



Master Thesis

submitted within the UNIGIS MSc programme
Interfaculty Department of Geoinformatics - Z_GIS
University of Salzburg

A Semantic Data Cube in Rasdaman

Building a geodata cube from semantically enriched satellite imagery and querying it with WCPS

by

M.Sc. Lena Elisabeth Krupp
104014

A thesis submitted in partial fulfilment of the requirements of
the degree of
Master of Science – MSc

Advisor:

Dr. Martin Sudmanns

Rostock, 30.04.2022

Science Pledge

I hereby declare that the thesis is entirely the result of my work. I have cited all sources I have used in my thesis, I have always indicated their origin. This thesis was not previously presented to another examination board and has not been published.

Rostock, 30.04.2022

A handwritten signature in black ink, appearing to be 'Krupp'.

Acknowledgments

I would like to thank Dirk Tiede and especially Martin Sudmanns for the excellent supervision of this thesis. I would also like to thank the rasdaman developers Bang Pham Huu, Dimitar Misev and Peter Baumann for the feedback and solutions for my questions posed in the rasdaman user group.

Ich bedanke mich bei Daniel Glup für seine allgemeine Unterstützung, sein offenes Ohr und Kartoffelgratin. Bei Utz Fleischer für seinen Humor. Bei Justus Winter für erhellende Gespräche. Und bei Dorle Osterode für schnelle Korrekturen, Inspiration und vor allem dem Interesse an dieser Arbeit.

Abstract

Nowadays, there are a variety of ways to get free access to satellite imagery of various Earth Observation missions via the Internet. The sheer mass of data often makes it difficult for users to search for the content that is relevant to them in a targeted manner. The classic web interfaces of satellite image providers do not have image understanding capabilities, which is why the filter options for selecting satellite scenes are limited to metadata and image-wide overview statistics. To remedy this shortcoming, the University of Salzburg used Open Data Cube (ODC) technology to develop several prototypical semantic data cubes that enable users to query Sentinel-2 data based on the image content. In their data cubes, satellite scenes are preclassified into categories representing semi-concepts equal or inferior to real-world classes using the Satellite Image Automatic Mapper™ (SIAM™) software. Those semi-concepts can be used as model building blocks for deriving real-world classes. However, this promising approach is not yet interoperable, which means that queries are limited to their existing ecosystem and cannot be applied to data cubes built with other technology. The aim of this work is to test how the concept of a semantic data cube developed by the University of Salzburg can be combined with the interoperability standards of the Open Geospatial Consortium (OGC) in order to promote the general accessibility of semantic data cube queries. The array database management system (DBMS) rasdaman supports OGC's standardized Web Coverage Processing Service (WCPS), a coverage processing language that enables the retrieval of raster data or information derived from such data from data cube hosting servers. This thesis shows how a semantic data cube can be implemented with rasdaman and how WCPS can be used in combination with SIAM™ semi-categories in order to build semantic queries.

Zusammenfassung

Heutzutage gibt es eine Vielzahl von Möglichkeiten, über das Internet freien Zugang zu Satellitenbildern verschiedener Erdbeobachtungsmissionen zu erhalten. Die schiere Masse an Daten erschwert es Nutzern oft, gezielt nach den für sie relevanten Inhalten zu suchen. Die klassischen Weboberflächen von Satellitenbildanbietern verfügen über keine Bildverständnisfunktionen, weshalb sich die Filtermöglichkeiten zur Auswahl von Satellitenbildszenen auf Metadaten und bildweite Übersichtsstatistiken beschränken. Um dieses Manko zu beheben, entwickelte die Universität Salzburg mithilfe der Open Data Cube-Technologie (ODC) mehrere prototypische semantische Datenwürfel, mit denen Benutzer Sentinel-2-Daten basierend auf dem Bildinhalt abfragen können. In ihren Datenwürfeln werden Satellitenszenen mit der Satellite Image Automatic Mapper™ (SIAM™) Software in als "Semi-Konzepte" bezeichnete Kategorien vorklassifiziert. Semi-Konzepten können eine reale Klasse abbilden, sind aber in vielen Fällen Vorprodukte die nicht als vollwertige Klassen behandelt werden, sondern als Modellbausteine für die Ableitung realer Klassen verwendet werden. Dieser vielversprechende Ansatz ist jedoch noch nicht interoperabel, was bedeutet, dass Abfragen auf ihr bestehendes Ökosystem beschränkt sind und nicht mit auf anderer Technologie basierenden Datenwürfeln kombiniert werden können. Das Ziel dieser Arbeit ist, zu testen, wie das Konzept eines semantischen Datenwürfels der Universität Salzburg mit Interoperabilitätsstandards des Open Geospatial Consortium (OGC) kombiniert werden kann, um die allgemeine Zugänglichkeit von semantischen Datencube-Abfragen zu fördern. Das Array Database Management System (DBMS) rasdaman unterstützt den standardisierten Web Coverage Processing Service (WCPS) von OGC, eine Coverage-Verarbeitungssprache, die den Abruf von Raster-Daten oder daraus abgeleiteten Informationen von den Hosting-Servern des Datenwürfels ermöglicht. Diese Arbeit zeigt, wie ein semantischer Datenwürfel mit rasdaman implementiert werden kann und wie WCPS in Kombination mit SIAM™-Semi-Kategorien verwendet werden kann, um semantische Abfragen zu erstellen.

Table of Contents

Acronyms.....	ix
1 Introduction.....	1
2 Background.....	6
2.1 Semantic data cubes based on SIAM™ semi-concepts.....	6
2.2 Rasdaman.....	11
2.3 Big raster data and standards to work with them.....	14
2.4 WCPS Syntax in rasdaman.....	17
2.4.1 Sources of information about WCPS in rasdaman.....	17
2.4.2 Executing WCPS queries.....	18
2.4.3 General query structure of WCPS.....	20
2.4.4 WCPS metadata/probing functions in rasdaman.....	23
2.4.5 Further WCPS constructs.....	25
3 Methods.....	26
3.1 Hardware setup and rasdaman installation.....	26
3.2 Data sets.....	26
3.3 Importing images with wcst-import.sh.....	29
3.4 Querying categorical data with WCPS in rasdaman.....	32
4 Results.....	34
4.1 Building the cube.....	34
4.2 Example queries tested.....	35
4.2.1 Structure of the results presented.....	35
4.2.2 Descriptive statistics of a 2D image.....	35
4.2.3 Simple selection and display of (composite) categories.....	37
4.2.4 Fusing Sentinel-2 and SIAM™ data sets.....	38
4.2.5 Investigating categorical trajectories.....	40
4.2.6 Time series analysis.....	41
4.2.7 Edge detection based on categorical data.....	45
5 Discussion.....	48
6 Conclusion.....	52

References.....	ix
Appendix A - WCPS Syntax.....	xv
A.1 Coverage Subsetting.....	xv
A.2 Let Clause.....	xvii
A.3 Deriving single bands and Multiband Constructor.....	xvii
6.1 A.4 Induced Operations.....	xviii
A.5 Conditional evaluation/Case distinction.....	xix
A.6 General Coverage Constructor.....	xix
A.7 Aggregation operations.....	xxi
A.8 Combining constructor and aggregation queries.....	xxiii
A.9 Coverage Filtering with where.....	xxiv
A.10 Clipping.....	xxv
A.11 Additional Functions.....	xxvi
Appendix B - Images in the data cube.....	xxviii
Appendix C - Ingredient files for data import.....	xxxv
C.1 Ingredients file for importing Sentinel-2 data.....	xxxv
C.2 Ingredient file for importing SIAM™ data.....	xxxvi
Appendix D - WCPS queries tested.....	xxxvii
D.1 Descriptive statistics of a 2D image.....	xxxvii
6.2 D.2 Simple selection and display of (composite) categories.....	xl
D.3 Fusing Sentinel-2 and SIAM™ data sets.....	xliv
D.4 Investigating categorical trajectories.....	l
6.3 D.5 Time series analysis.....	lviii
D.6 Edge detection based on categorical data.....	lxii

Table of Figures

Figure 2.1 The Rasdaman architecture.....	13
Figure 2.2 WCPS console available under http://localhost:8080/rasdaman/ows	19
Figure 3.1 Area of the Sentinel-2 granule used.....	28
Figure 3.2 Data importing process with <code>wcst_import.sh</code>	31
Figure 4.1 Sobel operation.....	45
Figure 4.2 Subregion used to test the Sobel operator in WCPS on SIAM™ semi-concepts in order to discriminate land from water.....	46

Index of Tables

Table 2.1 The 33 SIAM™ semi-concepts derived from Sentinel-2 data.....	9
Table 2.2 Basic WCPS queries.....	22
Table 2.3 WCPS probing functions in rasdaman.....	23
Table 2.4 Probing functions not yet implemented in rasdaman.....	24
Table 4.1 A cube with seasonal regularity.....	43

Acronyms

AOI	Area of Interest
ARD	Analysis Ready Data
BOA	Bottom of atmosphere reflectance
CRS	Coordinate Reference System
DBMS	Data Base Management System
EO	Earth Observation
EPSG	The EPSG Geodetic Parameter Dataset is a public registry of geodetic datums, spatial reference systems, Earth ellipsoids, coordinate transformations and related units of measurement originally created by the European Petroleum Survey Group. Each entity has an unique EPSG code by which it can be referenced
GUI	Graphical User Interface
MDD	Multidimensional Discrete Data
MIME type	A MIME type specifies a media type definition (acronym derived from Multipurpose Internet Mail Extensions) indicating the nature and format of a document, file, or assortment of bytes. Examples for MIME types are 'text/csv' or 'image/tiff'
MSI	Sentinel-2's multi-spectral instrument
ODC	Open Data Cube
OGC	Open Geospatial Consortium
OWS	OGC Web Services
QGIS	Free and Open Source Desktop Geographical Information System, formerly known as 'Quantum GIS'
SCBIR	Semantic Content-Based Image Retrieval
SIAM™	Satellite Image Automatic Mapper™ (SIAM™) software for preclassifying

	optical satellite data
TCI	Sentinel-2 True Color Image
TOA	Top of atmosphere reflectance
WCPS	Web Coverage Processing Service
WCS	Web Coverage Service
WCS-T	Web Coverage Service – Transaction Extension
WFS	Web Feature Service
WKT	Well Known Text
WMS	Web Map Service
XML	Extensible Markup Language

1 Introduction

Since the launch of the first Landsat satellite in July 1972, satellite remote sensing has provided valuable information about the state and evolution of Earth's surface, contributing to provide us insights into the physical and socio-economic processes that shape our world. Throughout the history of remote sensing, the degree to which civil users could profit from satellite data has always been connected to its availability and accessibility. This was and still is strongly but not solely determined by the state of the art technology. For example, albeit accessible for civilian use, the costs of acquiring satellite images remained a limiting factor for scientists from those early days until the first decade of this century, with prizes fluctuating considerably in periods in which the system was state owned and in which it was privatized (Morain, 1998). Widespread use of satellite images only began with the opening of the free Landsat archive in 2008. As it proved beneficial for creating economic value, this concept was later adapted by other national and international Earth Observation (EO) programs (Zhu *et al.*, 2019), the most well-known probably being the freely available data from the Copernicus program of the European Union, which sent its first satellite Sentinel-2A into Earth's atmosphere in June 2015 (The Sentinel missions, 2021). With the increase in the number of freely available satellite images, the number of people using the data increased significantly and the user base expanded from remote sensing specialists to scientists that were not necessarily experts in remote sensing but came from other domains such as Biology, Ecology, Geology or Marine Science (Sudmanns, Lang and Tiede, 2018).

Today, the remote sensing community is in the middle of a process of adapting to the sheer amount of data available that presents both an opportunity and a challenge for using satellite data effectively. With images coming from several hundreds of EO satellite sensors and comprising a huge variety of spatial, temporal, radiometric and spectral resolutions, the volume of overall remotely sensed data today is estimated by the Open Geospatial Consortium (OGC) to be likely surpassing one exabyte. Producing, processing and transmitting such big EO data at high velocity leads to high technological requirements for the storing, loading and processing capacity of infrastructures (Ma *et al.*, 2015; Esbrí, 2021). Traditionally, storage was connected to high costs. Thus remotely sensed data has mostly been stored in raw form on tape storage infrastructures hosted by various

Government Agencies. This made retrieval and data preparation tedious and time consuming. Consequently, only a small portion of the available images was being used, coining the term 'dark data' for the large amounts of potentially useful data not considered in research (Purss *et al.*, 2015). During the last years, a lot of work has been going into developing reliable infrastructure capable of efficiently storing, organizing, analyzing and retrieving large amounts of data. Since the handling of image data from satellite-based sensors is essential for EO, structures that deal with raster data are of utmost importance. Different approaches for storing and handling large amounts of raster data exist today. For example, raster data could be managed in data warehouse software systems such as Hive, Apache Spark and ClimateSpark, in a NoSQL database management system (DBMS) like MongoDB, or in an array-based DBMS such as rasdaman and SciDB (Baumann *et al.*, 1998; Thusoo *et al.*, 2009; Stonebraker *et al.*, 2013; Zaharia *et al.*, 2016; Hu *et al.*, 2018; Bradshaw, Brazil and Chodorow, 2019). In the remote sensing community, organizing satellite images in a data cube has proven to be a very useful structuring tool that has become increasingly popular over the last years (Baumann *et al.*, 2019).

