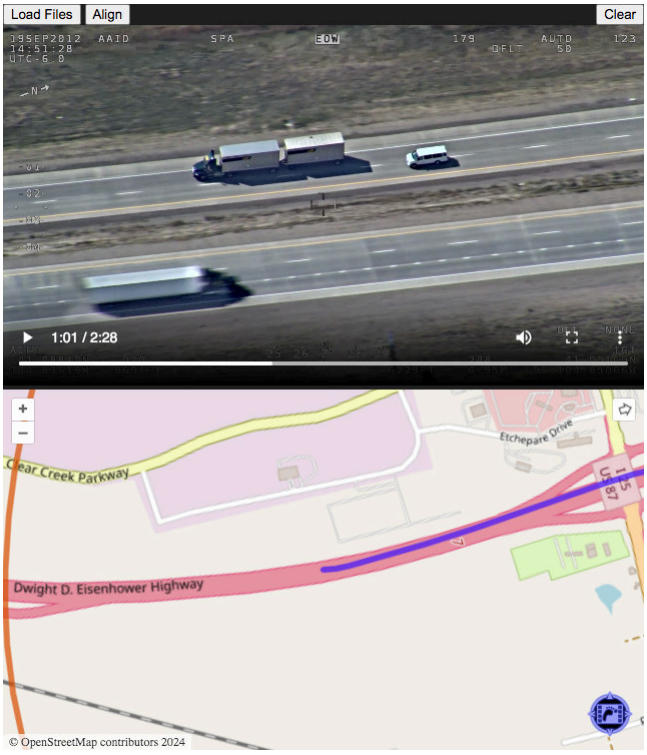


Figure H.5 — OpenLayers Map Rotation Aligned With Video Camera Heading

A WebVMT heading attribute was prototyped in OGC Testbed 19 for the sight lines used to track vehicles from video footage (OGC 23-042). These data were also included in the Hillyfields Bubble dataset that was published under an open source license in OGC Testbed 19, and subsequently proposed as a new feature in the WebVMT Community Group (CG). The CG

proposal was extended to include a heading attribute for map rotation and implemented as a prototype feature in the WebVMT engine.

A new utility was developed to calculate the heading of one WebVMT path from another. This was used to create WebVMT content centered on the truck trajectory with the map rotation automatically aligned to the camera heading.

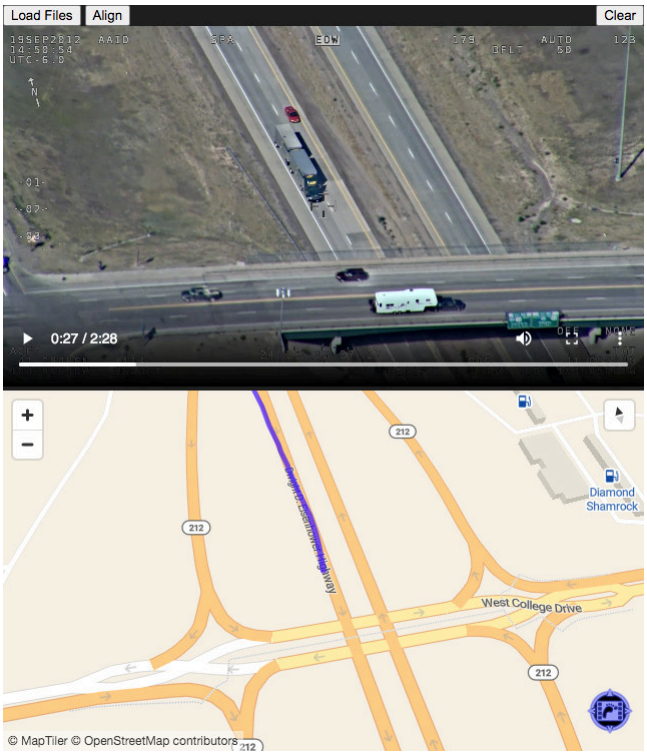


Figure H.6 — MapLibre View Aligned With Video Camera Orientation

Figure H.6 shows the frame center track on the map in blue which is marked by a white crosshair baked into the video frame above. This is a proxy for the truck's trajectory in this case.

