Flexible real-time data processing and visualization workflows emerging from OGC API modules

Publication Date: YYYY-MM-DD

Approval Date: YYYY-MM-DD

Submission Date: YYYY-MM-DD

Reference number of this document: 21-033

Category: Discussion Paper

Editors: Jérôme Jacovella-St-Louis

Title: Flexible real-time data processing and visualization workflows emerging from OGC API modules

---

Discussion Paper

LICENSE AGREEMENT

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

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

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

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

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the

Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Table of Contents

Chapter 1. Summary

1.1. Subject

This Discussion Paper documents the results and recommendations of the 2020-2021 GeoConnections project entitled *"Flexible real-time data processing and visualization workflows emerging from OGC API modules"*, also referred to by the shorter name *"Modular OGC API Workflows"*.

This project has been financially supported by GeoConnections, a national collaborative initiative led by Natural Resources Canada. GeoConnections supports the integration and use of the Canadian Geospatial Data Infrastructure (CGDI), an on-line resource that improves the sharing, access and use of open geospatial information.

Ecere Corporation led the project as the proponent, with the participation and contributions of collaborating OGC Members as well as two additional Canadian research centers:

- Université Laval - Département des sciences géomatiques
- Institut national de la recherche scientifique (INRS) - Centre Eau, Terre, Environnement
- EOX IT Services
- Spatialys
- rasdaman
- 52° North
- Pangaea Innovation
- Universitat Autònoma de Barcelona (UAB) - Center for Ecological Research and Forestry Applications (CREAF)
- CubeWerx
- Centre de Recherche en Informatique de Montréal (CRIM)

An important output of this project is draft specifications for an extension to *OGC API - Processes, Part 3: Workflows and Chaining* (OGC document 21-009 [https://docs.ogc.org/DRAFTS/21-009.html]), allowing to:

- chain nested processes,
- refer to external processes and collections accessible via OGC API standards, and
- trigger execution of processes through OGC API data delivery specifications (such as *OGC API - Features*, *Tiles*, *Maps* and *Coverages*).

During the project, the *OGC API - Processes* Standard Working Group has adopted a revised charter to continue developing this specification with the intent to eventually propose it to the OGC Technical Committee as an official OGC standard.

1.2. Executive Summary

The work on the project officially started on June 1st, 2020 and completed on March 31st, 2021. The collaborating organizations explored the use of modular OGC API specifications to empower geospatial researchers with the ability to easily define processing and visualization workflows.

Starting from the idea that it should be easy to discover and instantly bring together any available data sources, processing capabilities and algorithms, the team sought to prototype draft specifications for chaining processing and data delivery services together, and demonstrate their applications in specific domains of interest.

1.3. Recommendations

The following are key recommendations for changes and enhancements to existing OGC API specifications, and for future work exploring the potential of modular workflows.

- The participating organizations recommend that additional steps be taken by both the OGC Standards and Innovation Program to dive deeper into this approach in the form of additional interoperability experiments such as Code Sprints, as well as Testbeds for which Workflows may provide an interesting opportunity to connect the work of different work packages focusing on different OGC API data delivery and/or processing specifications.
- The OGC API - Processes Standard Working Group will continue the development of the Part 3 - Workflows and Chaining extension, for which two participants in this project volunteered as editors of the specifications. The project's recommendation is to continue the standardization effort with an objective to make it an official OGC standard.

1.4. Document contributor contact points

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

Contacts

Name	Organization	Role
Jérôme Jacovella-St-Louis	Ecere	Editor
Patrick Dion	Ecere	Contributor
Diego Caraffini	Ecere	Contributor
Réjean Loyer	Ecere	Contributor
Mir Abolfazl Mostafavi	Université Laval	Contributor
Saeid Doodman	Université Laval	Contributor
Saeid Homayouni	INRS-CETE	Contributor
Mohammad Rezaee	INRS-CETE	Contributor
Stephan Meißl	EOX	Contributor
Bernhard Mallinger	EOX	Contributor

Name	Organization	Role
Even Rouault	Spatialys	Contributor
Peter Baumann	rasdaman	Contributor
Bang Pham Huu	rasdaman	Contributor
Vlad Merticariu	rasdaman	Contributor
Benjamin Pross	52 North	Contributor
Matthes Rieke	52 North	Contributor
Christian Autermann	52 North	Contributor
Matthew B. J. Purss	Pangaea Innovation	Contributor
Joan Masó Pau	UAB-CREAF	Contributor
Núria Julià	UAB-CREAF	Contributor
Panagiotis (Peter) Vretanos	CubeWerx	Contributor
Tom Landry	CRIM	Contributor
Francis Charette-Migneault	CRIM	Contributor

Chapter 2. References

The following normative documents are referenced in this document.

NOTE: Only normative standards are referenced here, e.g. OGC, ISO or other SDO standards. All other references are listed in the bibliography.

- OGC API - Common - Part 1
- OGC API - Common - Part 2
- OGC API - Features - Part 1
- OGC API - Features - Part 2
- OGC API - Features - Part 3
- OGC API - Coverages - Part 1
- OGC API - Tiles
- OGC API - Maps
- OGC API - Styles
- OGC API - Processes - Part 1
- OGC API - Processes - Part 2
- OGC API - Processes - Part 3
- Coverage Implementation Schema
- Sensor Things API
- Topic 21 - Discrete Global Grid Systems - Part 1
- Topic 21 - Discrete Global Grid Systems - Part 4

Chapter 3. Terms and definitions

3.1. Definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

4.x
coordinate reference system
coordinate system that is related to the real world by a datum

(SOURCE: ISO 19111:2019 Geographic information — Referencing by coordinates)

4.x
coverage
feature that acts as a function to return values from its range for any direct position within its spatio-temporal domain

4.x
dataset
A dataset is a collection of data, published or curated by a single agent. Data comes in many forms including numbers, words, pixels, imagery, sound and other multi-media, and potentially other types, any of which might be collected into a dataset.

Note: There is an important distinction between a dataset as an abstract idea and a distribution as a manifestation of the dataset

(SOURCE: W3C Data Catalog Vocabulary (DCAT) - Version 2 [https://www.w3.org/TR/vocab-dcat-2/], 2020)

4.x
Data Store

A data store is a repository for persistently storing and managing collections of data which include not just repositories like databases, but also simpler store types such as simple files, metadata, models, etc.

(SOURCE: https://www.information-management.com/glossary/d.html:2020)

4.x
elevation
Synonym for “height”

(SOURCE: OGC Abstract Topic 6: Schema for coverage geometry and functions [https://portal.opengeospatial.org/files/?artifact_id=19820])

4.x
height
Distance of a point from a chosen reference surface measured upward along a line perpendicular

to that surface. Note 1 to entry: A height below the reference surface will have a negative value, which would embrace both gravity-related heights and ellipsoidal heights.

(SOURCE: ISO 19111:2019 Geographic information — Referencing by Coordinates)

4.x
Metadata
information that captures the characteristics of a resource to represent the ‘who’, ‘what’, ‘when’, ‘where’, ‘why’, and ‘how’ of that resource.

(SOURCE: National System for Geospatial Intelligence (NSG) Metadata Foundation (NMF) Version 3.0 [https://nsgreg.nga.mil/doc/view?i=4252&month=10&day=22&year=2019])

4.x
resource
identifiable asset or means that fulfills a requirement

NOTE	A web resource, or simply resource, is any identifiable thing, whether digital, physical, or abstract.

(SOURCE: ISO:19115-1:2014 Geographic information — Metadata — Part 1: Fundamentals)

- **tile**

  geometric shape with known properties that may or may not be the result of a tiling (tessellation)process. A tile consists of a single connected "piece" without "holes" or "lines" (topological disc).

NOTE	“tile” is NOT a packaged blob of data to download in a chunky streaming optimization scheme!

- **tiling**

  in mathematics, a tiling (tessellation) is a collection of subsets of the space being tiled, i.e. tiles that cover the space without gaps or overlaps.

3.2. Abbreviated terms

CRS	Coordinate Reference System
GPKG	GeoPackage
glTF	GL Transmission Format
LoD	Level of Detail
NRCan	Natural Resources Canada
PBR	Physically-Based Rendering (PBR)
STAC	SpatioTemporal Asset Catalog
TIFF	Tagged Image File Format
TMS	Tile Matrix Set

UML	Unified Modeling Language

Chapter 4. Overview

Section 5 Chaining data and processing services: A discussion of the challenges, solutions and opportunities for improved performance, findability, accessibility, interoperability and re-usability explored in this project.

Section 6 OGC API Workflows extension: A summary of the *OGC API - Processes - Part 3: Workflows* extension specifications initially developed in this project.

Section 7 Earth Observation application use case scenario: classifying crops with a process integrated in Workflows extension, using the Python Scikit-learn Random Forest Machine Learning classification algorithm

Section 8 3D Visualization of Urban Environments, Point Clouds & Accessible Routing: managing arbitrarily large point cloud datasets with multi-resolution pyramids, integrating height information from point clouds into vector features, visualizing extruded 3D buildings from OpenStreetMap data sources and taking road steepness into consideration for calculating more accessible routes.

Section 9 DGGS flooding scenario use case: Integrating multiple data sources using a Discrete Global Grid System to establish flooded zones based on rising water levels, as well as impacted buildings and roads.

Section 10 Implemented Server Components: A description of the different server components developed and enhanced during this project: GNOSIS Map Server, pygeoapi, TerraNexus, rasdaman, MiraMon and javaPS.

Section 11 Implemented Client Components: A description of the different client components developed and enhanced during this project: GNOSIS Cartographer, the GDAL OGCAPI driver & QGIS, MiraMon, and TerraNexus client.

Section 12 Technology Integration Experiments: A list and connectivity matrix of the different Technology Integration Experiments performed during this project.

Section 13 Project contributions to OGC API specifications: Details of the contributions to OGC API specifications made by participants in the context of this project, including *Processes*, *Common*, *Coverages*, *Tiles*, *Maps*, *Styles* and *DGGS*.

Section 14 Project contributions to Free and Open Source Software: Details of the contributions to Free and Open Source Software made by participants in the context of this project, including to GDAL, QGIS, pygeoapi, rasdaman, javaPS and the Ecere cross-platform Software Development Kit.

Section 15 Future Work: Recommended next steps which could be taken, building upon the achievements of this project, in the OGC Innovation and Standards program, by participants, the OGC membership, as well as additional stakeholders who could benefit from leveraging modular workflows.

Chapter 5. Chaining data and processing services

This chapter discusses the challenges, solutions and opportunities for improved performance, findability, accessibility, interoperability and re-usability explored in this project.

5.1. OGC API Family of Standards

The timing of the project was especially fitting, as the Open Geospatial Consortium is currently developing a new 'OGC API' family of standards, with the intent to facilitate implementation by adopting modern web development practices such as resource-oriented service architecture and API description using for instance OpenAPI.

Another important design objective of the OGC API is to encourage re-use of common building blocks, to provide both improved consistency within the family of standards, as well as interoperability when dealing with different types or different aspects of geospatial data.

This translates into the ability to re-use software components in both client and server implementations, making it easier to incrementally support additional building blocks from the OGC API family of standards, and this ultimately benefits end-users with improved Findability, Accessibility, Interoperability and Re-usability (FAIR).

Most of the organizations and individuals participating in this project are OGC members deeply involved in the OGC Standards Program. The initiative provided a unique and timely opportunity for these key contributors to different OGC API Standards Working Groups to collaborate together, review the current status of the relevant draft specifications and provide feedback on a question of critical importance, How can all these standards work together efficiently, and enable possibilities greater than the sum of all parts?

This question was first given some serious attention at the joint ESIP and OGC Coverage Processing and Analytics Sprint held in Bethesda, Maryland in January 2020, in a session focusing on the Convergence of OGC APIs, laying the foundation for this project.

The discussions in this session were also inspired by work done as part of previous Testbeds of the OGC Innovation Program, in particular Ecere’s proposal for a Unified Map Service as part of its participation in the 2017 Testbed 13 Vector Tiles work package, discussions that took place at the Testbed 15 Kickoff in early 2019 regarding a vision of how all the different OGC API building blocks could fit together, and a subsequent proposal more specifically focused on how chained workflows could be achieved with OGC API - Processes at the OGC API Hackathon which took place in London in June 2019.

5.2. Workflows

The project focused on how data can efficiently flow between these API building blocks, regardless of where its source or the processing capabilities are located, whether they are hosted on the same server or on different continents.

The team initiated the development of a new extension for the OGC API - Processes specifications, provisionally named Part 3 - Workflows and Chaining. This extension allows to elegantly define complex workflows by chaining together simpler processes, as each process can accept input either provided by the end-user as fixed parameters, from a local or remote data source, or from the output of another local or remote process.

In many ways, this is reminiscent of the pipelining capabilities supported by UNIX-like operating systems since the 1970s. The specification make it possible for the end-user to define and execute entire workflows without the need for any kind of preliminary setup on any of the data or processing services involved.

This is achieved by the client first registering a workflow by submitting it to the immediate node in the chain, that is the one performing the final steps before the client will receive the results. This immediate node will in turn perform registration and validation with the next node in the chain and so on.

If the workflow validates successfully, the client will then proceed to initiate processing requests, again by asking the immediate node, which will in turn request data from external nodes as required to fulfill its own requests.

Because all of these requests triggering processing reference a pre-registered and validated workflow, which already set up all processing aspects, the Workflows extension can specify the use of the regular data delivery OGC API specifications, such as OGC API - Features, Coverages and Tiles, for both the client retrieving final results as well as for intermediate processing services accessing input data from remote data sources and processes.

This greatly simplifies interoperability and how any OGC API services, for both data delivery and processing, can effortlessly be linked together to build these workflows. Furthermore, because the resulting output data can be accessed exactly like any other geospatial data available through OGC API data delivery services, they can be thought of as "virtual collections", in reference to the notion of a generic collection of geospatial data as defined by the OGC API - Common - Part 2 specifications.

5.3. Tiling, Caching and Performance considerations

Another advantage of leveraging the OGC API data delivery specifications to trigger the processing is that they enable the use of space partitioning mechanisms, such as data cube subsetting, multi-resolution tile pyramids, or discrete global grids, to request and process only the portion of the data relevant to the immediate area and resolution of interest. This makes it possible for the end-user to obtain immediate results for the portion of the data being visualized, even as the user explores different regions of the result set or tweak input parameters.

The use of multi-resolution tiles in particular greatly facilitates both the ability for servers to distribute processing a large quantity of data in parallel over different server instances, CPU or GPU cores, as well as to cache the final and intermediate results along the hops of the workflow. Because tiles separate space in a fixed predetermined manner, clients revisiting the same area of interest with the same inputs will be making the same identical requests, even though the actual area being visualized would not be exactly the same.

As a result of processing in direct response to client requests, and optionally by pre-empting future

requests, services can both ensure the latest available data sources are used while optimizing the allocation of available computing resources. These resources can also be prioritized based on the importance and urgency of the requests, as well as the credentials of the clients making them.

Although this is an alternative to batched processing, these workflow specifications do not preclude the use of batch execution. Batched processing could even be scheduled as background tasks whose results are combined with prioritized requests. In this way, requests would in effect prioritize part of an on-going batch execution as clients have an immediate need for them.

All together, this improved cacheability, distributiveness and agility in resource allocation can result in very significant gains in both performance and responsiveness, offsetting the lag typically associated with distributed workflows.

5.4. Re-usability

Since virtual collections resulting from Workflows can easily be accessed as regular OGC API data delivery services, publishing derived datasets as a workflow can guarantee the results are current with respect to the latest sources, while also making the methodology and sources visible, ensuring both transparency and re-usability.

As much as possible, workflow definitions would not hard-code parameters such the area or resolution of interest, or the specific API or format to use. Instead, those are left for negotiation between the requesting and providing nodes at each hop along the workflow, maximizing both interoperability and re-usability.

Chapter 6. OGC API Processes Workflows and Chaining extension

This chapter summarizes the OGC API - Processes - Part 3: Workflows extension specifications initially developed in this project.

6.1. Part 3 - Workflows and Chaining

In practical terms, Workflows extends the process execution request defined by OGC API - Processes - Core with relatively simple additions. First, a nested Process object is allowed as an additional type of inputs to a process. This Process object is identical to the top-level execution request, making the document itself recursive. In addition to processes local to the service to which a request is being submitted, these processes can refer to remote processes on a different server.

Secondly, another type of input object is introduced representing an OGC API collection. These collection can also refer to either local or remote data sources supporting OGC API data delivery specifications. This provides a more flexible mechanism to request data than the 'href' input type defined in Core, as different requests from the same collection can fill in additional details such as the area of interest, a selection filter or which API or format to use.

Finally, an additional request mode is introduced so that instead of immediately starting a synchronous or asynchronous processing job, the service defers execution to a subsequent OGC API request. Instead, upon submitting the execution request the service only validates the workflow and upon success responds with a document describing the resulting collection.

One more mode is planned whereas the initial response would be a top-level OGC API landing page representing the resulting dataset, which would be fully compliant with OGC API - Features and facilitate returning and selecting from multiple possible results from the processing workflow as separate virtual collections.

6.2. Relationship with Part 2 - Transactions for deployment

A Workflow document could also be used to create a new generic persistent process, using the Part 2 - Transactions for deployment extension to OGC API - Processes. For this purpose, an additional type of input is defined allowing to name parameterized inputs which can be inserted at any level of the workflow.

Any other types of processes deployed using the Part 2 extension can also be referenced from Workflows, for example processes defined using OGC Application Packages, or any other supported payload which may include the use of Common Workflow Language, Jupyter Notebooks, or simply Python source code.

In addition, the Workflows extension provides a simple mechanism to upload algorithms to the data by first submitting code to a general purpose process once as part of workflow registration, then subsequently simply requesting results from resulting virtual collection. Multiple experiments

during this project applied this approach, using Python, the Web Coverage Processing Service (WCPS) as well as generic coverage processing expressions.

Chapter 7. Earth Observation application use case scenario

This chapter explores classifying crops with a process integrated in Workflows extension, using the Python Scikit-learn Random Forest Machine Learning classification algorithm.

7.1. Crops classification

In collaboration with the Centre Eau, Terre, Environnement of the Institut National de la Recherche Scientifique in Quebec City, Ecere prototyped a crop classification process as a scenario to demonstrate the use of the Workflows extension.

7.1.1. Training

A model for the RandomForestClassfier, provided by the Scikit-learn Python library for Machine Learning, was first trained with reference imagery from sentinel-2 satellite data for different seasons, paired with a database of declared agricultural production parcels.

Each type of crops exhibiting a different signature over the multiple frequencies sensed by the sentinel-2 MultiSpectral Instrument for different times of the year, decision trees allow to uniquely classify, to a varying degree of confidence, the type of crops present in a given area.

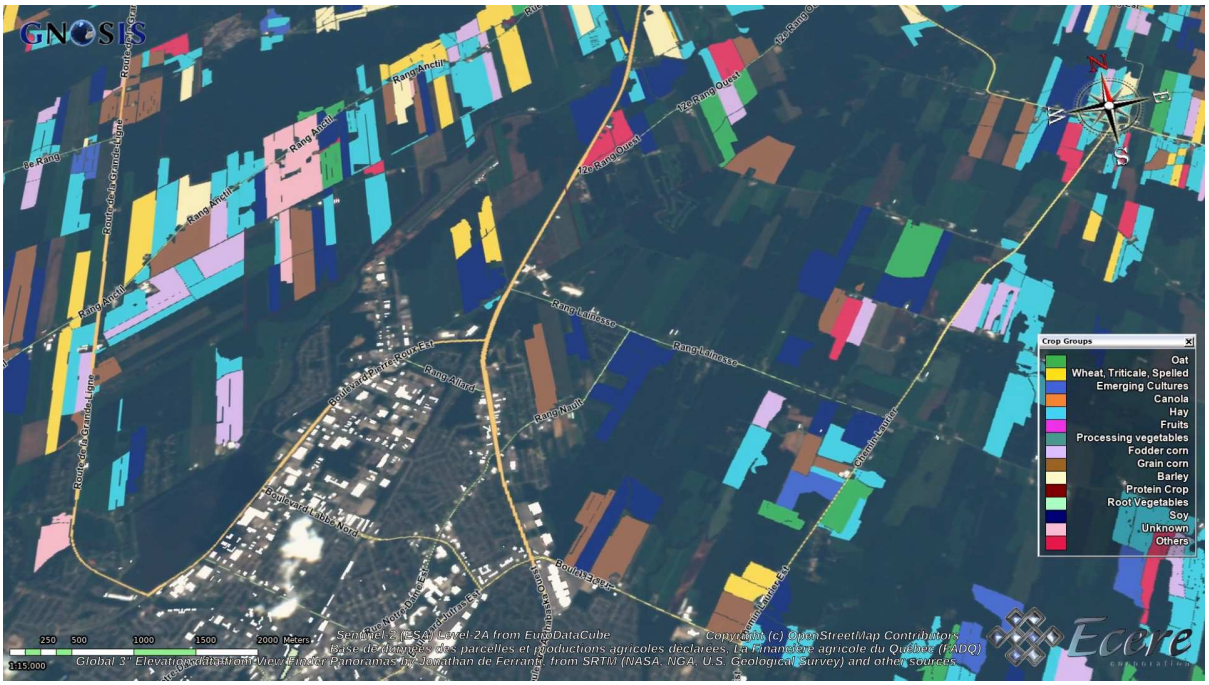

Figure 1. Crop parcels used for training

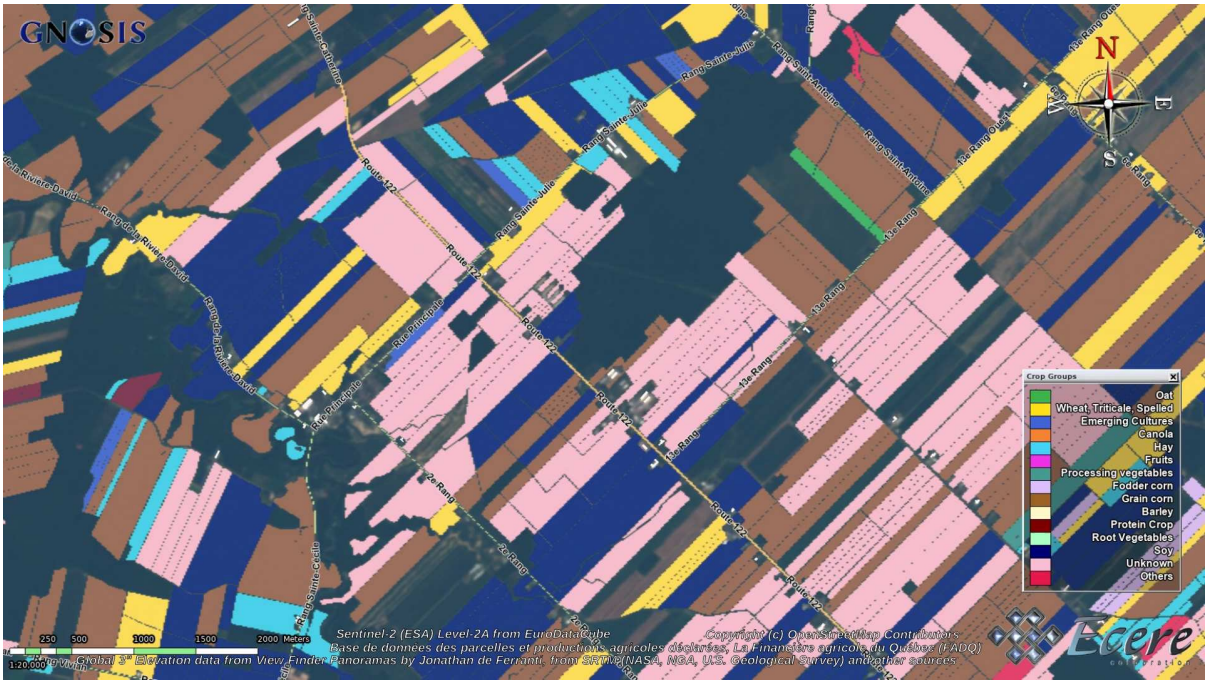

Figure 2. Crop parcels used for training

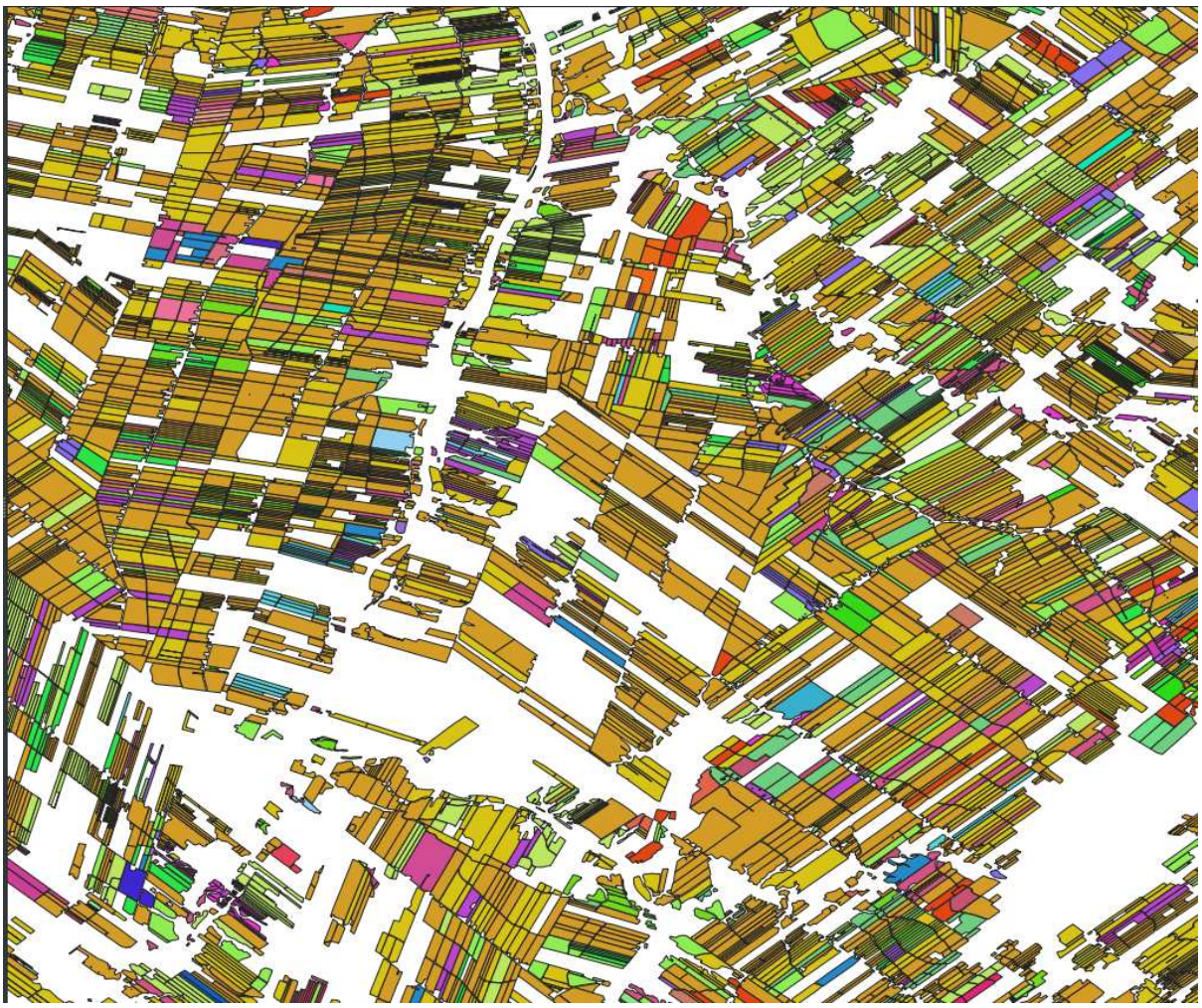

Figure 3. Crop parcels used for training

Figure 4. Spring

Figure 5. Summer

Figure 6. Fall

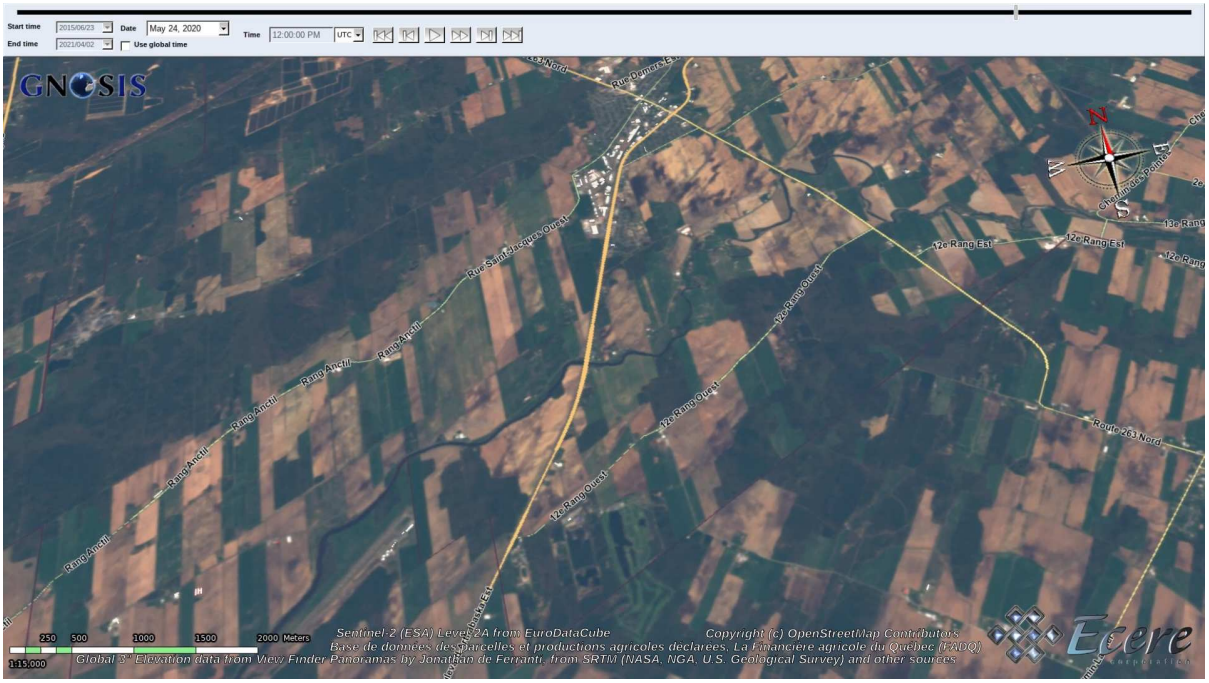

Figure 7. Spring

Figure 8. Summer

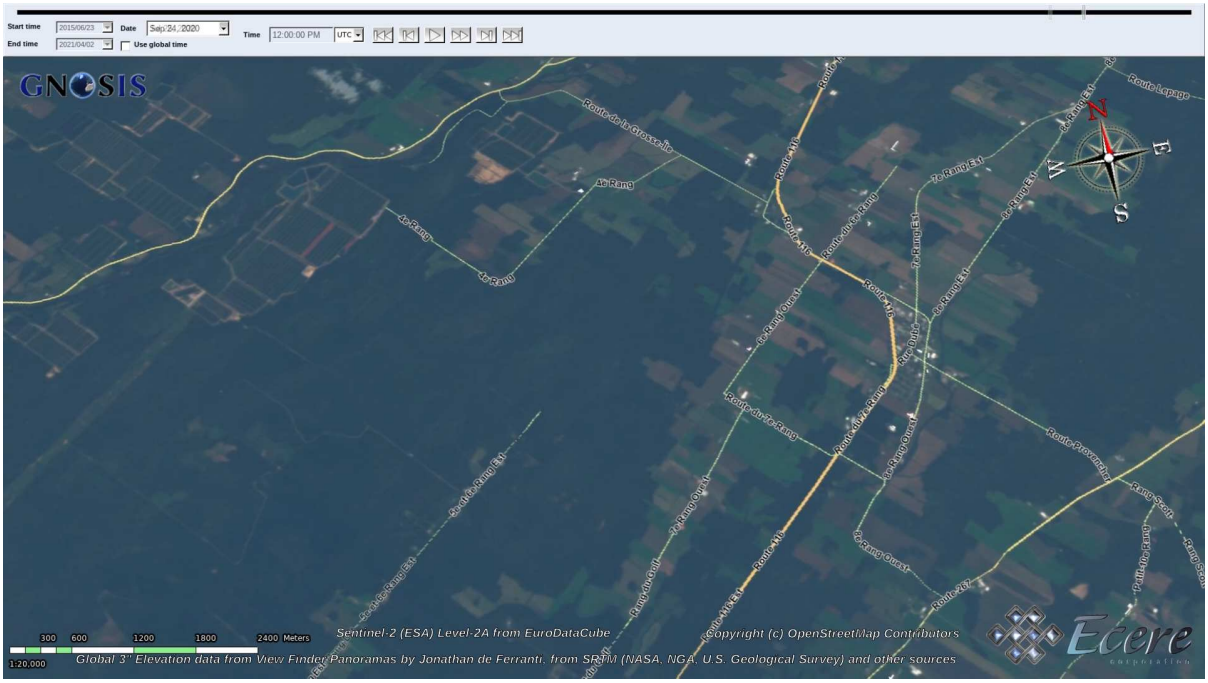

Figure 9. Fall

Figure 10. Enhanced Vegetation Index

In addition to feeding a combination of available bands to the classifier, the training and classification program calculates an Enhanced Vegetation Index utilizing the red, blue and near infra-red bands. The programs also masks out areas identified by a dedicated band as too cloudy.