Data cubes can be described as massive multi-dimensional arrays containing gridded data (raster data) organized along dimension axes (Baumann *et al.*, 2019). A cube supports at least one through four spatial, temporal and other dimensions, but may have even more (Baumann, 2017). A Geospatial Data Cube necessarily comprises two or three spatial dimensions, and a data cube used for EO has at least one non-spatial dimension, e.g. time (Sudmanns, Lang and Tiede, 2018). Organizing data in a cube does not only have the advantage of avoiding the traditional cumbersome looking through files stored in a file system with metadata encoded in file and directory names, but allows neat slicing and dicing, filtering, aggregation, and even ad hoc analysis of the data. This is extremely powerful when coupled with remotely sensed analysis ready data (ARD) that has undergone a series of corrections, from radiometric to geographic processing, making the data more accessible to a wider public of non-remote sensing experts (Killough, 2019; Kopp *et al.*, 2019). A data cube containing ARD can function as a data center where data is served according to the needs of the users who can query exactly the data or results they need without the need for excessive downloads or in some cases, without any downloads at all (Baumann *et al.*, 2019). Using data cubes as web-based access points or 'geospatial web services' is a step away from traditional geoscience applications that have

been developed for siloed environments towards distributed services fostering technical interoperability and thus accessibility – from GISystems to GIServices (Yue et al., 2015). Standards for implementing web services dealing with raster data have been developed by the OGC and are currently in the process of being improved and extended, such as OGC Web Map Service (WMS), Web Coverage Service (WCS) and Web Coverage Processing Service (WCPS) (Wagemann et al., 2018).

Several data cubes are used in the remote sensing community today, implementing OGC standards to different levels. The multidimensional database system **rasdaman (Raster Data Manager)** already existed in the 90s (Baumann, 1994; Baumann et al., 1998). While the data cube was originally derived from the OLAP (Online Analytical Processing) cubes for business and statistic applications (Baumann et al., 2021; Strobl et al., 2017), rasdaman developers sought to close the technological gap that hindered the use of database technology in scientific applications due to a lack of support for ordered data structures in database management systems. It is not explicitly designed for geodata but for a variety of scientific disciplines (Baumann and Holsten, 2011). Another well-known data cube project that targets science in general and can handle satellite data is SciDB (Stonebraker et al.; 2013, Joshi et al., 2019). A third popular cube, Open Data Cube (ODC) has its origins closer tied to geosciences: It is inspired of and extended from the Australian Geoscience Data Cube, that marked an important step in the advancement of data cube technology as it showed how big time series of satellite images could effectively be stored and worked with. Referring to this best practice example, the Committee on Earth Observations Satellites' (CEOS) data cube team established the Open Data Cube initiative to foster the creation of similar data cubes (ODC, 2018; Kopp et al., 2019).

Currently, the potential of semantic data cubes for EO is being researched at the Department of Geoinformatics at the University of Salzburg, known as Z_GIS. A semantic data cube can be defined as 'a spatio-temporal data cube containing EO data, where for each observation at least one nominal (i.e., categorical) interpretation is available and can be queried in the same instance', an approach that goes beyond providing ARD (Augustin et al., 2019). While optical EO data might be ARD, it is still merely digital numbers representing spectral values. It has to go through further processing in some way to make it meaningful, useful, valuable and relevant to become information and/or even knowledge.

(Rowley, 2007; Sudmanns, Lang and Tiede, 2018). In a semantic data cube, EO data has gone through an interpretation process that loaded a satellite scene with meaning. This is also called semantic enrichment. At Z_GIS, Sen2Cube.at was developed as the prototype of a web-application that allows users to access several semantic data cubes and provides a Graphical User Interface (GUI) to interact with them (Sudmanns *et al.*, 2021). While smaller cubes have been designed for subregions of Syria and Afghanistan, the main data cube accessible on the Sen2Cube.at platform is a cube of Austria comprising semantically enriched categorical Sentinel-2 satellite data from the start of the mission until today. To produce the semantically enriched layers, the Satellite Image Automatic Mapper (SIAM™) software was used on the input Sentinel-2 data that provides sensor-agnostic classification capabilities. SIAM™ separates pixels into categories based on their spectral characteristics using a prior knowledge-based decision-tree preliminary classifier. This works automatically and in near real-time (Baraldi, 2019). It should be emphasized, that the resulting categories are not equivalent to real world land cover classes a user might be interested in. Instead, the resulting semantic categories are regarded as generic 'semi-concepts' equal or inferior to land cover classes but globally valid (Baraldi and Boschetti, 2012a; Augustin *et al.*, 2018). These semi-concepts can be combined much like building blocks to create models that produce real-world land cover classes of interest for a specific EO task. (Augustin *et al.*, 2018). In Sen2Cube.at models can be build using a model builder GUI (Sudmanns *et al.*, 2021). They are stored within the application and are reusable. This way, a model can be applied to different data cubes that might be based on different sensor data, as long as semantic enrichment was done in the same way. This ultimately allows for automatic processing. In Sen2Cube.at, the original data and their derived first stage information SIAM™ layers as well as optional additional layers such as a Digital Elevation Model (DEM) stored in the cube are therefore called the 'factbase'. In addition, there is also the knowledgebase, in which the models are stored as rules. An inference engine allows to combine the two by applying the rules to the facts (Sudmanns *et al.*, 2021). Successfully processed inferences can be downloaded either as TIFF file or as QGIS project. Additionally, an OGC WMS link is provided.

Even though an early approach for a semantic data cube has been created using rasdaman (Sudmanns *et al.*, 2017), Z_GIS now has adopted Open Data Cube as the underlying technology for this purpose. Open Data Cube provides a common analytical

framework with a series of data structures and tools to facilitate the organization and analysis of large gridded data collections (ODC, 2021). Sen2Cube.at is accessible to people who aren't used to programming and working with command-line interfaces, as it allows you to create models and start inferences with a GUI. However, it still remains an isolated environment to some extent. Even though there is the possibility to add a web service extension to ODC, Sen2Cube.at does currently not support geospatial standards like OGC WCS and WCPS that foster interoperability between geographic information systems. The increasing number of platforms providing access to EO data holds the negative potential of developing a user-unfriendly landscape of geo-applications in which users have to tediously apply separate workflows for each system. In other words 'Standalone array stores form just another silo, even with query capabilities' (Baumann *et al.*, 2021). This is where the important role of interoperability becomes evident. Interoperability can be described as the 'The ability of systems to provide services to and accept services from other systems and to use the services so exchanged to enable them to operate effectively together' ISO TC204, document N271 as cited by (Kuhn, 2005). The extent to which remote sensing data and derived products will be accessible to a wide range of users in the future will be positively influenced if standardized interfaces are omnipresent in geodata technology.

This work aims to explore how semantic image retrieval can be combined with the concept of interoperability to enable better general accessibility of EO data. It investigates how the concept of a data cube semantically enriched with SIAM™ can be applied to an array DMBS with OGC WCPS capabilities in order to find out whether WCPS provides adequate functionality to deal with semantic semi-concepts. As system, the rasdaman array DBMS was chosen, as it is the reference implementation for OGC WCS and in addition, supports WCPS. With the exception of a data cube set up to test the observation of flooding events in Somalia (Sudmanns *et al.*, 2017), no semantic data cube has been realized so far with rasdaman. Two topics were researched. First, the process of building a cube by loading semantically enriched SIAM™ layers together with Sentinel-2 data into a rasdaman array database was explored. As a proof of concept for a working semantic data cube, a small prototype of a cube with only one Sentinel granule and 13 satellite scenes was implemented. Second, it was examined how the WCPS constructs currently implemented in rasdaman can be used to query SIAM™ semi-concepts. The goal was to develop

sample queries that could be components of models intended to map real-world land cover classes, similar to what can be done in Sen2Cube.at.

The thesis is structured as follows: Chapter 2 provides background information on the approach of creating a semantic data cube based on SIAM™ classification, rasdaman's technological structure as well as OGC standards. Special attention is paid to the OGC WCPS query language. Its syntax as implemented in rasdaman is explained in a quite detailed manner Chapter 2.4 and appendix A in order to foster better understanding of queries presented in the result part. Chapter 3 demonstrates the actual data structure setup as it gives an overview over the system and data sets used (all images can be viewed in appendix B), presents the conceptual data cube model used in this thesis and describes the data import process applied. In addition, six topics for which queries were tested are presented. The queries examined consisted only of those that returned results with 0 to 2 dimensions. In Chapter 4, the testing results are depicted. This includes the description of caveats for the process of loading data into the rasdaman array store as well as the presentation of sample queries tested. For readability reasons, import scripts and the code of the tested queries have been largely outsourced to Appendices C and D. In Chapter 5, the results are further analyzed with regard to the strengths, weaknesses, future opportunities and potential obstacles of using rasdaman with SIAM™ data, before the work is summarized in the final chapter.

2 Background

2.1 Semantic data cubes based on SIAM™ semi-concepts

Offering free available satellite images, processed in an ARD manner and stored conveniently in data cubes that can be accessed via web interfaces, has facilitated access to remotely sensed satellite data and fostered general awareness of available data which has been argued as contributing to 'democratizing' satellite data (Kopp *et al.*, 2019). However, to become truly accessible to a broad range of users that are no remote sensing experts, producing retrievable information goes beyond producing ARD. Even though many possibilities to access satellite data exist, relevant data can be hard to find in the vast amount of data offered online. Some Content Based Image Retrieval (CBIR)

mechanisms based on metadata in the form of text information or image-wide summary statistics are integrated into web interfaces offering satellite images such as the Earth Explorer platform (EarthExplorer, n.d.) run by the US Geological Survey (USGS) or the EO browser featured at the Sentinel Scientific Data Hub (Sentinel-hub EO-Browser, n.d.) run by the European Space Agency (ESA). Here, the scene selection via geographic area of interest (AOI), timespan and some textual metadata like mission name, image path/row, data category is possible and helps users to reduce the amount of data they have to look through for finding relevant data. However, they lack real image understanding capabilities (Tiede *et al.*, 2017), meaning that a selection of satellite scenes based on the geographical entities they contain is not possible. A semantic EO data cube that allows for Semantic Content Based Image Retrieval (SCBIR) is proposed as a solution. In a semantic data cube, at least one categorical interpretation of its spectral characteristic is available for each pixel (Augustin *et al.*, 2019). Filtering scenes can therefore happen on the basis of these categories.

To be useful for large amounts of data, the generation of a categorical interpretation for a EO satellite must be as automated as possible. However, this is not an easy task. As vision is an ill-posed problem in the Hadamard sense, it is not possible to recreate an unambiguous real world model based on an 2D image by trying to inductively infer it from spectral pixel information only. When reducing a 4-dimensional world to a 2-dimensional image, information is lost. The resulting information gap cannot be compensated for by deriving an interpretation of the world exclusively on the basis of quantitative sensory data, as this is not sufficient to infer stable symbolic percepts of the world (Tiede *et al.*, 2016). Each time when using machine learning to derive real world classes from satellite images for a selected region, manual user input is needed in the form of providing labeled training or validation data by contributing reference data collected from the field, from existing maps or from tabular data. Therefore, classification cannot happen in a fully automated way, making the analysis of large amounts of satellite data costly in terms of labor and time (Baraldi, Girona and Simonetti, 2010). To circumvent the problem, a solution inspired by human vision can be applied. In the much-quoted seminal work of Marr, human vision is presented as going from a first-stage primal sketch based on feature extraction of fundamental components of a scene right to symbolic almost immediately and without losing information (Marr, 1982). To achieve this, the human vision system consists of a

preattentive (low-level) vision first phase, in which our brain extracts picture primitives based on general-purpose image processing criteria independent of the scene under analysis, and an attentive (high-level) vision second phase, in which our brain operates as a careful scanning system employing a focus of attention mechanism based on prior knowledge (Baraldi and Humber, 2014). To process what we see, our brain combines inductive inference (progressing from particular cases, or sub-symbolic true facts to a general model of our environment) with deductive inference (progressing from a general model of the world which is based upon the knowledge acquired through our experience to a particular case). For example, if we look at a landscape in a park, we could identify a series of linear lines that border contiguous areas of light and dark color (the sub-symbolic true facts) as a park bench (entity with meaning also called a 'symbol'). Based on our knowledge about the world, we can discriminate it from its surroundings immediately. Thus, human vision can be described as 'a symbolic hybrid (combined deductive and inductive) inference system where (symbolic) prior knowledge is injected into the sensory data interpretation process starting from the preattentive vision first stage' (Baraldi and Boschetti, 2012a, 2012b).