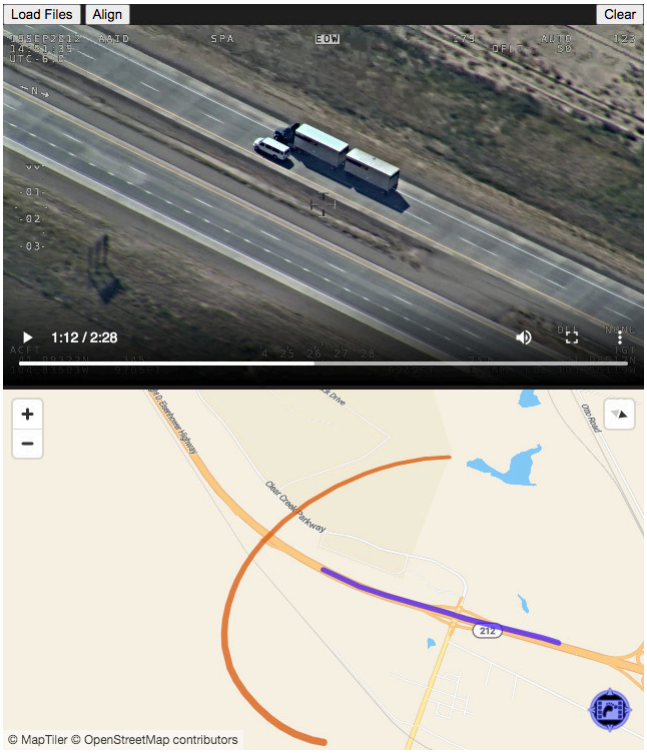


Figure H.7 — Comparison Between Paths For Camera (Orange) & Frame Center (Blue)

There are two ways in which to verify that the map rotation accurately reflects the camera orientation in this example:

1. The video includes a compass indicator which is baked into the image on the left-hand edge towards the top of the video frame in Figure H.7. This indicator matches the compass direction shown in the top right corner of the map element. These indicators are also visible in Figure H.6 and Figure H.5.

2. Figure H.7 shows the paths of the camera in orange and frame center in blue. The tip of the blue path is always vertically above the tip of the orange path which also confirms that the map rotation is correct.

H.4.1.2. Cognitive Analysis

Loading these same video and VMT files into a web page with photogrammetric data can assist cognitive analysis of features that are visible in the video, though may not be marked on a map.