7.1.2. Prediction

The classification process integrates within the GNOSIS Map Server a Python program making use of the Scikit-learn RandomForestClassifier prediction.

The workflow used to demonstrate this applications refers to an OGC API collection as a source for the satellite imagery made provided by EOX [https://eox.at] from Euro Data Cube (EDC) [https://eurodatacube.com] and implemented using the pygeoapi [https://pygeoapi.io] server software to implement support for OGC API - Coverages.

Visualization clients requesting tiled results of the crop classification process will trigger a workflow registered with the GNOSIS Map Server, which will first request the source imagery for the given tile from the EuroDataCube before executing the Python classification program running the Random Forest classification algorithm, then return the results. Multiple tiles can be requested at once, causing the server to process tiles in parallel.

A further step could chain an additional process to convert the classified result from raster form to polygons, tagged with the crop type. If the intent is to classify parcels where crops are known to be cultivated rather than detecting new crops or areas no longer cultivated, classification could also be done more efficiently on a parcel basis, rather than at an individual pixel level.

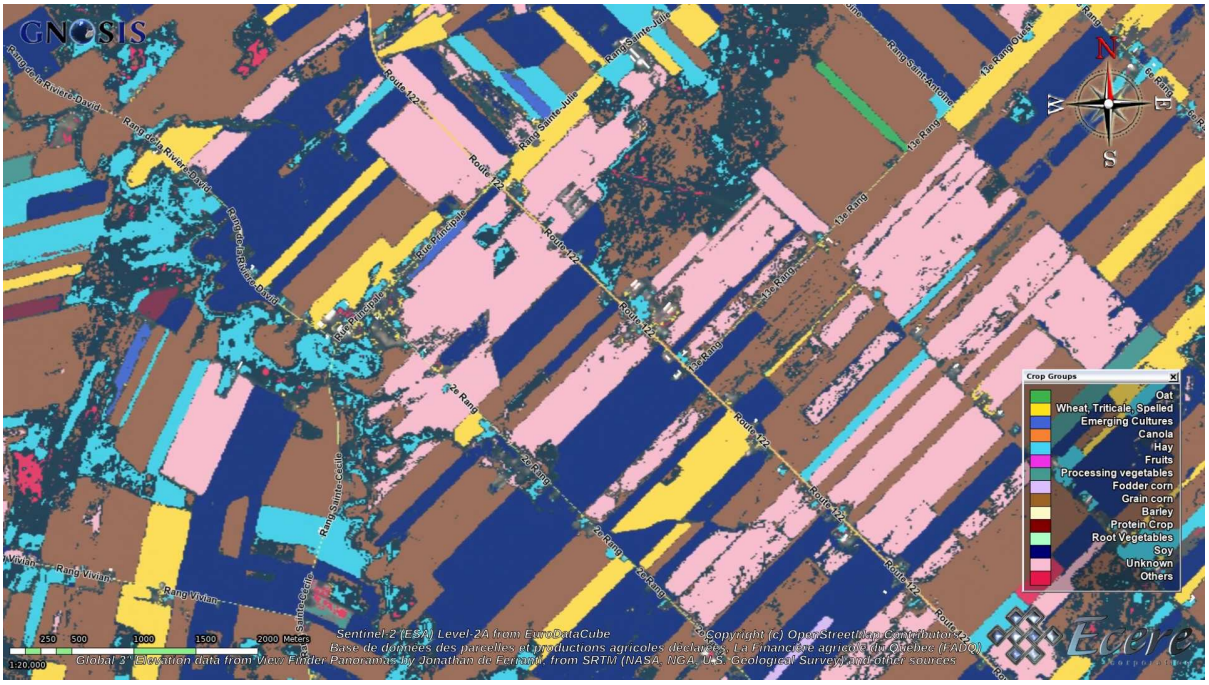

Figure 11. Classification results in GNOSIS Cartographer (full screen)

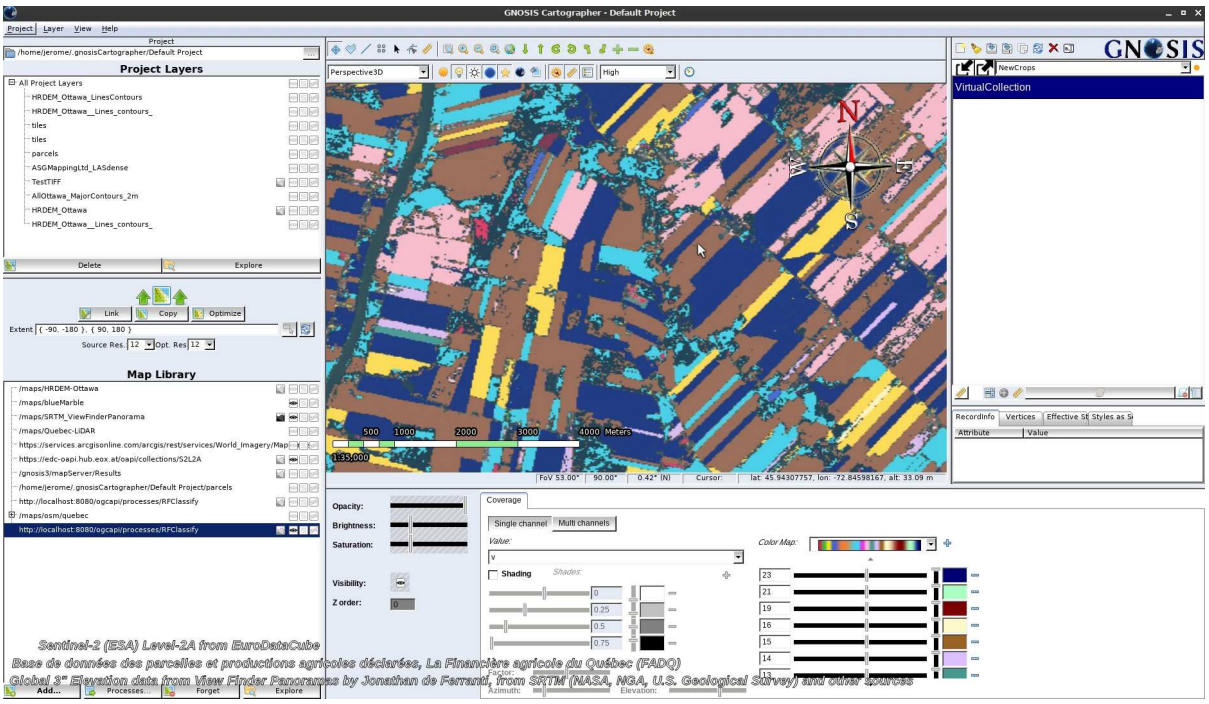

Figure 12. Classification results in GNOSIS Cartographer

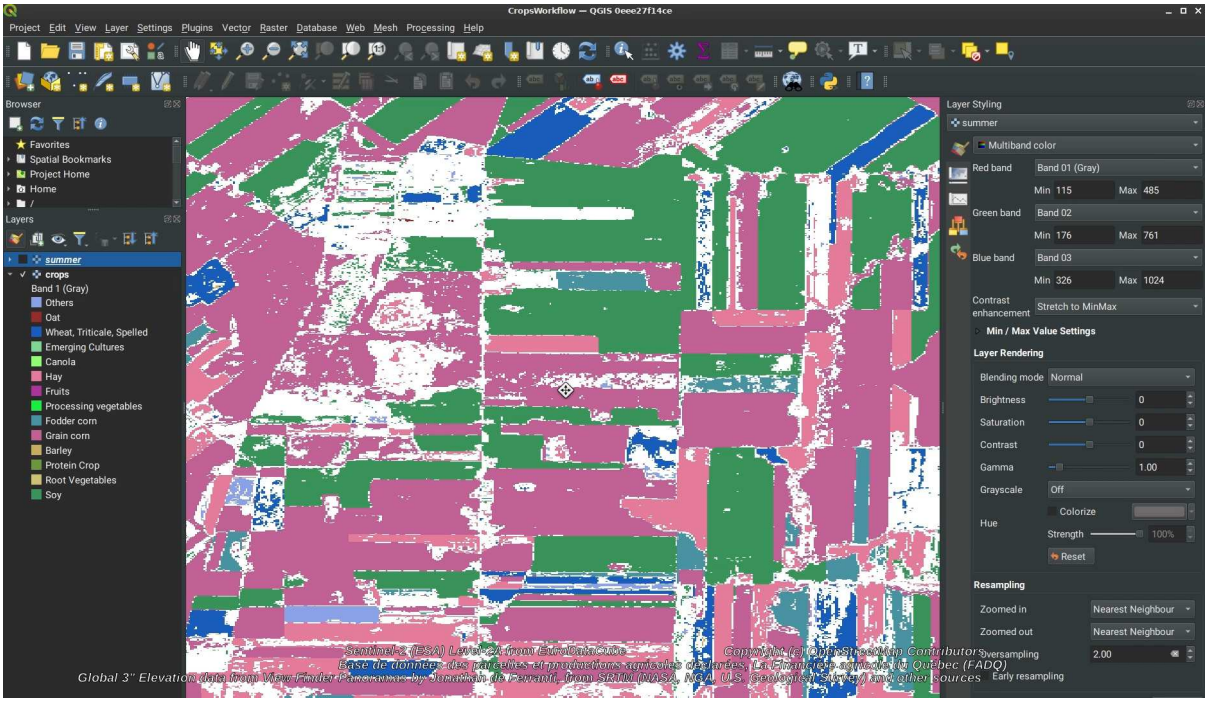

Figure 13. Classification results in QGIS

7.2. An introduction to RF classifier for earth observations

Introduction

Image classification algorithms can be broadly categorized into supervised and unsupervised approaches. Supervised classifiers are preferred when sufficient amounts of training data are available. Parametric and non-parametric methods categorize classification algorithms based on data distribution assumptions. Prior to the introduction of Random Forest (RF), Support Vector Machine (SVM) has been in the spotlight for RS image classification, given its superiority compared to Maximum Likelihood Classifier (MLC), K-Nearest Neighbours (KNN), Artificial Neural Networks (ANN), and Decision Tree (DT) for RS image classification. Since then, it has drawn attention within the remote sensing community, as it produced classification accuracies comparable with those of SVM [1-2].

Theoretical Background

RF is an ensemble learning approach developed by [3] for solving classification and regression problems. Ensemble learning is a machine learning scheme to boost accuracy by integrating multiple models to solve the same problem. In particular, multiple classifiers participate in ensemble classification to obtain more accurate results than a single classifier. In other words, the integration of multiple classifiers decreases variance, especially in unstable classifiers, and may produce more reliable results. Next, a voting scenario is designed to assign a label to unlabeled samples [4], [5], [6]. The commonly used voting approach is majority voting, which assigns the label with the maximum number of votes from various classifiers to each unlabeled sample [7]. The popularity of the majority voting method is due to its simplicity and effectiveness. More advanced voting approaches, such as the veto voting method, wherein one single classifier vetoes other classifiers' choice, can be considered an alternative for the majority voting method [8].

The widely used ensemble learning methods are boosting and bagging. Boosting is a process of building a sequence of models, where each model attempts to correct the error of the previous one in that sequence. AdaBoost was the first successful boosting approach developed for binary classification cases [8]. However, the main problem of AdaBoost is model overfitting [5]. Bootstrap Aggregating, known as Bagging, is another ensemble learning method [9]. Bagging is designed to improve integrated models' stability and accuracy while reducing variance. Bagging is more robust against the overfitting problem than the boosting approach [3].

RF was the first successful bagging approach developed based on the combination of Breiman's bagging sampling approach, random decision forests, and the random selection of features independently introduced by [10-12]. RFs with significantly different tree structures and splitting variables encourage overfitting and outliers among the various ensemble tree models. Therefore, the final prediction voting mitigates the overfitting in the classification problem, while the averaging is the solution for the regression problems. Within the generation of these individual decision trees, each time, the best split in the random sample of predictors is chosen as the split candidates from the full set of predictors. A fresh sample of predictors is taken at each split utilizing a user-specified number of predictors (Mtry). By expanding the random forest up to a user-specified number of trees (Ntree), RF generates high variance and low bias trees. Therefore, new sets of input (unlabeled) data are assessed against all decision trees generated in the ensemble, and each tree votes for a class's membership. The membership with the majority votes will be the one that is eventually selected [13], [14]. This process, hence, should obtain a global optimum [15]. To reach a global optimum, two-third of the samples, on average, are used to train the bagged trees, and the remaining samples, namely the out-of-bag (OOB), are employed to cross-validate the quality of the RF model independently. The OOB error is used to calculate the prediction error and then to evaluate Variable Importance Measures (VIMs) [16], [17].

Of particular RFs' characteristic is variable importance measures (VIMs) [18]–[21]. Specifically, VIM allows a model to evaluate and rank predictor variables in relative significance [22], [23]. VIM calculates the correlation between high dimensional datasets based on internal proximity matrix measurements [24] or identifying outliers in the training samples by exploratory examination of sample proximities using variable importance metrics [25]. The two primary variable importance metrics are: Mean Decrease in Gini (MDG) and Mean Decrease in Accuracy (MDA) [26]–[28]. MDG measures the mean decrease in node impurities resulting from splitting and computes the best split selection. MDG switches one of the random input variables while keeping the rest constant. It then measures the decrease in the accuracy, which has taken place through the OOB error estimation and Gini Index decrease. In a case where all predictors are continuous and mutually uncorrelated, Gini VIM is not supposed to be biased [22]. MDA, however, considers the difference between two different OOB errors, the one that resulted from a data set obtained by randomly permuting the predictor variable of interest and the one that resulted from the original data set [3].

Two parameters have to be set to run the RF model: The number of trees (Ntree) and the number of randomly selected features (Mtry). RFs are reported to be less sensitive to the Ntree compared to Mtry [29]. Reducing Mtry parameter may result in faster computation, but reduces both the correlation between any two trees and every tree's strength in the forest and, thus, has a complex in□uence on the classification accuracy [13]. Since the RF classifier is computationally efficient and does not overfit, Ntree can be as large as possible [30]. Several studies found 500 as an optimum number for the Ntree because the accuracy was not improved by using Ntrees higher than this number [15]. Another reason for this value being commonly used is that 500 is the default value in

the software packages like R package, “randomForest” [31]. In contrast, the number of Mtry is an optimal value and depends on the data at hand. The Mtry parameter is recommended to be set to the square root of the number of input features in classification tasks and one-third of the number of input features for regression tasks [3]. Although methods based on bagging such as RF, unlike other methods based on boosting, are not sensitive to noise or overtraining [32], [33], the above-stated value for Mtry might be too small in the presence of a large number of noisy predictors, i.e., in the case of non-informative predictor variables, the small Mtry results in building inaccurate trees [22].

The RF classifier has become popular for classification, prediction, studying variable importance, variable selection, and outlier detection since its emergence in 2001 by [3]. They have been widely adopted and applied as a standard classifier to a variety of prediction and classification tasks, such as those in bioinformatics [34], computer vision [35], and remote sensing land cover classification [36]. RF has gained its popularity in land cover classification due to its straightforward and understandable decision-making process and excellent classification results [15], and easy implementation of RF in a parallel structure for geo-big data computing acceleration [37]. Other advantages of RFclassifier can be summarized as (1) handling thousands of input variables without variable deletion, (2) reducing the variance without increasing the bias of the predictions, (3) computing proximities between pairs of cases that can be used in locating outliers, (4) being robust to outliers and noise, (5) being computationally lighter compared to other tree ensemble methods, e.g., Boosting. As such, many research works have illustrated the great capability of the RS classifier for classification of Landsat archive, hyper and multispectral image classification [38], and Digital Elevation Model (DEM) data [39]. Most RF research has demonstrated that RF has improved accuracy compared to other supervised learning methods and provides VIMs crucial for multi-source studies, where data dimensionality is considerably high [40].

References

[1] M. Belgiu and L. Drăguţ, “Random forest in remote sensing: A review of applications and future directions,” ISPRS J. Photogramm. Remote Sens., vol. 114, pp. 24–31, 2016.

[2] P. Mather and B. Tso, Classification methods for remotely sensed data. CRC press, 2016.

[3] L. Breiman, “ST4_Method_Random_Forest,” Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1017/CBO9781107415324.004.

[4] K. Fawagreh, M. M. Gaber, and E. Elyan, “Random forests: from early developments to recent advancements,” Syst. Sci. Control Eng. Open Access J., vol. 2, no. 1, pp. 602–609, 2014.

[5] S. Banks et al., “Wetland Classification with Multi-Angle/Temporal SAR Using Random Forests,” Remote Sens., vol. 11, no. 6, p. 670, Mar. 2019, doi: 10.3390/rs11060670.

[6] Z. Bassa, U. Bob, Z. Szantoi, and R. Ismail, “Land cover and land use mapping of the iSimangaliso Wetland Park, South Africa: comparison of oblique and orthogonal random forest algorithms,” J. Appl. Remote Sens., vol. 10, no. 1, p. 015017, Mar. 2016, doi: 10.1117/1.JRS.10.015017.

[7] L. Lam and S. Y. Suen, “Application of majority voting to pattern recognition: an analysis of its behavior and performance,” IEEE Trans. Syst. Man Cybern.-Part Syst. Hum., vol. 27, no. 5, pp. 553–568, 1997.

[8] R. K. Shahzad and N. Lavesson, “Comparative analysis of voting schemes for ensemble-based malware detection,” J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl., vol. 4, no. 1, pp. 98–117, 2013.

[9] L. Breiman, “Bagging predictors,” Mach. Learn., vol. 24, no. 2, pp. 123–140, 1996.

[10] T. K. Ho, “Random decision forests,” in Proceedings of 3rd international conference on document analysis and recognition, 1995, vol. 1, pp. 278–282.

[11] T. K. Ho, “The random subspace method for constructing decision forests,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 20, no. 8, pp. 832–844, 1998.

[12] Y. Amit and D. Geman, “Shape quantization and recognition with randomized trees,” Neural Comput., vol. 9, no. 7, pp. 1545–1588, 1997.

[13] J. M. Klusowski, “Complete Analysis of a Random Forest Model,” vol. 13, pp. 1063–1095, 2018.

[14] L. Breiman, “Random forests,” Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[15] M. Belgiu and L. Drăgu, “Random forest in remote sensing: A review of applications and future directions,” ISPRS J. Photogramm. Remote Sens., vol. 114, pp. 24–31, 2016, doi: 10.1016/j.isprsjprs.2016.01.011.

[16] R. Genuer, J. M. Poggi, and C. Tuleau-Malot, “Variable selection using random forests,” Pattern Recognit. Lett., vol. 31, no. 14, pp. 2225–2236, 2010, doi: 10.1016/j.patrec.2010.03.014.

[17] Ö. Akar and O. Güngör, “Integrating multiple texture methods and NDVI to the Random Forest classification algorithm to detect tea and hazelnut plantation areas in northeast Turkey,” Int. J. Remote Sens., vol. 36, no. 2, pp. 442–464, Jan. 2015, doi: 10.1080/01431161.2014.995276.

[18] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, “Variable selection using random forests,” Pattern Recognit. Lett., vol. 31, no. 14, pp. 2225–2236, Oct. 2010, doi: 10.1016/j.patrec.2010.03.014.

[19] H. Ishida, Y. Oishi, K. Morita, K. Moriwaki, and T. Y. Nakajima, “Development of a support vector machine based cloud detection method for MODIS with the adjustability to various conditions,” Remote Sens. Environ., vol. 205, pp. 390–407, Feb. 2018, doi: 10.1016/j.rse.2017.11.003.

[20] E. Izquierdo-Verdiguier, V. Laparra, L. Gomez-Chova, and G. Camps-Valls, “Encoding Invariances in Remote Sensing Image Classification With SVM,” IEEE Geosci. Remote Sens. Lett., vol. 10, no. 5, pp. 981–985, Sep. 2013, doi: 10.1109/LGRS.2012.2227297.

[21] C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis, “Conditional variable importance for random forests,” BMC Bioinformatics, vol. 9, no. 1, p. 307, Dec. 2008, doi: 10.1186/1471-2105-9-307.

[22] A.-L. Boulesteix, S. Janitza, and J. Kruppa, “Overview of Random Forest Methodology and Practical Guidance with Emphasis on Computational Biology and Bioinformatics,” Biol. Unserer Zeit, vol. 22, no. 6, pp. 323–329, 2012, doi: 10.1002/biuz.19920220617.

[23] E. M. Abdel-Rahman, O. Mutanga, E. Adam, and R. Ismail, “Detecting Sirex noctilio grey-attacked and lightning-struck pine trees using airborne hyperspectral data, random forest and

support vector machines classifiers," ISPRS J. Photogramm. Remote Sens., vol. 88, pp. 48–59, Feb. 2014, doi: 10.1016/j.isprsjprs.2013.11.013.

[24] A. Verikas, A. Gelzinis, and M. Bacauskiene, "Mining data with random forests: A survey and results of new tests," Pattern Recognit., vol. 44, no. 2, pp. 330–349, 2011, doi: 10.1016/j.patcog.2010.08.011.

[25] J. M. Corcoran, J. F. Knight, and A. L. Gallant, "Influence of multi-source and multi-temporal remotely sensed and ancillary data on the accuracy of random forest classification of wetlands in northern Minnesota," Remote Sens., vol. 5, no. 7, pp. 3212–3238, 2013, doi: 10.3390/rs5073212.

[26] A. Behnamian, K. Millard, S. N. Banks, L. White, M. Richardson, and J. Pasher, "A Systematic Approach for Variable Selection With Random Forests: Achieving Stable Variable Importance Values," IEEE Geosci. Remote Sens. Lett., vol. 14, no. 11, pp. 1988–1992, Nov. 2017, doi: 10.1109/LGRS.2017.2745049.

[27] X. Shen and L. Cao, "Tree-Species Classification in Subtropical Forests Using Airborne Hyperspectral and LiDAR Data," Remote Sens., vol. 9, no. 11, p. 1180, Nov. 2017, doi: 10.3390/rs9111180.

[28] P. S. Gromski, Y. Xu, E. Correa, D. I. Ellis, M. L. Turner, and R. Goodacre, "A comparative investigation of modern feature selection and classification approaches for the analysis of mass spectrometry data," Anal. Chim. Acta, vol. 829, pp. 1–8, Jun. 2014, doi: 10.1016/j.aca.2014.03.039.

[29] T. Berhane et al., "Decision-Tree, Rule-Based, and Random Forest Classification of High-Resolution Multispectral Imagery for Wetland Mapping and Inventory," Remote Sens., vol. 10, no. 4, p. 580, Apr. 2018, doi: 10.3390/rs10040580.

[30] H. Guan, J. Li, M. Chapman, F. Deng, Z. Ji, and X. Yang, "Integration of orthoimagery and lidar data for object-based urban thematic mapping using random forests," Int. J. Remote Sens., vol. 34, no. 14, pp. 5166–5186, 2013, doi: 10.1080/01431161.2013.788261.

[31] P. Probst, M. N. Wright, and A. L. Boulesteix, "Hyperparameters and tuning strategies for random forest," Wiley Interdiscip. Rev. Data Min. Knowl. Discov., vol. 9, no. 3, pp. 1–15, 2019, doi: 10.1002/widm.1301.

[32] J. C. W. Chan and D. Paelinckx, "Evaluation of Random Forest and Adaboost tree-based ensemble classification and spectral band selection for ecotope mapping using airborne hyperspectral imagery," Remote Sens. Environ., vol. 112, no. 6, pp. 2999–3011, 2008, doi: 10.1016/j.rse.2008.02.011.

[33] M. Pal, "Random forest classifier for remote sensing classification," Int. J. Remote Sens., vol. 26, no. 1, pp. 217–222, 2005, doi: 10.1080/01431160412331269698.

[34] C. Strobl, A. L. Boulesteix, A. Zeileis, and T. Hothorn, "Bias in random forest variable importance measures: Illustrations, sources and a solution," BMC Bioinformatics, vol. 8, 2007, doi: 10.1186/1471-2105-8-25.

[35] C. Lindner, P. A. Bromiley, M. C. Ionita, and T. F. Cootes, "Robust and Accurate Shape Model Matching Using Random Forest Regression-Voting," IEEE Trans. Pattern Anal. Mach. Intell., 2015,

doi: 10.1109/TPAMI.2014.2382106.

[36] M. Mahdianpari et al., “Big Data for a Big Country: The First Generation of Canadian Wetland Inventory Map at a Spatial Resolution of 10-m Using Sentinel-1 and Sentinel-2 Data on the Google Earth Engine Cloud Computing Platform: Mégadonnées pour un grand pays: La première carte d’inventaire des zones humides du Canada à une résolution de 10 m à l’aide des données Sentinel-1 et Sentinel-2 sur la plate-forme informatique en nuage de Google Earth EngineTM,” Can. J. Remote Sens., pp. 1–19, 2020.

[37] M. N. Wright and A. Ziegler, “Ranger: A fast implementation of random forests for high dimensional data in C++ and R,” J. Stat. Softw., vol. 77, no. 1, pp. 1–17, 2017, doi: 10.18637/jss.v077.i01.

[38] J. S. Ham, Y. Chen, M. M. Crawford, and J. Ghosh, “Investigation of the random forest framework for classification of hyperspectral data,” IEEE Trans. Geosci. Remote Sens., vol. 43, no. 3, pp. 492–501, 2005, doi: 10.1109/TGRS.2004.842481.

[39] P. O. Gislason, J. A. Benediktsson, and J. R. Sveinsson, “Random forests for land cover classification,” Pattern Recognit. Lett., vol. 27, no. 4, pp. 294–300, 2006, doi: 10.1016/j.patrec.2005.08.011.

[40] P. O. Gislason, J. A. Benediktsson, and J. R. Sveinsson, “Random forests for land cover classification,” Pattern Recognit. Lett., vol. 27, no. 4, pp. 294–300, 2006, doi: 10.1016/j.patrec.2005.08.011.

Chapter 8. 3D Visualization of Urban Environments, Point Clouds & Accessible Routing

This chapter describes managing arbitrarily large point cloud datasets with multi-resolution pyramids, integrating height information from point clouds into vector features, visualizing extruded 3D buildings from OpenStreetMap data sources and taking road steepness into consideration for calculating more accessible routes.

8.1. Processing and visualizing large points clouds

In collaboration with the Centre de recherche en donnees et intelligence geospatiales of the Université Laval in Québec City, Ecere developed capabilities for visualizing, processing and streaming very large point clouds datasets in its GNOSIS Map Server and GNOSIS Cartographer visualization tool.

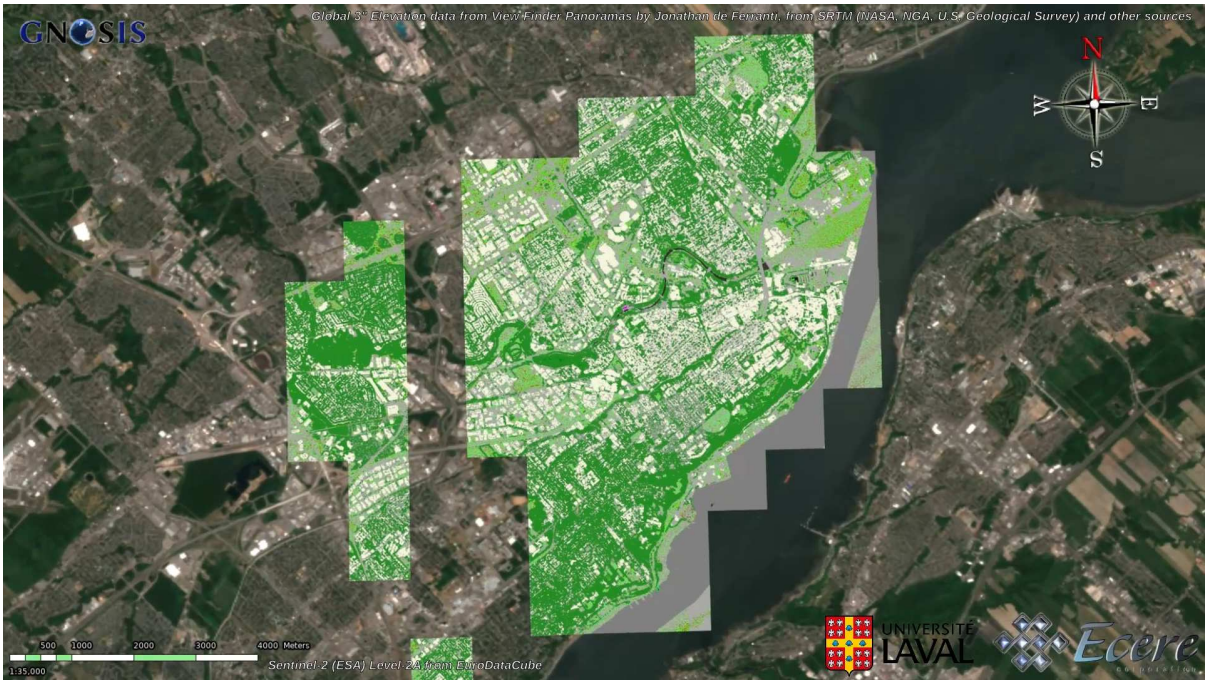

Figure 14. Visualization of classified multi-resolution point cloud (~1 billion points) provided by Université Laval

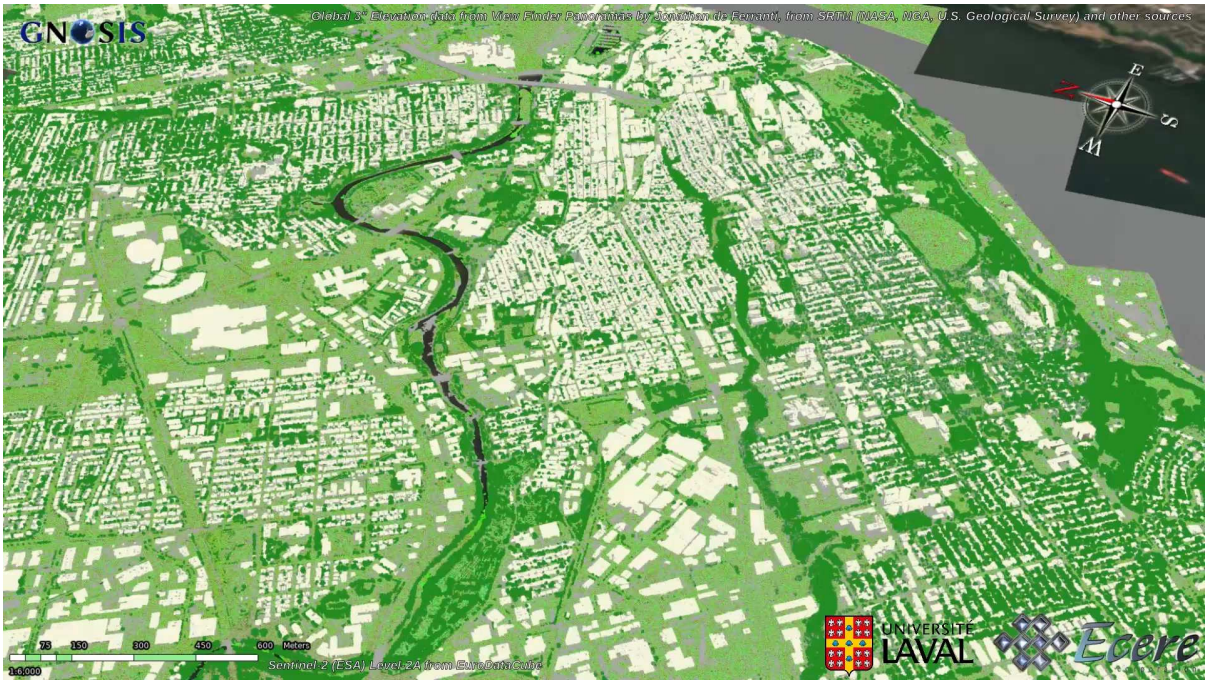

Figure 15. Visualization of classified multi-resolution point cloud (~1 billion points) provided by Université Laval

Figure 16. Visualization of classified multi-resolution point cloud (~1 billion points) from LiDAR provided by Université Laval

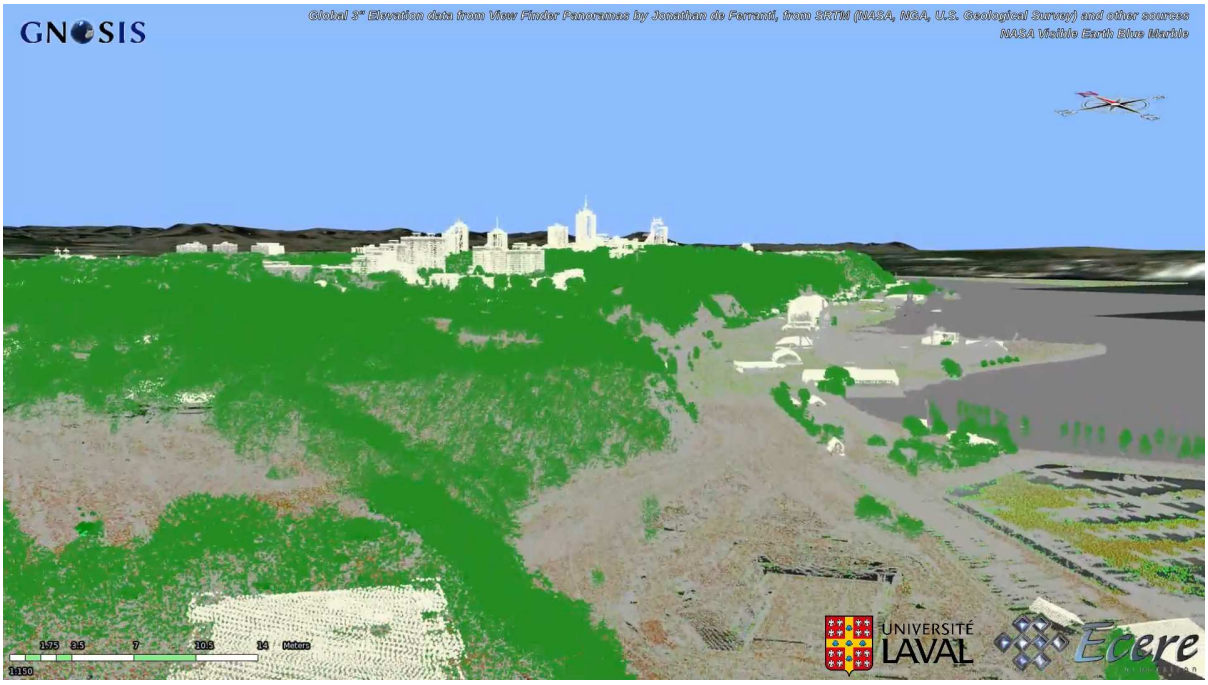

Figure 17. Visualization of classified multi-resolution point cloud (~1 billion points) from LiDAR provided by Université Laval

Figure 18. Visualization of multi-resolution point cloud from photogrammetry provided by ASG Mapping, including RGB values

8.2. Height information

The team also developed a process to extract height information from point clouds to be integrated as part of vector features such as building footprints, allowing to extrude them as 3D buildings of the proper heights. The process, integrated within the GNOSIS Map Server, enabled the visualization of an OpenStreetMap dataset of the whole Province of Québec, integrating elevation information from a point cloud of Québec City provided by the Université Laval.