It is an important finding, that a priori knowledge of the world needs to be incorporated into a computer vision system for it to produce useful results (Mulier and Cherkassky (2007) as cited by (Baraldi and Humber, 2014). It is necessary that top-down knowledge partly guides lower level processing, as is the case in a hybrid system (Vecera and Farah, 1997). The Satellite Image Automatic Mapper™ (SIAM™) software used to produce classified input raster layers for the Sen2Cube.at data cubes as well as for the data cube of this thesis mimics the preattentive first phase of human vision. Its output can thus be viewed as the 'primal sketch' of a semantic interpretation of the input satellite scene. SIAM™ is a preclassifier of satellite data that does not require any user input in the form of training and supervision to run. Instead, it relies on spectral categorization through following non-adaptive decision trees that are based on prior knowledge. The resulting symbolic preattentive categories belong to a discrete, mutually exclusive and finite set of fuzzy semi-concepts which are greater than zero and equal or inferior to a real world class (Baraldi and Boschetti, 2012b, 2012a). In other words, they have a 1:1 or many:1 relationship to a real world class. Mapping back from such a SIAM™ preliminary classification map to the input image value domain should create a piecewise constant

approximation of the input scene much like an edge-preserving smoothing filter with image details that represent high spatial-frequency components clearly recognizable (Baraldi and Boschetti, 2012a). The SIAM™ classification system is pixel-based and as such relies purely on spectral (color) information and is insensitive to context such as size, shape, location, texture and semantic information. No reference data set or supervised data learning mechanism is used (Baraldi *et al.*, 2006). SIAM™ is sensor-agnostic when it comes to handle data that has been calibrated to Top-of-Atmosphere (TOA) reflectance, but it's input requirements are based on Landsat high-resolution data. When used with Sentinel-2 data, six bands of the sensor's multi-spectral instrument (MSI) are used to derive SIAM™ categories. These are the blue, green, red and near infrared band as well as two medium infrared bands which are resampled from 20m pixels to 10m pixels to meet the input criteria. A constant is working as an input placeholder for the thermal band, which Sentinel-2's MSI does not possess but is included to apply additional thermal decision rules in SIAM™ with other sensors (Augustin *et al.*, 2018; Baraldi and Boschetti, 2012). Following the SIAM™ decision rules, the Sentinel-2 bands are converted to several products: four semi-concept granularities providing 18, 33, 48 and 96 symbolic variables (categories) as well as four additional information layers comprising sub-symbolic variables. These are the 'binary vegetation mask' based on the vegetation-related semi-concepts, the 'pentanary haze mask', a greenness index and panchromatic brightness image. Furthermore, text files providing image-wide summary statistics are produced (Augustin *et al.*, 2018; Baraldi, 2019) Only the 33 categories product has been used for this thesis. Table 2.1 shows the categories together with their ID and short name. The SIAM™ raster layer gets returned in pseudo-colors that match what people generally associate with a semi-concept (e.g. green for vegetation related semi-concepts, blue for water-related semi-concepts etc.).

Table 2.1 The 33 SIAM™ semi-concepts derived from Sentinel-2 data. Adapted from (Sen2Cube.at Manual, 2022)

N°	Spectral Category	Short
1	Strong vegetation with high NIR	SVHNIR
2	Strong vegetation with low NIR	SVLNIR
3	Average vegetation with high NIR	AVHNIR

4	Average vegetation with low NIR	AVLNIR
5	Weak vegetation	WV
6	Shadow area with vegetation	SHV
7	Shrub Rangeland with high NIR	SHRBRHNIR
8	Shrub Rangeland with low NIR	SHRBRLNIR
9	Herbaceous Rangeland	HRBCR
10	Weak Rangeland	WR
11	Pit or bog	PB
12	Greenhouse	GH
13	Very bright barren land or built-up	VBBB
14	Bright barren land or built-up	BBB
15	Strong barren land or built-up	SBB
16	Average barren land or built-up	ABB
17	Dark barren land or built-up	DBB
18	Weak barren land or shadow area with barren land	WBBorSHB
19	Near infrared-peaked barren land or built-up	NIRPBB
20	Burned area	BA
21	Deep water or shadow	DPWASH
22	Shallow water or shadow	SLWASH
23	Turbid water or shadow	TWASH
24	Salty shallow water	SASLWA
25	Cloud	CL
26	Smoke plume	SMKPLM
27	Thin clouds over vegetation	TNCLV
28	Thin clouds over water area or barren land or built-up areas	TNCLWA_BB
29	Snow or water ice	SN
30	Shadow snow	SHSN
31	Shadow areas	SH
32	Flame	FLAME
33	Unknown	UN
255	No data	NO DATA

Due to the ill-posed nature of vision, it is important to stretch that the fuzzy spectral categories derived by SIAM™ can not be understood as real world classes but can be the building blocks for them. Semi-concepts derived by SIAM™ still have to be coupled with a system that acts as an attentive vision second phase. Therefore, the semantic data cube concept of the Sen2Cube.at platform developed at the University of Salzburg offers users the opportunity to create rules that combines the SIAM™ semi-concepts in a meaningful way in order to obtain real land cover classes. In this way, prior knowledge finds its way into the classification. Once rules have been defined to create land cover classes for a specific application, they can be saved as a model. Saved models can be used in an automated way for the fast derivation of real-world land cover classes of Sentinel-2 data. Chapter 4.2 shows which sample queries were used in this thesis to test whether the coverage processing standard WCPS is suitable for building models based on SIAM™ semi-concepts.

2.2 Rasdaman

Rasdaman (Raster Data Manager) is a domain independent array DBMS which is capable of managing array data of arbitrary size and structure, from medical images to geospatial data. While it's conceptual roots date back to the late 80s (Baumann *et al.*, 2021), rasdaman has been developed since 1996 as a product of a series of EU funded projects. A commercial version was marketed by the research spin-off rasdaman GmbH since 2003. Since rasdaman GmbH and Jacobs University teamed up in 2008/2009 there is in an open source community version onto which the work of this thesis is based (Rasdaman developers, 2022, Chapter 1). Rasdaman is designed to handle multidimensional data. While the word 'cube' invokes the mental image of three dimensions, the arrays managed by rasdaman can have 1 to 4 and even more dimensions and are therefore described as Multidimensional Discrete Data (MDD), as raster data or regularly gridded data. Sometimes, the description 'massive multi-dimensional array' is used to emphasize on the capability of a cube to store data of sizes that can go significantly beyond the main memory resources of the server hardware (Baumann, 2017); (Baumann *et al.*, 2019). The array possesses n-dimensional axes which allows for unambiguous querying of cell

values. Each dimension axis of an array has an extent with an upper and lower bound known as the axis' domain. The index values of a domain have to be integers but can be negative. The cell values (pixel or voxel values depending on the array's dimension) of the array can either be a base type as used in the C/C++ language or a composite type (struct). All values within one array have to have the same type. The possibility to slice and trim axes allows for a quick access of subsets based on selected timespans and areas of interest (Baumann, 2017). Several arrays are stored as collections which can be compared to a table in a relational database but have only two columns/attributes: there is an object identifier (OID) which is maintained by the rasdaman system, as well as the array itself (Rasdaman developers, 2022a, Chapter 1). Due to that structure it is possible to link the arrays via foreign keys with conventional tables and therefore reference particular array objects or subsets of them by querying a specific domain specification. Rasdaman partitions arrays into subarrays called tiles and stores each tile either in a file system directory maintained by rasdaman with metadata stored in an SQLite instance embedded in rasdaman, or in a binary large object (BLOB) with a spatial index in a Postgres database that allows to maintain arrays of unlimited size. The storage variant can be chosen during installation in custom installations. When installing from a preconfigured package, the SQLite variant is installed by default (Rasdaman developers, 2022a, Chapter 2.5.3). The partitioning scheme can be adapted by the rasdaman user (Rasdaman developers, 2022a Chapter 1). The rasdaman array management system can be run in a distributed way on different servers thus leveraging computing power. Figure 2.1 shows its architecture. There is a central master node, the rasdaman host, which is the central request dispatcher and controls all server processes. It accepts requests sent by the client node and assigns server processes to handle them. The server processes are resolving the assigned requests and produce calls to one or more relational servers each belonging to a database host. The relational server retrieves the requested data from a relational data base store and sends it back to the client node using the network (Hu et al., 2018).

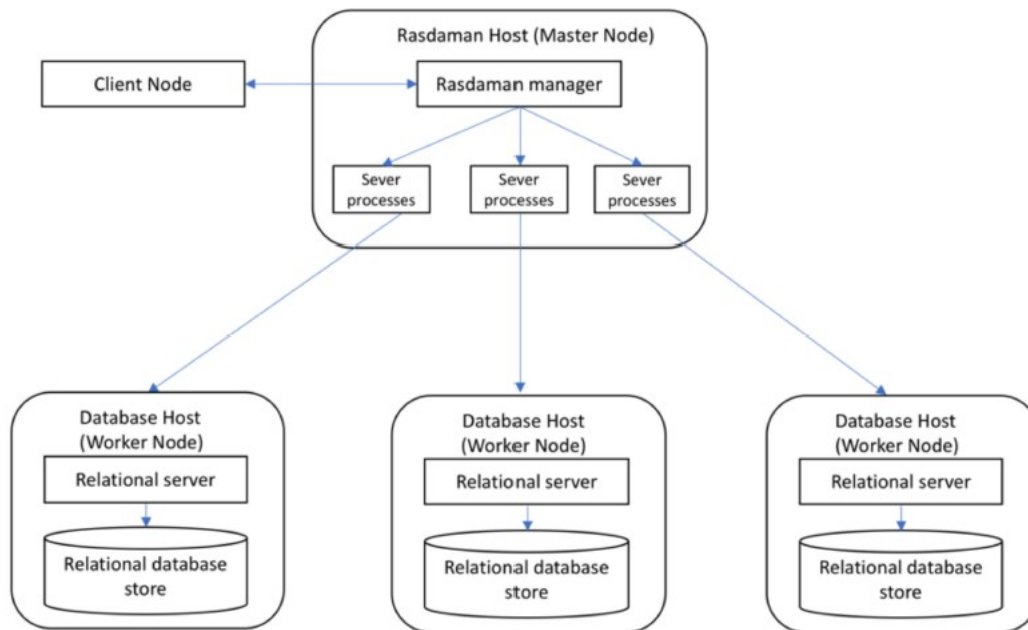


Figure 2.1 The Rasdaman architecture. Retrieved from (Hu et al., 2018)

As the rasdaman array DBMS is domain agnostic, it does not provide specific semantics of space and time – these are ‘outsourced’ to the rasdaman petascope module, which is built on top of the array engine (Rasdaman developers, 2022a, Chapter 5). Petascope provides geospatial query functionality by employing raster related OGC Web Services as presented in Chapter 2.3. Petascope’s ability to identify, understand and create Coordinate Reference Systems (CRSs) is based on SECORE (Semantic Coordinate Reference System Resolver). SECORE was created by the rasdaman developer team from Jacobs University Bremen. It resolves conforming URLs that contain a CRS ID to their CRS definitions expressed in Geography Markup Language (GML) available in a BaseX XML database deployed as registry service (Misev, Rusu and Baumann, 2012). A public registry service that can be used for deriving CRS definitions is the ‘OGC Definitions Server’ by the OGC Name Authority. It provides URLs for CRS definitions beginning with ‘https://www.opengis.net/def’ (OGC 09-048r5, 2019; OGC Naming Authority SC, 2022). For example, a coverage can contain the the URL shown in Listing 2.1. By requesting this URL, the CRS gets resolved to its GML definition which will help to identify the coverage as WGS 84.

<https://www.opengis.net/def/crs/EPSG/0/4326>

Listing 2.1

SECORE is currently operated for the OGC by its member rasdaman GmbH and Jacobs University (Hobona, 2021). When rasdaman is installed, it will install a local BaseX XML database containing CRS definitions by default, so CRS resolving can also be deployed locally (rasdaman team, 2022, Chapter 5).

To query data, rasdaman provides the declarative query language rasql that allows for query rewriting through the server for more efficient requests (Baumann *et al.*, 2021). Rasql is much like SQL but for multidimensional arrays and is in fact the reference model for the ISO SQL/MDA (“Multi-Dimensional Arrays”) standard (ISO/IEC 9075-15:2019). On top of that, querying data in rasdaman can also be done using the OGC standardized WCPS language. In rasdaman, WCPS queries get internally transformed into rasql queries. WCPS is closely related to Xquery. Its syntax is presented separately in Chapter 2.4 and Appendix A.

2.3 Big raster data and standards to work with them

Geodata come in a variety of forms and formats. To foster interoperability between applications for presenting and processing them, the OGC consisting of enterprises, government organizations, research organizations and universities in the geospatial domain, oversees the development of open geospatial standards (OGC, n.d.). A common classification of geodata in the GIS world is a separation between 'discrete' vector data and 'continuous' raster data. Within a vector data model, the spatial characteristics of a discrete real-world phenomenon like buildings, streams and measurement stations are represented as one or more geometric primitives such as points, curves, surfaces or solids (OGC 07-011, 2006). Those primitives are understood to be a geospatial 'feature' (an abstraction of real world phenomena) with their additional characteristics being stored as feature attributes (OGC 06-103r4, 2011). Contrary, raster data is used to store real-world phenomena that vary continuously over space such as temperature, soil composition or elevation (OGC 07-011, 2006). In fact, a raster is not the only data format that can be used for storing continuous data. Other formats such as point clouds might be suitable as well, and therefore the term coverage is applied by the OGC to generally define coverages as

'[...] digital geospatial information representing space/time-varying phenomena, specifically spatio-temporal regular and irregular grids, point clouds, and general meshes' (OGC 17-089r1, 2018). The defining commonality of these different formats is that the values stored in them have a geospatial unique position and, moreover, can only accept values within a clearly defined range: 'A coverage is a feature that associates positions within a bounded space (its domain) to feature attribute values (its range). In other words, it is both a feature and a function. [...] A coverage may represent a single feature or a set of features' (OGC 07-011, 2006). They can be multidimensional (OGC 09-146r8, 2019). Coverage values possess the same type, known as the coverage's range type. Admissible types have named components called fields (also known as bands or channels) with a unique field name which can contain either (atomic) numeric or Boolean types. The range fields of a coverage are not required to be of the same type (OGC 08-068r3, 2021).