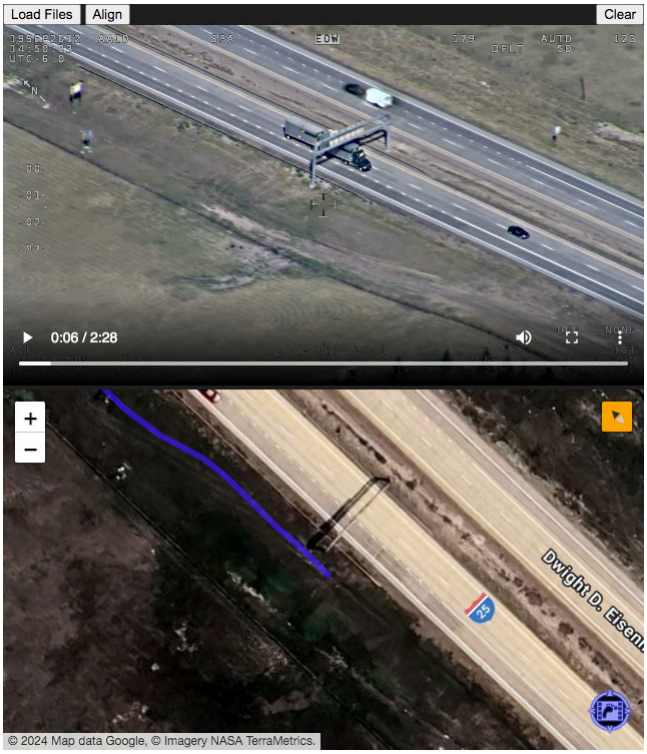


Figure H.8 — Truck Video With Rotated Photogrammetry Data

The top video panel in Figure H.8 shows the truck passing under a sign that spans the road on a gantry. The bottom panel shows the photogrammetry data synchronized to the video with WebVMT which includes several notable features.

1. The sign gantry is visible in the center of the photogrammetry data and can be recognized from the shadow that it casts. This prominent feature provides a simple check that the data are probably in the right location.

2. A paler area of ground that is parallel to the highway and next to the gantry can be seen beyond the fence line in the foreground of the video frame. This paler area extends to the near side of the fence line to form a shape that is similar to the letter 'J'. A feature with this same unique shape can be seen in the lower map panel below the end of the blue line which provides good evidence that these two images show exactly the same location.

3. The tip of the frame center trajectory shown in blue in the map panel accurately matches the location of the crosshair in the video frame above. The relative location of the trajectory tip to the foot of the gantry support indicates that this is accurate to within a meter or so. This provides a metric of the data accuracy and demonstrates good synchronization.

4. Having established that the video and photogrammetry images show the same location, less obvious features can also be identified. Crash barriers can be seen on either side of the near carriageway which lead up the sign gantry and then end a few meters beyond this point. These features are visible in both images and their locations can be accurately identified.

5. The inside lane markings change from short dashed lines to a continuous white line on both carriageways shown on the right side of the video image. The same changes in lane markings can be seen towards the right of the photogrammetry data. The locations of these points can be accurately identified.

H.4.2. Results

Aligning map rotation with camera orientation improves geospatial cognition of geotagged images and simplifies GEOINT analysis. Vehicle satellite navigation systems and smartphone navigation applications have adopted the same proven technique to display geospatial information in a concise form that is aligned with the user's heading.

A new utility was developed to calculate the heading of one WebVMT path from another to support camera orientation. The data produced by this tool successfully demonstrated that WebVMT can synchronize and interpolate dynamic data, including camera orientation, to match video imagery with frame-level accuracy. Accurately synchronizing photogrammetry data with a video frame enables location verification, provides accuracy metrics and facilitates cognition of details that might otherwise have been overlooked in GEOINT analysis.

H.5. Litter Monitoring Use Case

Geotagged video observing target litter items was captured by front and rear dash cams at the Ordnance Survey headquarters near Southampton, UK to demonstrate how roadside litter can be monitored using footage from a vehicle fleet.

H.5.1. Data Capture

In April 2024, Away Team collaborated with Ordnance Survey (OS) at their headquarters site near Southampton, UK to capture geotagged video for a roadside litter monitoring use case.

Target litter items were placed at known roadside locations on site and filmed using vehicles equipped with dash cams. The litter included tins, drink cans and plastic bottles in a variety of sizes and colors, and traffic cones. The cones were identified in OGC Testbed 19 (section 7.4, OGC 23-042) by the UK National Highways Agency as an item of interest for collection.

Two vehicles were equipped with front- and rear-facing cameras to capture geotagged video.

1. A car with a Nextbase 322GW GPS HD dash cam and rear window cam.
2. The OS Streetdrone with IMU and extra cameras: Ring RBGDC50 at the front and Zed 2i stereo at the rear.

Figure H.9 — Ordnance Survey StreetDrone Equipped With Extra Cameras

Five test runs were completed to capture data from the same road circuit. Alternate vehicles were used for each successive run.

H.5.2. Analysis

Each vehicle produced two separate geotagged video files corresponding to its front and rear cameras. Video output from the Nextbase system was automatically segmented into files with a maximum duration of one minute. This automatic feature allows short video files to be submitted as police evidence without any editing process to ensure that their forensic integrity is preserved.