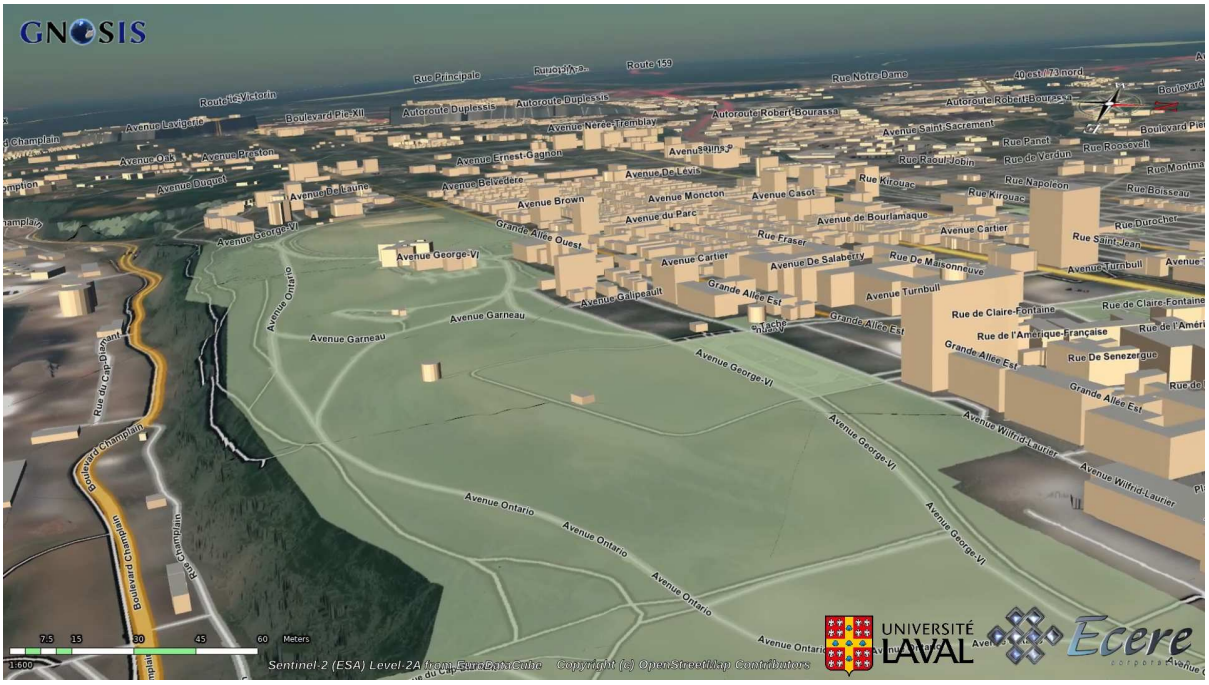

Figure 19. Building elevation inferred from point cloud integrated with OpenStreetMap building footprints

Figure 20. OpenStreetMap dataset for the whole province of Québec

8.3. Gridification

Ecere developed an additional process allowing to gridify point clouds, to turn a point cloud into a 2-dimensional Digital Elevation Model or orthorectified imagery. Featuring an option to select specific classes of a classified point cloud, the process which can be used as part of workflows, provides the flexibility to generate a Digital Terrain Model or Digital Surface model, or also generate building footprints.

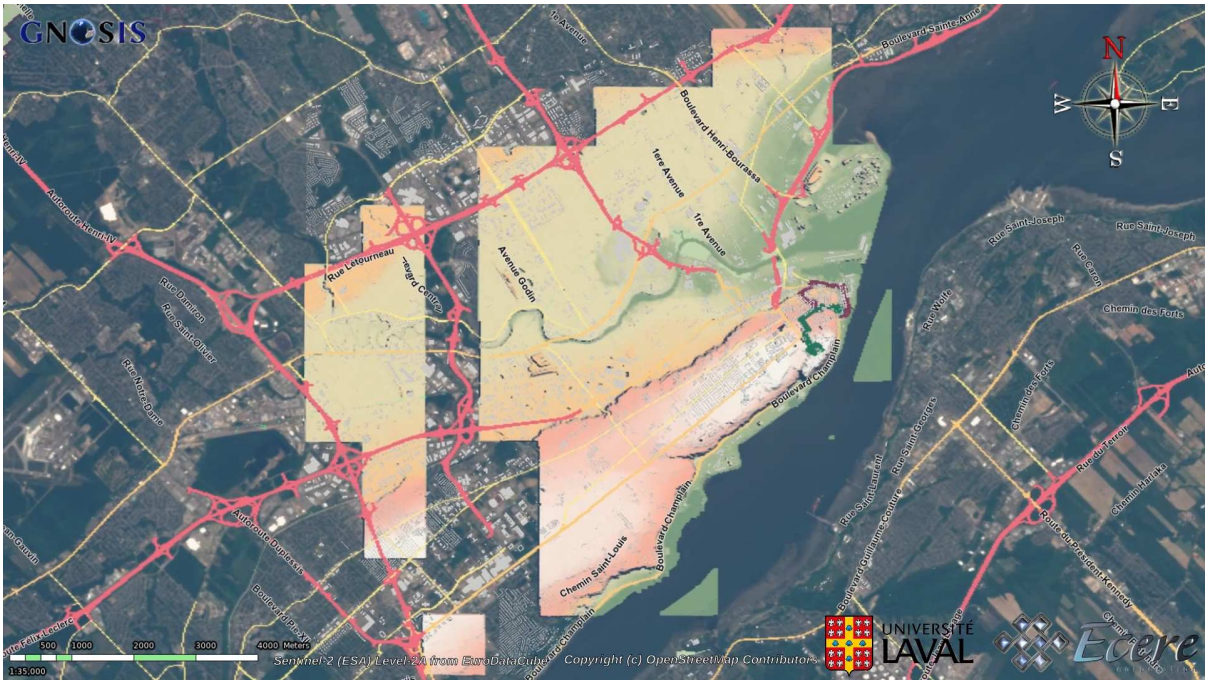

Figure 21. Digital Terrain Model (DTM) generated on-the-fly from point cloud

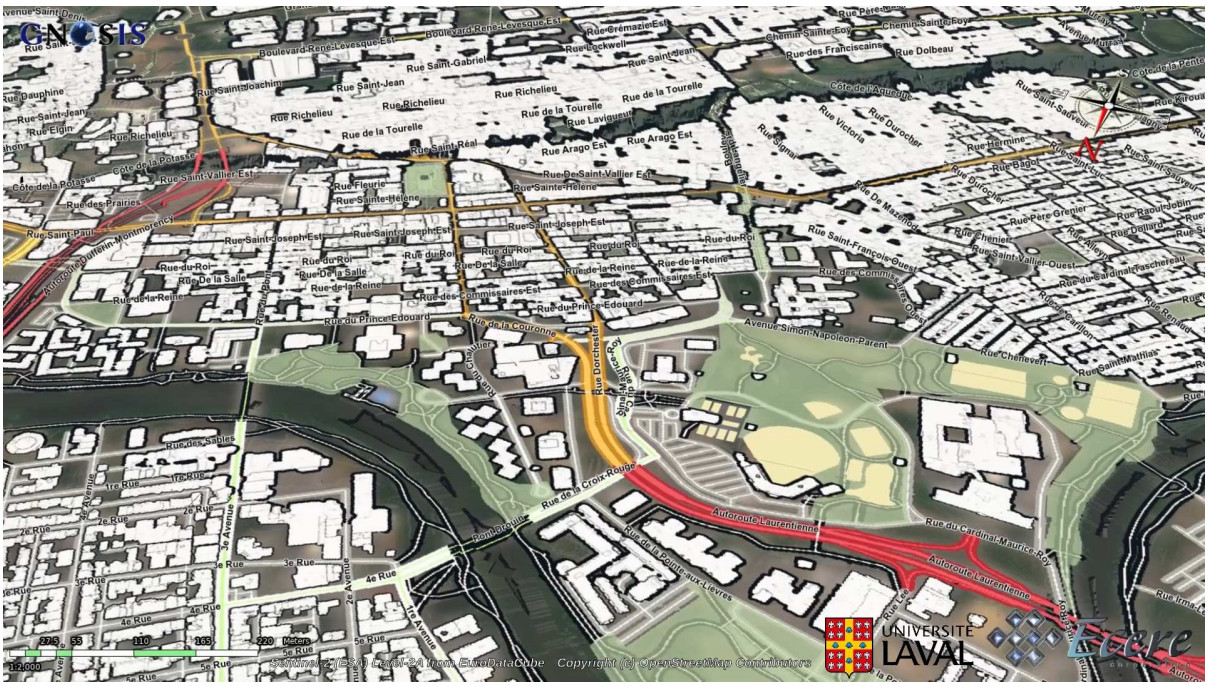

Figure 22. Building footprints generated on-the-fly from point cloud

8.4. Accessible routing

Finally, a last aspect of this scenario focused on the use of the elevation information extracted from point clouds to inform the calculation of more accessible routes for pedestrians, by considering the steepness on the route and integrating this information with a roads network built from the OpenStreetMap dataset. Ecere enhanced its OSMERE routing engine with this new capability.

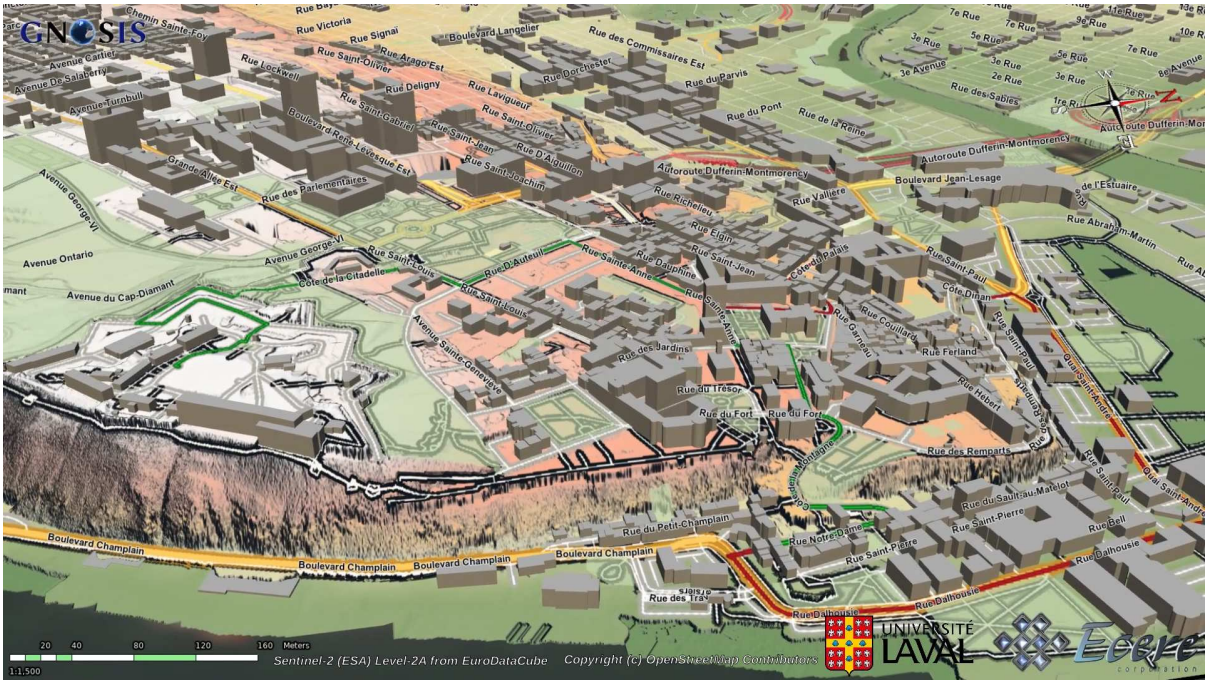

Figure 23. Shorter and less steep routes calculated based on DTM generated from point cloud

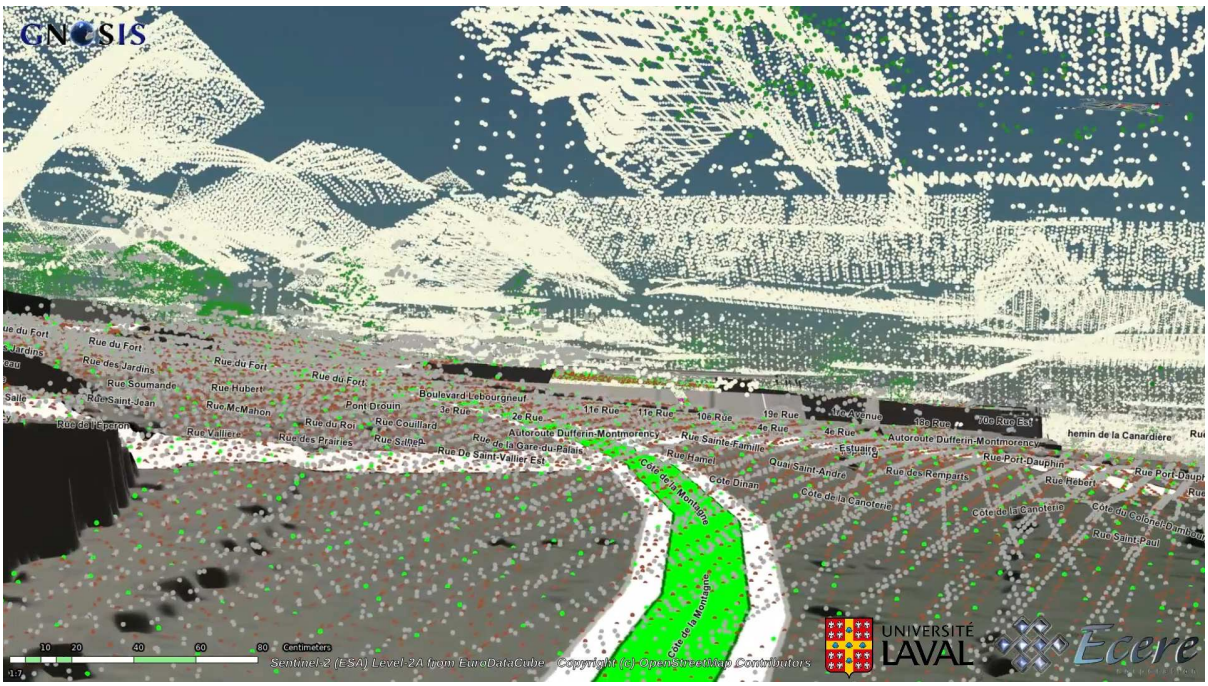

Figure 24. Shorter route

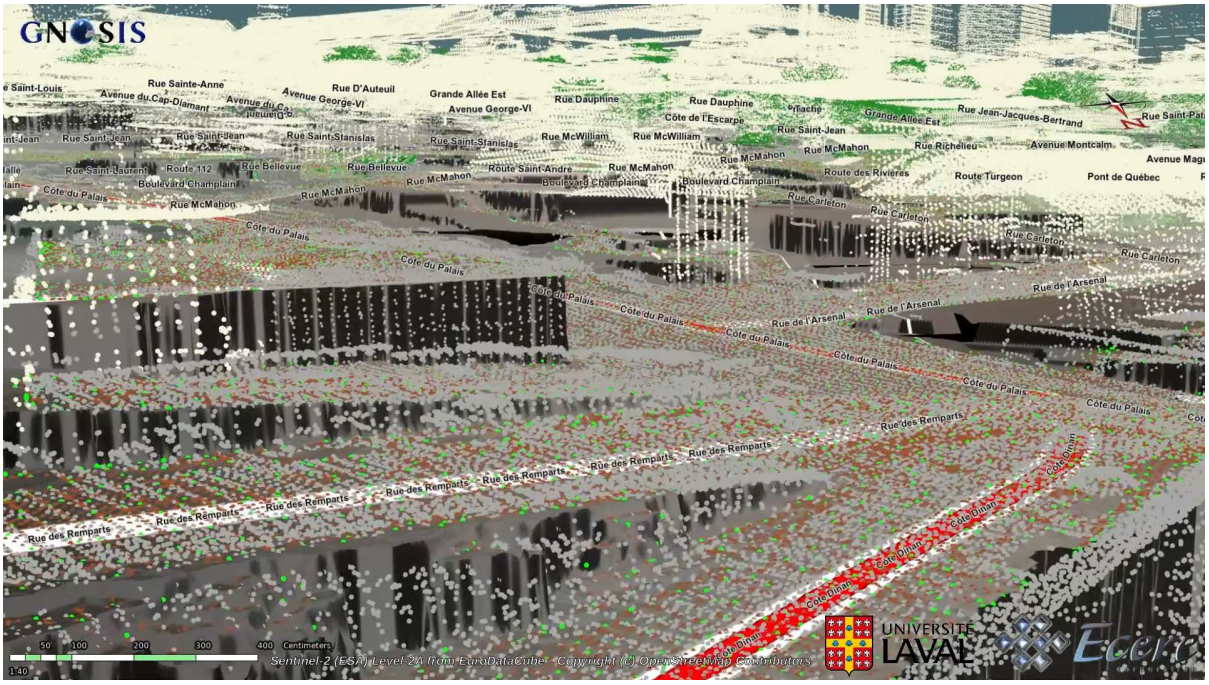

Figure 25. Route avoiding steeper hills

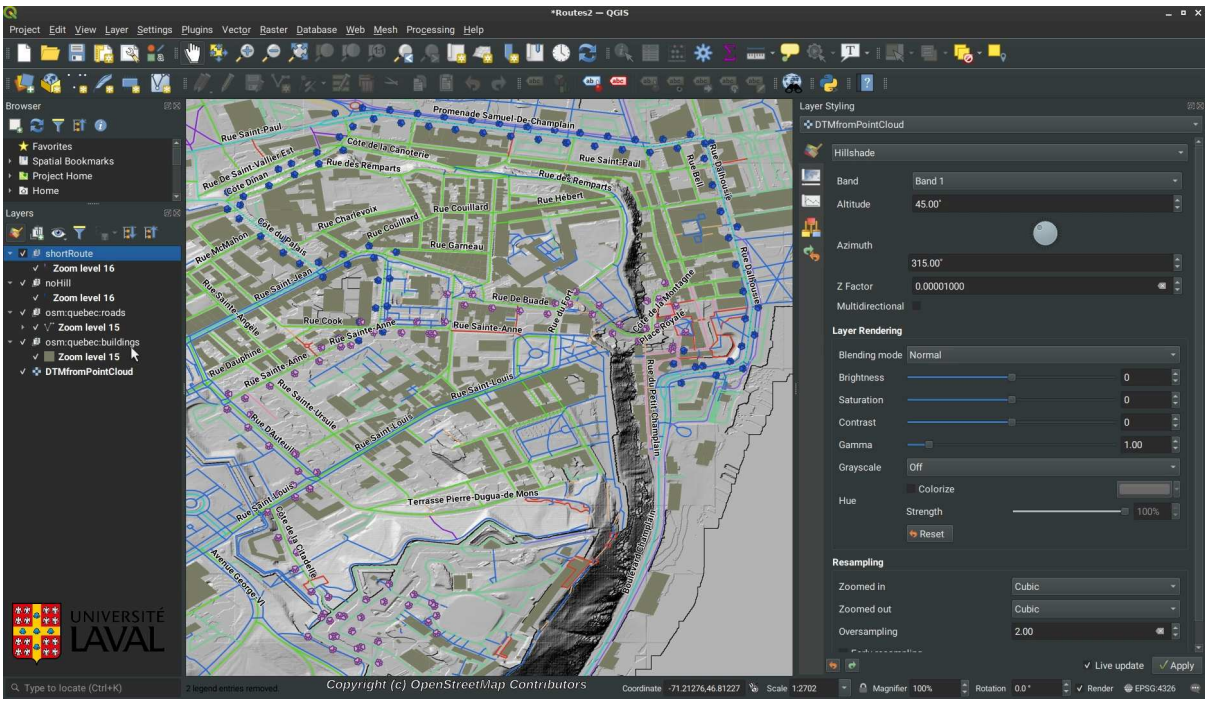

Figure 26. Shorter and less steep routes calculated based on DTM generated from point cloud visualized in QGIS

8.5. Point cloud processing workflows

Very large point clouds provide a good example application of the workflows extension, as various operations are often done in stages to generate them, post-process them, and potentially converting them to other types of data such as elevation models, orthorectified imagery, 3D terrain meshes, or detailed 3D models.

Chapter 9. DGGS flooding scenario use case

This chapter describes integrating multiple data sources using a Discrete Global Grid System to establish flooded zones based on rising water levels, as well as impacted buildings and roads.

9.1. DGGS for data integration

Pangaea Innovations prototyped an implementation of the Workflows extensions, developing capabilities focusing on the integration of multiple vector datasets using its TerraNexus implementation of a discrete global grid system, or DGGS. A DGGS tesselates the world in hierarchical cells of equal area at each level of the hierarchy, which corresponds to a particular resolution. This greatly facilitates data integration and simplifies analytics.

To demonstrate the DGGS capability for data integration via the execution of processing workflows, as well as to prototype a new OGC API for DGGS, the team developed a flood scenario focused on the Ottawa river.

The workflow combines elevation datasets of the area, a contour generation process provided by Ecere's GNOSIS Map Server, together with a polygon intersection process powered by the TerraNexus DGGS, and makes use of an OpenStreetMap dataset of Gatineau-Ottawa providing building footprints and a road network to determine which buildings and roads would be inundated based on the Ottawa river's water level raising by a specified height.

The workflow considers two classes of inputs:

1) Source inputs: In a flood scenario, these represent the datasets containing features that will be filtered to find those that intersect with a flood footprint.

2) Intersecting inputs: In a flood scenario, these represent the datasets containing features that will define the flood footprint.

For this scenario, building footprints and road networks (sourced from the Ecere's instance of Open Street Map datasets

roads [https://maps.ecere.com/ogcapi/collections/osm:ottawa:roads] and buildings [https://maps.ecere.com/ogcapi/collections/osm:ottawa:buildings]) were included as "Source" inputs and topographic contours (sourced from the Ecere OGC API process https://maps.ecere.com/ogcapi/processes/ElevationContours and an Ecere instance of the Ottawa city topographic contours dataset https://maps.ecere.com/ogcapi/collections/AllOttawa_MajorContours_2m) were included as the "Intersecting" inputs.

TerraNexus performed the DGGS enabled data integration of these datasets to identify buildings and roads that are affected by a given flood footprint defined by topographic elevation contours.

Chapter 10. Implemented Server Components

This chapter describes the different server components developed and enhanced during this project: GNOSIS Map Server, pygeoapi, TerraNexus, rasdaman, MiraMon and javaPS.

10.1. GNOSIS Map Server

Ecere developed support for the Workflows extension in its GNOSIS Map Server, and successfully tested execution of various processes within both its GNOSIS Cartographer client as well as within QGIS using the new OGC API driver. The newly developed processing capabilities supporting workflows included its OpenStreetMap routing engine, rendering raster and vector layers as maps, generation of elevation contours, extracting elevation from point clouds, turning point clouds into digital elevation models and ortho photos, crop classification as well as the adapters for WCPS and Processes - Core.

A list of demo processes can be found at this endpoint: https://maps.ecere.com/ogcapi/processes/. These include ElevationContours, MOAWAdapter, OSMERE, PointCloudElevation, and RenderMap processes. Workflows can be freedly edited then executed in the browser by pressing the "set up execution" button. Generated executions become accessible at the https://maps.ecere.com/ogcapi/scratch/ endpoint with a unique ID.

Support for Tile, Coverage, Map, Map Tile, Coverage Tile APIs for serving data has been implemented and can be accessed at the https://maps.ecere.com/ogcapi/ landing page. Data collections can be retrieved at https://maps.ecere.com/ogcapi/collections. Styles for a given collection can be accessed at http://maps.ecere.com/ogcapi/collections/{collectionID}/styles.

The server supports various parameters with which to filter requests including format, time, bbox, width and height. The general formats for requests are as follows.

Vector tiles: http://www.maps.ecere.com/ogcapi/collections/{collectionID}/tiles/{tileMatrixSet}/{tileMatrix}/{tileRow}/{tileCol}.{format}

Map tiles: http://www.maps.ecere.com/ogcapi/collections/{collectionID}/map/tiles/{tileMatrixSet}/{tileMatrix}/{tileRow}/{tileCol}

Coverage tiles: http://www.maps.ecere.com/ogcapi/collections/{collectionID}/coverage/tiles/{tileMatrixSet}/{tileMatrix}/{tileRow}/{tileCol}?f={format}

Maps: http://www.maps.ecere.com/ogcapi/collections/{collectionID}/map?f={format}

Coverages: http://maps.ecere.com/ogcapi/collections/{coverageID}/coverage.{format}

Example requests are as follows:

```
https://maps.ecere.com/ogcapi/collections/blueMarble/map?f=json&crs=CRS84&bbox=-90.00,-90.00,0.00,0.00&width=256&height=256&f=png -- This requests the blueMarble collection as a map with specifications for the coordinate reference system as CRS84, the bounding box extents, width and height of tiles in pixels, and map format as PNG.
```

```
https://maps.ecere.com/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/WebMercatorQuad/0/0/0?f=png -- This requests coverage tiles for the SRTM collection using the WebMercator tiling scheme, with the specified tile matrix, tile row and tile column values.
```

```
https://maps.ecere.com/ogcapi/collections/blueMarble/coverage.png?subset=time(%222004-01-01%22) -- this requests the blueMarble collection as a coverage, in png format, at a specified slice of time being 2004-01-01.
```

```
https://maps.ecere.com/ogcapi/collections/NaturalEarth:cultural/styles/continents/tiles/GNOSISGlobalGrid/0/1/2.mvt -- This requests vector tiles in mvt format for the cultural dataset with the continents style applied, for the specified tile matrix, row and column values.
```

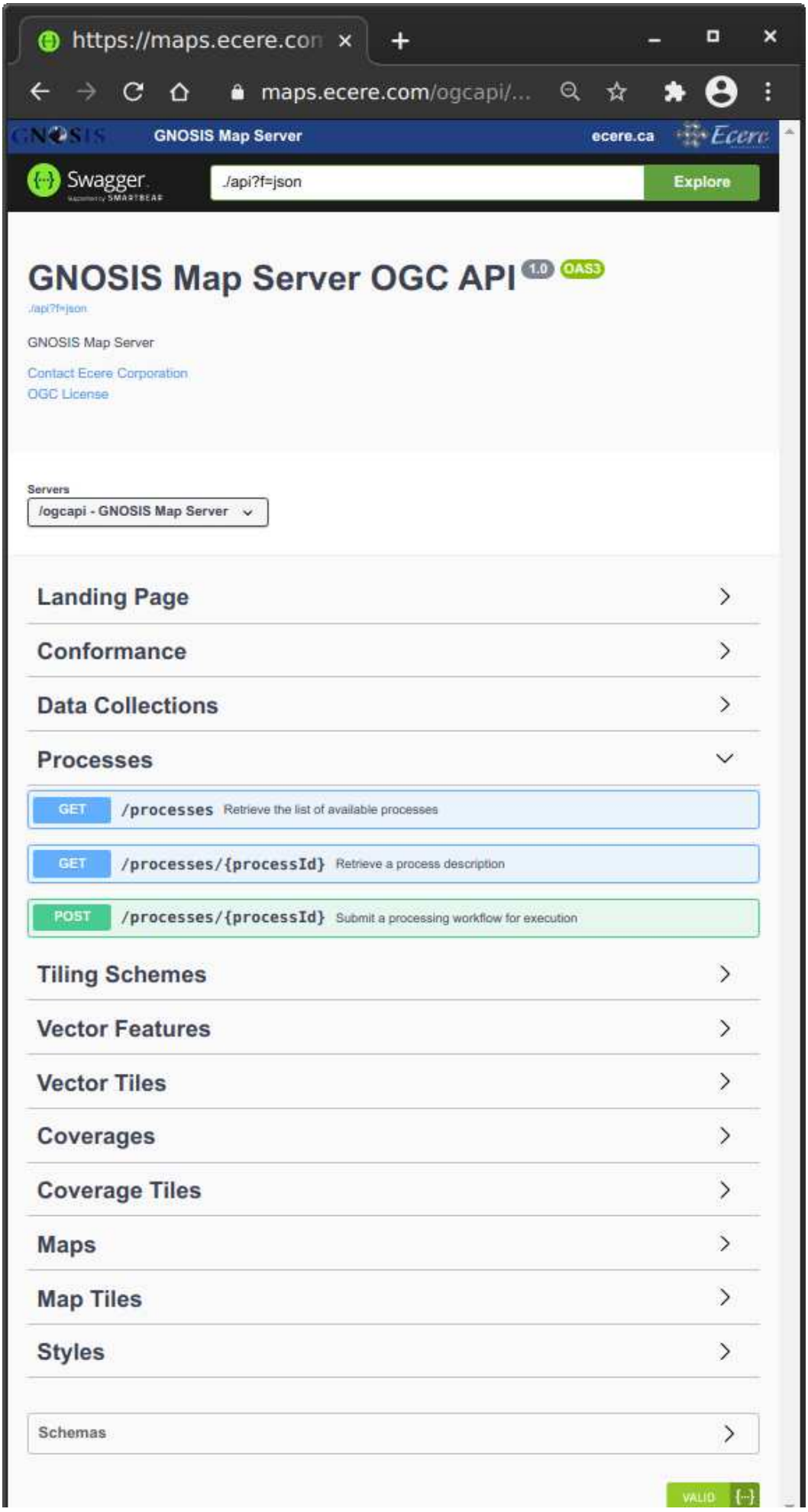

Figure 27. Documentation for the API, defined using OpenAPI and presented using SwaggerUI