Coverages can be offered via a web service that allow access to resources by specifying GET or POST HTTP requests. Several OGC standards exists for doing so in a well-defined manner. For example, OGC WMS is a visualization service that produces 2D maps targeted at human consumption (rasdaman team, 2022, Chapter 5). Another example, the OGC WCS provides the core functionality for any web server offering coverages suitable fur further processing. The primary goal of WCS is to enable simple but effective data retrieval with the focus on spatio-temporal subsetting, range subsetting (in some domains equal to "band selection"), reprojection, scaling and data format encoding (Baumann, 2010). Rasdaman GmbH and Jacobs University Bremen were submitting organizations of this standard (OGC 17-089r1, 2018). The OGC Web Coverage Service – Transaction Extension (WCS-T or WCS Transaction) is an extension of the WCS Core that defines how to modify a WCS server's coverage offering. Three request are defined (OGC 13-057r1, 2016):

- The 'InsertCoverage' request allows to add a coverage as parameter to the WCS server's coverage offering which can then be accessed by using WCS operations
- Similarly, 'DeleteCoverage' allows to remove a coverage from the WCS offering
- The 'UpdateCoverage' request allows to modify parts of a coverage offered by the WCS server

The OGC Web Coverage Processing Service (WCPS) is defined as a language for retrieval and processing of multi-dimensional geospatial coverages that might represent

sensor, image or statistics data. WCPS offers functionality similar to WCS requests but goes well beyond that. It is designed to support more powerful processing capabilities such as advanced extraction and server-side analysis of large, possibly multi-dimensional coverage repositories. To achieve that, it provides further coverage processing primitives and allows the nesting of functions which enables arbitrarily complex requests for a range of imaging, signal processing and statistical operations in order to be rendered, inserted into scientific models or used in some other client applications (Baumann, 2010; OGC 08-068r3, 2021). For example, provided that a web service offers a multi-spectral coverage comprising red and near-infrared bands, a coverage showing the Normalized Difference Vegetation Index (NDVI) could easily be returned by making use of the WCPS query capabilities (Baumann, 2010). In fact, in rasdaman WCS is implemented in WCPS (which in turn is internally implemented in rasql) (Misev, 2020). Both WCS-T and WCPS are integrated into rasdaman. Notably, rasdaman founder Peter Baumann served as an editor of WCS-T extension standard and WCPS Language Interface Standard which underlines the close relationship between software and standard.

2.4 WCPS Syntax in rasdaman

2.4.1 Sources of information about WCPS in rasdaman

Chapter 2.4 is dedicated to how WCPS queries are written and executed in rasdaman. In fact, there is already a lot of material published on this. For example, the official rasdaman documentation v10.0 is available at <https://doc.rasdaman.org/> includes two sections on WCPS dealing with WCPS syntax (Chapter 5 and Chapter 11) (rasdaman team, 2022). However, some information on syntax used in this thesis could not be found there and were derived from other sources closely related to the rasdaman project. For instance, the possibility to construct a coverage defining a list using < > was featured in the WCPS standard (OGC 08-068r3, 2021, Chapter 7.1.30). Even though it was not described in the current rasdaman WCPS documentation it could be found in the rasdaman training material from EarthServer¹. To better understand the WCPS grammar currently implemented in rasdaman and needed for building the semantic queries that are forming the objective of this thesis, WCPS constructs were gathered and tested in advance. They are presented in Chapter 2.4.3 and Appendix A. Wherever possible, a working example is shown. The sample data used for this is based on the sample files available in petascope by default when installing rasdaman from a preconfigured package. It is very important to note that all WCPS constructs have been tested in a rasdaman v10.0.0-beta3 version and not in a stable release. Some of the problems encountered with queries may have been solved in the stable version of rasdaman that has since be published, but could not be taken into account in this work due to time constraints. The following resources about coverages and WCPS are all created by or are maintained under the responsibility of rasdaman head developer Peter Baumann and were consulted for the subchapters of this chapter :

- WCPS standard 1.1 (OGC 08-068r3, 2021)
- Rasdaman web documentation v10.0 , especially Chapter 5.5 and 11.2 (rasdaman team, 2022)
- (Baumann, 2010)

¹ The EarthServer federation is an organization whose members are large-scale Earth data providers. It offers a single point of access where members can publish their data and services so that they can be mixed and matched (Earth Server Federation, n.d.)

- EarthServer webinars (Baumann *et al.*, n.d.)
- EarthServer Webinars WCPS 1 'Introduction to WCPS - Part 1: Basic' and 'Introduction to WCPS - Part 2: Advanced' uploaded to youtube.com by Peter Baumann on 29.07.2015 (Baumann and Merticariu, 2015a, 2015b)
- Hands-On Demos about INSPIRE² conform coverages (Baumann, Schleidt and Escriu, n.d.) available at <https://inspire.rasdaman.org/> (last accessed on 20.04.2022)
- Rasdaman workshop page provided by the rasdaman developer team (*Rasdaman Workshop*, n.d.)
- Rasdaman user forum available at <https://groups.google.com/g/rasdaman-users> (last accessed 24.04.2022, requires registration)

2.4.2 Executing WCPS queries

In rasdaman, a WCPS request can be submitted to Petascope as GET request using the WCS GET/KVP protocol binding (Rasdaman developers, 2022, Chapter 5.5.1). For specifying the WCPS query in a GET/KVP request, either the keyword query or the non-standard shortcut q can be used (Listings 2.2, 2.3). Only one query parameter in an URL is allowed.

```
http://localhost:8080/rasdaman/ows?service=WCS&version=2.0.1&
REQUEST=ProcessCoverage&query=<wcps-query>
```

Listing 2.2

```
http://localhost:8080/rasdaman/ows?service=WCS&version=2.0.1&
REQUEST=ProcessCoverage&q=<wcps-query>
```

Listing 2.3

Rasdaman provides access to several OGC web services (WMS, WCS, WCS-T, WCPS) via an OGC Web service (OWS) client that gets installed as a servlet with every rasdaman installation (rasdaman team, 2022, Chapters 2.4.2 and 11.2). It is available under the URL

2 The INSPIRE directive (Infrastructure for Spatial Information in Europe) aims at creating a common spatial data infrastructure in the EU to form a basis for a common environmental policy (European Commission, 2022)

shown in Listing 2.4. Here, a console can be found into which WCPS queries can be typed and executed (Figure 2.2).

<http://localhost:8080/rasdaman/ows#/services>

Listing 2.4

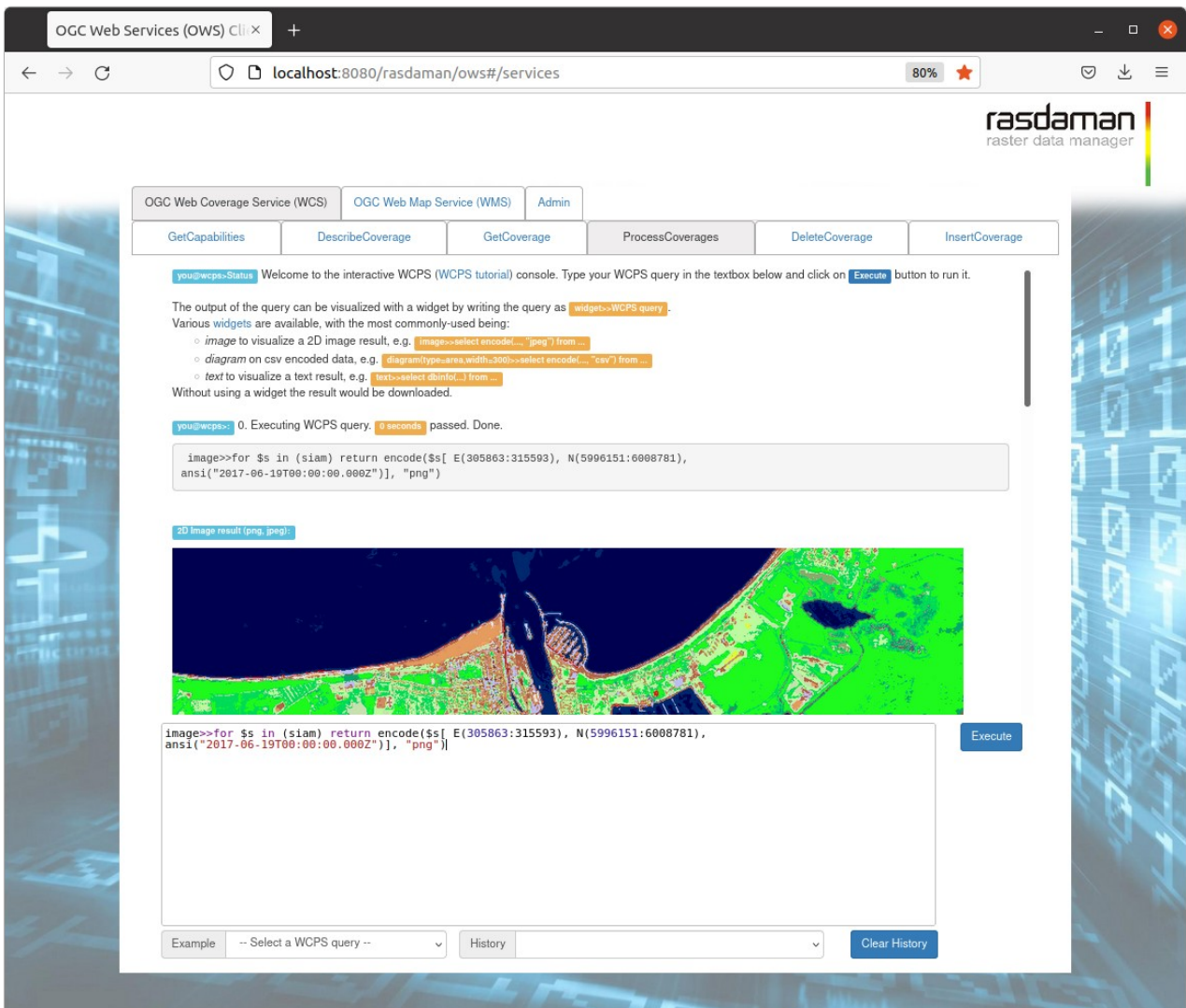


Figure 2.2 WCPS console available under <http://localhost:8080/rasdaman/ows>

When using the rasdaman OWS client, WCPS queries can be prefixed with the desired output style (diagram or image) and two right pointing single angle quotation marks (Listing 2.5) that enable the result of the query to be returned directly in the query window as shown in Figure 2.2 (There is even a possibility of projecting images on a globe by using the command `wwd(specifiedBoundingBox)>>`).

```
image >> [WCPS query expression]
```

Listing 2.5

2.4.3 General query structure of WCPS

Inspired by Structured Query Language (SQL), WCPS was designed to be a declarative query language that allows for server-side optimization and parallelization. WCPS queries either written by human users or automatically generated by a client during interaction are shipped to the data on a server and processed there (Baumann *et al.*, 2019). The declarative style of the language allows users to define what the result of a query should look like, rather than dictating each processing step. This makes queries easier to write and allows servers to flexibly optimize queries by rearranging the evaluation sequence in order to calculate faster results. WCPS is 'safe in evaluation' meaning that every admissible request can be evaluated in a finite number of steps and will terminate after finite time (Baumann, 2010). This avoids the possibility of Denial of Service (DoS) attacks on the level of a single request which, like in SQL, is achieved by avoiding explicit loop and recursion constructs (Baumann, 2010). Despite this it is possible to send requests with high workload to a server (OGC 08-068r3, 2021, Chapter 7 Note 2). The language is semantically closely related to XQuery, a XML query language that is specified by the World Wide Web Consortium (W3C) for XML databases. A future integration with XQuery is planned to allow for integrated data and metadata retrieval in WCPS 2.0 (Baumann and Merticariu, 2015a, 5:25). The heart of a XQuery expression is a FLWOR statement, meaning that it can contain the following components:

- **FOR**
- **LET**
- **WHERE**
- **ORDER**
- **RESULT**

WCPS is based on this schema except that explicit ORDER statements are not currently implemented to the author's knowledge. The meaning of the individual components will be explained in more detail in the course of this chapter as well in Appendix A. The current

WCPS syntax is defined in the WCPS 1.1 standard published in 2021. To date, not all constructs are implemented in rasdaman, as currently only WCPS 1.0 is implemented, WCPS 1.1 being still under work (Baumann, 2022a). In some cases rasdaman does also feature WCPS functions that are currently not mentioned in the WCPS standard, such as the clip function explained in Appendix A.