Figure H.10 — Nextbase Front And Rear Cameras In Situ

The first challenge was to handle the file segmentation process seamlessly.

H.5.2.1. Segmentation Issues

Video metadata were exported to WebVMT files in two stages.

1. In-band video location metadata were exported to GPX files using the open source exiftool.

2. GPX data were converted to WebVMT files using a new utility based on the GPX Importer community tool.

Data integrity was verified by confirming that the location data for each front camera video file matched the equivalent rear camera video file. However, video file segmentation was found to produce spurious segmentation of the vehicle trajectory.

Analysis showed that segmentation of a time sequence — such as a trajectory — into one-minute sections can, for example, produce segments that each contain six values at intervals of ten seconds. However, these data only represent five intervals, since the current section includes the start value of the sixth interval but not *end* value. This 'missing' value is included in the *next* section. Incorrect handling of this final interval can lead to spurious segmentation of the trajectory.

An ingestion process was devised to update the previous segmented file when the following file segment was received to ensure that trajectories remained continuous.

H.5.2.2. Litter Categorization

The next challenge was to identify litter observations from the video footage.

Classification of litter items is logistically important to ensure that the assigned collection team is properly equipped to remove the identified items. Glass bottles are heavier and bulkier to carry than their plastic counterparts, and a broken glass bottle is more hazardous to collect than a plastic one.

Figure H.11 — Target Litter Items Used In Data Capture

Reference photos were used to create a catalogue of litter locations and classify each item with a unique identifier such as 'clear-plastic-bottle-01'. This allowed the same target item to be identified from multiple observations.

Being able to categorize and count litter items is important to successfully monitor accretion and deploy collection teams efficiently. Tracking individual small items such as bottles or cans is impractical and unnecessary. However, tracking larger objects such as a mattress or a wooden pallet is important since these items can pose a significant threat to the safety of road users. Larger items may require specific equipment to safely collect and remove them, so this information is vital for collection team deployment.

H.5.2.3. Video Metadata Accuracy Issues

Litter observations were identified from manual inspection of the footage. This provided valuable insight into issues involved in identifying litter objects from video and highlighted issues that also affect automated processes and manual verification.

Finding the right tool for video metadata analysis is essential. The key features that have been identified are as follow.

1. **Millisecond Timing**: Video frame rates are typically captured at 24 or 30 frames per second (fps) so a single frame lasts around 0.042 or 0.033 seconds respectively. Millisecond resolution is required to accurately synchronize metadata with video. This accuracy is reflected in the cue times used by WebVTT and WebVMT for video metadata on the web.

2. **Frame-By-Frame Control**: Precise control over the individual video frame displayed is vital. Comparison between the current frame and *both* adjoining frames — before and after — can quickly reveal changes to the visibility of an object. This control is required to determine the exact moment when an observation starts or ends.

3. **Playlist Support**: Segmentation of video files can be addressed by use of a playlist to reassemble the segments in the correct order. Such playlists should also provide accurate timing and frame control — particularly with respect to the transitions between file segments.

4. **Frame Zoom & Pan**: Modern cameras typically produce high-resolution output which includes fine levels of detail that may be difficult to see. Manual inspection of this information requires a viewer that can magnify part of the video frame. This control should be independent of the screen size to properly examine details within the image.

H.5.2.4. Geotagging Observations

Once the tools were in place, the next challenge was to identify when observations started and ended.

Visual observation of objects in motion imagery are affected by several factors.

1. **Camera Field of View**: Observations pass into or out of a sensor's field of view when the camera passes too close to a target object or when the camera heading and target bearing diverge sufficiently. This transition provides a clear start or end time for the observation.

2. **Video Resolution & Compression**: Distant or small objects may be within the sensor's field of view but are not visible due to limitations of the camera's resolution. This transition is gradual so the start or end time of the observation may be unclear or subjective. Image compression can contribute to this process

when fine detail is obscured by noise in the compression algorithm, though this can make the transition more abrupt.