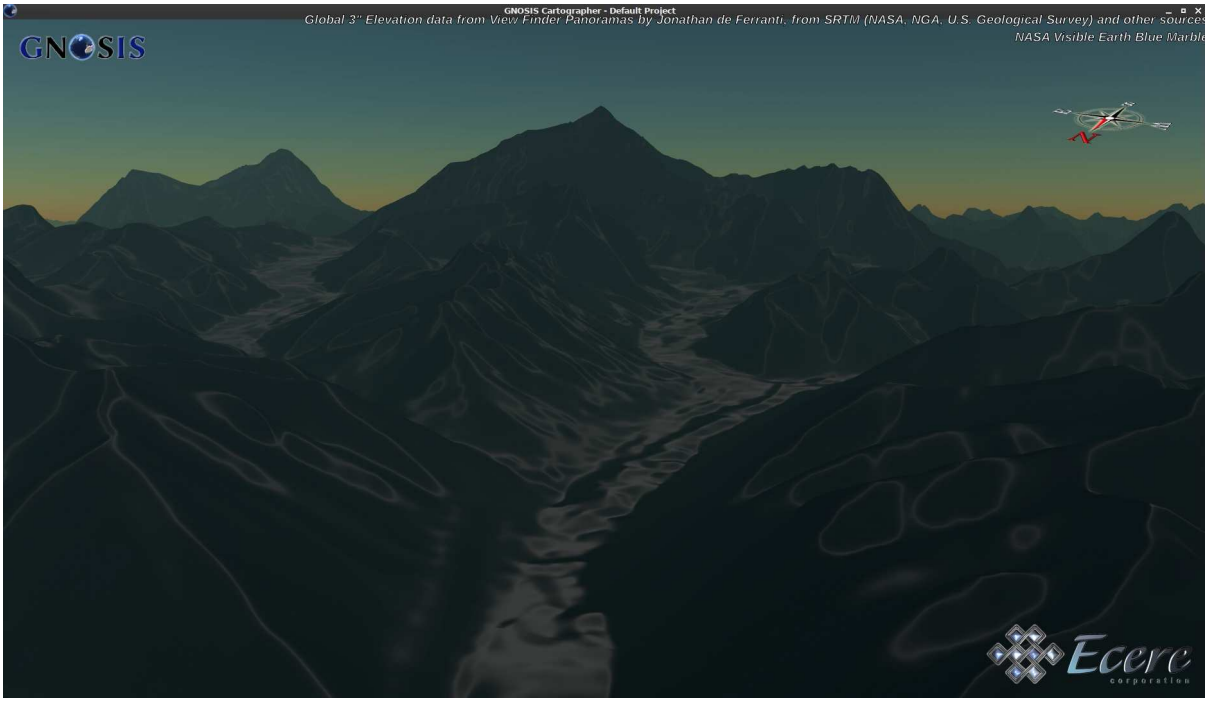

Figure 28. Global 3" elevation from ViewFinderPanorama served using OGC API - Coverages (Mount Everest)

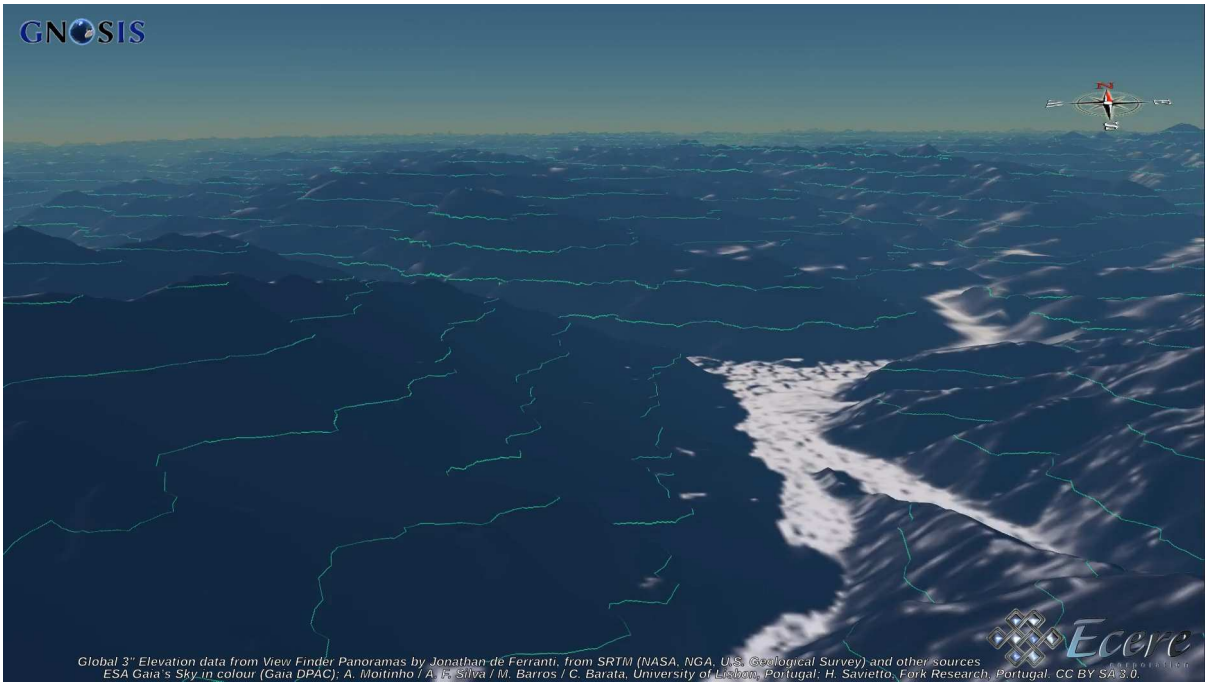

Figure 29. Elevation contours generated on-the-fly from elevation model via tile-based processing

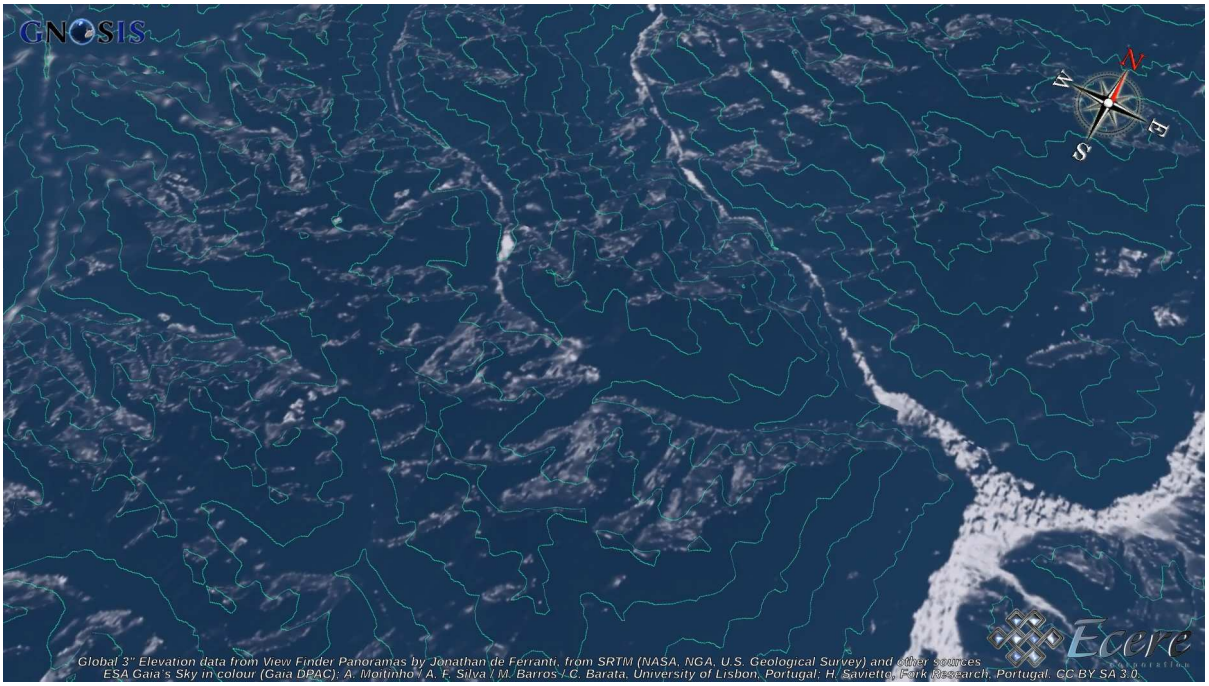

Figure 30. Elevation contours generated on-the-fly from elevation model via tile-based processing

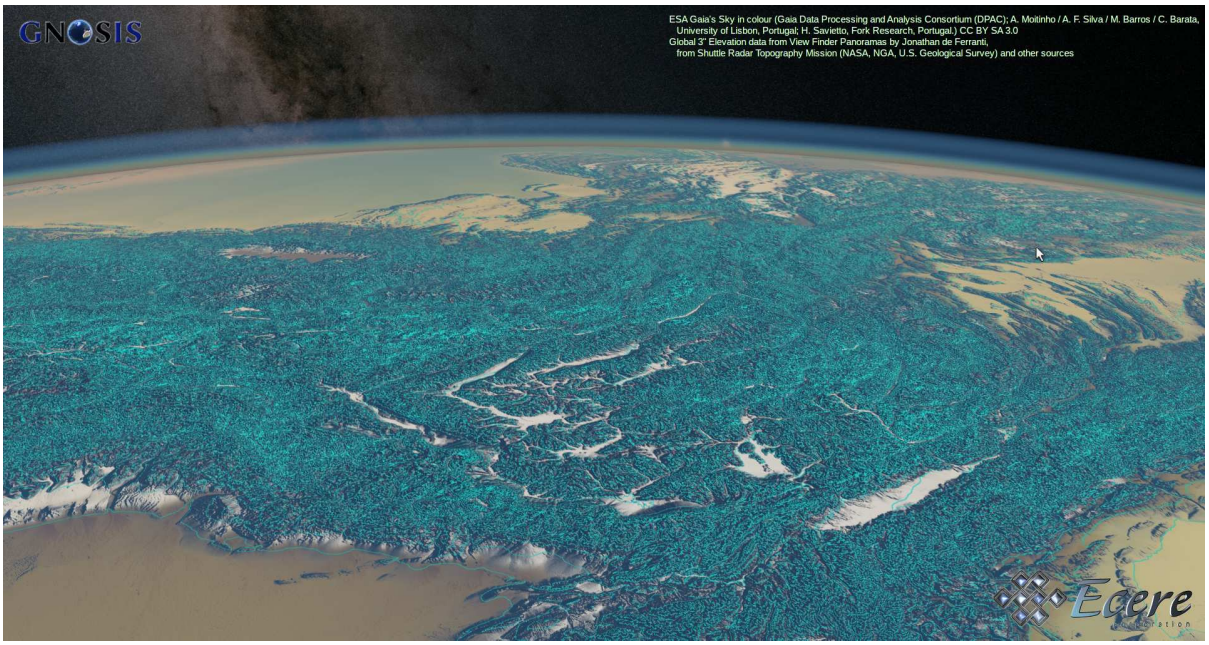

Figure 31. Elevation contours generated on-the-fly from elevation model via tile-based processing

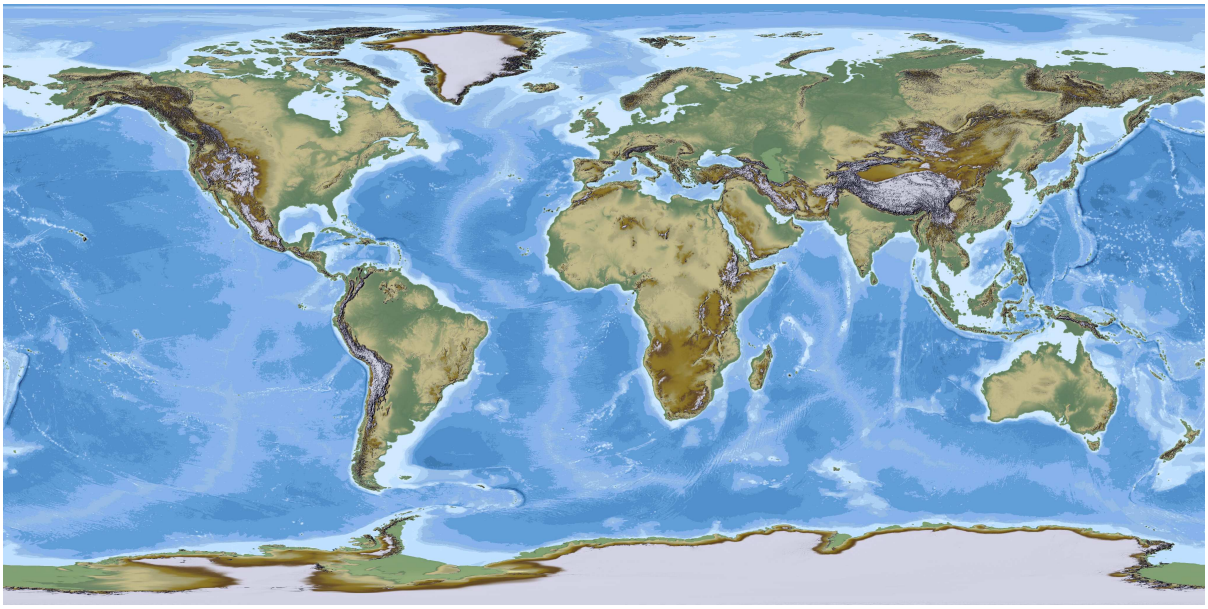

Figure 32. Vector bathymetry (Natural Earth) & elevation coverage (View Finder Panoramas by Jonathan de Ferranti, from SRTM and other sources)

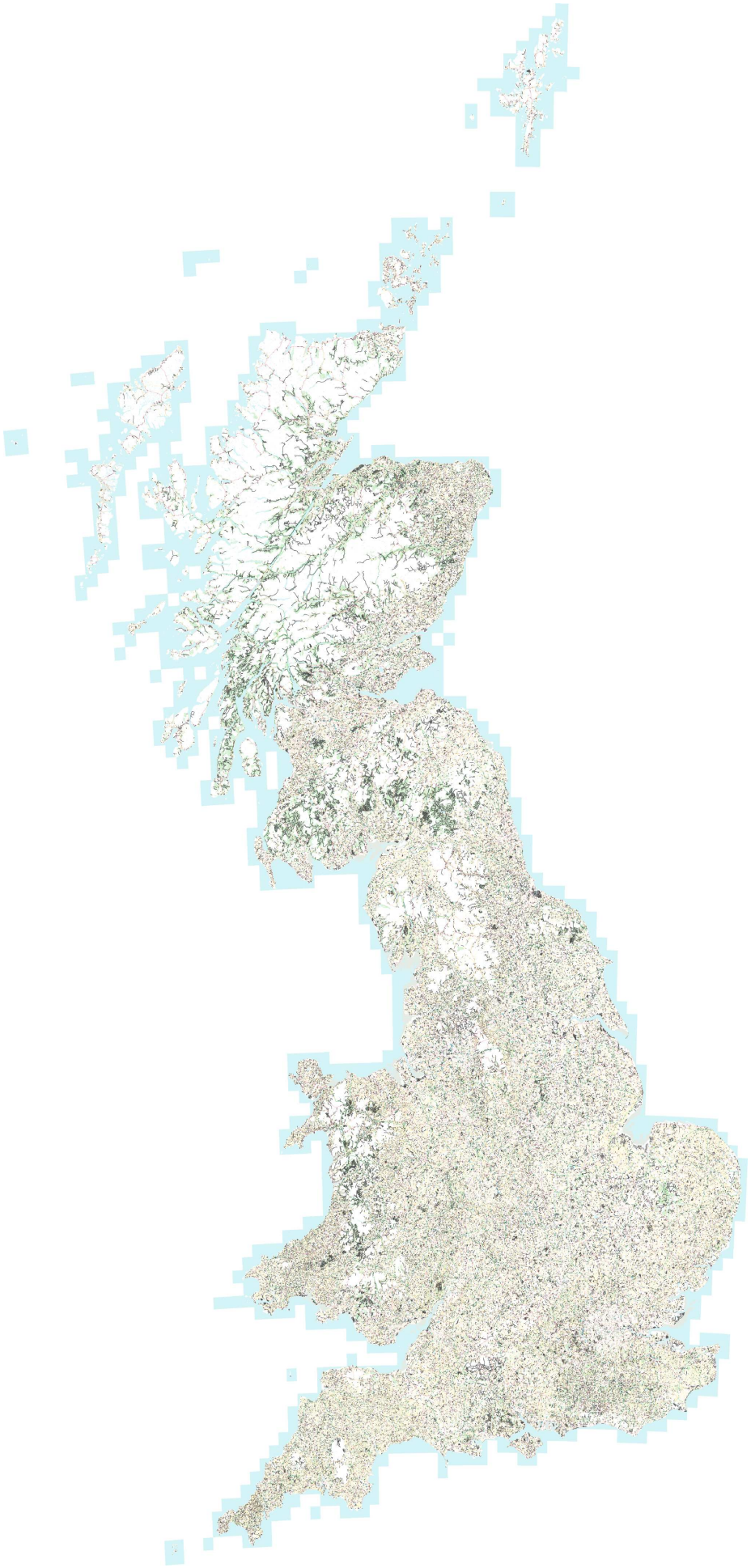

Figure 33. Whole of Great Britain, OpenMap Local from Ordnance Survey (© Crown Copyright and database right 2020)

Figure 34. OpenMap Local (zoomed in) from Ordnance Survey (© Crown Copyright and database right 2020)

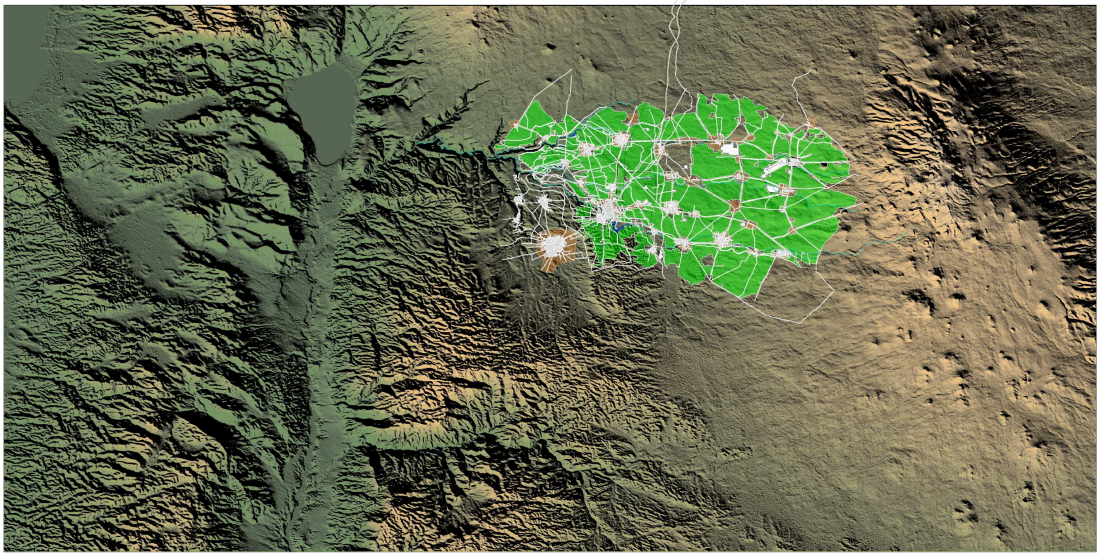

Figure 35. Server-side rendered map of Daraa from DTED & NGA Topographic Data Store, Copyright (c) OpenStreetMap Contributors

Figure 36. Ottawa - Map Tile Copyright (c) OpenStreetMap Contributors

Figure 37. Ottawa - Map Tile (next zoom level) Copyright (c) OpenStreetMap Contributors

10.2. pygeoapi

The server provided by EOX [https://eox.at] uses the software pygeoapi [https://pygeoapi.io] to provide the API interfaces. EOX improved support for OGC API - Coverages and Processes in pygeoapi and implemented support for the Workflows extensions as well.

Server instances are provided on the Euro Data Cube (EDC) [https://eurodatacube.com] exploitation platform. Each interested user can ask to get their server instance, the MOAW app, deployed and enabled which is run in the user's EDC EOxHub Workspace [https://eurodatacube.com/marketplace/infra/edc_eoxhub_workspace]. The workspace is offering a managed JupyterLab [https://eurodatacube.com/marketplace/apps/edc_jupyterlab] instance with curated base images ready for usage besides the runtime for apps like the MOAW one.

This JupyterLab serves two main purposes in the MOAW context, to develop a process or algorithm as well as to inspect and debug executed processes or jobs.

Under the hood the Euro Data Cube (EDC) exploitation platform is based on a Kubernetes cluster. Figure 1 below shows the architecture of the MOAW app as deployed on EDC. The figure can be read from left to right to follow the execution of a workflow.

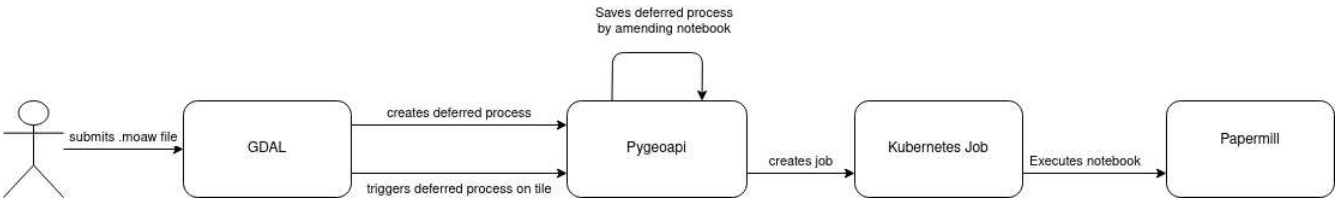

Figure 38. Architecture of server instance provided by EOX

The workflow execution starts with a user submitting a workflow as `.moaw` file using for example GDAL. The workflow is sent to a generic process `python-coverage-processor` provided by pygeoapi as shown in the code snippet below. This generic process supports:

- data input via references to collections

- selection of bands from the input `sourceBands`
- python functions `bandsPythonFunctions`

```
$ docker run --rm -it -v ${PWD}:/data osgeo/gdal:ubuntu-full-latest bash -c 'export
CPL_VSIL_NETWORK_STATS_ENABLED=YES && export CPL_VSIL_SHOW_NETWORK_STATS=YES && export
CPL_DEBUG=ON && gdal_translate -if OGCAPI -of COG /data/eox.moaw /data/test.tif
-projwin 19.99 60 20 59.99'
HTTP: Establish persistent session named 'OGCAPI:0x561a43528590'.
HTTP: Fetch(https://edc-oapi.hub.eox.at/oapi/processes/python-coverage-processor)
HTTP: libcurl/7.68.0 GnuTLS/3.6.13 zlib/1.2.11 brotli/1.0.7 libidn2/2.2.0
libpsl/0.21.0 (+libidn2/2.2.0) libssh/0.9.3/openssl/zlib nghttp2/1.40.0 librtmp/2.3
HTTP: These POSTFIELDS were sent:
{
    "process": "https://edc-oapi.hub.eox.at/oapi/processes/python-coverage-processor",
    "inputs": {
        "data" : [ { "collection": "https://edc-
oapi.hub.eox.at/oapi/collections/S2L2A" } ],
        "sourceBands" : [ { "value" : [ "B04", "B08" ] } ],
        "bandsPythonFunctions" : {
            "value" : {
                "ndvi" : "return (ds[0].B08.astype(float) - ds[0].B04.astype(float)) /
(ds[0].B04.astype(float) + ds[0].B08.astype(float))",
                "b4" : "return (ds[0].B04) * 5.0"
            }
        }
    }
}

HTTP: These HTTP headers were set: Accept: application/geo+json, application/json
```

The workflow is executed in two steps:

1. create or register a deferred process
2. trigger or execute the deferred process on a tile

The registered workflow or process from the first step is stored as Jupyter notebook with the parameters encoded as shown in Figure 2. This stored notebook is used by pygeoapi afterwards to create Kubernetes jobs which executes the stored notebook in a headless manner using papermill.

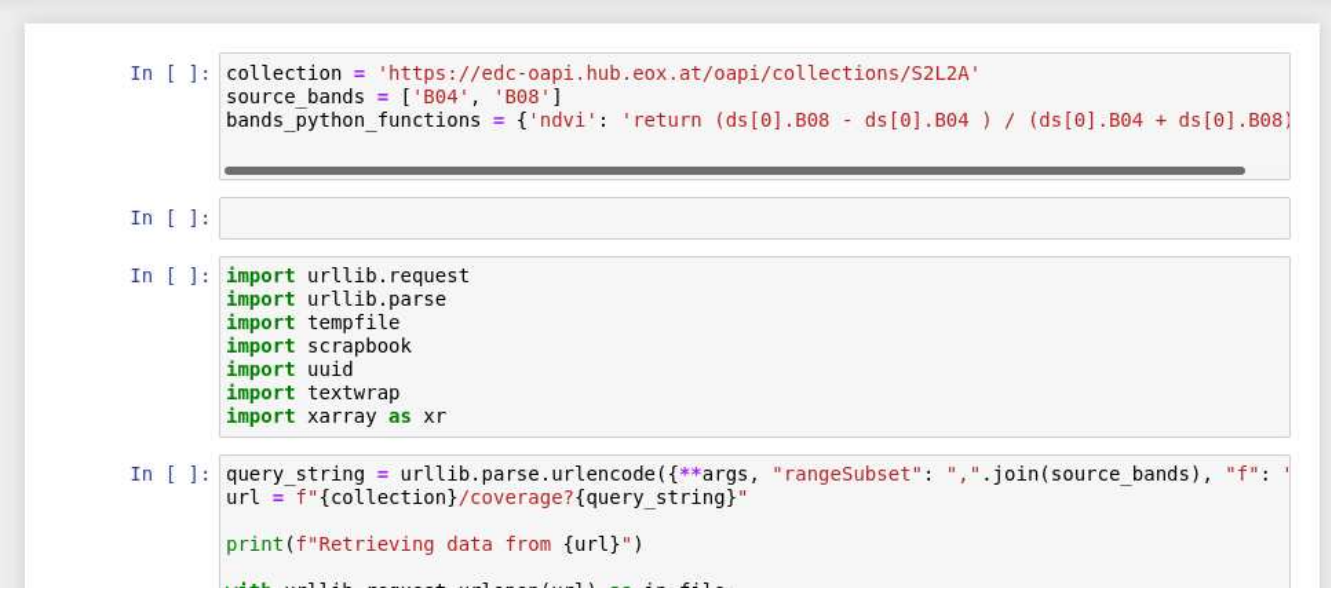

Figure 39. Workflow as notebook

The first step returns the URL to the newly registered process using `/deferred/<Id>` nested below the generic `python-coverage-processor` process as shown in the code snippet below. Note that this code snippet is the continuation of the output from the GDAL command executed in the previous code snippet. This API endpoint features a fully fledged OAPI - Coverage endpoint including metadata combined from the input data collection (area, time) and the process itself (bands).

```
OGCAPI:
<GDAL_WMS>
    <Service name="OGCAPICoverage">
        <ServerUrl>https://edc-oapi.hub.eox.at/oapi/processes/python-coverage-
processor/deferred/e687e24286a272cc5fbd4d17cda659bbbc753747bd7cbe6de21cd89137bc8da3/co
verage?subset=Lon(${minx}:${maxx}),Lat(${miny}:${maxy})&amp;scaleSize=Lon(${width}),La
t(${height})</ServerUrl>
    </Service>
    <DataWindow>
        <UpperLeftX>-180</UpperLeftX>
        <UpperLeftY>90</UpperLeftY>
        <LowerRightX>180</LowerRightX>
        <LowerRightY>-90</LowerRightY>
        <SizeX>4007502</SizeX>
        <SizeY>2003751</SizeY>
    </DataWindow>
    <OverviewCount>14</OverviewCount>
    <BlockSizeX>256</BlockSizeX>
    <BlockSizeY>256</BlockSizeY>
    <BandsCount>2</BandsCount>
    <DataType>UInt16</DataType>
    <MaxConnections>5</MaxConnections>
    <Accept>image/tiff;application=geotiff</Accept>
    <Cache />
</GDAL_WMS>

GDAL: GDALOpen(/data/eox.moaw, this=0x561a43528590) succeeds as OGCAPI.
```

```
Input file size is 4007502, 2003751
COG: Generating final product: start
GTiff: No source overviews to copy
GDAL: Use hashset band block cache
GDAL: GDAL_CACHEMAX = 783 MB

HTTP: Requesting [1/1] https://edc-oapi.hub.eox.at/oapi/processes/python-coverage-
processor/deferred/e687e24286a272cc5fbd4d17cda659bbbc753747bd7cbe6de21cd89137bc8da3/co
verage?subset=Lon(19.9807760545097608:20.0037729238812574),Lat(59.9890854702006209:60.
0120823395721317)&scaleSize=Lon(256),Lat(256)

HTTP: Request [0] https://edc-oapi.hub.eox.at/oapi/processes/python-coverage-
processor/deferred/e687e24286a272cc5fbd4d17cda659bbbc753747bd7cbe6de21cd89137bc8da3/co
verage?subset=Lon(19.9807760545097608:20.0037729238812574),Lat(59.9890854702006209:60.
0120823395721317)&scaleSize=Lon(256),Lat(256) : status = 200, type = image/tiff, error
= (null)

GDAL: GDALOpen(/vsimem/wms/0x7f80c037f010/wmsresult.dat, this=0x561a442627e0) succeeds
as GTiff.
GDAL: GDALClose(/vsimem/wms/0x7f80c037f010/wmsresult.dat, this=0x561a442627e0)
WMS: Clean cache
0...10...20...30...40...50...60...70...80...90...100 - done.
COG: Generating final product: end
GDAL: GDALClose(/data/test.tif, this=0x561a4356c6a0)
HTTP: Ended persistent session named 'OGCAPI:0x561a43528590'.
GDAL: GDALClose(/data/eox.moaw, this=0x561a43528590)
GDAL: In GDALDestroy - unloading GDAL shared library.
```

As written above, the actual processing is triggered by pygeoapi. pygeoapi uses an interface via a Kubernetes back-end and leverages Kubernetes native jobs. Kubernetes automatically selects which node to run the job on and provides autoscaling if needed. The stored notebook is executed by papermill. Papermill stores the resulting notebook which can further be used for logging or debugging purposes. Figure 3 below shows the pygeoapi GUI to inspect executed jobs.

Via pygeoapi an integrated storage can be accessed. It provides access to synchronous result, e.g. as GeoTIFFs, as well as asynchronous results which are exposed on an S3 bucket or in the cluster storage for access within the Euro Data Cube EOxHub Workspace.

pygeoapi

Home / Processes / Python Coverage Processor / Jobs / 1abc690a-87e1-11eb-9aea-7e2f079691de

Job status

Status: successful

Progress: 100%

Message

Completed

Parameters


```
{
  "args": {
    "subset": "Lon(19.9807760545097608:20.0037729238812574),Lat(59.9890854702006209:60.0120823395721317)",
    "scaleSize": "Lon(256),Lat(256)"
  }
}
```


Progress

Duration

0:00:10.895044

Figure 40. pygeoapi job status

A summary of contributions to the Free and Open Source Software pygeoapi is given in the respective section for pygeoapi of Chapter 14.

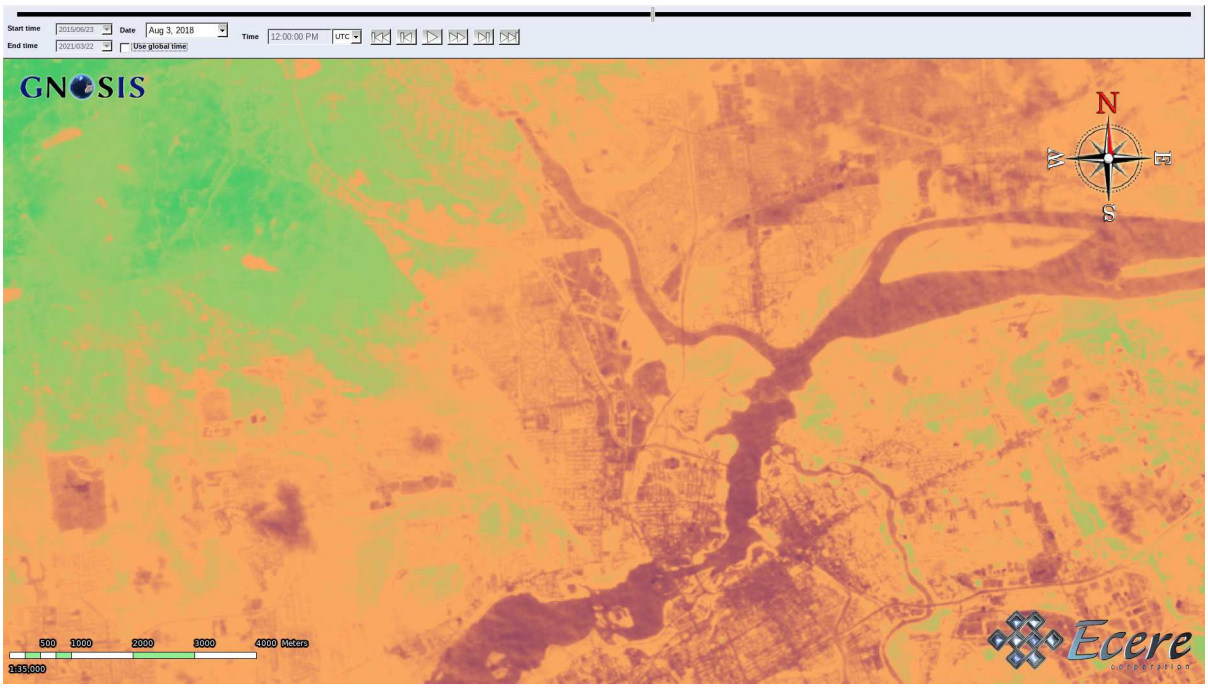

Figure 41. Visualization of NDVI calculation via modular workflow execution of python coverage processor from EOX's pygeoapi server, defining coverage bands using custom Python functions

10.3. TerraNexus

Pangaea Innovations implemented support for OGC API - Features in its TerraNexus server, and successfully achieved OGC certification for its implementation. The server utilizes a Discrete Global Grid System to optimize data retrieval and processing.