A WCPS processing request consists of a `processCoveragesExpr` which allows clients to query of one or more coverages offered on a WCPS server and returns an ordered sequence of one or more scalar values or coverages (OGC 08-068r3, 2021). The frame for each WCPS query are the **for** and the **return** clause. Queries iterate over lists of coverages specified in the **for** clause; here a coverage object A or multiple coverage objects A, B, C etc. are sequentially tied to an iterator variable `$c` on which the query is applied. Prefixing the variable with '\$' is not mandatory, but a rasdaman convention to resemble to a XQuery-style syntax (Baumann, 2010). It is used like this throughout this thesis. If the query output is a simple scalar or a list of scalars which can be Boolean, numeric, or a string, it gets returned as ASCII text. If the result is a coverage or a list of coverages, then a format encoding has to be specified as MIME type using an `encode` statement in the **return** clause (OGC 08-068r3, 2021). Coverage expressions support multidimensionality but attention has to be paid to chose the right MIME type that can handle the respective dimensionality of the output for a successful query (Baumann, 2010). For example, a three dimensional output could be stored in a netCDF file but cannot be stored in a TIFF, so the query would fail if 'image/tiff' or 'tiff' was specified. The WCPS standard stipulates that data items returned can have different dimensions, domains, range types and thus be heterogeneous in size and in structure (OGC 08-068r3, 2021, Chapter 7 Note 3). This is not well implemented in rasdaman v10.0.0-beta3 and problems were observed when multiple results were to be returned as can be seen in Table 2.2. WCPS 1.1 also stipulates a `store()` function that allows an encoded coverage to be stored on the server side for a retrieval at a later time when specifying the returned URL (OGC 08-068r3, 2021). However, this function is not available in rasdaman yet. It is possible to combine coverages. The WCPS equivalent to a SQL join operation is defining two iterator variables and combining them in queries. This translates to nested loops (Baumann and Merticariu, 2015b, 8:12; rasdaman team, 2022, Chapter 11.2).

Table 2.2 Basic WCPS queries

Syntax	Rasdaman Example
for \$c in (A) return scalar	for \$c in (mean_summer_airtemp) return 42 → 42
for \$c in (A, B, C) return scalar	for \$c in (AverageChlorophyll, AverageTemperature) return 42 → 42 42
for \$c in (A) return encode(\$c, „image/tiff“)	for \$c in (mean_summer_airtemp) return encode(\$c, "image/tiff") → returns the input coverage in tiff format
for \$c in (A, B, C) return encode(\$c, „image/tiff“)	for \$c in (AverageChlorophyll, AverageTemperature) return encode(\$c[ansi("2015-01-01T00:00:00.000Z")] , "image/tiff") → Caveat: The result here is likely a bug. Instead of returning two separate and distinct tiff files, two times the first coverage is returned and written into one tiff file. This results in a broken tiff whose content can be opened in a text editor, but not with an image program or a GIS.
for \$c in (A) return store(encode(\$c, „image/tiff“))	This query should return an URL under which the server stores the tiff-encoded result coverage. This functionality is not implemented in rasdaman yet.
for \$a in (A ₁ ,A ₂ ,...,A _n), \$b in (B ₁ ,B ₂ ,...,B _n), ..., \$n in (N ₁ ,N ₂ ,...,N _n) return f(\$a,\$b,...,\$n)(same as above)	for \$c in (-AverageTemperature), \$d in (AverageTemperature) return encode((unsigned char) \$c.Red[ansi("2012-12- 01T20:07:00.500Z")] * 2 - \$d.Red[ansi("2012-12-09T20:47:12.500Z")] , "tiff")

2.4.4 WCPS metadata/probing functions in rasdaman

According to the WCPS 1.1 standard, the focus of WCPS lies on providing coverage processing functionality and metadata functions are only integrated to the extent necessary for a coherent service in order to fit into the OGC standard family (OGC 08-068r3, 2021 Chapter 7 Note 4) Nevertheless, some probing functions are very useful to better understand how to work with a specific coverage and some might be useful for processing constructs as well. Table 2.3 shows probing functions currently implemented in rasdaman. Table 2.4 shows functions featured in the WCPS 1.1 standard currently not implemented in rasdaman.

Table 2.3 WCPS probing functions in rasdaman. Adapted from (rasdaman team, 2022, Chapter 11.2.4)

Syntax	Result
imageCrsDomain(\$c)	Returns a list of comma-separated axes bounds
imageCrsDomain(\$c, a)	Returns the low and high (lo, hi) grid bounds for axis a
imageCrsDomain(\$c, a).x	Returns the upper or lower grid bounds. For x, either 'lo' or 'hi' has to be specified
domain(\$c)	Returns a list of comma-separated axes bounds according to the coverage's CRS orders respectively. Each list element contains an axis a with the lower and upper bounds in the axis CRS Caveat: While for a geospatial axis domain(\$c, a) it returns the domain limits in geocoordinates, it does return image coordinates here as well for the time axis.
domain(\$c, a)	Returns the low and high (lo, hi) geo for axis a
domain(\$c, a).x	Returns the upper or lower geo bounds. For x, either 'lo' or 'hi' has to be specified

Syntax	Result
domain(\$c, a, c)	Returns the geo (lo, hi) bounds for axis a in CRS c
domain(\$c, a, c).x	Returns the upper or lower geo bounds for axis a in CRS c. For x, either 'lo' or 'hi' has to be specified
crsSet(\$c)	Returns a set of CRS identifiers
imageCrs(\$c)	Returns the grid CRS (CRS:1)
nullSet(\$c)	Returns a set of null values
identifier(\$c)	Returns the name of the coverage
describe(\$c, "application/json", "outputType=GeneralGridCoverage")	Returns a coverage description for a requested coverage without the range set in JSON
describe(\$c, "application/gml+xml"))	Returns a coverage description for a requested coverage without the range set in GML

Table 2.4 Probing functions not yet implemented in rasdaman. Adapted from (OGC 08-068r3, 2021, Chapter 6.2 Table 3)

Syntax	Result
value(\$c,p)	Returns the coverage grid point ("pixel"), "voxel",...) values, of data type rangetype(\$c) for all $p \in \text{imageCrsDomain}(\$c)$
dimensionList(\$c)	Returns an unordered list (i.e., set) of all of the coverages dimension names
rangeType(\$c)	Returns the data type of the coverage's grid point values, given as a set of pairs of field Name and (atomic) data type

rangeFieldType(\$c, f)	Returns the data type of one coverage range field, given as some atomic type name
rangeFieldNames(\$c)	Returns a set all of the coverage's range fields names
nullSet(\$c,r)	Returns a set of all values that represent null as coverage range field value for all $r \in \text{rangeType}(\$c)$
interpolationDefault(\$c,r)	Returns the default interpolation method per coverage field for all $r \in \text{rangeType}(\$c)$
interpolationSet(\$c,r)	All interpolation methods applicable to the particular coverage range field for all $r \in \text{rangeType}(\$c)$; must list at least the default interpolation method
interpolationType(im)	Interpolation type of a particular interpolation method for all $\text{im} \in \text{interpolationList}(\$c)$
nullResistance(im)	Null resistance level of a particular interpolation methods for all $\text{im} \in \text{interpolationList}(\$c)$

2.4.5 Further WCPS constructs

Central WCPS functionality comprises trimming and slicing as well as conditional evaluation. On top of that, two kind of operations are particularly noteworthy when handling an EO data cube: Operations for constructing coverages and operations for summarizing or condensing them. Furthermore, special operations like scaling and reprojection can be applied (Baumann, 2010). Examples for all of these operations can be found in Appendix A.

3 Methods

3.1 Hardware setup and rasdaman installation

Several ways of installing rasdaman are offered:

- Prepackaged installation for CentOS or Debian/Ubuntu
- Downloading a Virtual Machine with a fully configured system with a rasdaman install ready to run
- Downloading and compiling yourself

For this thesis, a prepackaged rasdaman v10.0.0-beta3 install for Ubuntu 20.04 'Focal Fossa' was chosen, as it provided more WCPS functionality than the latest stable rasdaman release of version 9.8 from 25.07.2019. The stable release of rasdaman v10.0.0 did only come out at the 18.03.22 and is now also available for newer Ubuntu version 22.04 (rasdaman team, n.d.). The data cube tested was deployed locally on a laptop with a 11th Gen Intel® Core™ i7-1165G7 @ 2.80GHz processor with 8 threads, a disk storage capacity of approximately 512 GB and memory of 32 GiB.

3.2 Data sets

A small data set was sufficient for the test purposes of this thesis. As test area, one Sentinel-2 granule (the minimum indivisible partition of a Sentinel product, also known as tile) was selected, extending from approximately 11.9469° E to 13.5885° E and from 54.0214° N to 55.0388° N. As can be viewed in Figure 3.1, the area covered by the granule comprised a part of the German Baltic coast, stretching approximately from the city of Rostock to the island of Hiddensee. In some images a large part of the island of Rügen can be seen, in others, this area is not included anymore. In the North-East corner, parts of the Danish islands Møn and Falster are visible but the most central and prominent feature is the German peninsula Fischland-Darß. The red outline in Figure 3.1 marks the borders of an area that has been used as an Area of Interest (AOI) for some of the sample queries presented in the result chapter.

13 Sentinel-2 scenes were downloaded from the Copernicus Open Access Hub for a time period from 09.08.2015 to 13.02.2021 (*Open Access Hub*, 2022). The images were selected manually. With the exception of 2015, the year in which Sentinel-2A, the first satellite of the Copernicus EO program was launched on June 23, a 'winter' image and a subsequent 'summer' image were selected for each year to introduce some form of regularity to the data cube. With the exception of one scene that has been taken at the end of March, the winter pictures are from the months of January and February. The summer images are from June with one exception where the scene has been taken in May. The choice of scenes to consider also depended on other factors. For example, care was taken to ensure that some winter images contained snow and ice in order to have some easy to spot land cover variability in the scenes. In addition, some images were desired to have some clouds for testing purposes, but the images should not have too much cloud cover overall. Sentinel-2 images are offered at Copernicus hub at two different processing levels. Images that are processed to meet level L1C are geometrically corrected to cartographic geometry and radiometrically corrected to Top-of-Atmosphere (TOA) reflectance (*Sentinel-2 User Handbook*, 2015). L2A products additionally went through an atmospheric correction and represent Bottom-of-Atmosphere (BOA) reflectance. Since automatic processing of L2A was not available in the first years of the Sentinel-2 mission, the BOA products are only available for more recent images. Thus, all scenes were acquired in their L1C variant. As SIAM™ can produce a preliminary classification from both L1C and L2A data, this was not a problem. The size of a L1C satellite scene comprising 13 bands and an additional true color image (TCI) varied between 443,8 MB and 724,3 MB with individual bands having sizes from 1,1MB (usually Band 10, 60 m Short Wave Infrared with Central Wavelength 1375 nm) to 135,6 MB (TCI). To save storage space, not all bands were imported into rasdaman. Only the bands with a spatial resolution of 10 m were completely loaded into the DBMS. They required 12.54 GB of storage space. For the bands with a spatial resolution of 60 m, only one scene was imported to save disk space. This accounted for 20.09 MB of storage used. Bands with a resolution of 20 m were not imported at all.

All Sentinel-2 images downloaded were undergoing a semantic enrichment process with SIAM™ which was done at Z_GIS. While by default several products are produced, only layers categorized into 33 semi-concepts were used for this thesis. The size of each data

set with 33 categories was 120.56 MB so the total semantically enriched data loaded into rasdaman had a size of 1.57 GB. For all images the CRS used was EPSG 32633. All 13 Sentinel-2 scenes and their semantically enriched counterparts can be viewed in Appendix B.



Figure 3.1 Area of the Sentinel-2 granule used. The red-bordered polygon marks the AOI used in query examples that demonstrate WCPS clipping functionality

3.3 Importing images with `wcst-import.sh`

Satellite images in several formats such as TIFF, netCDF, GRIB and others can be imported into the rasdaman array store using WCS-T functionality. For easier import, rasdaman offers the `wcst_import.sh` utility. It hides the more complex underlying WCS-T requests from the user and additionally maintains the geo-related metadata in rasdamans petascopedb. The import process is shown in Figure 3.2. The rasdaman documentation describes the `wcst_import.sh` tool to be based on the following two concepts:

- **Recipe** - A recipe defines how a set of data files can be combined into a well-defined coverage (e.g. a 2-D mosaic, regular or irregular 3-D timeseries, etc.);
- **Ingredients** - A JSON file that configures how the recipe should build the coverage (e.g. the server endpoint, the coverage name, which files to consider, etc.)' (rasdaman team, 2022, Chapter 5.7.1).

Recipes are written in python and contain classes that define how to validate and read input data containing specifications for bands, axes, metadata, CRS, etc. Rasdaman already provides some predefined recipes, e.g for importing regular and irregular time series and for importing Sentinel-1 or 2 data. Furthermore, users can create their own custom recipe using python. Ingredients are JSON files that contain parameters in which input options have to be set. An ingredient file contains several sections. In the config section, the service URL of the server on which the coverages shall be offered is specified and the importing behavior can be influenced. For example, data can be imported blockwise or file by file. Furthermore, there is an input section where the coverage ID and the paths to the input files have to be defined. There is also a recipe section in which the recipe used for the import is specified. Optionally, there might also be a hook section in which shell commands that shall run before or after the data import can be set. Ingredient files are run together with a bash script using the shell command in Listing 3.1 (It can also be run in the background as a daemon by adding the flag '`--daemon start`'). While some ingredient options are found and need to be set in every ingredient file, the availability of other additional options is dependent on the recipe file the ingredient file is chosen to work with.

```
$ wcst_import.sh path/to/my_ingredients.json
```

Listing 3.1

Satellite data loaded into rasdaman are stored in a (multidimensional) array also known as collection. New data can be added to an existing collection as long as its data type, spatial resolution and CRS match the existing collection. A geospatial collection complies with the OGC coverage definition cited above (see Chapter 2.3) and can thus be regarded as a coverage (Rasdaman also allows for the import of non-geospatial raster data using its native query language rasql. Such rasters are not considered to be coverages). It can also be called a data cube. However, the prototypical data cube designed for this work is intended to contain multi-spectral Sentinel-2 data as well as categorical SIAM™ data. Can these data sources with their different types be merged into one cube at all? Indeed, rasdaman supports the creation of custom composite data types. Theoretically, it would thus be possible to create a collection with values that include a custom type with multi-spectral band values as well as a categorical value. However, creating such a cube would be impractical, as it would lead to a large overhead in preparation of the data imported. Fortunately, this is not necessary. It is sufficient to load the images whose characteristics match into a common collection. In the end, several collections form a 'virtual' data cube in rasdaman. The use of WCPS allows to retrieve, process and fuse the data from the different collections by providing adequate functionality. This concept can be compared to views in relational DBMS (Baumann *et al.*, 2021).