3. **Obstructions**: Objects that obscure the camera's sight line to a target object also affect the start and end times of an observation. Background obstructions such as hedges and fences at the roadside typically alter the start or end time of an observation. Foreground obstructions such as bushes, lamp posts and other vehicles can cause temporary obstructions where an observed object is briefly hidden and then reappears. Care should be taken not to interpret a temporary obstruction as two observations of different litter items.

H.5.2.5. Data Synchronization

WebVMT negative cue times were prototyped in OGC Testbed 19 (section 7.4, OGC 23-042) for non-destructive synchronization of data. These were used in OGC Testbed 20 to synchronize the cumulative dash cam trajectory data with each video segment file.

Figure H.12 — Front Video Segment With Litter Observations (Red) & Dash Cam Trajectory (Blue)

Figure H.12 shows a video segment synchronized with the camera trajectory — in blue — which extends prior to the start of that file without loss of data integrity.

Litter observations were also synchronized and aggregated with the camera trajectory. An active observation at the start of a new video segment file can be recorded with a negative cue start time. This prevents truncation due to segmentation and ensures that data integrity is retained.

Figure H.13 — Rear Video Segment With Litter Observations (Red) & Dash Cam Trajectory (Blue)

Figure H.13 shows a target litter item — in red — observed by the rear camera. This same item was also observed by the front camera in Figure H.12.

These data were ready for further study but insufficient time remained in OGC Testbed 20. This analysis could be deferred to OGC Testbed 21 as described in Future Work.

H.5.3. Results

A number of video players were evaluated for use as an inspection tool and compared against the requirements identified for metadata analysis. These features are:

- Millisecond timing display.
- Forward and backward frame step control.
- Playlist support with frame step integration.
- Image magnification with pan.

Table H.1 — Key Analysis Features Supported by Video Players.

PLAYER (VERSION)	MILLISECOND DISPLAY	FRAME STEP	PLAYLIST	ZOOM & PAN
mpv (0.33.1)	Yes	Yes	No previous frame step between files	Yes
IINA (1.3.1)	Frame accuracy bug	Yes	No previous frame step between files	Yes
VLC (3.0.21)	Not updated by frame step (extension)	No previous frame step	No frame step between files	Yes
ffplay (4.4.1)	10ms resolution	No previous frame step	No playlist support	No zoom support

The mpv application offered the best support for the identified analysis requirements. Zoom and pan support were provided by adding the following customized keyboard shortcuts:

```
q   add video-zoom -.25
e   add video-zoom .25

w   add video-pan-y .05
d   add video-pan-x -.05
x   add video-pan-y -.05
a   add video-pan-x .05

s   set video-pan-x 0; set video-pan-y 0; set video-zoom 0
```

Listing H.1 — Zoom/Pan Keyboard Shortcuts For mpv (input.conf)

The IINA application is based on mpv and offered comparable support for the analysis requirements but was hindered by a bug in the timing display.

Factors affecting observation visibility in the video were identified including field of view, obstruction by foreground and background objects, and video resolution and compression.

Front and rear litter observations have been geotagged in segmented video files without loss of data integrity. This dataset is ready for further study but insufficient time remained in OGC Testbed 20 to complete this analysis. This analysis could be deferred to OGC Testbed 21 as described in Future Work.

H.6. Video Metadata Search Use Cases

Benchmark tests were identified for video metadata queries which provide structured metrics for accessing metadata in GIMI video files with associated use cases.

These tests are equally applicable to in-band and out-of-band metadata queries and serve to highlight the differences between them.

H.6.1. GIMI Design Considerations For Video Metadata

In-band and out-of-band metadata encodings are different approaches to the same problem that are not mutually exclusive. Both have strengths and weaknesses, and the best choice should be determined from analysis of the individual use case.