Pangaea Innovations also implemented support for the Workflows extension, for use within a flood scenario demonstration.

Figure 42. OpenStreetMap buildings available from TerraNexus Features API

10.4. rasdaman

Rasdaman ("raster data manager") is the pioneer datacube engine giving access to massive multi-dimensional raster data through OGC standards, in particular: the Web Coverage Processing Service (WCPS) datacube analytics language.

Open-source rasdaman community is residing at https://rasdaman.org/, including its [https://doc.rasdaman.org] documentation].

OAPI-coverages has been implemented based on some version of the evolving specification. This includes WCS-like single parametrized requests and in particular WCPS analytics of any complexity illustrated by the following example (linebreaks introduced for better understanding):

```
https://oapi.rasdaman.org/rasdaman/oapi/wcps?
    q=for $c in (mean_summer_airtemp),
          $d in (decode($1))
      return encode( $c+$d-60, "image/jpeg")
```

A [demo service](https://oapi.rasdaman.org/rasdaman/oapi/collections) offers data sets S2_NDVI_84, S2_FALSE_COLOR_84, AverageTemperature, AverageChlorophyll, mean_summer_airtemp.

The following request examples illustrate access:

- `/` - landing page
  - Example: https://oapi.rasdaman.org/rasdaman/oapi/
- `/collections` - returns the list of collections hosted by the server
  - Example: https://oapi.rasdaman.org/rasdaman/oapi/collections
- `/collections/{coverageId}` - returns the collection object
  - Example: https://oapi.rasdaman.org/rasdaman/oapi/collections/S2_NDVI_84
- `/collections/{collectionId}/coverage` - returns the coverage object represented by this collection
  - Example: https://oapi.rasdaman.org/rasdaman/oapi/collections/S2_NDVI_84/coverage
  - Additional parameters:
    - subset: any number of subset parameters, or 1 single subset parameter with multiple values, comma-separated by axis
    - f: format parameter
    - Examples:
      - https://oapi.rasdaman.org/rasdaman/oapi/collections/AverageTemperature/coverage?subset=ansi(%222012-12-01T20:07:00.500Z%22)&f=image/png
      - https://oapi.rasdaman.org/rasdaman/oapi/collections/S2_NDVI_84/coverage?subset=Lat(51.9:52.1)&subset=Long(-4.1:-3.9)&subset=ansi(%222018-11-14%22)&f=image/png
      - which is the same as https://oapi.rasdaman.org/rasdaman/oapi/collections/S2_NDVI_84/coverage?subset=Lat(51.9:52.1),Long(-4.1:-3.9),ansi(%222018-11-14%22)&f=image/png
- `/collections/{collectionId}/coverage/domainset` - returns the domain set of the coverage with the specified id
  - Example: https://oapi.rasdaman.org/rasdaman/oapi/collections/S2_NDVI_84/coverage/domainset
- `/collections/{collectionId}/coverage/rangetype` - returns the range type of the coverage with the specified id
  - Example: https://oapi.rasdaman.org/rasdaman/oapi/collections/S2_NDVI_84/coverage/rangetype
- `/collections/{collectionId}/coverage/rangeset` - returns the range set of the coverage with the specified id
  - Additional parameters:
    - subset: any number of subset parameters, or 1 single subset parameter with multiple values, comma-separated by axis
  - Examples:
    - https://oapi.rasdaman.org/rasdaman/oapi/collections/mean_summer_airtemp/coverage/

rangeset?subset=Lat(-14:-13)&subset=Long(130:131)

- https://oapi.rasdaman.org/rasdaman/oapi/collections/S2_NDVI_84/coverage/rangeset?subset=Lat(51.9:52)&subset=Long(-4.1:-4.1)&subset=ansi(%222018-11-14%22)

- `/collections/{coverageId}/coverage/metadata` - returns the metadata of the coverage with the specified id
  - Example: https://oapi.rasdaman.org/rasdaman/oapi/collections/S2_NDVI_84/coverage/metadata
- Content negotiation
  - Currently rasdaman partially supports content negotiation in the Accept header as per RFC 2616 https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html. In the Accept request header field a comma-separated list of media-range parameters (i.e., MIME types) is evaluated. Any accept-params are ignored. Further, media-range parameters may not contain "*" wildcards.
  - Example:

```
        curl -H "Accept: image/tiff;application=geotiff;q=1.0,image/png;q=0.5"
"https://oapi.rasdaman.org/rasdaman/oapi/collections/mean_summer_airtemp/coverage"
--output test.tif
```

- `/wcps` - executes a WCPS query
  - Parameters:
    - Q: WCPS query, properly URL escaped
    - 1, 2, 3, ...: each value is either textual or binary coverage. The query can reference these as $1, $2, $3, etc. String parameters are copied directly through textual substitution in the query string, coverages need to be decoded through `decode($1)`. See documentation for details: https://doc.rasdaman.org/05_geo-services-guide.html?highlight=post#positional-parameter-in-wcps
  - Example: Visualize NDVI into an 8-bit greyscale JPEG:
    - Query:

```
for $c in (S2_FALSE_COLOR_84)
return
  encode(
          128 * ( 1 + ($c.0 - $c.1) / ($c.0 + $c.1) )
          [Lat(51.9:52.1), Long(-4.1:-3.9), ansi("2018-11-14")],
          "image/jpeg")
```

- URL: https://oapi.rasdaman.org/rasdaman/oapi/wcps?Q=for%20%24c%20in%20(S2_FALSE_COLOR_84)%20return%20encode(128%2A(1%20%2B%20(%24c.0%20-%20%24c.1%20%2B%201)%20/%20(%24c.0%20%2B%20%24c.1))%20%5BLat(51.9:52.1),%20Long(-4.1:-3.9),%20ansi(%222018-11-14%22)%5D,%22image/jpeg%22)
  - Example: Upload coverage from a 2D GeoTiff file for processing, doing data fusion with

server-side stored coverages (both coverages have same rangetypes, same domainsets):

```
            curl -s "https://oapi.rasdaman.org/rasdaman/oapi/wcps" \
                 -F 'q=for $c in (mean_summer_airtemp), $d in (decode($1)) return
encode($c - 10 + $d - 50, "jpeg")' \
                 -F "1=@/home/rasdaman/Downloads/test.tif" \
                  > test.jpeg
```

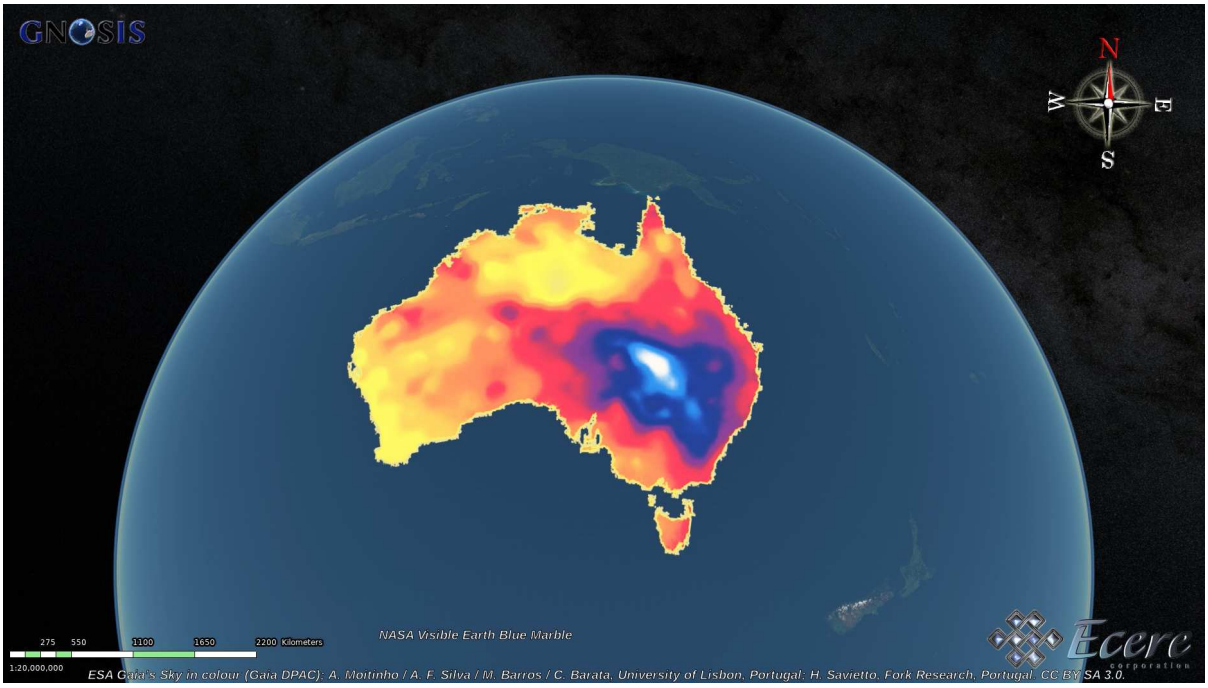

Figure 43. Results (passed through values) from rasdaman Web Coverage Processing Service (WCPS) integrated within the OGC API

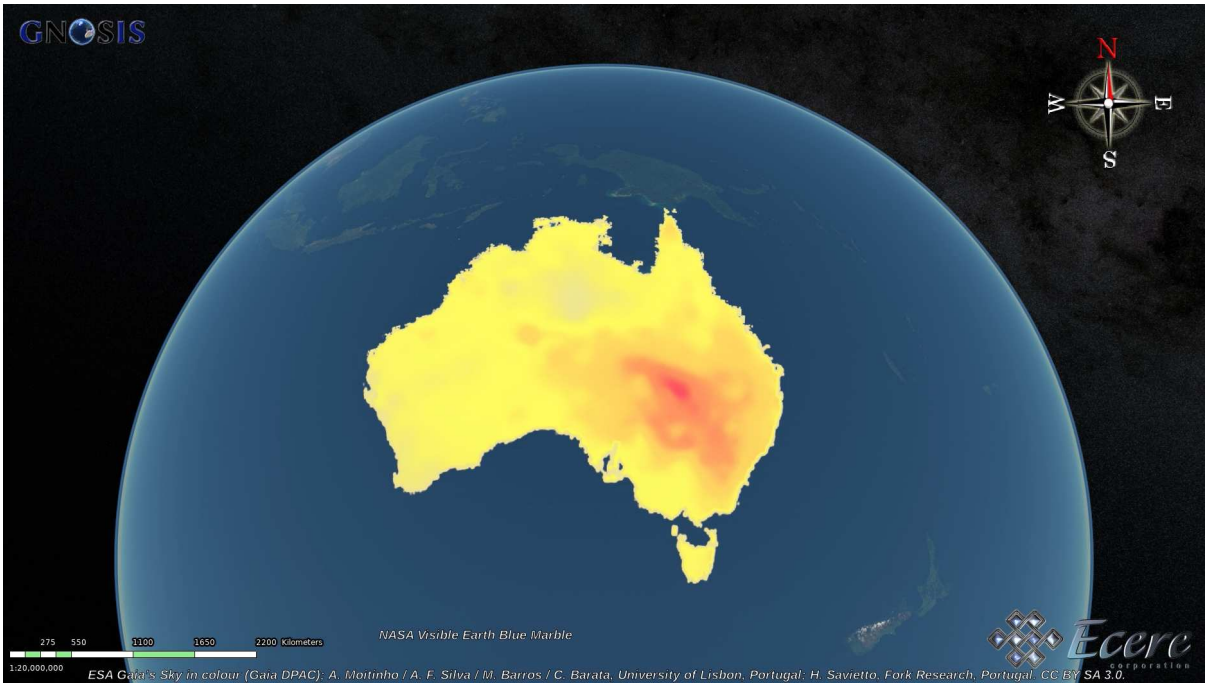

Figure 44. Results (modified values) from rasdaman Web Coverage Processing Service (WCPS) integrated within the OGC API

10.5. CubeSERV

CubeSERV implements support for several OGC APIs, including Features, Coverages, Maps, Tiles, Processes and Records.

CubeWerx initiated development to support the Workflows extension. Technology Integration Experiments using the Features, Coverages, Maps and Tiles APIs were performed with the GNOSIS Cartographer and MiraMon visualization clients.

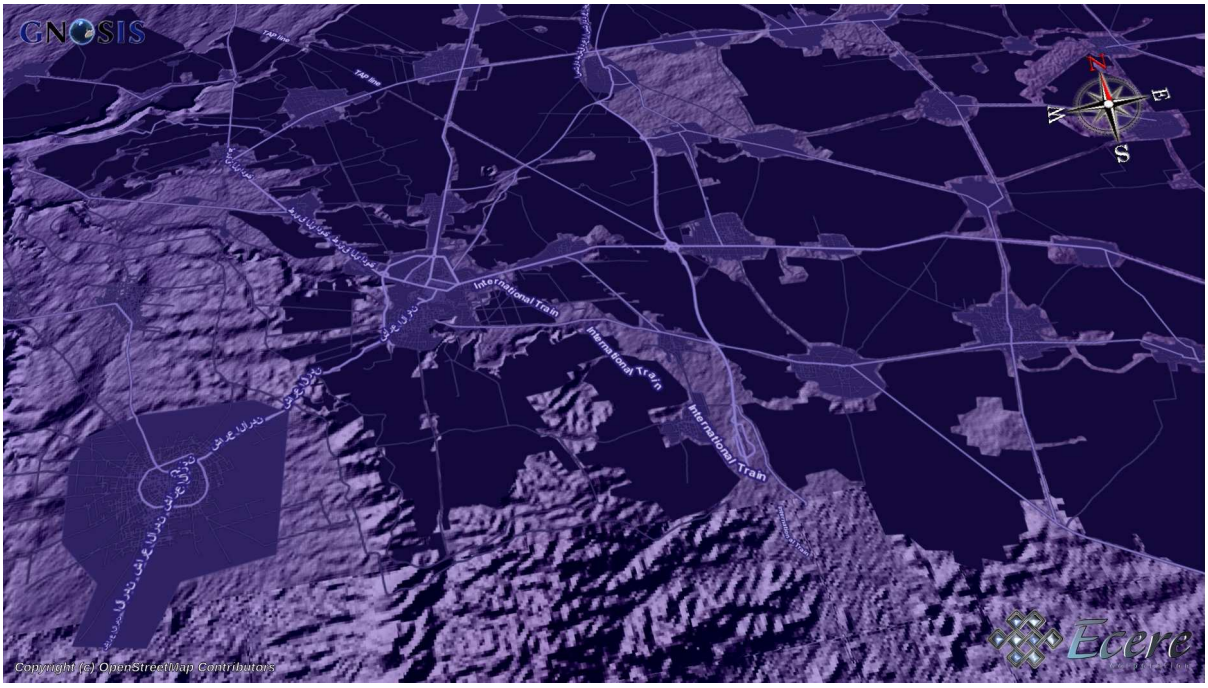

Figure 45. OpenStreetMap and DTED elevation rendered server-side by CubeSERV, accesed using OGC API - Maps in GNOSIS Cartographer

10.6. MiraMon

The Center for Ecological Research and Forestry Application of the Autonomous University of Barcelona (UAB-CREAF) initiated the development of *OGC API - Maps & Tiles* server support in the MiraMon server. However, since a large amount of efforts was diverted to focus on the specifications development, issues still remained at the end of the project preventing from achieving successul technology integration experiments.

10.7. javaPS

52 North provided access to the javaPS service, with support for *OGC API - Processes - Part 1: Core*. In order to integrate the service within modular workflows, Ecere provide an adapter process running on the GNOSIS Map Server through which both QGIS/GDAL and GNOSIS Cartographer, as well as cascading services, could interface with the javaPS processes.

The service provided two processes in particular: a Normalized Difference Vegetation Index calculation (NDVI), running in JupyterLab, deployed using an Application Deployment Execution System (ADES), as well as a routing process supporting developed for the OGC Open Routing Pilot. 52 North deployed multiple flavors of the routing process, one of them leveraging Ecere's

OpenStreetMap Routing Engine (OSMERE), which was used for the experiments.

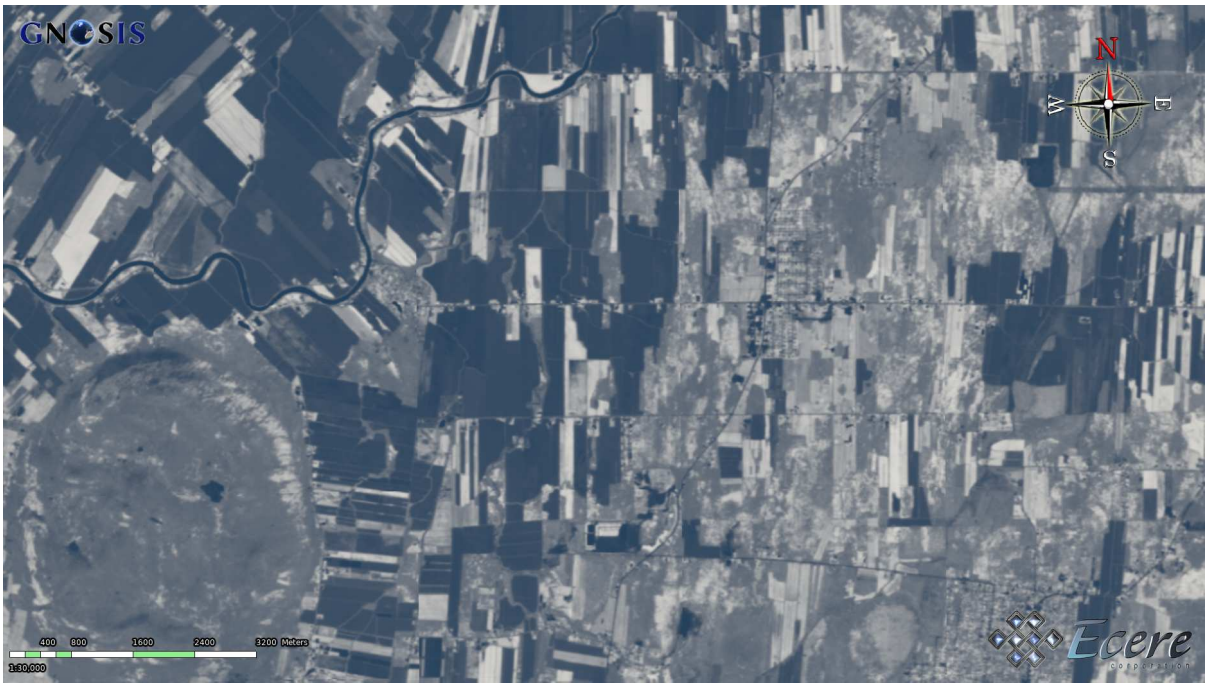

Figure 46. NDVI Calculation performed by javaPS, from sentinel-2 imagery retrieved from EOX's pygeoapi server on the EuroDataCube, accessed via the MOAWAdapter process from GNOSIS Map Server, and visualized in GNOSIS Cartographer

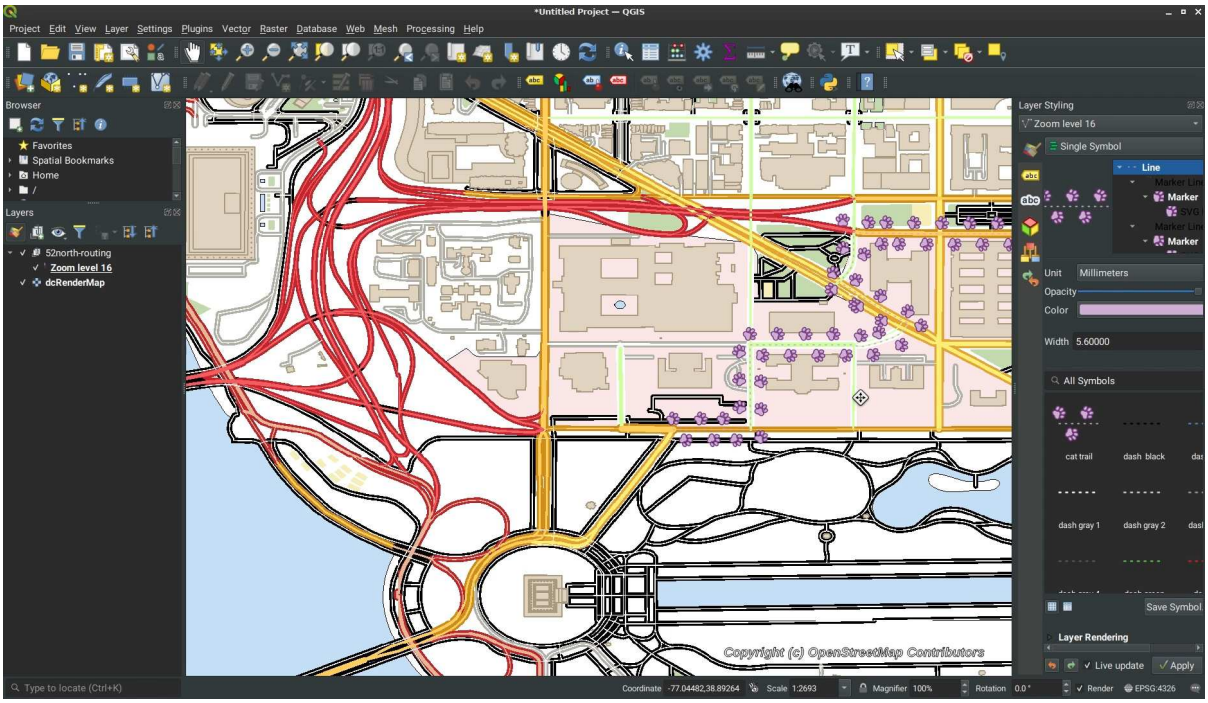

Figure 47. Routing performed on javaPS, leveraging OSMERE routing engine, accessed via the MOAWAdapter process from GNOSIS Map Server, and visualized in QGIS/GDAL on top of a map of Washington D.C. rendered server-side on the GNOSIS Map Server. Data Copyright (c) OpenStreetMap contributors

Chapter 11. Implemented Client Components

This chapter describes the different client components developed and enhanced during this project: GNOSIS Cartographer, the GDAL OGCAPI driver & QGIS, MiraMon, and TerraNexus client.

11.1. GNOSIS Cartographer

Cartographer supports APIs for vector features, vector tiles, coverages, maps, coverage tiles and map tiles in multiple formats. These include GNOSIS' own GMT (Gnosis Map Tile) format, Mapbox Vector Tiles, PNG, JPEG, GeoTIFF and GeoJSON.

The client will recognize a process from an server endpoint, as well as pregenerated workflows in JSON documents with the ".moaw" extension. Workflows can be manually edited from the Workflow Editor JSON tab. In the Editor tab, connecting to a process hyperlink will generate configurable inputs. Accepting the configuration will close the editor and add the executed result as a feature layer to the layers panel for visualization. This layer is subject to changes in visibility, styling and priority as with any other layer in Cartographer.

Ecere also implemented support for Coverages, including support for temporal datasets, in its GNOSIS Cartographer client, and also improved support for other OGC API data delivery specifications including Maps, Tiles and Features, as part of the unified OGC API client module of its GNOSIS Software Development Kit. The team performed experiments with multiple combinations of the different services.

In GNOSIS Cartographer, Ecere also developed a new Workflow editor, featuring in its initial version a basic User Interface to edit workflows by connecting to processing services, parsing the process description information and presenting a list of inputs for which the user can select appropriate values, including the option of pointing to OGC API datasets or to specify a nested process as an input. These workflows can be exported, for example to be visualized with GDAL in QGIS, or their results can be directly visualized in GNOSIS Cartographer.

Ecere also developed capabilities for visualizing, processing and streaming very large point clouds datasets in its GNOSIS Map Server and GNOSIS Cartographer visualization tool.

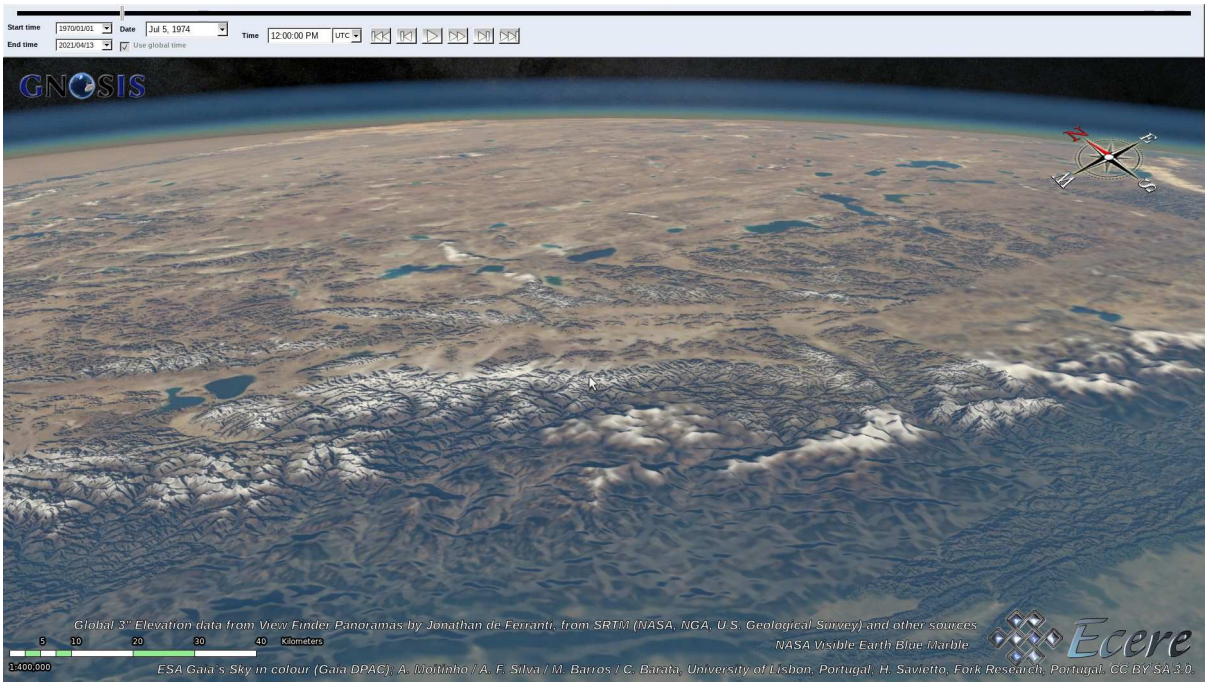

Figure 48. NASA Blue Marble with temporal support & 3D Terrain generated from global elevation model, both accessed via OGC API - Tiles in GNOSIS Cartographer

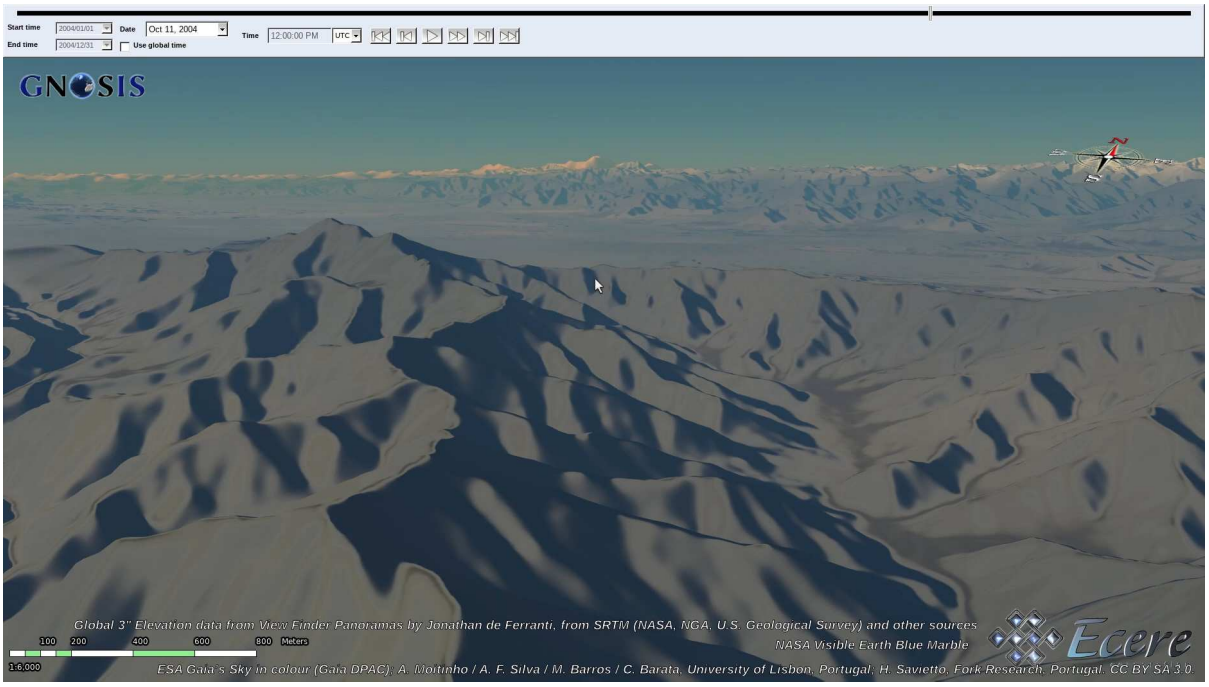

Figure 49. NASA Blue Marble with temporal support & 3D Terrain generated from global elevation model, both accessed via OGC API - Tiles in GNOSIS Cartographer

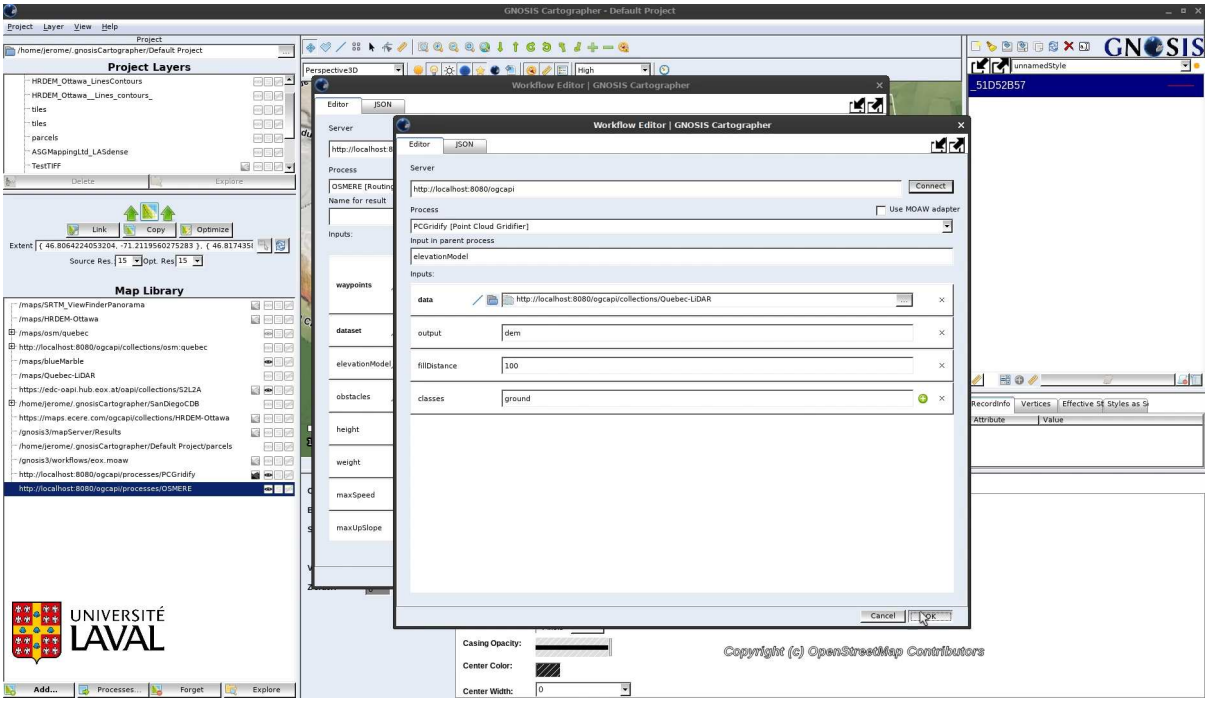

Figure 50. Processing Workflow Editor in GNOSIS Cartographer, showing nested process for generating elevation model from gridifying point cloud, itself an input to routing process

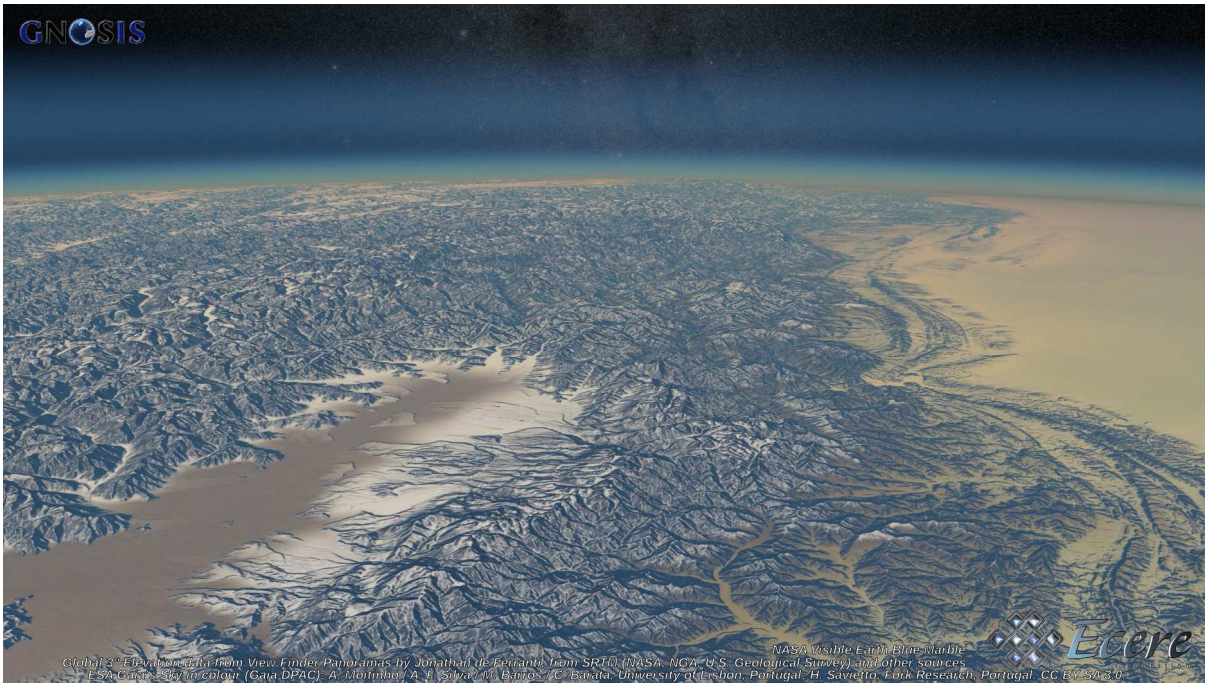

Figure 51. NASA Blue Marble & 3D Terrain generated from global elevation model, both accessed via OGC API - Tiles in GNOSIS Cartographer, with ESA's Gaia Sky in Colour in the background

11.2. GDAL (OGCAPI driver) & QGIS

11.2.1. GDAL

Spatialys developed a new unified OGCAPI client driver [https://gdal.org/drivers/raster/ogcapi.html] for GDAL in the context of this project, with support for OGC API - Maps, Coverages, Tiles, including support for Mapbox and GeoJSON vector tiles. The driver supports accessing an OGC API landing page, collection or a Workflow as defined by the extension loaded from a JSON document with the .moaw extension. The driver was also succesfully used within QGIS for experiments with all

servers.