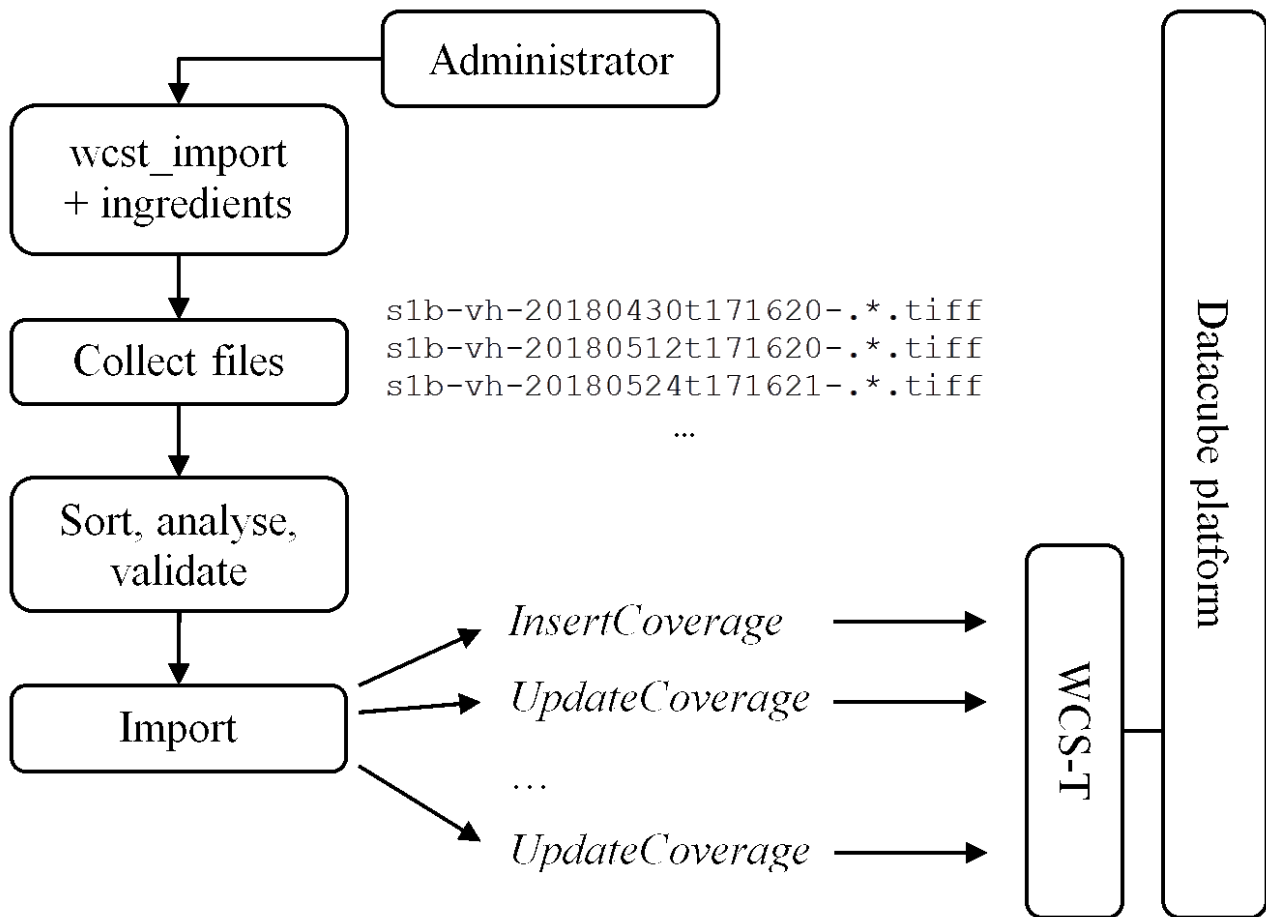


Figure 3.2 Data importing process with `wcst_import.sh`. Retrieved from (rasdaman team, 2022, Chapter 5.7.1 Figure 5.1).

3.4 Querying categorical data with WCPS in rasdaman

Working with SIAM™ semi-concepts in the rasdaman implementation of WCPS has been tested by investigating several queries that are potentially useful for displaying land cover data, analyzing it or preparing it for further analysis. Test queries were developed along the following six topics:

1. Descriptive statistics of a 2D image

- Calculating histograms and statistical values for a 2D image sliced along the time axis.
- Potential use case: Gathering general information about the occurrence of land cover classes that have been derived from SIAM™ semi-concepts for a selected date. For example, calculating the occurrence of snow for the 21.02.2017.

2. Simple selection and display of (composite) categories

- Displaying a selection of SIAM™ categories in 2D maps. Composite categories means that more than one SIAM™ semi-concept are combined to form a composite class potentially representing a real-world land cover class.
- Potential use case: Creating an overview map for a land cover type of interest. For example, select all areas classified as having strong or average vegetation with high NIR.

3. Fusing Sentinel-2 and SIAM™ data sets

- Displaying SIAM™ data on top of a Sentinel-2 background in 2D maps.
- Using SIAM™ data to select Sentinel-2 data and vice versa which can be used for further analysis.
- Potential use case: Creating a map that shows selected land cover classes in a specified color on top of a custom Sentinel-2 band combination. For example: Select all categories that are not related to vegetation and display them black so as not to take the focus of the vegetation classes displayed as a Sentinel-2 color infrared band combination.

4. Investigating categorical trajectories

- Combining different time slices of the same SIAM™ coverage to gain insights about land use change trajectories.

- Potential use case: Tracing how certain land cover types have changed into other land cover types. For example: Check where areas classified as deep water in 2015 have changed into areas classified as shallow water and where they changed into land.

5. Time series analysis

- Creating time series of categorical change and/or calculating descriptive statistical values condensed along the time axis.
- Potential use case: Calculating land cover statistics per time period. For example, generating a time series with the yearly maximum snow cover or producing a map that shows areas frequently covered with snow.

6. Edge detection based on categorical data

- Applying a Sobel operator to detect edges of land use classes that are based on SIAM™ categories.
- Potential use case: Extracting borders between land cover classes. For example, extracting a shoreline.

As part of the results presented in the respective chapter, a catalog of sample queries tested can be found in Appendix D. For many queries only a subset of the total satellite scene was processed to speed up response times. The focus of this thesis was on queries that return 1 and 2 dimensional data. Queries with 3 dimensional output were not tested.

4 Results

4.1 Building the cube

For this thesis, three collections have been created by running the import script based on the data sets presented in Chapter 3.2. Two collections consisted of Sentinel-2 data: one containing the four bands with 10 m resolution (Band 2-4 and Band 8) and one with the three bands at 60 m resolution (Band 1, Band 9, Band 10). To import them, the Sentinel-2 recipe already predefined in rasdaman was used. An ingredient file in JSON format was created by only slightly adapting the Sentinel-2 sample ingredient file featured in Chapter 5.7.9 of the rasdaman documentation (rasdaman team, 2022). The changed parameters were 'coverage_id', 'paths', 'resolutions' and 'levels'. The parameter 'crss' was left blank in order to import the Sentinel-2 scenes in the CRS they were delivered in. A sample ingredient file for importing the scenes with the 10 m and 60 m resolution bands at once can be found in Appendix C.1. The third collection was the SIAM™ coverage consisting of only one band. The generic 'general coverage recipe' was chosen to import the SIAM™ data. It offers a multitude of setting options that have not been fully investigated in this thesis. The ingredient file used for importing SIAM™ data can be found in Appendix C.2. According to the test character of this work, the raster data was imported to localhost. As storage space was scarce, all 13 scenes taken from different dates between 2015 and 2021 were only imported for the SIAM™ and Sentinel-2 10 m resolution collections. For the Sentinel-2 coverage with the 60 m resolution only the one date that has been used in the test queries was imported. The fact that each of the collections that make up the virtual cube can be easily changed without affecting other parts proved to be advantageous here. After importing, the resulting 'virtual cube' was a 3D cube with two spatial and a temporal axes made from three actual cubes.

Problems encountered and caveats

It is important to note that in the rasdaman community version, the chronological order of image recording dates must be adhered to when importing coverages. Images with timestamps earlier than the image with the most recent timestamps fail to be imported, as this function is only supported in the commercial rasdaman version (Pham Huu, 2022).

A known limitation is that `rasdaman` can currently not deal with categorical data. Categorical IDs can thus only be expressed in a number format and the meaning of an ID can only be transferred in the metadata of the coverage. It is also worth mentioning that the explanation of some input parameters in the ingredient files was not perceived as simple to understand. For example, the need to set the `axis order` parameter option remained unclear because omitting the parameter or arbitrarily changing the axis order when importing a sequence of 2D scenes did not appear to affect the result. This may be different when importing a 3D `netCDF` file, but has not been tested because it was beyond the scope of this work.

4.2 Example queries tested

4.2.1 Structure of the results presented

With a few exceptions where queries are depicted in their entirety, only the essential parts are shown in the WCPS listings of the subsections of chapter 4.2. If a sample query with a query ID is referenced, its complete syntax can be found in Appendix D. For each query, the result is also displayed there and peculiarities and caveats are pointed out. The three coverages that make up the virtual data cube of this thesis are referred to as 'siam', 'sen2_10m' and 'sen2_60m' in the queries shown in the rest of this chapter and in Appendix D.

4.2.2 Descriptive statistics of a 2D image

For each date in the data cube, a 2D scene could be generated with WCPS, optionally trimming the latitude and longitude for the requested area (Listing 4.1). The specified MIME type can optionally be shortened to 'tiff' instead of 'image/tiff' (or to 'csv' instead of 'text/csv').

```
for $s in ( siam )
return encode(
  $s[ E(305863:315593), N(5996151:6008781),
  ansi("2017-06-19T00:00:00.000Z")], "tiff")
```

Listing 4.1

Several descriptive statistical values can be created to describe the content of a scene. The total pixel count of a category that is relevant for calculating the area covered can simply be queried by using a conditional statement on the coverage subset in the return clause (Listing 4.2). As SIAM™ categories are semi-concepts and more often than not need to be combined to form a real-world class, 'composite' categories are generally generated by querying multiple categories together using an 'or' operator (Listing 4.3).

```
for $s in (siam)
return encode (count(
($s[ansi("2017-01-10T00:00:00.000Z")]=29)), "csv" )
```

Listing 4.2

```
for $s in (siam)
return encode (count(
($s[ansi("2017-01-10T00:00:00.000Z")]=21 or
$s[ansi("2017-01-10T00:00:00.000Z")]=22) ), "csv" )
```

Listing 4.3

It is possible to use arithmetic operations on the pixel count to convert it to m² or some other square measure (**Q1**). The count of a category can also be performed for an AOI only by using a clip function with the vertices of the target polygon specified in the text markup language Well Known Text (WKT) (**Q3**). The query for getting back the share of a single category is shown in **Q2**. Values representing a histogram showing the pixel count for a category in the image can also be derived. For this, a coverage constructor iterating over the IDs of the 33 categories of the SIAM™ layer and counting each pixel per bucket can be used. The part of the query showing only the coverage constructor can be seen in listing 4.4, the full query is presented in **Q4**.

```
coverage histogram
over $bucket x( 1 : 33)
values count($s[ansi("2017-01-10T00:00:00.000Z")] = $bucket)
```

Listing 4.4

The values returned come without axis information (unless the 'diagram >>' command precedes the query in the OWS console, as described in Chapter 2.4.2). This needs to be

considered when interpreting the result. In order to query the pixel count of the 'no data' value (SIAM™ category 255) it is possible to change the bucket span from 1:33 to 1:255, but the result will contain a lot of empty buckets in between. Iteration is only possible over a range, not over fixed indices. A workaround for building a histogram based on categories whose IDs are not neighbors is iterating over the number of categories and querying the category indexes via a switch statement and conditional phrases as shown in listing 4.5 (Q5).

```
coverage histogram
over $bucket x( 1 : 2)
values switch
case $bucket=1 return count(($s[ansi("2017-01-10T00:00:00.000Z")] = 25))
default return count(($s[ansi("2017-01-10T00:00:00.000Z")] = 29))
```

Listing 4.5

By combining the coverage constructor of Listing 4.4 with a division operation and a Boolean expression that excludes pixels that are not within the 1:33 category range, the percentage of each category of the overall scene can be queried (Listing 4.6 and Q6).

```
coverage histogram
over $bucket x( 1 : 33)
values (count($s[ansi("2017-01-10T00:00:00.000Z")] = $bucket)) /
count(($s[ansi("2017-01-10T00:00:00.000Z")]>0) and ($s[ansi("2017-01-
10T00:00:00.000Z")]<34))*100
```