H.6.1.1. In-Band and Out-of-Band Metadata Comparison

The Society of Motion Picture and Television Engineers (SMPTE) KLVs adopted by MISB are a good choice for encoding in-band metadata within a video file and certainly have advantages for real-time use cases where timely delivery is important, such as broadcasting. However, many web and geospatial use cases, such as web crawling, are better served by out-of-band metadata.

By definition, any in-band encoding requires users to read and decode the video file structure in order to access the metadata as discussed in section 8.2 of OGC Testbed 16 D021 Engineering Report (OGC 20-036). This imposes significant access overheads — both in terms of bandwidth and accessibility.

1. Metadata typically constitutes less than 1% of a video file, so 99% of the file data is not relevant in a web crawler use case.
2. In-band KLVs are embedded within the video file, so access to them is dependent on successfully navigating the video encoding format.

By contrast, out-of-band metadata is stored in a sidecar file which eliminates both these issues and utilizes the inherent hyperlink nature of the web. This file only contains the metadata, so access is low-bandwidth and agnostic of the video encoding.

In addition, out-of-band encoding can help speed up the implementation process. For example, video encoding is a function of smartphone operating systems such as Android and iOS, so implementation of changes to in-band metadata encoding is completely reliant on Google and Apple respectively. However, out-of-band metadata solutions can be implemented by any developer and thus avoid such bottlenecks.

H.6.1.2. JSON Encoding of SMPTE KLV

A simple solution is to encode KLVs as key-value objects in JSON format, and synchronize these with video files using the WebVMT sync command.

```
NOTE KLVs encoded in WebVMT

00:00:12.345 -->
{ "sync": { "type": "org.ogc.testbed-20.gimi.example.klvs", "data":
  { "key-1": "value-1",
    "key-2": "value-2",
    "key-3": {"key-3a": "value-3a", "key-3b": "value-3b" } }
```

```
} }
```

Listing H.2 — JSON Encoding of SMPTE KLVs in WebVMT

H.6.1.3. TAI & UTC Timestamps

WebVMT supports Coordinated Universal Time (UTC) timestamps and uses WebVMT media start time to set the date-time offset of the media start.

```
WEBVMT

MEDIA
url:my-video.mp4
mime-type:video/mp4
start-time:2018-02-19T12:34:56.789Z

NOTE Zero on the media timeline for the my-video file corresponds to UTC
timestamp 2018-02-19T12:34:56.789Z
00:00:00.000 -->
...
```

Listing H.3 — UTC Timestamp Support in WebVMT

This design mirrors the 'timeline offset' property of `HTMLMediaElement` which is exposed through the `getStartDate()` method in HTML.

HTML also supports negative cue times for metadata timestamps that occur prior to the media start time. This was identified as a WebVMT requirement in OGC Testbed 19 for synchronization of traffic data and a prototype solution was used extensively in the Hillyfields Bubble dataset.

Negative cue time support has been proposed in the WebVMT Community Group and is planned for inclusion in WebVMT. HTML already supports both negative cue times for non-destructive synchronization and unbounded cues for live streaming use cases. The IANA media type `text/vtt` supports neither feature and WebVTT is unlikely to include these in future due to a lack of use case requirements for timed text.

A new IANA media type `text/vmt` should be considered to support these two breaking changes to `text/vtt` and improve timed video metadata integration with web browsers.

H.6.2. Video Search Benchmark Analysis

WebVMT offers a common format for sharing timed video metadata on the web that is agnostic of video encoding and accessible to web crawlers. This enables online search engines to handle metadata-related queries for video data such as GIMI files.

Efficient access to metadata-related content in a video file is an important benchmark for the proposed GIMI design and serves to highlight key differences between the in-band and out-of-band metadata approaches.

The search process has two stages:

1. Access metadata to query matching patterns; and

2. Retrieve video content for positive matches.

These two steps may be combined for in-band metadata use cases to help mitigate the access overhead. Selective query benchmarks may have similar file access requirements to the first benchmark. However, they also serve to quantify the relative sizes of the metadata access overhead, query selection and retrieval steps in the overall process.