GDAL 3.2, released in November 2020, incorporated this driver. The driver is built-in by default in all popular binary builds of GDAL, and is thus easily available.

This driver supports the following APIs:

- Tiles, raster (PNG, JPEG) and vector (GeoJSON, Mapbox Vector Tiles)
- Maps (PNG, JPEG)
- Coverages: GeoTIFF

The driver recognizes a filename (with .moaw extension) containing a JSON document, like the following, specifying a deferred processing.

```
{
   "process" : "https://maps.ecere.com/ogcapi/processes/RenderMap",
   "inputs" : {
      "transparent" : { "value" : false },
      "background" : { "value" : "navy" },
      "layers" : [
         { "collection" :
"https://maps.ecere.com/ogcapi/collections/NaturalEarth:physical:bathymetry" },
         { "collection" :
"https://maps.ecere.com/ogcapi/collections/SRTM_ViewFinderPanorama" }
      ]
   }
}
```

It also recognizes strings formatted like “OGCAPI:{url}” where {url} is the URL to a OGC API landing page, or the URL to the collection description pages. In that case the driver will return subdatasets, that is a list of raster layers, with the dataset name of different collections.

```
$ gdalinfo OGCAPI:https://maps.ecere.com/ogcapi

Driver: OGCAPI/OGCAPI
Files: none associated
Size is 512, 512
Origin = (0.000000000000000,0.000000000000000)
Pixel Size = (1.000000000000000,1.000000000000000)
Subdatasets:

SUBDATASET_1_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth?f=json
  SUBDATASET_1_DESC=Collection NaturalEarth

SUBDATASET_2_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth:raster
?f=json
  SUBDATASET_2_DESC=Collection raster

SUBDATASET_3_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth:raster
:NE1_HR_LC_SR_W_DR?f=json
  SUBDATASET_3_DESC=Collection NE1_HR_LC_SR_W_DR

SUBDATASET_4_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth:raster
:NE2_HR_LC_SR_W_DR?f=json
  SUBDATASET_4_DESC=Collection NE2_HR_LC_SR_W_DR

SUBDATASET_5_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth:raster
:HYP_HR_SR_OB_DR?f=json
  SUBDATASET_5_DESC=Collection HYP_HR_SR_OB_DR

SUBDATASET_6_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth:cultur
al?f=json
  SUBDATASET_6_DESC=Collection cultural

SUBDATASET_7_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth:cultur
al:ne_10m_admin_0_disputed_areas_scale_rank_minor_islands?f=json
  SUBDATASET_7_DESC=Collection ne_10m_admin_0_disputed_areas_scale_rank_minor_islands
[...]

SUBDATASET_285_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/QuebecCity?f=json
  SUBDATASET_285_DESC=Collection QuebecCity
  SUBDATASET_286_NAME=OGCAPI:https://maps.ecere.com/ogcapi/collections/HRDEM-
Ottawa?f=json
  SUBDATASET_286_DESC=Collection HRDEM-Ottawa
```

The user can then inspect the metadata of a given collection:

```
$ gdalinfo
"OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth:raster:NE1_HR_LC_SR_W_D
R?f=json"
```

```
Driver: OGCAPI/OGCAPI
Files: none associated
Size is 32768, 16384
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)"],
        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 = (-180.000000000000000,90.000205823663904)
Pixel Size = (0.010986328125000,-0.010986340687479)
Metadata:
  TITLE=NE1_HR_LC_SR_W_DR
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (-180.0000000,  90.0002058) (180d 0' 0.00"W, 90d 0' 0.74"N)
Lower Left  (-180.0000000, -90.0000000) (180d 0' 0.00"W, 90d 0' 0.00"S)
Upper Right ( 180.0000000,  90.0002058) (180d 0' 0.00"E, 90d 0' 0.74"N)
Lower Right ( 180.0000000, -90.0000000) (180d 0' 0.00"E, 90d 0' 0.00"S)
Center      (   0.0000000,   0.0001029) (  0d 0' 0.01"E,  0d 0' 0.37"N)
Band 1 Block=256x256 Type=Byte, ColorInterp=Red
  Overviews: 16384x8192, 8192x4096, 4096x2048, 2048x1024, 1024x512, 512x256, 256x128
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: 16384x8192, 8192x4096, 4096x2048, 2048x1024, 1024x512,
512x256, 256x128
Band 2 Block=256x256 Type=Byte, ColorInterp=Green
  Overviews: 16384x8192, 8192x4096, 4096x2048, 2048x1024, 1024x512, 512x256, 256x128
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: 16384x8192, 8192x4096, 4096x2048, 2048x1024, 1024x512,
```

```
512x256, 256x128
Band 3 Block=256x256 Type=Byte, ColorInterp=Blue
  Overviews: 16384x8192, 8192x4096, 4096x2048, 2048x1024, 1024x512, 512x256, 256x128
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: 16384x8192, 8192x4096, 4096x2048, 2048x1024, 1024x512,
512x256, 256x128
Band 4 Block=256x256 Type=Byte, ColorInterp=Alpha
  Overviews: 16384x8192, 8192x4096, 4096x2048, 2048x1024, 1024x512, 512x256, 256x128
```

and proceed for example with extracting a subset of it:

```
$ gdal_translate -projwin -5 55 15 40 -outsize 1024 0 \

"OGCAPI:https://maps.ecere.com/ogcapi/collections/NaturalEarth:raster:NE1_HR_LC_SR_W_D
R?f=json" \
     out.tif
```

The driver incorporates logic to decide automatically which API to use, and which format of data to select when several ones are possible. The user has the ability to specify open options to decide which API and/or format must be used precisely, the choice of the tile matrix set for the Tiles API, if tiles must be cached or not, restrict the bounding box of interest, etc.

11.2.2. QGIS

Due to the abstraction offered by GDAL and the generic use that QGIS makes of it, no particular change was required in QGIS to be able to support OGC API, by using the GDAL OGCAPI driver underneath. In particular, it is accessible through the QGIS 3.18 Windows installer. Users can easily drag and drop a .moaw file into QGIS user interface and see the result of the workflow displayed in the QGIS map canvas.

An enhancement was however submitted and incorporated into QGIS to better report the name of GDAL datasets with multiple raster layers.

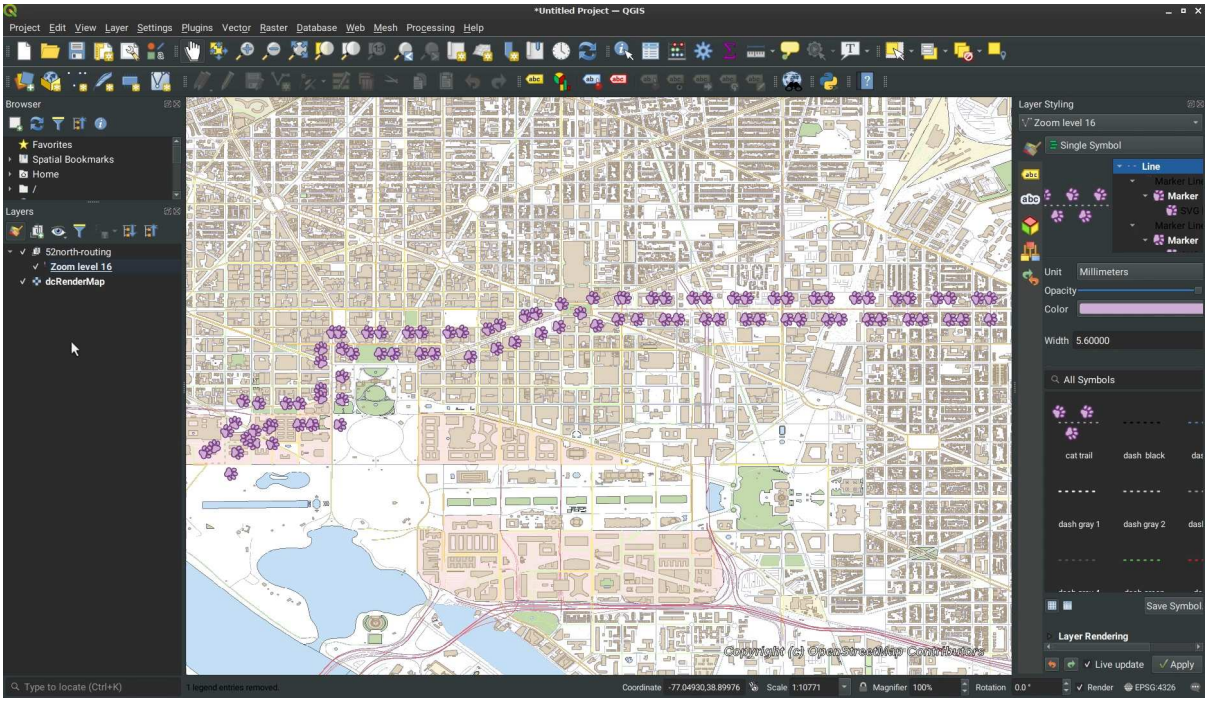

Figure 52. Visualizing route calculation workflow results (vector tiles) in QGIS (using the new GDAL OGC API driver) from javaPS process accessed via MOAWAdapter on GNOSIS Map Server, on top of a map rendered server-side on GNOSIS Map Server, using OGC API - Tiles

11.3. TerraNexus Client

11.4. MiraMon Client

UAB-CREAF implemented support for *OGC API - Maps and Tiles* in MiraMon client.

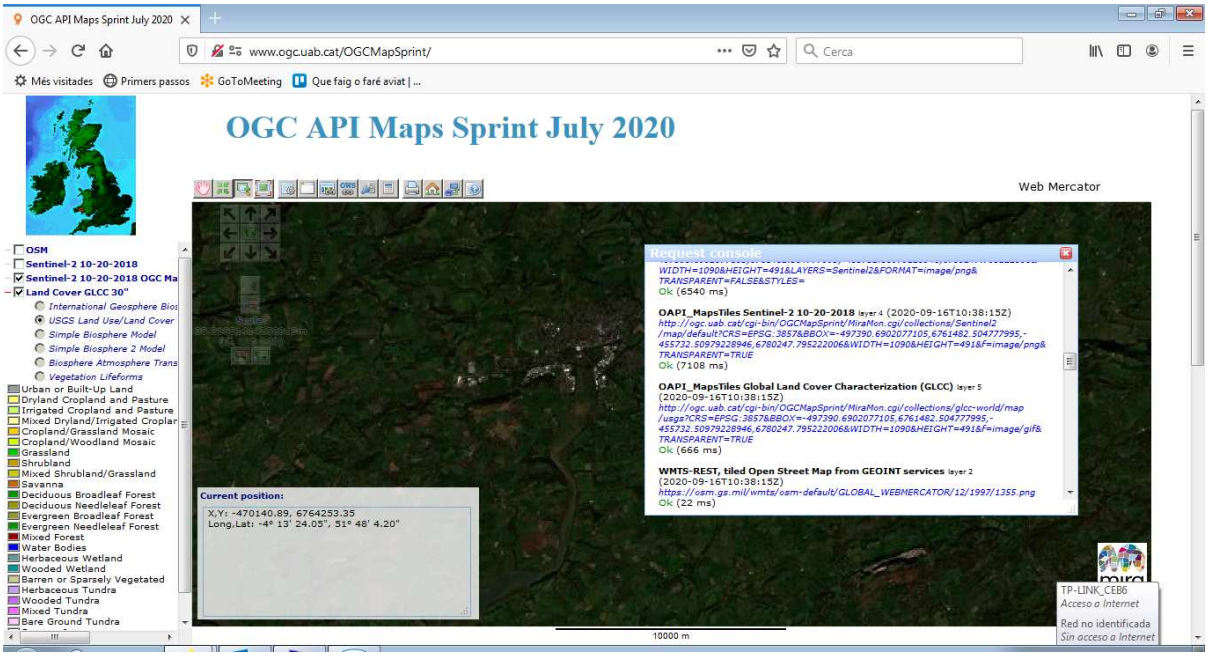

Figure 53. MiraMon client accessing data using OGC API - Maps

Chapter 12. Technology Integration Experiments

This chapters lists the different Technology Integration Experiments performed during this project, and provide a matrix of the successful connectivity and interoperability achieved between the implemented components.

12.1. TIE matrix

These tables are organized per client, for each the different OGC APIs and services.

Table 1. TIEs with GNOSIS Cartographer as a client

GNOSIS Cartographer	**GNOSIS**	**pygeoapi**	**rasdaman**	**MiraMon**	**TerraNexus**	**javaPS**	**CubeSERV**
Tiles	**Success**		**Success via WCPS adapter**	Progress		**Success via MOAW adapter**	Progress
Maps	**Success**		**Success via WCPS adapter**	Progress		**Success via MOAW adapter**	**Success**
Features	**Success**				**Success**	**Success via MOAW adapter**	**Success**
Coverages	**Success**	**Success**	**Success**			**Success via MOAW adapter**	**Success**
Processes / Workflows	**Success**	**Success**	**Success via WCPS adapter**		Progress	**Success via MOAW adapter**	Progress

Table 2. TIEs with GDAL / OGR (QGIS) as a client

GDAL / OGR (QGIS)	**GNOSIS**	**pygeoapi**	**rasdaman**	**MiraMon**	**TerraNexus**	**javaPS**	**CubeSERV**
Tiles	**Success**		**Success via WCPS adapter**	Progress		**Success via MOAW adapter**	Not tested
Maps	**Success**		**Success via WCPS adapter**	Progress		**MOAW adapter**	Supported but not tested

GDAL / OGR (QGIS)	GNOSIS	pygeoapi	rasdaman	MiraMon	TerraNexus	javaPS	CubeSERV
Features	**Success**				**Success**	**Success via MOAW adapter**	Supported but not tested
Coverages	**Success**	**Success**	**Success**			**Success via MOAW adapter**	Supported but not tested
Processes / Workflows	**Success**	**Success**	**Success via WCPS adapter**		Progress	**Success via MOAW adapter**	Progress

Additionally, the **MiraMon** client was successfully tested with the Ecere and CubeSERV servers for *OGC API - Maps & Tiles*.

12.2. Results and screenshots

In addition to these demonstrated scenarios, the project collaborators performed several additional Technology Integration Experiments where visualization clients accessed services providing processing and data delivery capabilities, including components developed by different organizations, as well as services acting as clients when cascading to upstream services.

12.2.1. Workflows Extension TIEs

GNOSIS Map Server

Ecere developed support for the Workflows extension in its GNOSIS Map Server, and successfully tested execution of various processes within both its GNOSIS Cartographer client as well as within QGIS using the new OGC API driver. The newly developed processing capabilities supporting workflows included its OpenStreetMap routing engine, rendering raster and vector layers as maps, generation of elevation contours, extracting elevation from point clouds, turning point clouds into digital elevation models and ortho photos, crop classification as well as the adapters for WCPS and Processes - Core.

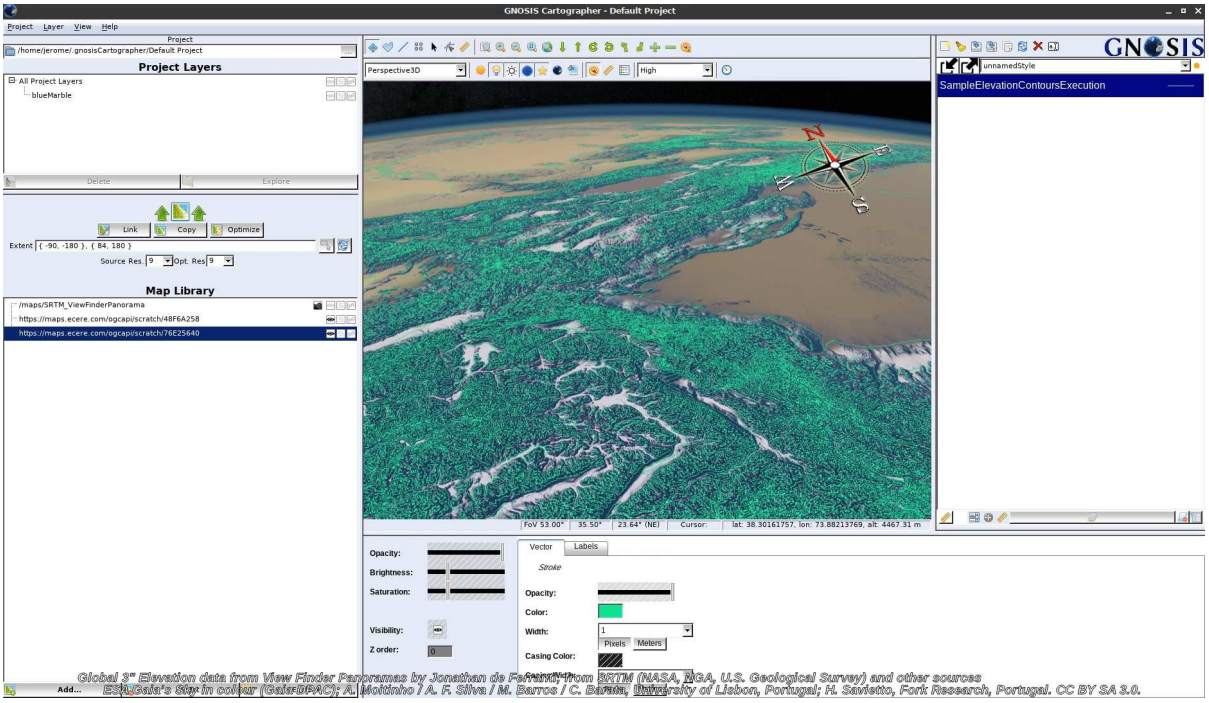

Figure 54. Elevation contours and map generated on-the-fly from elevation model via Workflows extensions, executed and accessed per tile on the GNOSIS Map Server

pygeoapi

EOX [https://eox.at] successfully implemented the Workflows extension in its server integrated within Euro Data Cube (EDC) [https://eurodatacube.com], leveraging the pygeoapi [https://pygeoapi.io] server software and the JupyterLab environment to execute processing requests on coverage data. Clients can write custom Python functions as part of the workflow execution request to define the values of bands for the resulting coverage, which could be sourced from one or more data sources. Please see pygeoapi for further details.

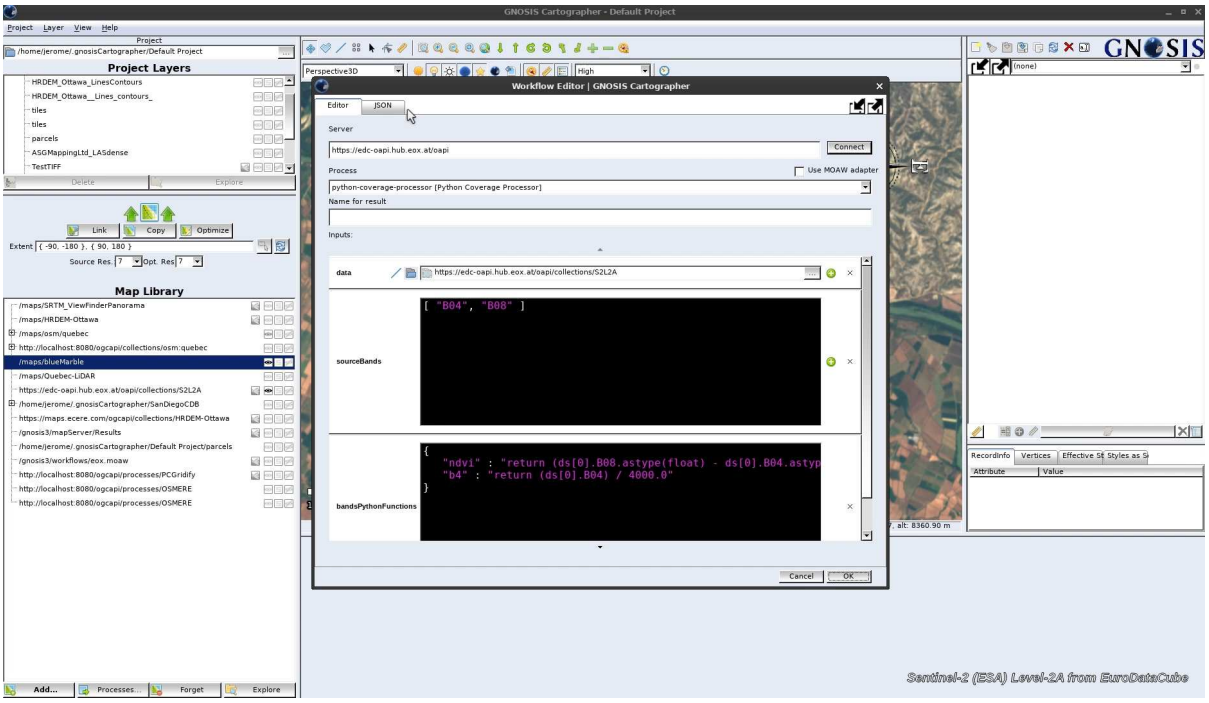

Figure 55. Setting up a workflow for Python Coverage Processor on EOX/pygeoapi service in GNOSIS Cartographer

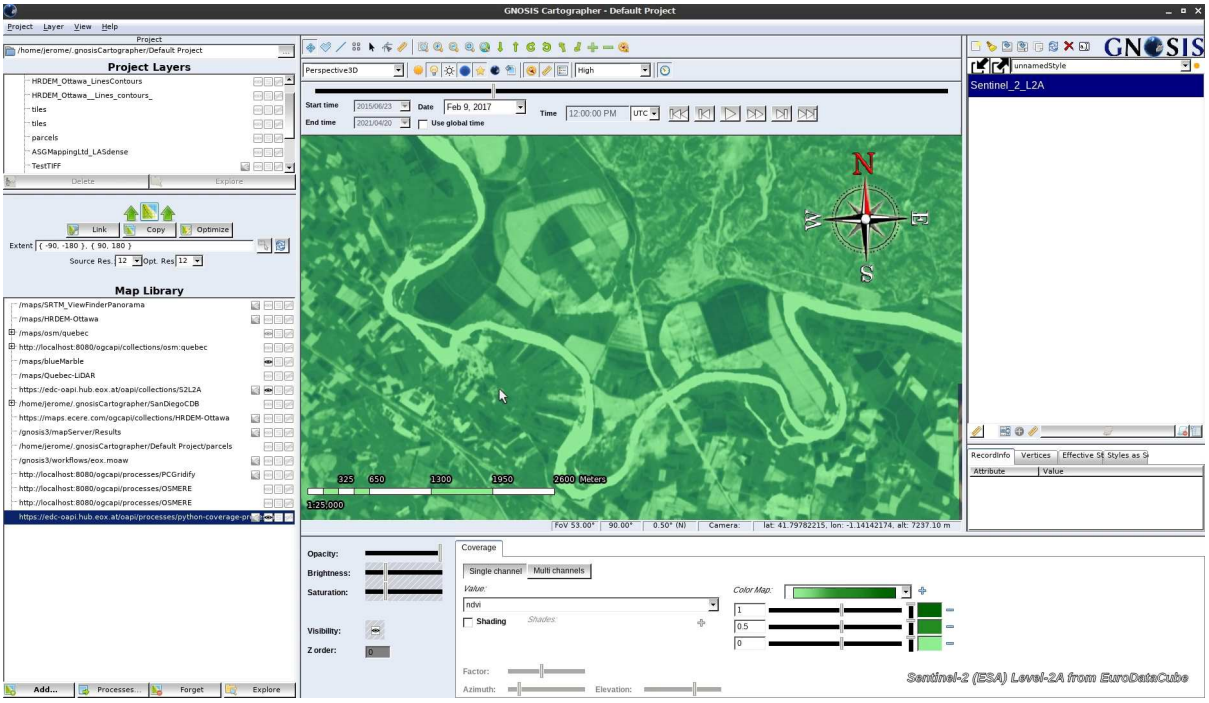

Figure 56. Visualizing results of the Python Coverage Processor on EOX/pygeoapi, using Workflows extension, in GNOSIS Cartographer

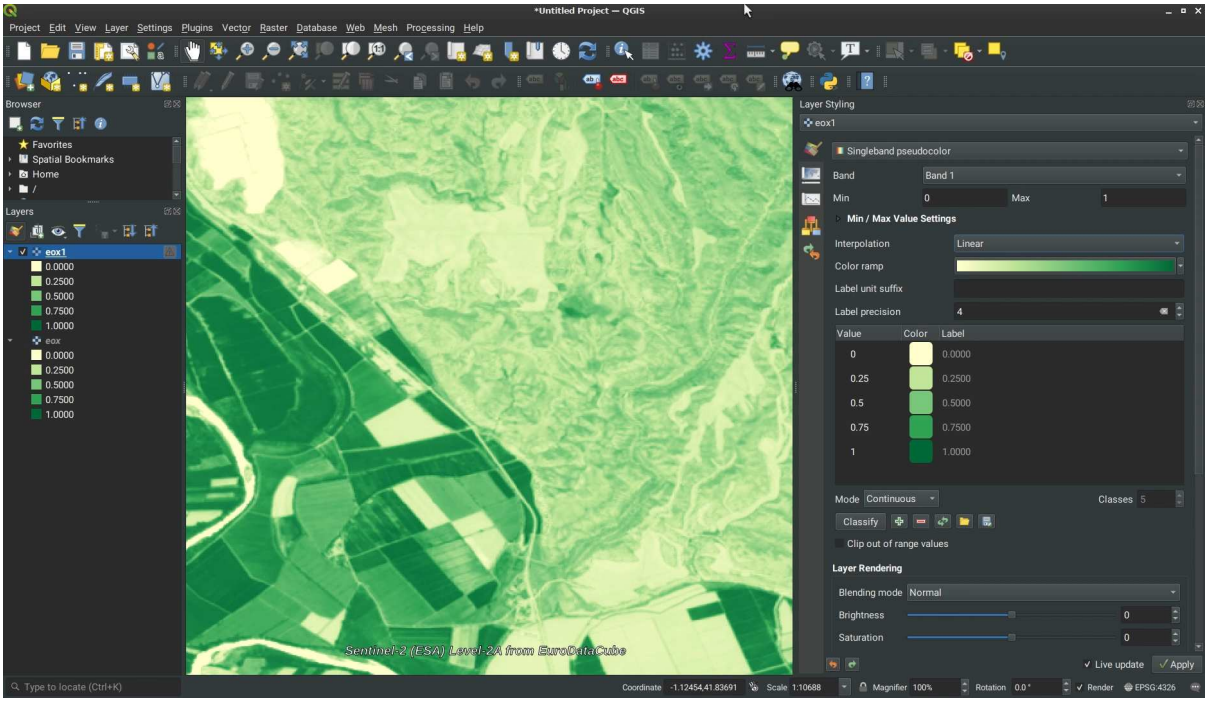

Figure 57. Visualizing results of the Python Coverage Processor on EOX/pygeoapi, using Workflows extension, in QGIS

CubeWerx

CubeWerx initiated the development of support for the Workflows extension in its CubeSERV server product.

TerraNexus

Pangaea Innovations initiated the development of support for the Workflows extension in its TerraNexus server.

12.2.2. TIEs Using Workflows/Core Adapter

As some of the server providers did not have enough resources allocated to the project to implement support for the Workflows extension, Ecere developed a couple of Adapter processes enabling the use of these services as part of Workflows through the use of these processes. One such process allowed the use of 52 North's javaPS OGC API - Processes - Core implementation.

javaPS

Experiments involving the javaPS server focused on two processes in particular. The first was a routing engine developed for the OGC Open Routing Pilot, which in one flavor also involved OSMERE, Ecere's OpenStreetMap Routing Engine, as a route calculation backend. The second was a process deployed using an Application Deployment Execution System, or ADES*, to compute a Normalized Difference Vegetation Index.