Listing 4.6

4.2.3 Simple selection and display of (composite) categories

A simple, relatively intuitive query is the basic selection of categories for a selected timestamp (Q7). The query can be extended by a switch statement to color code the requested categories as shown in listing 4.7. For each category, the output color is defined by defining the respective RGB value (Q8). As with descriptive statistics, the queries for a custom AOI can be done by using the clip function (Q9).

```
((unsigned char)
switch
case $$[$sub]=25 return {red:255; green: 255; blue: 255}
case $$[$sub]=29 return {red:0; green: 255; blue: 255}
default return {red: 0; green: 0; blue: 0}
)
```

Listing 4.7

The importance of keeping track of data types in queries is paramount to working with rasdaman. The result of a Boolean expression is returned as true and false values and needs to be explicitly cast to unsigned character here, as rasdaman otherwise throws the error message shown in listing 4.8.

```
<ows:ExceptionText>Execution error 457 in line 1, column 8, near token encode:
MDD has a non-char cell type, cannot apply color palette table.</ows:ExceptionText>
```

Listing 4.8

4.2.4 Fusing Sentinel-2 and SIAM™ data sets

There are several ways of mixing a Sentinel-2 coverage with a SIAM™ coverage in a 2D output map. The overlay function is probably best used for displaying selected and color-coded SIAM™ categories on a Sentinel-2 'background' consisting of a RGB image (**Q10**) or any other custom chosen band combinations. While the operation name 'overlay' suggests that there are two coverages lying on top of each other, it should be noted that in fact there is only one output coverage. The color values of the selected SIAM™ categories assigned to the RGB channels in the switch statement blend in with the Sentinel-2 band values. The notion 'background' might therefore be misleading. The mixing of values in one result coverage is also the reason that the coverages to be fused have to have the same data type. While the SIAM™ coverage is of type 'Byte - Eight bit unsigned integer' with a maximal value of 255, the Sentinel-2 coverage is of type 'UInt16 - Sixteen bit unsigned integer' with the maximal values for each band ranging between 4998 (Band 3) and 7849 (Band 4). To fuse the two coverage, both data sets need to be cast to the same data type. For example, both coverage could be cast to unsigned short (16 bit) like in listing 4.9. In **Q10** and **Q11**, both coverages are cast to unsigned char. A histogram stretch

formula is applied to all Sentinel bands to allow for a translation of 16 bit values into a 8 bit color space. It should be noted that as the color values assigned to the categorical values blend in with the continuous satellite values, they have an influence on the image statistics and thus on the way the 'background' is displayed when opening the coverage in a GIS.

```
((unsigned int
switch
case
$c[$sub]=21 return {red:1; green: 1; blue: 255}
default return {red: 0; green: 0; blue: 0}
)
overlay
(unsigned int){
red: $s2[$sub].B4 ;
green: $s2[$sub].B3;
blue: $s2[$sub].B2
}
```

Listing 4.9

When combining bands from coverages that have different pixel resolutions, the scaling function needs to be applied as shown in **Q11**. Here, a Sentinel-2 band combination of B4, B3 and the coastal Aerosol band B1 was chosen as 'background'. Contrary to B3 and B4, the coastal Aerosol band has a spatial resolution of 60 m, is thus stored in another coverage and is therefore resampled with the scale function to match with the 10 m resolution bands.

When using the overlay function it is possible to assign values to SIAM™ categories that can later be excluded from being displayed by defining them as Null-values in a GIS. In this way, the SIAM™ categories, similar to a cookie cutter, can be used as a preselection for Sentinel-2 areas that should be subjected to further analysis. However, there is a more straight-forward way to use SIAM™ categories to achieve this. Two coverages can simply be combined by multiplying a Boolean mask that is the result of some condition based on SIAM™ categories with a Sentinel-2 scene of the same extent like in listing 4.10 (**Q12**). Of course, it is also possible to derive only SIAM™ categories for areas for which a condition based on a Sentinel band is true. For example, all SIAM™ categories for locations where a specified Sentinel-2 band has a value that is above a certain threshold could be queried (**Q13**). Chaining conditions by multiplying multiple masked coverages with an original

coverage is also possible, as **Q14** shows. Here, only values of Band 2-4 of the Sentinel-2 coverage are shown for pixels which are not part of SIAM™ category 21 or 22 and also do not possess B4 values greater or equal to 500.

```
(unsigned char) (($s[ansi("2021-06-18T00:00:00.000Z")] = 21) *  
($s2[ansi("2021-06-18T00:00:00.000Z")]))
```

Listing 4.10

4.2.5 Investigating categorical trajectories

Rasdaman allows for a quick investigation of land cover change. Analogously to multiplying a SIAM™ coverage that is masked using a Boolean expression with a Sentinel-2 image, the masked coverage can be multiplied with itself at different timestamps. In **Q15**, it can be seen that it is easily possible to find out to which categories pixels belong to in the newest image of the data cube that have been deep water in the oldest image by using the query syntax of listing 4.11.

```
($s[$sub_lo] = 21) * $s[$sub_hi]
```

Listing 4.11

Using the subsetting function to slice the coverage at different timestamps in combination with the switch statement, pixels can be color coded according to their land cover change trajectory like in listing 4.12. For example, pixels that have been classified as deep water in the earliest satellite scene and became shallow water can be assigned a different color than those that became land or those that stayed the same (**Q16**, **Q17**). Many different trajectories could be distinguished in this way, but with each additional case, the query becomes more confusing, at least for human readers (**Q18**). It should be noted that the overlay function can be used to achieve a similar goal (**Q19**), but the syntax for this purpose is not as clear and straight-forward compared to the switch statement. The final land use change trajectory example shown in **Q20** demonstrates how the switch statement in combination with time slices can be used to compare at which time intervals deep water changed into land. Two periods are considered: Period one starts with the earliest image in the cube and ends in 19.06.2017. Consequently, period two starts the day period one ends

and ends with the latest image in the cube. Pixels that have changed from deep water into shallow water during the first period and on to land in the second period are colored yellow. Those that were changing to land in the first period right away and stayed land in the second period are displayed in red.

```
switch
case $s[$sub_lo] = 21 and $s[$sub_hi] = 21 return {red:0; green: 0; blue: 100}
case $s[$sub_lo] = 21 and ($s[$sub_hi] = 22 or $s[$sub_hi] = 23 or
    $s[$sub_hi] = 24 return {red:0; green: 100; blue: 200}
default return {red: 0; green: 0; blue: 0}
```

Listing 4.12

4.2.6 Time series analysis

For each geographic coordinate in a data cube represented by a pixel, it is possible to calculate how often it belonged to a particular SIAM™ (composite) category for all the timestamps within a time interval by summing up each occurrence of the category along the timeline using a condenser function as demonstrated in listing 4.13. The result is a 2D map with each pixel containing its value count for the specified category. For example, by condensing the water occurrence of several years in one map, we get the frequency with which water was observed per pixel (**Q21**). This water count example with rasdaman has been shown in scientific literature before (Sudmanns *et al.*, 2017).

```
condense +
over $t ansi(imageCrsDomain($sub[ansi("2017-06-19T00:00:00.000Z":
"2021-06-18T00:00:00.000Z")], ansi))
using $sub[ansi($t)]=21 or $sub[ansi($t)]=22 or $sub[ansi($t)]=23
or $sub[ansi($t)]=24
```

Listing 4.13

The occurrence of a category in a scene over a period of time can be viewed by combining a coverage constructor with the count condenser function to form a time series (Listing 4.14). As with the histograms describing a 2D image for a specified time slice, this can be done for both absolute values and shares (**Q22**, **Q23**). By simply enclosing the time series

constructor with a maximum/minimum/ average pixel count function as depicted in listing 4.15, one can derive the respective aggregate value (**Q24**, **Q25**).

```
coverage timeseries
over $p ansi(imageCrsDomain($s[ansi("2017-01-01T00:00:00.000Z":
"2020-12-31T00:00:00.000Z")], ansi))
values count($s[ansi($p)] = 29)
```

Listing 4.14

```
max(coverage timeseries
over $p ansi(imageCrsDomain($s[ansi("2017-01-01T00:00:00.000Z":
"2017-12-31T00:00:00.000Z")], ansi))
values count($s[ansi($p)] = 29))
```

Listing 4.15

Unfortunately, *rasdaman* lacks an index querying functionality. Only the pixel/voxel values can be queried based on axis index values, but the opposite is not true. One can neither query the timestamp or timespans for which a pixel/voxel condition is true, nor (geographical) coordinates or bounding boxes. It is thus not possible to find out at which date the maximum categorical count was observed directly. Here the user has to resort to the second best solution and stick with the time series to find out when the maximum happened. It should be noted, that especially for irregular data cubes with many timestamps this might not be a trivial task as the values delivered come without any time indices. Therefore, the information has to be extracted elsewhere (e.g. by using the WCPS describe function) and applied separately.

To the author's knowledge it is not possible to form queries based on the temporal units of the time index directly. We cannot address a certain month precisely which would help when building a query that returns a time series based only on the categorical count for ice pixels for each February from 2016 to 2021. It is also not easily feasible to build a time series from the maximum ice cover per year for the same timespan. However, depending on the cube design, a workaround can solve this problem. For example, in a regular data cube comprising exactly one image per month, retrieving a time series over several years which is always showing the target value count of the same month can be done by using a coverage constructor. The newly constructed index in the over clause of the coverage constructor needs to be set to match the total numbers of years that are of interest. In the

values clause, an arithmetic operation where each year index \$t is multiplied by 12 and added by an additional offset can then be used to find the right month to slice for each year in the original coverage. Choosing the right offset value is important in order to get the values of the right month. Listing 4.16 would return February values, if the first image in a regular data cube was taken in January. However, if the first image stored was taken in July, it would return an August value. Hence, the queries are not transferable to another cube without adapting them to this cube's temporal settings. It should also be remembered that the query can fail if no image data is available for a month in the last year of the query period. Therefore, it is important to always know the boundaries of the temporal axis domain of a data cube.

```

for $s in (oneImagePerMonthCube)
return
encode(coverage timeseries
over $t year(0:9)
values count($s[ansi($t*12+1)] = 29)
,"text/csv")

```

Listing 4.16

As the irregular cube built for this thesis was not suitable for querying the snow trajectory for one month over several years, an alternative query has been produced to present an example using the same query structure as in Listing 4.16. The test cube can be understood as alternately having a summer and a winter month. So apart from the first year, every year comprises two seasonal scenes (instead of 12 months). Table 4.1 gives an overview of this division. Thus, when iterating over the 6 years in the cube, the query structure of Listing 4.17 is used in **Q26a**. Of course, aside from pixels that have probably been misclassified, there won't be a high number of snow pixels for the summer season. But to get the values for the winter month, we just have to change the slice on the temporal axis 'ansi' to (\$t*2+1) as was done in **Q26b**. Attention should be paid to stay within the domain boundaries of the temporal axis for the query to work.

Table 4.1 A cube with seasonal regularity

ansi	index	season
"2015-08-09T00:00:00.000Z"	0	Summer
"2016-01-06T00:00:00.000Z"	1	Winter

"2016-05-12T00:00:00.000Z"	2	Summer
"2017-01-10T00:00:00.000Z"	3	Winter
"2017-06-19T00:00:00.000Z"	4	Summer
"2018-01-07T00:00:00.000Z"	5	Winter
"2018-05-30T00:00:00.000Z"	6	Summer
"2019-02-16T00:00:00.000Z"	7	Winter
"2019-06-26T00:00:00.000Z"	8	Summer
"2020-03-27T00:00:00.000Z"	9	Winter
"2020-06-15T00:00:00.000Z"	10	Summer
"2021-02-13T00:00:00.000Z"	11	Winter
"2021-06-18T00:00:00.000Z"	12	Summer

```
coverage timeseries
over $t year(0:6)
values count($s[ansi($t*2)] = 29)
```

Listing 4.17

Using this schema, the query can be adapted to get an aggregate value such as a maximum/minimum or average of timestamps grouped together into some temporal unit, e.g. a year, a season or a month. Listing 4.18 shows how the maximum snow per year is calculated in **Q28**.

```
coverage timeseries
over $t year(1:6)
values max( count($s[ansi($t*2)] = 29) , count($s[ansi($t*2+1)] = 29) )
```

Listing 4.18

Listing 4.19 shows this concept applied to a hypothetical regular cube containing one image per month for the maximum snow count in the winter season. The winter season here consists of the months December, January and February. The first image of the imaginary cube with the temporal image was taken in January.

```

for $s in (oneImagePerMonthCube)
return
encode(coverage timeseries
over $t year(0:9)
values max(count($s[ansi($t*12+0)] = 29), count($s[ansi($t*2+1)] = 29),
count($s[ansi($t*12+11)] = 29))
,"text/csv")

```

Listing 4.19

Irregular cubes without any patterns are difficult to query for time series, as there is no real understanding of the time axis currently implemented in rasdaman. It should be noted that supposedly regular data cubes are not always so in reality. A data cube containing one satellite image scene per day has 356 images in a normal year and 366 in a leap year. Months have 28-31 days. That could make it complicated to summarize the images in a yearly or monthly value. A potentially viable approach could be to employ switch statements to define different calculating branches, similar to the case distinction which returned condenser values in **Q5**. However, the feasibility was not tested in this work.