To compare in-band and out-of-band queries, identical metadata should be encoded into both formats for direct comparison.

H.6.2.1. Benchmark 1: Query For All Video Metadata

The first benchmark measures the time taken to access all in-band metadata in a video file. This corresponds to the export use case where all in-band video metadata are exported to an out-of-band format such as WebVMT.

This code provides a useful steppingstone in the development process by forming a chassis on which to build the filtering steps needed for later benchmarks. The export metric quantifies the overhead incurred by parsing timed metadata encoded within a video file.

H.6.2.2. Benchmark 2: Video Metadata Query For an Exact Match

This benchmark measures the time taken to find metadata values which exactly match a target pattern in a video file.

The simplest query is for an exact match where the metadata value is identical to the target pattern. An example of this is the Automatic Number Plate Recognition (ANPR) use case for Law Enforcement and Public Safety.

Video can be searched for footage showing a vehicle whose registration plate matches the target value to track its location. In traffic flow use cases, vehicle registration data are often obfuscated with hash values for privacy and security reasons. This protects personal data, stops speculative queries and prevents partial matches, while still permitting exact matches for a hash code when the complete registration is known.

H.6.2.3. Benchmark 3: Video Metadata Query For a Threshold Match

This benchmark measures the time taken to find metadata values that exceed a threshold in a video file.

A simple numerical query is for a metadata value that exceeds a target threshold. The vehicle collision monitoring use case provides an example of this requirement.

A dash camera with an accelerometer can identify when a vehicle has been involved in a collision. An impact can be identified from a characteristic spike in accelerometer data which exceeds a given threshold value. Video footage can then be preserved to provide forensic evidence of the incident and help to identify what happened, when and where this occurred

and who was responsible. This feature is often included in consumer devices and can be used commercially to monitor vehicle fleets to safeguard staff and protect company assets.

H.6.2.4. Benchmark 4: Video Metadata Query For a Location Match

This benchmark measures the time taken to find metadata values that fall within a geospatial region of interest in a video file.

A basic location query is for metadata values that fall within a given radius of a geospatial target point. An example of this is the missing person use case.

The last sighting of the missing person provides the starting location and time elapsed since that sighting can be used to estimate how far they may have traveled in the interim. A geotagged video search can be limited to a given locality to eliminate irrelevant data and locate the missing person more quickly.

This benchmark could be refined to discriminate between two- and three-dimensional location data, since height is only required in the latter case. Geofencing offers another refinement where more complex proximity calculations are required.

H.6.3. Results

Use of out-of-band video metadata can improve data privacy and security since the sidecar file has security permissions that are separate from the video file. This enables the two files to have different access permissions. More open permissions may be granted to access the sidecar file which enables search queries to determine whether the video file is of interest. This approach does not require direct access to the video content which may contain personally identifiable information such as images of faces or vehicle registrations.

Out-of-band metadata search queries only require access to the associated video file when a positive result is returned. Monitoring this file access pattern could identify malign access and help to curb web scraping activities employed by unscrupulous organizations abusing Big Data and Artificial Intelligence (AI) technologies.

In addition, use of out-of-band metadata queries should reduce file access bandwidth since a negative result only requires access to the smaller sidecar file and should never access the larger video file.

No benchmark metrics were calculated in OGC Testbed 20 due to the lack of suitable GIMI video files for analysis.

H.7. Conclusions

GEOINT Imagery Media for ISR (GIMI), encapsulated within NGA Standard 0076 is designed to revolutionize management, integration and utilization of still and motion imagery in the defense and intelligence communities.

Identifying relevant use cases is important to ensure that GIMI design is fit for purpose, particularly for Intelligence, Surveillance and Reconnaissance (ISR) applications. Alignment with web-based technologies — and HTML in particular — is key to design interoperability that will integrate smoothly with other systems.