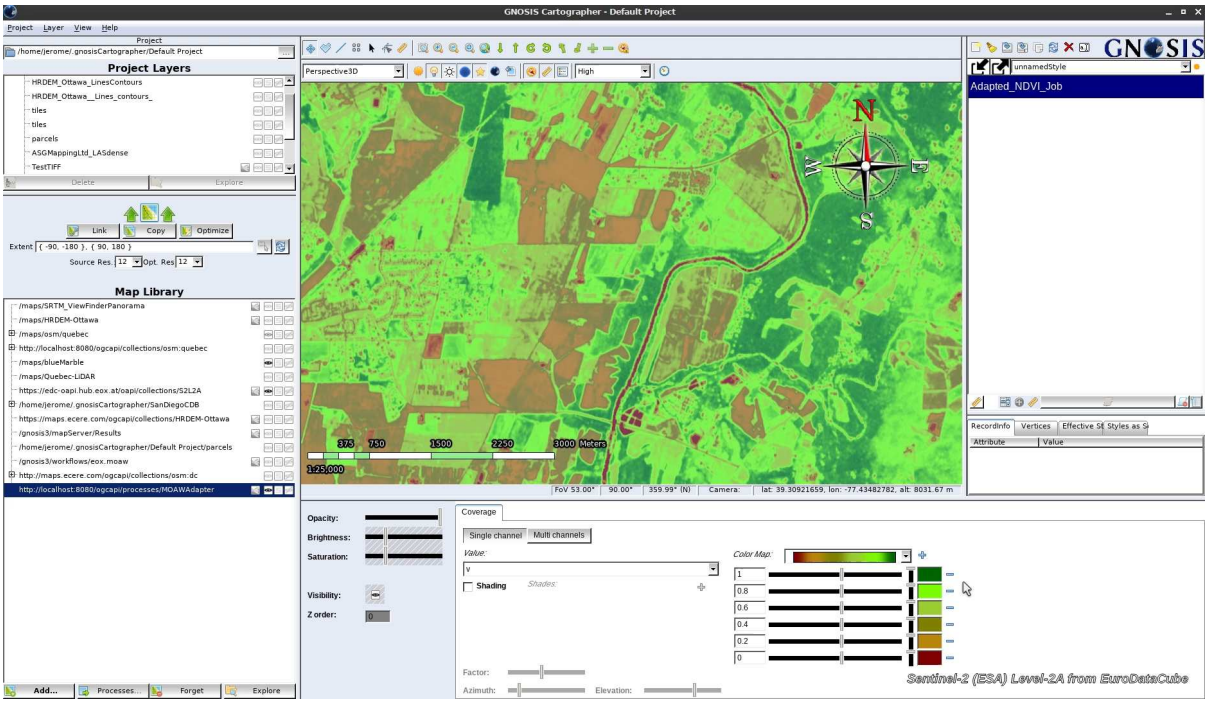

Figure 58. NDVI calculation performed in javaPS on sentinel-2 imagery retrieved from EOX's pygeoapi service via the GNOSIS Map Server Workflows/Core adapter visualized in GNOSIS Cartographer

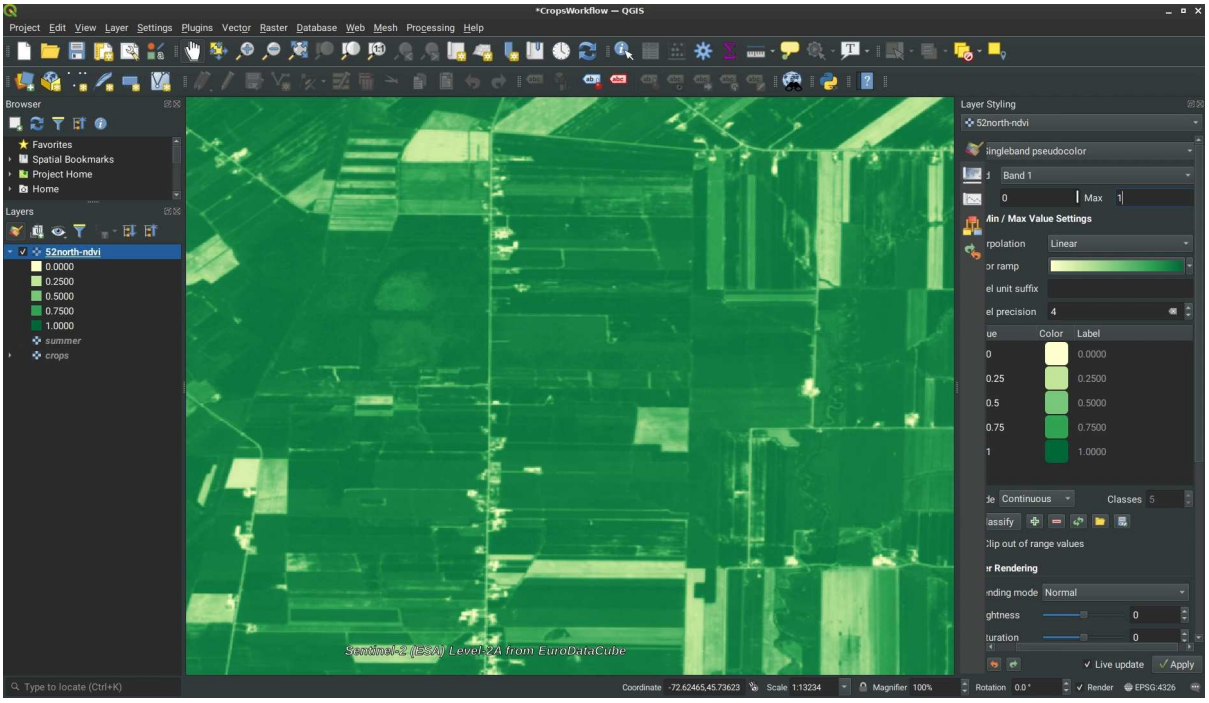

Figure 59. NDVI calculation performed in javaPS on sentinel-2 imagery retrieved from EOX's pygeoapi service via the GNOSIS Map Server Workflows/Core adapter visualized in QGIS

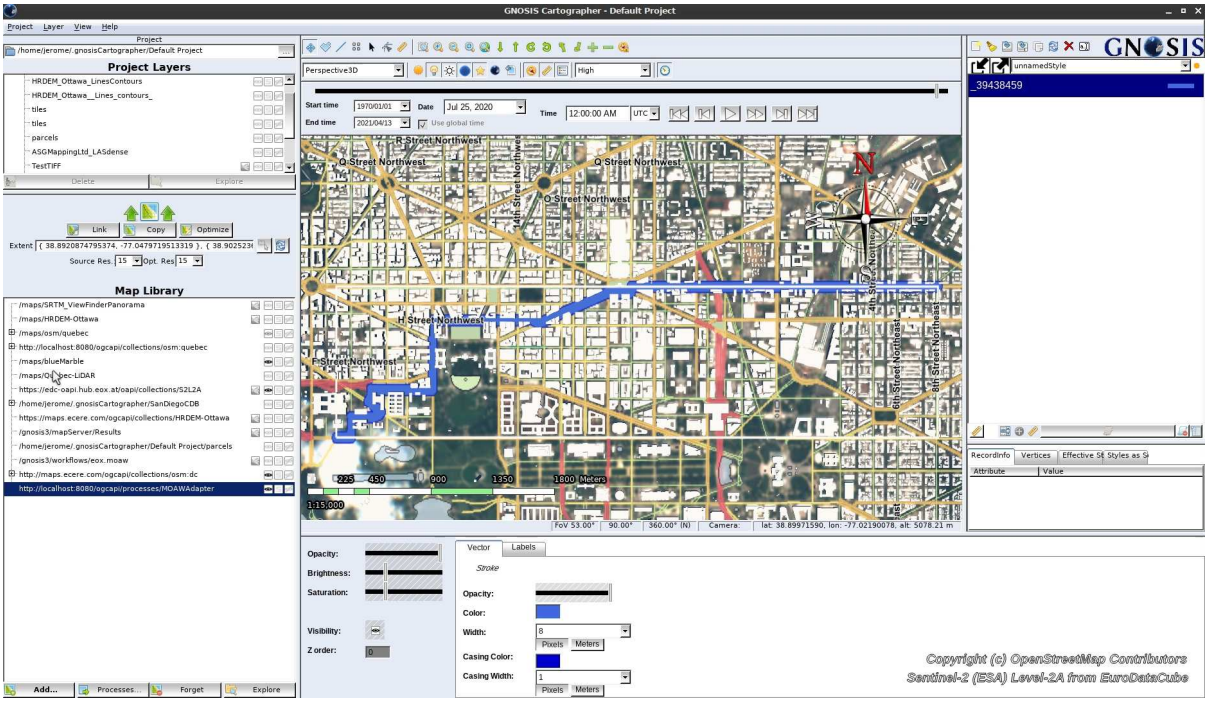

Figure 60. Route calculation performed in javaPS, leveraging Ecere's routing engine, accessed via the GNOSIS Map Server Workflows/Core adapter with results visualized in GNOSIS Cartographer

pygeoapi

In addition to directly executing processes using the *Workflows* extension on the pygeoapi service deployed by EOX, other experiments also used the Workflows/Core adapter.

12.2.3. TIEs Using Workflows/WCPS Adapter

Another adapter process targeted the Web Coverage Processing Service (WCPS) as implemented by the rasdaman OGC API service.

rasdaman

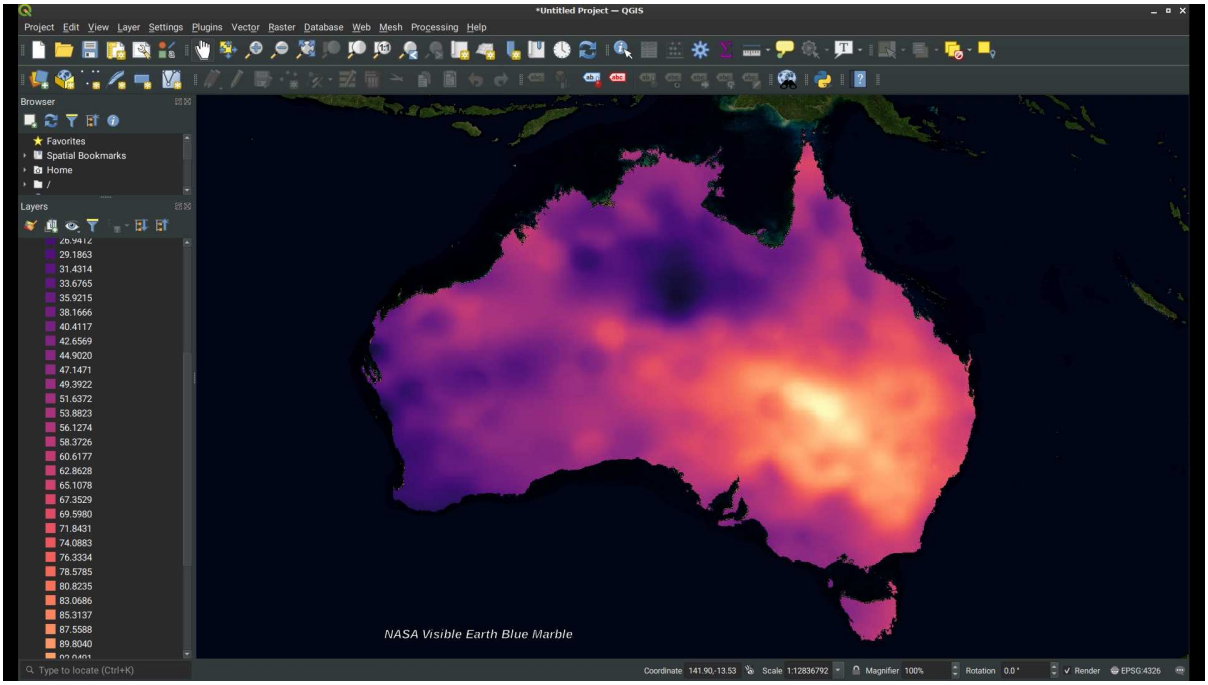

Figure 61. Coverage processing using a WCPS script executed using Workflows, via the GNOSIS Map Server WCPS adapter, visualized in QGIS

12.2.4. Cascaded services TIEs

The following TIEs involved more than a single service to fulfill requests:

- The WCPS adapter implemented on the GNOSIS Map Server, cascaded requests to rasdaman's service implementing WCPS.
- The *Processes - Core* adapter implemented on the GNOSIS Map Server cascaded requests to a the javaPS or pygeoapi service implementing Processes Core.
- The *Processes - Core* adapter directs the javaPS service to cascade to the pygeoapi service implementing *OGC API - Coverages* to retrieve sentinel-2 data.
- The version of the javaPS routing process used invoked Ecere's OSMERE routing engine to perform the actual route calculations.
- The crop classification process on the GNOSIS Map Server cascaded to pygeoapi using *OGC API - Coverages* to request data.
- The map rendering process on the GNOSIS Map Server rendered maps including data sources retrieved from other services via OGC API - Tiles, but would also *OGC API - Maps, Coverages or Features*.
- The contours rendering process on the *GNOSIS Map Server* could also be used to generate contours sourced from other services supporting *OGC API - Coverages*.
- The TerraNexus server cascaded to the *GNOSIS Map Server* to retrieve buildings, roads and elevation contours as part of the flood scenario workflow.

Although not involving a separate service, routes were calculated using the routing process on the *GNOSIS Map Server* with a nested process generating an elevation model from a point cloud for building a road network including elevation informatino.

12.2.5. OGC API - Coverages TIEs

Ecere, EOX and rasdaman developed support for the draft OGC API - Coverages specifications in their services and successfully achieved interoperability with the GNOSIS Cartographer and GDAL clients as part of the project.

GNOSIS Map Server

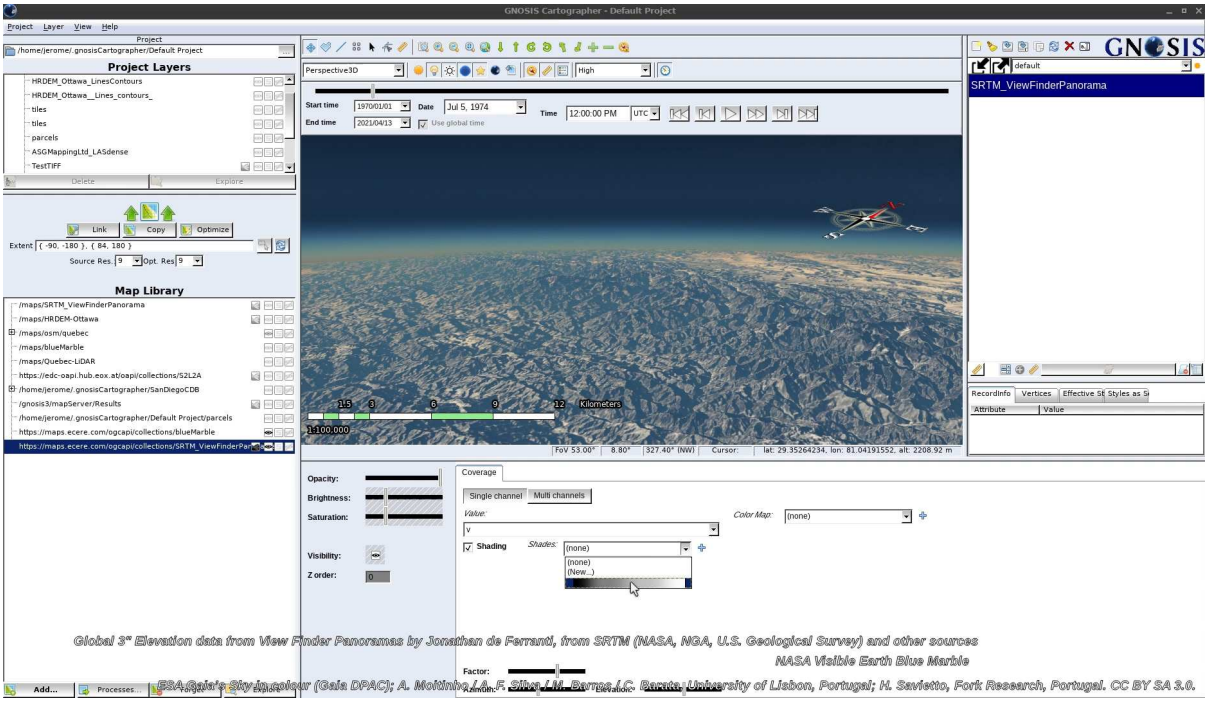

Figure 62. Global elevation data retrieved from the GNOSIS Map Server using OGC API - Coverages / Tiles, visualized in GNOSIS Cartographer

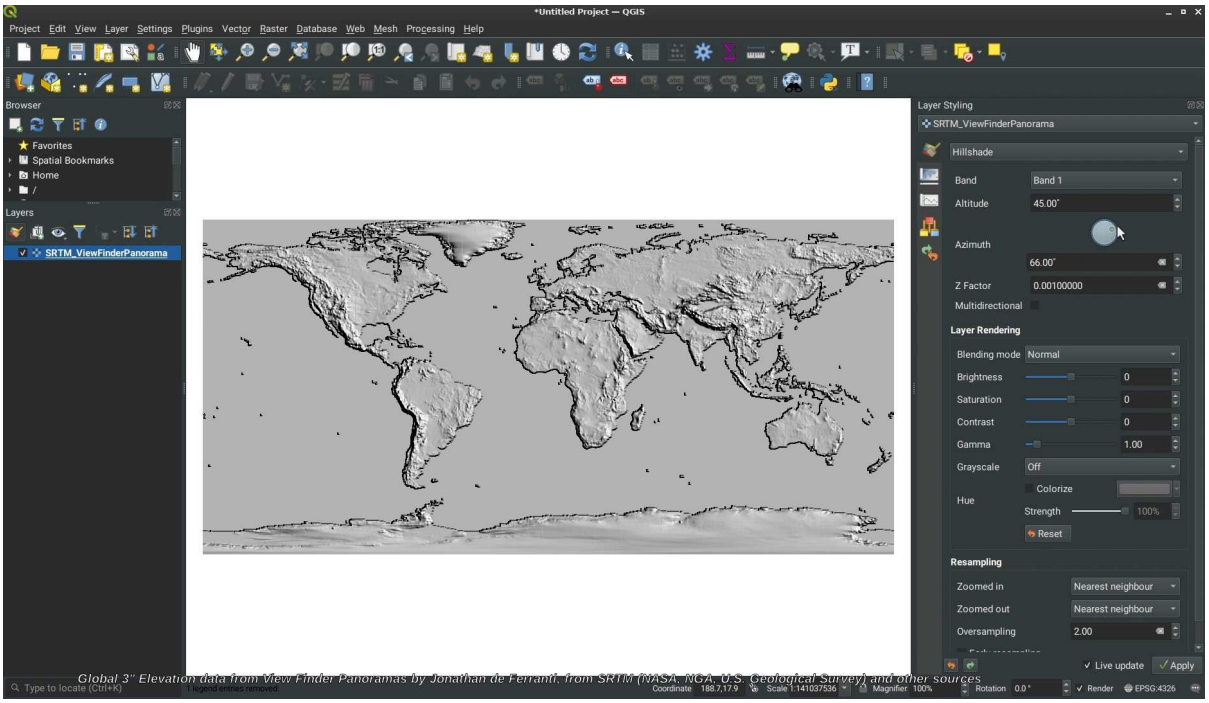

Figure 63. Global elevation data retrieved from the GNOSIS Map Server using OGC API - Coverages, visualized in QGIS

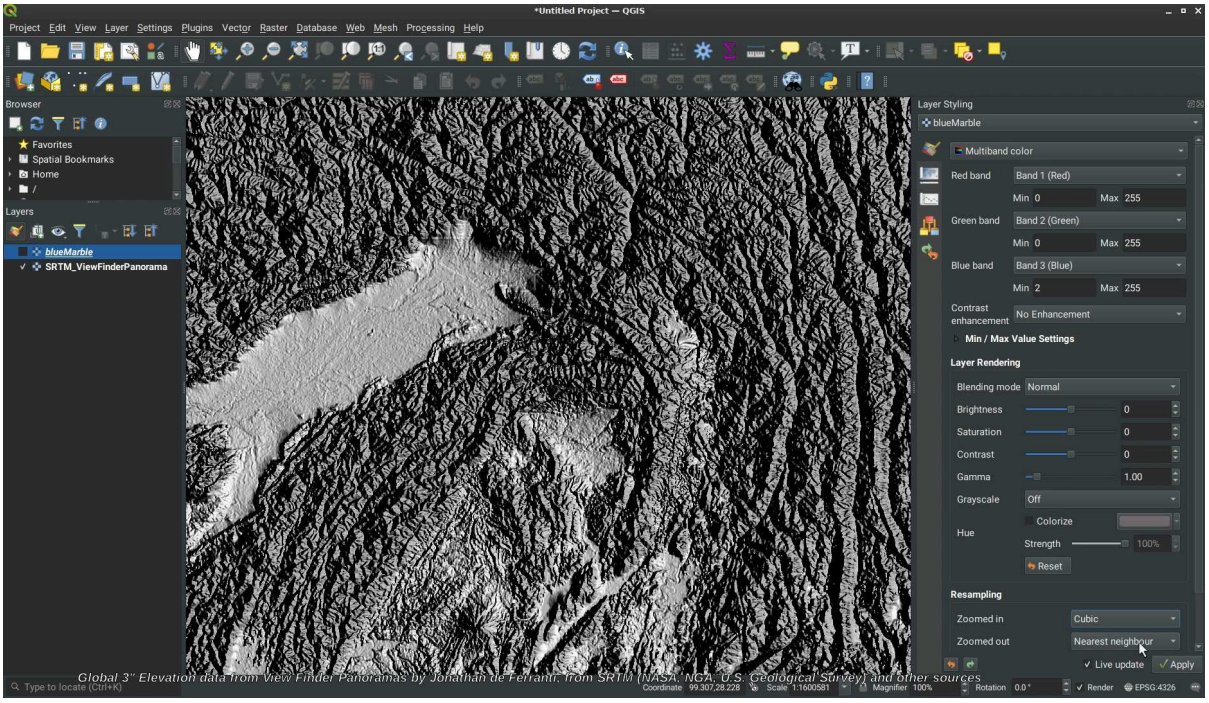

Figure 64. Global elevation data retrieved from the GNOSIS Map Server using OGC API - Coverages, visualized in QGIS (zoomed in)

pygeoapi

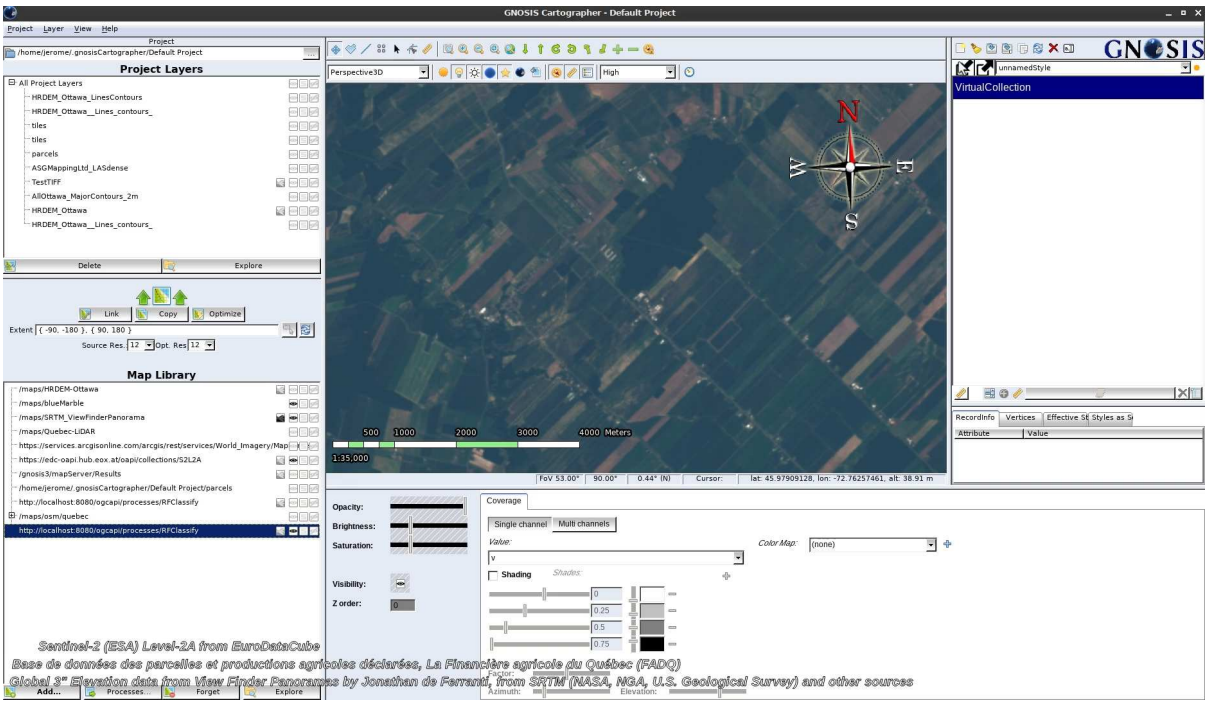

Figure 65. Sentinel-2 imagery retrieved from EOX's deployed pygeoapi service using OGC API - Coverages, visualized in QGIS

rasdaman

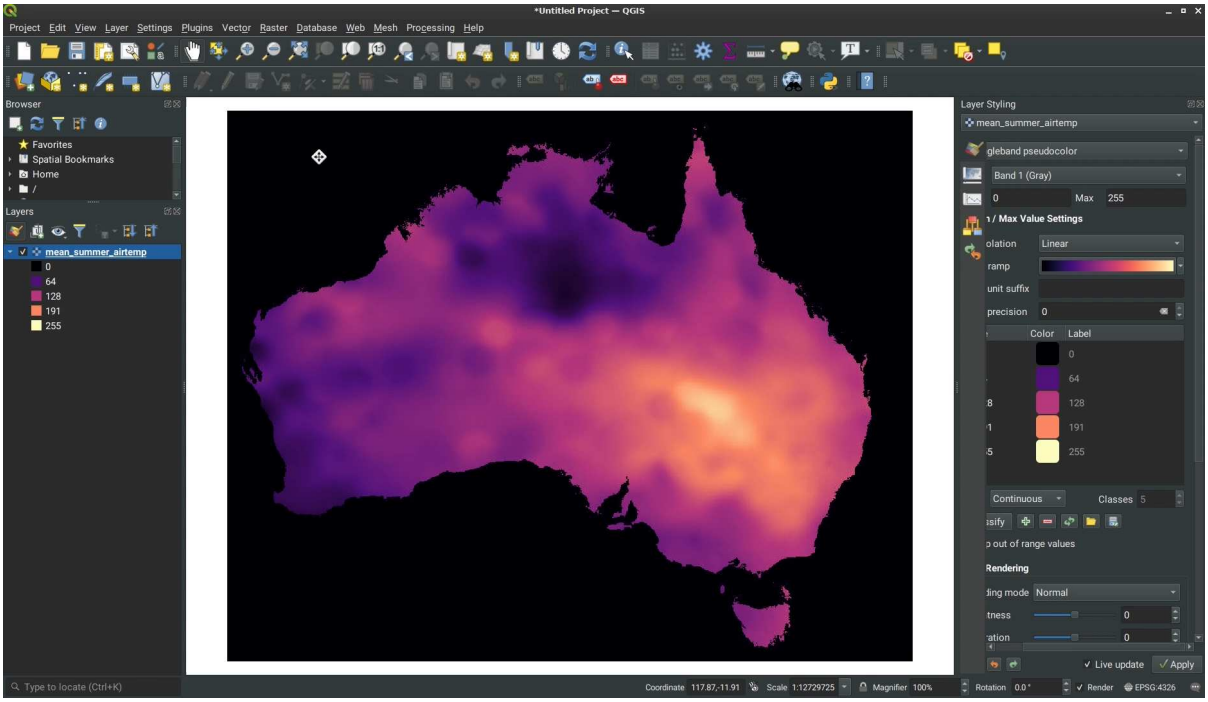

Figure 66. Mean air summer temperature coverage retrieved from rasdaman using OGC API - Coverages, visualized in QGIS

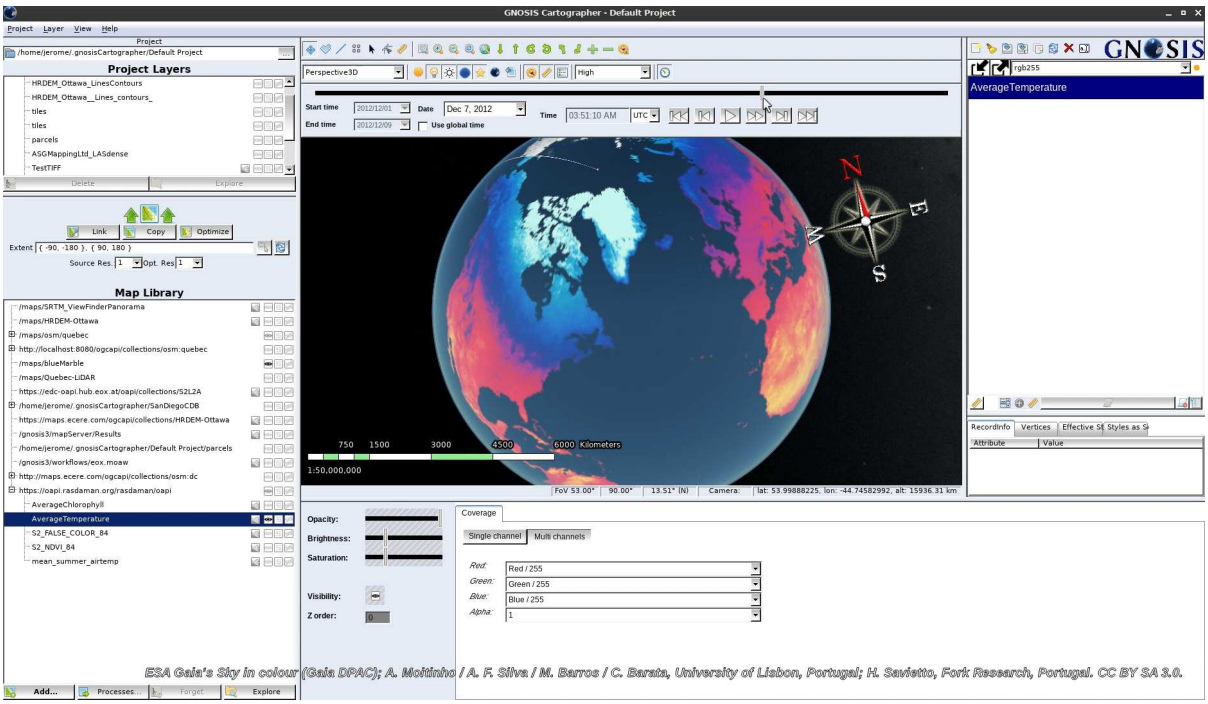

Figure 67. Global average monthly temperatures retrieved from rasdaman using OGC API - Coverages, visualized in GNOSIS Cartographer

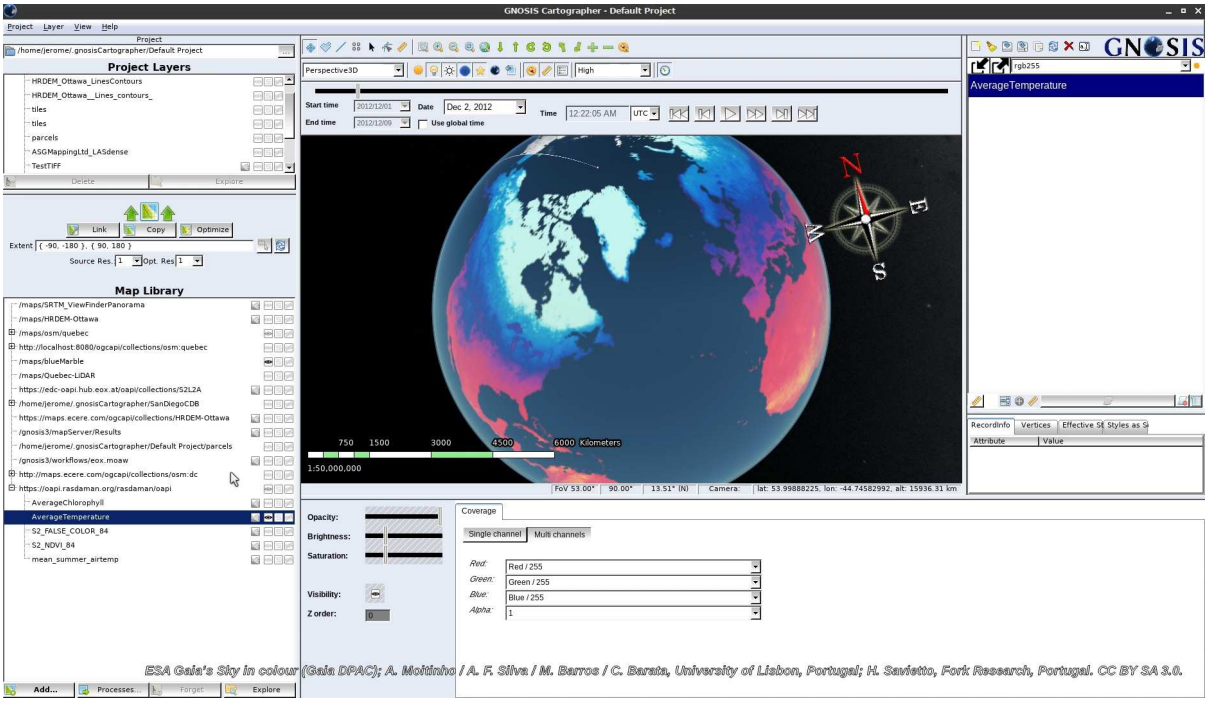

Figure 68. Global average monthly temperatures retrieved from rasdaman using OGC API - Coverages, visualized in GNOSIS Cartographer

CubeSERV

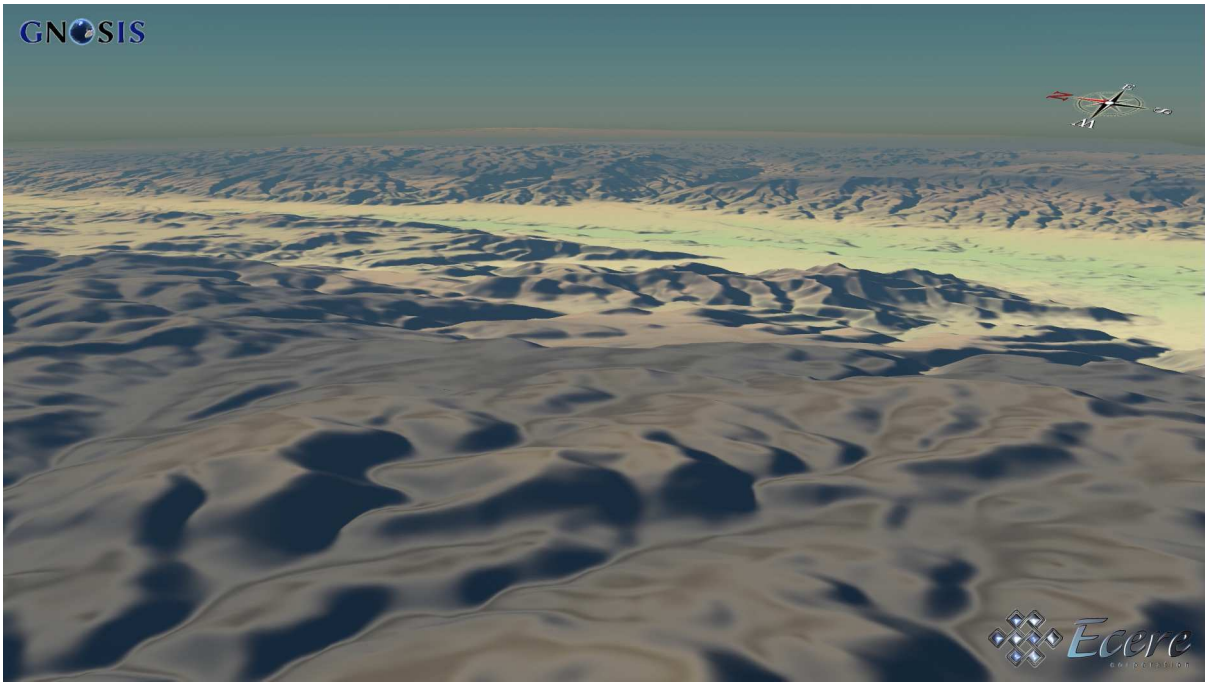

Figure 69. Elevation coverage retrieved from CubeWerx's CubeSERV using OGC API - Coverages, visualized in GNOSIS Cartographer

12.2.6. OGC API - Features TIEs

During the initiative, Ecere and Pangaea Innovations both greatly improved the reliability, performance and correctness of their implementation of OGC API - Features and successfully passed the compliance tests to become certified implementations of the standard.

GNOSIS Map Server

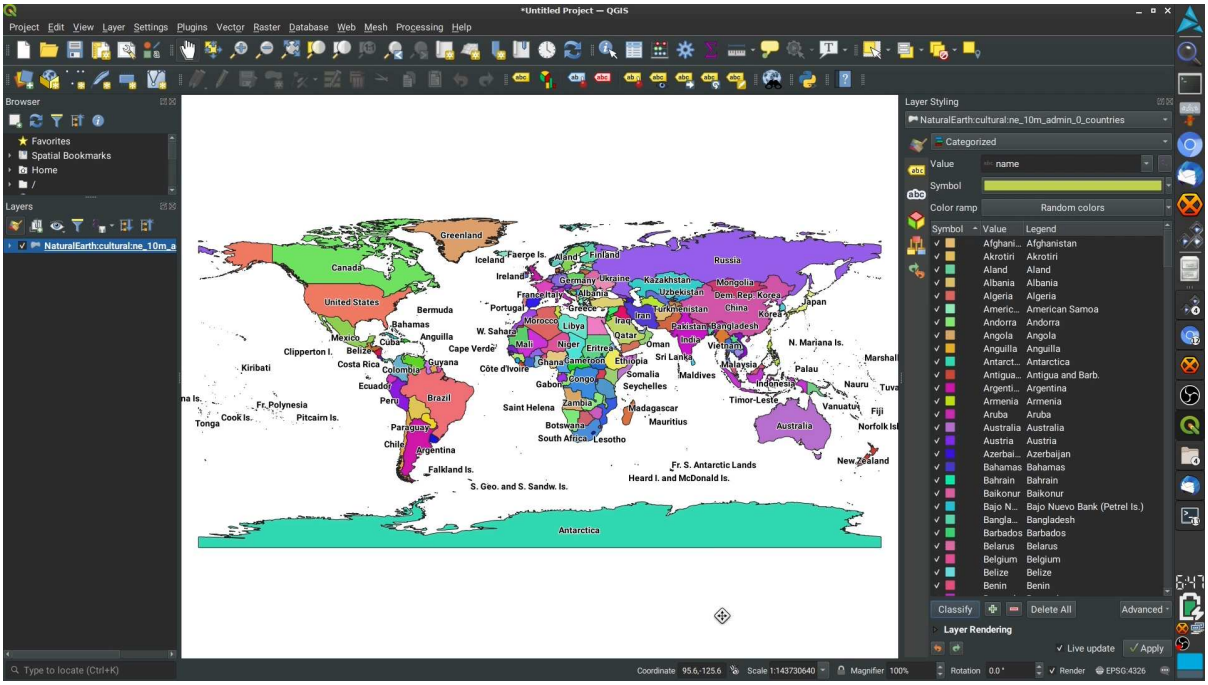

Figure 70. World Countries from Natural Earth retrieved from GNOSIS Map Server using OGC API - Features, visualized in QGIS

TerraNexus

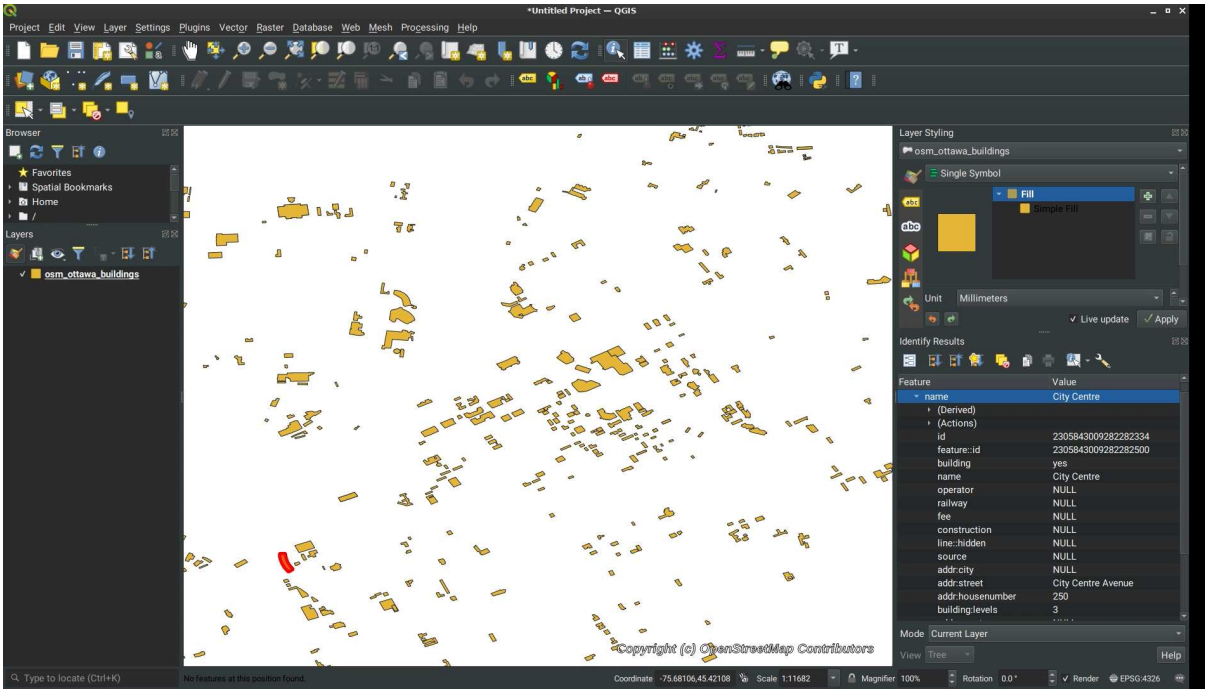

Figure 71. OpenStreetMap buildings retrieved from TerraNexus server using OGC API - Features, visualized in QGIS

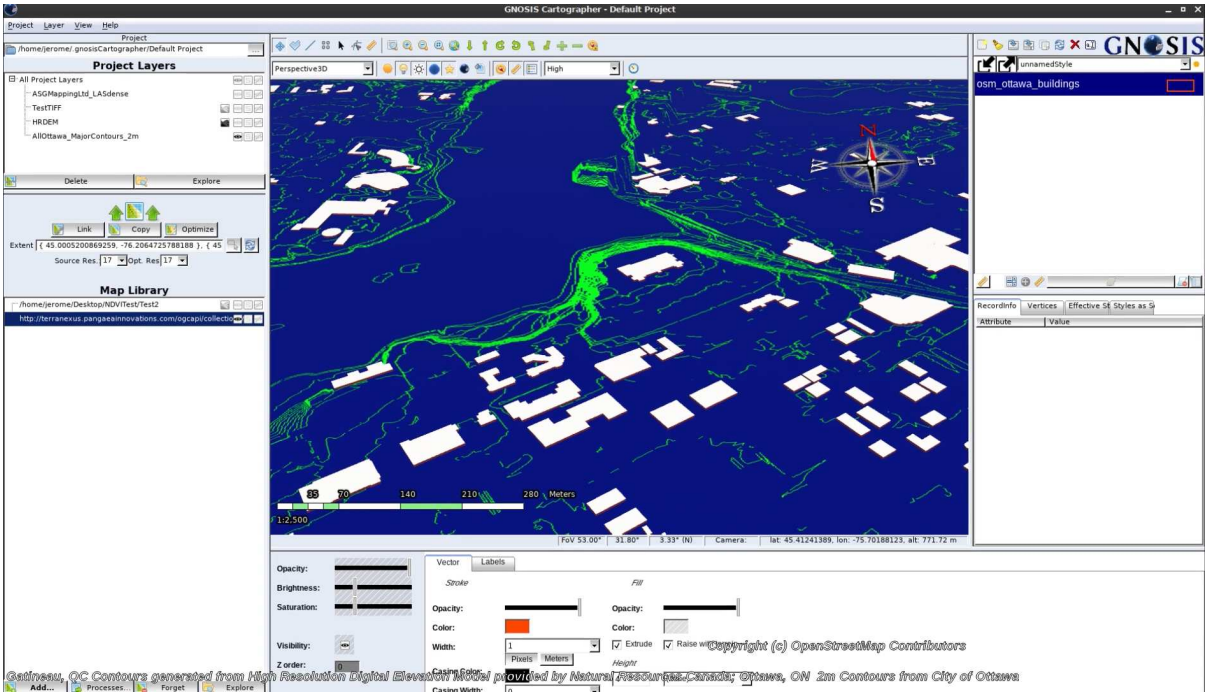

Figure 72. OpenStreetMap buildings retrieved from TerraNexus server using OGC API - Features, visualized in GNOSIS Cartographer

12.2.7. OGC API - DGGS TIEs

TerraNexus

12.2.8. OGC API - Maps & Tiles TIEs

GNOSIS Map Server

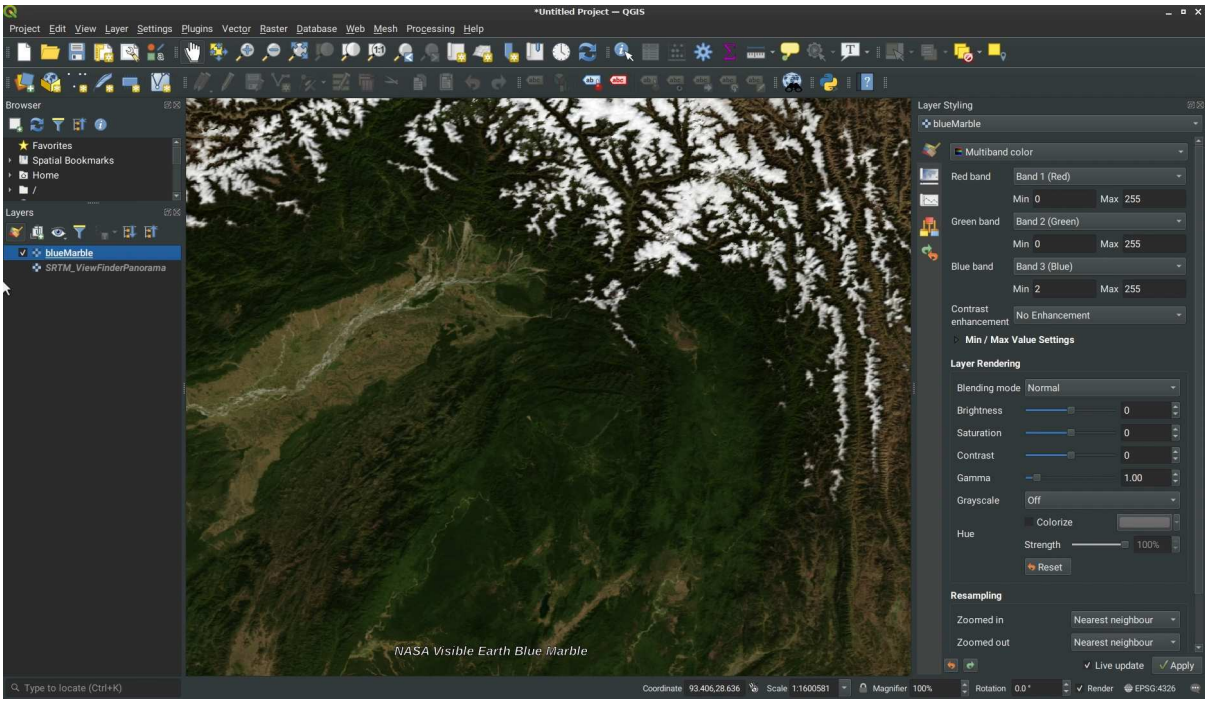

Figure 73. NASA VisibleEarth Blue Marble retrieved from GNOSIS Map Server using OGC API - Map & Tiles, visualized in QGIS

CubeSERV

Ecere also achieved success in visualizing data using the Maps, Coverages and Features API from the CubeServ service. Miramon sucessfully visualized data from the CubeSERV service using *OGC API - Maps*.

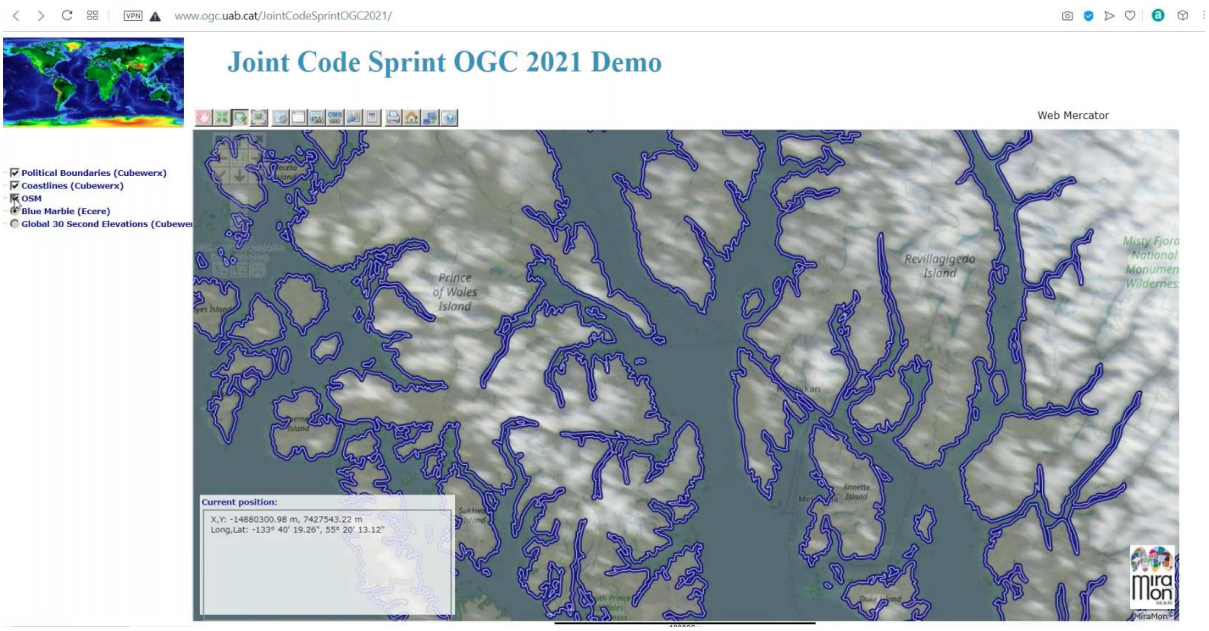

Figure 74. Map tiles retrieved from CubeWerx's CubeSERV visualized in MiraMon client, together with NASA VisibleEarth Blue Marble also retrieved as map tiles from GNOSIS Map Server

Chapter 13. Project contributions to OGC API specifications

This chapter details the contributions to OGC API specifications made by participants in the context of this project, including *Processes*, *Common*, *Coverages*, *Tiles*, *Maps*, *Styles* and *DGGS*.

13.1. Processes

13.1.1. Part 1: Core

The team insisted on taking the opportunity of a fresh start to ensure the new specifications being defined are simple, elegant and straightforward, sometimes benefiting from taking a new perspective untainted by biases from the WxS specifications and their associated XML encodings, which preceded these new OGC API specifications. Several improvements to the Processes specifications in particular were suggested and approved, as clean and simple execution requests was considered key to facilitate managing the definition of complex workflows.

13.1.2. Part 3: Workflows

The most significant contribution of this project to open standards is the creation of the new Part 3 extension to OGC API - Processes for Workflows and Chaining, as well as its inclusion in the Charter of the OGC API - Processes Standard Working Group as a work item to further develop and standardize.

However, this extension is relatively simple and relies heavily on the capabilities defined in all other OGC API specifications, providing only the means to connect these building blocks in an interoperable manner.

As such, collaborators contributed significantly to the development of these other OGC API specifications, in particular to OGC API - Common, Coverages, Tiles, Maps, Processes and DGGS, and to a lesser extent to OGC API - Features, Records, Styles, Routes as well as to future specifications for accessing Bounding Volume Hierarchy of 3D data, such as 3D Tiles and i3s, currently referred to as the GeoVolumes API.

13.2. Common

An important aspect of these contributions focused on advocating to provide as much consistency as possible across the OGC API family of standards.

13.2.1. Part 1: Core

The Core of the OGC API - Common standard provides an entry point into an OGC API as well as a mechanism to declare conformance classes and define the API. Collaborators participated in the development of the specification throughout hte project, and highlighted the importance to account for proper service and dataset metadata.

13.2.2. Part 2: Geospatial Data

A multi-year effort to convince the OGC membership of the importance of the concept of a generic collection of geospatial data, which may be accessed using one or more OGC API access mechanism, culminated in a resolution to define what is now known as the OGC API - Common - Part 2 - Geospatial Data draft specifications.

This same OGC API collection concept is a data resource which can be accessed using the OGC API - Features, Coverages, Tiles, Maps, EDR and GeoVolumes (Bounding Volume Hierachies of 3D data). It is also central to the Workflows extensions, bridging all these APIs with Processes by allowing processes to both use collections as inputs, as well as to produce virtual collections as outputs.

13.3. Coverages

Collaborators participated in the development of the Coverages API specifications, presenting a simplified approach for the resources to be implemented and encouraging alignment with the other OGC API specifications, while suggesting the importance of including support for subsetting, scaling and tiles conformance classse.

13.4. Tiles

Project participants suggested a better organization of resources and conformance class for the Tiles API centered around the a dedicated resource for tilesets with proper metadata, informed from the results of the second OGC Vector Tiles Pilot initiative.

13.5. Maps and Styles

Collaborators suggested a reorganization and simplification of the map resources, where the listing of styles is leveraged from the Styles API and a map resource can be attached to a particular style, as it can be attached to a collection or a dataset.

13.6. Discrete Global Grid Systems

Collaborators proposed two basic approaches to services powered by Discrete Global Grid Systems: on one hand they can power APIs such as Features and Coverages, improving performance and facilitating data integration, while on the other a dedicated API can be provided for clients and cascading services understanding the same DGGS. Participants highlighted that making resources and operations DGGS-specific only when required facilitates integration with components not supporting DGGS, or supporting a different DGGS.

13.7. Other OGC Standards

Additionally, collaborators contributed to other important OGC API specifications providing a foundation for these APIs, such as the OGC 2D Tile Matrix Set and TileSet Metadata Standard, powering the Tiles API, the OGC Coverage Implementation Schema providing the foundation for Coverages, as well as OGC GeoPackage, an interoperable geospatial data format and container, and

OGC CDB, specifications for storing very detailed 3D content for use cases such as a data repository, training and simulation.

Figure 75. San Diego CDB provided by CAE for OGC Interoperable Simulation & Gaming sprint visualized in GNOSIS Cartographer

Chapter 14. Project contributions to Free and Open Source Software

This chapter details the contributions to Free and Open Source Software made by participants in the context of this project.

The collaborators made several important contributions to Free and Open Source Software as part of this project. Software projects for which code was directly contributed included GDAL, QGIS, pygeoapi, javaPS, rasdaman, as well as the Ecere Cross-Platform SDK and eC programming language.

14.1. GDAL & QGIS

Contributions to GDAL and QGIS concerned the development of a new unified driver for the OGC API family of standards, with new support for:

- OGC API - Tiles (for both raster & vector tiles, in Mapbox Vector Tile or GeoJSON format)
- OGC API - Maps
- OGC API - Coverages

as well as the ability to execute processing workflows and retrieve their results in any of those APIs.

14.2. pygeoapi

EOX implemented improved support for OGC API - Coverages and Processes in pygeoapi and implemented support for the Workflows extensions as well.

The server instance provided by EOX uses the software pygeoapi to provide the API interfaces.

Several improvements and fixes have been submitted to pygeoapi directly.

The MOAW specific implementations are currently in the main branch of the EDC pygeoapi fork [https://github.com/eurodatacube/pygeoapi].

To view the actual changes compare this branch with upstream [https://github.com/geopython/pygeoapi/compare/master%2E%2E%2Eeurodatacube:main].

Depending on the adoption of the MOAW extensions in OAPI - Processes these changes will be adapted and potentially submitted for inclusion in the official pygeoapi repository.

As described in the section pygeoapi, processes are run in a kubernetes cluster as native kubernetes jobs. Each job is actually an execution of the corresponding Jupyter notebook with the invocation parameters via papermill. The software extending pygeoapi with a manager to run kubernetes jobs executing notebooks via papermill is released under MIT license at https://github.com/eurodatacube/pygeoapi-kubernetes-papermill.

The data access is using concepts from the forthcoming OAPI - Coverages specification. The code is

currently in the oapi branch of the ogc-edc repository [https://github.com/eurodatacube/ogc-edc] also available under MIT license.

14.3. rasdaman

rasdaman implemented support for the OGC API - Coverages in its service, as a façade on top of WCS, the Web Coverage Service.

14.4. javaPS

52 North improved its javaPS OGC API - Processes implementation

14.5. Ecere cross-platform SDK

Ecere made several improvements to its general purpose cross-platform Ecere Software Development Kit, including bug fixes, improvements to its eC programming language and compiler, its Integrated Development Environment, its automated bindings generator for C, C++ and Python, its Cross-platform Graphical User Interface, optimizing its 3D rendering engine, and improving the eC Cascading Style Sheets forming the basis of the Cascading Map Style Sheets (CMSS) for defining styling and symbology.

Chapter 15. Future Work

This chapter recommends next steps which could be taken, building upon the achievements of this project, in the OGC Innovation and Standards program, by participants, the OGC membership, as well as additional stakeholders who could benefit from leveraging modular workflows.

Although the collaborators achieved many successful Technology Integration Experiments, and produced very concrete results in the form of specifications for defining Workflows for processing and visualization and improving related OGC API specifications, they consider this success to be only the starting point for exploring the potential that this modular approach to OGC API Workflows has to offer.

15.1. OGC Innovation & Standards Program

The participating organizations recommend that additional steps be taken by both the OGC Standards and Innovation Program to dive deeper into this approach in the form of additional interoperability experiments such as Code Sprints, as well as Testbeds for which Workflows may provide an interesting opportunity to connect the work of different work packages focusing on different OGC API data delivery and/or processing specifications.

15.2. Participants

The OGC API - Processes Standard Working Group will continue the development of the Part 3 - Workflows and Chaining extension, for which two participants in this project volunteered as editors of the specifications. The project's recommendation is to continue the standardization effort with an objective to make it an official OGC standard.

15.3. Other stakeholders

The project succesfully demonstrated the viability of the approach and participating organizations encourage stakeholders interested in high performance client-driven distributed workflows to support the efforts of the SWG, the OGC as well as present and future collaborators on this road to standardization, experimentation, implementation and adoption of Modular OGC API Workflows.

Annex A: Project Schedule & Milestones

The project schedule was divided in four (4) Milestones with the following objectives:

Milestone 1: Specifications, servers & 3D (June 1st – July 31st 2020)

Key outputs:

- Developing API Specifications for complex distributed chained processing workflows
- Initial success executing these workflows in processing and data delivery services
- Improved 3D visualization capabilities

Milestone 2: Clients & initial integration (August 1st – October 15th 2020)

Key outputs:

- Workflow capable visualization clients
- Workflow editing capabilities
- Successful Technology Integration Experiments

Milestone 3: Evaluation & improvements (October 16th - January 15th, 2021)

Key outputs:

- Improved clients & services
- additional, more complex workflow execution TIEs
- resolved issues related to additional TIEs
- resolved issues encountered during evaluation of the clients and services by the researchers (INRS and Universite Laval)
- Improved user-friendliness of clients, e.g. straightforward data discovery and workflow editing
- Established Earth Observation & Smart City scenarios

Milestone 4: Application, demos and outreach (January 15th – March 31st, 2021)

Key outputs:

- Demonstration videos showcasing the capabilities
- Insights on future steps (both for OGC standards and collaborators' platforms)
- Publication detailing the project results (this document)

[[Workflow Execution Documents]]

Annex B: Workflows

B.1. WCPS Adapter

```
{
   "process" : "https://maps.ecere.com/ogcapi/processes/WCPSAdapter",
   "inputs" : {
      "wcps" : "https://ows.rasdaman.org/rasdaman/oapi/wcps",
      "data" : [
            { "collection" :
"https://ows.rasdaman.org/rasdaman/oapi/collections/Temperature4D" },
            { "collection" :
"https://ows.rasdaman.org/rasdaman/oapi/collections/RadianceColor" }
         ],
      "code" :
         "for $te in ( %{0} ),\n     $ra in ( %{1} )\n    let $et := %{timeRange},
$st := \"2012-05-01T00:00:00.000Z\", $ec := [Lat(%{latRange}), Long(%{lonRange})]
\nreturn\n  encode(\n  scale(\n    extend(\n      (\n        condense +\n        over
$t ansi(imageCrsDomain($ra, ansi))\n        using (($ra.0 + $ra.1 + $ra.2)[ansi($t)] *
(($te.1 + $te.2 + $te.0)[ansi($et),elev(100)]))\n      )[$ec]\n    ,\n    {
Lat(%{tileLat}), Long(%{tileLon}) }),\n  { Lat:\"CRS:1\"(1:%{pxHeight}), Long:\"
CRS:1\"(1:%{pxWidth}) } ),\n  %{encFormat})"
   }
}
```

B.2. MOAW Adapter / 52 North NDVI

```
{
   "process" : "https://maps.ecere.com/ogcapi/processes/MOAWAdapter",
   "inputs" : {
      "data" : {
         "process" :
"https://testbed.dev.52north.org/ades/rest/processes/org.n52.geotiff.ndvi",
         "inputs" : {
            "source" : { "collection" : "https://edc-
oapi.hub.eox.at/oapi/collections/S2L2A" },
            "red_source_band" : 4,
            "nir_source_band" : 8,
            "nir_factor" : 1.0,
            "red_factor" : 1.0
         }
      }
   }
}
```

B.3. MOAW Adapter / 52 North Routing

```
{
   "process" : "https://maps.ecere.com/ogcapi/processes/MOAWAdapter",
   "inputs" : {
      "data" : {
         "process" : "https://testbed.dev.52north.org/orp/ecere/processes/routing",
         "inputs" : {
            "waypoints" : { "value" : {
                  "type" : "MultiPoint",
                  "coordinates" : [
                     [-77.047712, 38.892346],
                     [-76.99473, 38.902629]
                  ]
               }
            }
         }
      }
   }
}
```

B.4. RenderMap

```
{
   "process" : "https://maps.ecere.com/ogcapi/processes/RenderMap",
   "inputs" : {
      "transparent" : true,
      "layers" : [
            { "collection" : "https://maps.ecere.com/ogcapi/collections/osm:dc" }
      ]
   }
}
```

B.5. EOX Python Coverage Processor

```
{
   "process" : "https://edc-oapi.hub.eox.at/oapi/processes/python-coverage-processor",
   "inputs" : {
      "bandsPythonFunctions" : {
         "value" : {
            "ndvi" : "return (ds[0].B08.astype(float) - ds[0].B04.astype(float)) /
(ds[0].B04.astype(float) + ds[0].B08.astype(float))",
            "b4" : "return (ds[0].B04) / 4000.0"
         }
      },
      "data" : [
         { "collection" : "https://edc-oapi.hub.eox.at/oapi/collections/S2L2A" }
      ],
      "sourceBands" : [
         [ "B04", "B08" ]
      ]
   }
}
```

B.6. Digital TerrainModel from Point Cloud

```
{
   "process" : "https://maps.ecere.com/ogcapi/processes/PCGridify",
   "inputs" : {
      "data" : { "collection" : "https://maps.ecere.com/ogcapi/collections/Quebec-
LiDAR" },
      "output" : "dem",
      "fillDistance" : 100,
      "classes" : [ "ground" ]
   }
}
```

B.7. Integrating Point Cloud Elevation into OSM Features

```
{
  "process" : "https://maps.ecere.com/ogcapi/processes/PointCloudElevation",
  "inputs" : {
    "features" : { "collection" :
"https://maps.ecere.com/ogcapi/collections/osm:quebec:buildings" },
    "pointCloud" : { "collection" : "https://maps.ecere.com/ogcapi/collections/Quebec-
LiDAR" }
  }
}
```

B.8. Routing Engine - with elevation model derived from point cloud

```
{
  "process" : "https://maps.ecere.com/ogcapi/processes/OSMERE",
  "inputs" : {
    "dataset" : { "collection" :
"https://maps.ecere.com/ogcapi/collections/osm:quebec:roads" },
    "elevationModel" :
    {
       "process" : "https://maps.ecere.com/ogcapi/processes/PCGridify",
       "inputs" : {
          "data" : { "collection" : "https://maps.ecere.com/ogcapi/collections/Quebec-
LiDAR" },
          "output" : "dem",
          "fillDistance" : 100,
          "classes" : [ "ground" ]
       }
    },
    "preference" : "shortest",
    "mode" : "pedestrian",
    "waypoints" : { "value" : {
        "type" : "MultiPoint",
        "coordinates" : [
          [ -71.20290940, 46.81266578 ],
          [ -71.20735275, 46.80701663 ]
        ]
      }
    }
  }
}
```

B.9. Routing Engine - Avoiding steep hills with elevation model derived from point cloud

```
{
  "process" : "https://maps.ecere.com/ogcapi/processes/OSMERE",
  "inputs" : {
    "dataset" : { "collection" :
"https://maps.ecere.com/ogcapi/collections/osm:quebec:roads" },
    "elevationModel" :
    {
       "process" : "https://maps.ecere.com/ogcapi/processes/PCGridify",
       "inputs" : {
          "data" : { "collection" : "https://maps.ecere.com/ogcapi/collections/Quebec-
LiDAR" },
          "output" : "dem",
          "fillDistance" : 100,
          "classes" : [ "ground" ]
       }
    },
    "preference" : "shortest",
    "maxUpSlope" : 17,
    "maxDownSlope" : 13,
    "mode" : "pedestrian",
    "waypoints" : { "value" : {
        "type" : "MultiPoint",
        "coordinates" : [
          [ -71.20290940, 46.81266578 ],
          [ -71.20735275, 46.80701663 ]
        ]
      }
    }
  }
}
```

B.10. Crops Classification with Random Forest Classifier

```
{
   "process" : "https://maps.ecere.com/ogcapi/processes/RFClassify",
   "inputs" : {
      "data" : { "collection" : "https://edc-oapi.hub.eox.at/oapi/collections/S2L2A" }
   }
}
```

Annex C: Revision History

Table 3. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
June 2020	J. St-Louis	.1	all	Created initial content
March 2021	J. St-Louis	.2	all	First complete draft for project completion deliverable.
April 2021	J. St-Louis	.3	all	Second draft version of the document.
May 2021	J. St-Louis	.4	all	Third draft version of the document.

Annex D: Bibliography

[1] STAC: Spatio Temporal Asset Catalogs. [https://stacspec.org/] (2017).