H.7.1. Maritime

Away Team has successfully demonstrated how geospatial data from multiple sources can be aggregated and accurately synchronized with video using WebVMT. Web browser integration with web technologies such as OpenSeaMap has made data more accessible, helped to guide GEOINT analysis and significantly enriched data using geospatial techniques.

H.7.2. Camera Orientation

Combining camera orientation information with map and photogrammetry data can significantly improve cognition of imagery including features that are not normally marked on maps such as trees and road furniture. This demonstration highlights the importance of camera orientation data in the GIMI standard for still and motion imagery, and how GEOINT analysis can benefit from this information.

H.7.3. Litter Monitoring

Video player requirements have been identified for geotagged video analysis including millisecond display, frame-level control, playlist support for segmented video, and zoom and pan to identify visual details within video frames.

Factors affecting observation visibility in video have been identified including field of view, obstruction by foreground and background objects, video resolution and compression.

The litter monitoring dataset is ready for investigation but there was insufficient time in OGC Testbed 20 for further study. This analysis could be deferred to OGC Testbed 21 as described in Future Work.

H.7.4. GIMI Video Metadata Considerations

MISB Class 3 Imagery is an increasingly important resource due to proliferation of location-aware devices with cameras such as drones, dash cams, body-worn video and smartphones.

In-band and out-of-band metadata have strengths and weaknesses which depend on the individual use case. These two approaches are not mutually exclusive, and GIMI video metadata design should address both options.

JSON encoding of SMPTE KLVs can be facilitated by WebVMT which is well-integrated with HTML and CSS for access, sharing and searching online.

TAI and UTC timestamps can be integrated with GIMI for in-band and out-of-band video metadata.

H.7.5. Video Metadata Search

Out-of-band metadata allows web crawlers to access video metadata on the web using sidecar files in a format that is agnostic of the video encoding. This enables online search engines to support video metadata queries.

Use of out-of-band metadata queries can improve privacy and security of personally identifiable information contained in video files, help to identify malign access and reduce file access bandwidth.

Benchmark tests have been designed to provide metrics for video metadata search access with identified use cases.

H.7.6. WebVMT Enhancements

Analysis of these video use cases has highlighted new requirements for the WebVMT design which have been implemented as prototypes in the engine and tools.

OpenLayers and MapLibre APIs have been integrated with the WebVMT JavaScript playback engine to support a wider range of map data and interface capabilities.

A new WebVMT feature has been prototyped to facilitate integration with maps that have customization requirements such as OpenSeaMap.

A new WebVMT utility has been created to calculate the heading of one path from another to support camera orientation.

WebVMT tools developed in OGC Testbeds 17, 19 and 20 have been updated into a single Perl command-line utility designed to create and manipulate WebVMT files to facilitate video metadata analysis.

The online community tool for importing GPX data has been ported to a command-line utility and extended to read CSV files to help simplify data migration to WebVMT.

Negative cue times and unbounded cues are breaking changes to WebVTT and the `text/vtt` MIME type. A new IANA media type `text/vmt` should be considered to support these HTML features and improve timed video metadata integration with web browsers.

H.8. Future Work

A number of topics are recommended for investigation in OGC Testbed 21.

H.8.1. Data Analysis of Litter Monitoring Use Case

Analyze data gathered in OGC Testbed 20 to demonstrate how roadside litter can be monitored using dash cam footage from a vehicle fleet.

1. Show how a database of litter observations can be accessed by a web search engine to demonstrate the video metadata search use case.
2. Demonstrate how an online database can be used to monitor litter accretion rates.
3. Quantify how data from rear-facing dash cams can add value to front dash cam footage for litter monitoring.

H.8.2. Benchmark Metadata Search Use Cases With GIMI Video

Identify suitable files for video metadata benchmark testing and calculate metrics to compare in-band and out-of-band search request to:

1. Return all metadata for a video file;
2. Query video metadata that exactly matches a target value;
3. Query video metadata that exceeds a numerical target value; and
4. Query video metadata that is within a geospatial target region.