4.2.7 Edge detection based on categorical data

The Sobel operator is an edge detection algorithm used to find vertical and horizontal edges in an image. An example of the Sobel operator working at a RGB image can be found at https://standards.rasdaman.com/demo_convolution.html (Jacobs University & rasdaman GmbH, 2020).

$$G = \sqrt{G_x^2 + G_y^2} \text{ with } G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \text{ and } G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Figure 4.1 Sobel operation. A = 2D input image, G_x = convolution kernel 1, G_y = convolution kernel 2, G = filtered 2D output. Figure taken from (Baumann et al., n.d.).

Two 3x3 convolution kernels with values representing different weights are moved in a moving window style across the input image, in order to add the weights to each input pixel. Usually, the Sobel operator is run on input images with values representing different levels of brightness and detects edges where these values differ the most. It was tested how it could work together with SIAM™ categories by trying to extract the shoreline for an

extent of the Sentinel-2 granule that covers a small part in the east of the most northern point of the Fischland-Darß Peninsula (Figure 4.2). The SIAM™ scene investigated was from the 19.06.2017. For this query, everything that is dark blue (21) and turquoise (30) is considered to be water³, all other categories shown are thought to be land. The idea to derive the shoreline was to mask the water values and apply the Sobel operator to the result. Three different approaches were investigated.

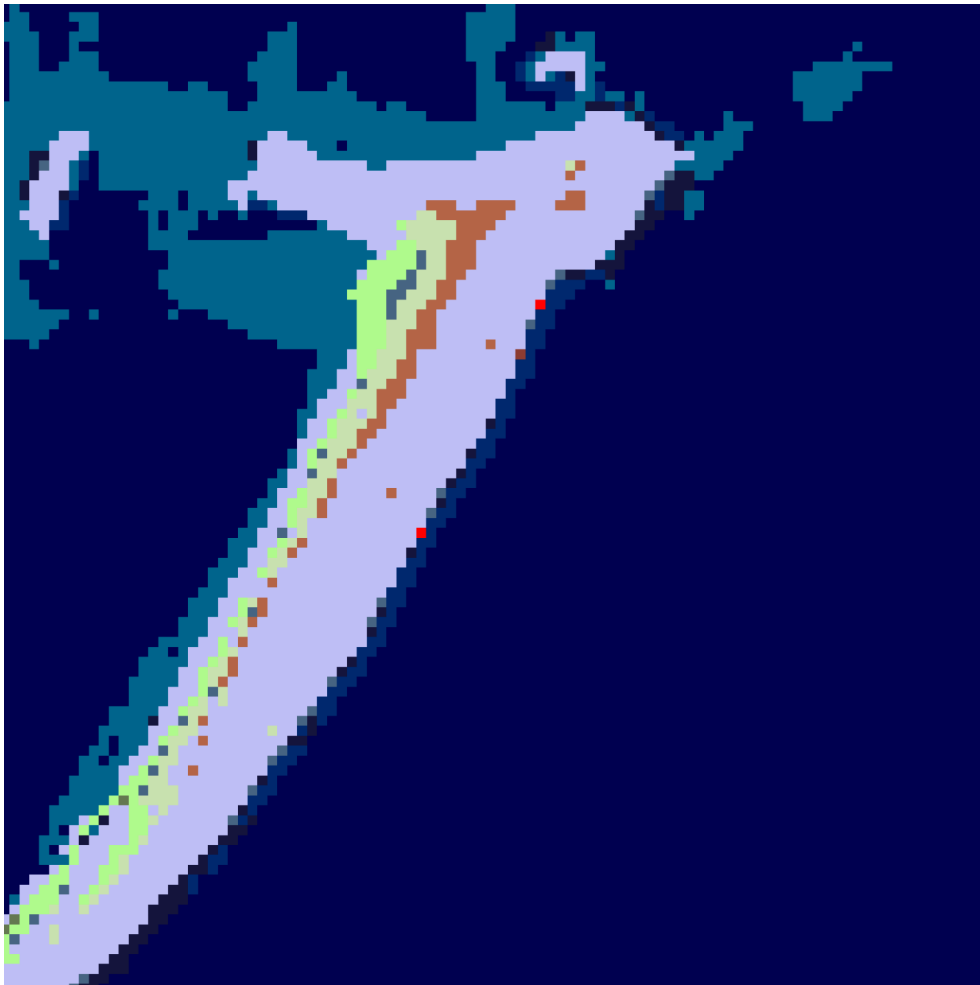


Figure 4.2 Subregion used to test the Sobel operator in WCPS on SIAM™ semi-concepts in order to discriminate land from water

3 SIAM™ category 30 actually corresponds to the semi-concept 'Shadow snow' (SHSN). It was prominent in the classification product of the 19.06.2017 in areas that are most likely shallow water. As the focus of this thesis is on the mechanics of WCPS in combination with SIAM™ semi-concepts, and not on the evaluation of semi-concepts themselves, this has not been further investigated.

The first approach was to mask the water categories by applying conditional phrases in the where-clause in the coverage condensers. When working with categorical data represented by integers, the integer is falsely interpreted as a brightness value. So when applying the Sobel operator to a coverage by using a condition in the where-clause of a coverage condenser that leads to only pixels of category 21 being considered (**Q29**), the filter's weight is applied to pixel either of value 0 or 21. This works well enough as long as only category is incorporated. But when using two or more categories in the condition of the condenser's where clause in order to simulate a composite category, this likely poses a problem. Presumably, the Sobel operator detects edges between categories that we would lump together as one, with the edges getting more defined the bigger the difference between categories get. For example, when extracting a shoreline and consider the categories 21 and 30 as water and everything else as land, the Sobel filter might find edges (= fake coastlines) between those categories as shown in **Q30**. Better results could be achieved with the second approach by employing the Boolean condition to select the categories in the using-clause of the coverage constructor in order to multiply the kernel with a mask (**Q31, Q32**). This is similar to what has been done when fusing SIAM™ and Sentinel-2 data or when querying land use change trajectories without the overlay function. The third approach tested failed. Here, approach a mask coverage has been predefined using a coverage constructor in the let statement using either a switch statement (**Q33a**) or a simple condition (**Q34a**). While this was possible, the next step of running the Sobel filter on the mask was unsuccessful in both cases because they ended up in long running WCPS queries that eventually lost the connection to the rasdaman host and returned an error (**Q33b, Q34b**). As the data cube was running on a local machine with limited resources, this might have contributed to this outcome. As can be seen in **Q33b** and **Q34b**, it was tried to reduce the query complexity by using only one kernel for the Sobel operator, but to no avail. However, the reason remained unclear. A further investigation of this issue was not carried out, as this would have gone beyond the scope of this work. It is important to note, that always when a new coverage is created by using a coverage constructor like an image that has gone a Sobel filter operation, the geographic CRS is lost and it can therefore not be placed correctly in a GIS.

5 Discussion

The results showed that it is possible to set up a semantically enriched data cube with SIAM™-classified data in the rasdaman array DBMS and query it with WCPS. To subsume the findings, they are discussed by structuring them into strengths, weaknesses, future opportunities and potential obstacles presented below.

Strengths

Importing Sentinel-2 data and SIAM™ data to the cube

- Importing the Sentinel-2 data and the SIAM™ data was straightforward as soon as it was clear which parameters need to be used and adapted in the ingredients file in order to load valid coverages. It was sufficient to slightly adapt some parameters and employ the general coverage recipe to import SIAM™ data as well as the Sentinel-2 recipe for the Sentinel-2 data. Both recipes are shipped with the rasdaman installation when installing from a preconfected package. Rasdaman's additional functionality for writing a custom recipe file did not need to be used.
- New collections containing data with different resolutions or different data types can be added to rasdaman anytime without changing the data structure of the other collections in the array store. They can be queried in combination with the other collections using data fusion syntax of WCPS. Therefore, the 'virtual cube' can easily be extended without breaking anything in the existing collections.

Querying functionality

- In a Geospatial Data Cube, descriptive statistics like pixel counts and shares for composites made from SIAM™ semi-concepts can be easily derived for a 2D coverage slice of a selected date. Returned results are either scalars or a list of histogram values in csv format. Calculating a histogram for categories whose IDs are neighbors is straightforward when employing a combination of coverage constructor and condenser functions. For categories with IDs not next to each other or composite categories, calculating a histogram is more difficult but can still be achieved.
- Composite SIAM™ categories can easily be displayed in a 2D map derived by slicing the coverage at a timestamp. If needed, data can be color coded using

conditional branching and shown on background Sentinel-2 data. WCPS provides convenient functionality to build and display custom band combinations from Sentinel-2 data. Conditional branching can also be used to compare different land cover change trajectories.

- Using conditional masking on a SIAM™ coverage sliced at a timestamp and multiplying the result with a slice of a different timestamp is a very powerful tool for land cover change trajectory observation. The same syntax can be used for preselecting a Sentinel-2 coverage based on SIAM™ categories (or vice versa). It is also very useful in a Sobel operator where a conditional mask based on selected semi-concepts can be multiplied by a convolution kernel. This makes edge detection based on composite SIAM™ categories possible that can for example be useful in shoreline detection.
- Rasdaman allows for neat condensing of the time axis to produce a 2D map of aggregated values for a given period of time. Time series can easily be produced for composite categorical counts and percentages provided that for each timestamp in the cube a data point shall be displayed. Time series that only take into account certain timestamps and skip others (e.g. a time series with February values for snow cover for the last ten years) or time series that should show aggregated values (e.g. showing the maximum snow cover for each year in the last ten years) can be produced with reasonable effort provided that the data cube has regular characteristics in the time period targeted.
- WCPS in rasdaman is currently actively developed and new helpful functionality is expected to be available in rasdaman with the full integration of WCPS standard 1.1.
- The query functionality provided together with the possibility to run rasdaman in a distributed system makes it likely that a big data cube similar to the Austrian Sentinel-2 data cube can be implemented in rasdaman.

Weaknesses

Importing Sentinel-2 data and SIAM™ data to the cube

- Importing files that are not in the right temporal order is not possible in the rasdaman community version.

- Currently there is no true support of categorical values in rasdaman, so information on categories have to be included in the metadata of a coverage
- The effect of some import parameters was not clear.

Querying functionality

- As WCPS does not natively support categorical data, care should be taken that rasdaman does not mistakenly interpret the ID of a category as a continuous cell value.
- Using the coverage constructor has the downside of not resulting in georeferenced images. In the future, the rasdaman developers plan to integrate a function that allows to set a geographic or temporal CRS to a coverage to comply with WCPS 1.1 (Baumann, 2022b).
- A functional limitation of WCPS is that currently, there is no way of querying the indices of a dimension axis. With such a functionality not available, users must forego the ability to query for timestamps and time periods on which a certain cell value can be observed. For example, the direct query 'Return all dates when more than 1000 pixels of category 29 (representing snow or water ice) have been present' is not possible.
- There is also no option of querying a month directly or aggregating values for a common time interval such as month or year. Workarounds are possible for regular cubes, but require good knowledge of a data cube with regard to its axes domain borders. For irregular cubes, a workaround might not always be possible.
- With increasing query complexity, WCPS syntax can get very verbose and thus difficult to read and write for humans.
- It is not yet clear until when the functionality from the new WCPS Standard 1.1. will be available in rasdaman.

Future Opportunities

- WCPS can be used flexibly for working with SIAM™ coverages. Queries build can be reused equivalent to the models build in Sen2Cube.at. The query possibilities have certainly not been exhausted in this work. In the future a repository with WCPS queries for SIAM™data could be created, much like the Sen2Cube.at knowledgebase. It might even be possible to build a translator to transform models

created in the Sen2Cube.at GUI into WCPS queries. This could make the GUI independent on the structure of the underlying data cube (provided the underlying cube implements WCPS)

- Extending WCPS functionality to query dimension axes based on SIAM™ composite categories or otherwise derived land cover classes would mean a large increase in potential applications. Coupled with real time understanding that allows for selecting and aggregating values for common time intervals would make it very powerful. Semantic content based image retrieval could be further developed to become ad hoc semantic content based image analysis.

Possible Obstacles

- Ad hoc investigation of data using WCPS is tedious for a user inexperienced with the language. For a user without programming background at all, making good use of WCPS might be difficult. The incentive to engage with query language in the future depends on how widely used WCPS will be. Only if it is implemented by many applications a true interoperability of systems will be achieved
- Theoretically, it should be possible with WCPS to fuse coverages from different servers and different data holders to generate combined results. However, it is not clear how coverages stored on different servers could be addressed in one query to be merged together.

Based on a small prototypical data cube, this thesis demonstrated that rasdaman is well suited to create a semantic data cube with SIAM™ data. It could be seen, that it has a flexible import mechanism that can be adjusted relatively easily to load SIAM™ raster data. On top of that, the variety of WCPS queries presented showed that rasdaman's WCPS in combination with SIAM™ semi-concepts enables a range of semantic queries that can be used and extended for SCBIR. A next step would now be to use the knowledge gained to build a semantically enriched 'big' rasdaman data cube containing a denser time series with more satellite scenes and set up on an external server or a group of servers to simulate use in a production environment. To further extend the code base for WCPS sample queries it would be helpful to gather common workflows and requirements in the field of land cover analysis from the remote sensing community and beyond to build

WCPS queries that are truly needed. The full potential of currently possible query constructs has not been investigated in this thesis. For example, queries that produce 3D results have not yet been taken into account. Further testing based on computational linguistic knowledge would need to be performed to get a good estimate of which query targets could be achieved with WCPS in combination with semantically enriched SIAM™ data sets. This could also help to identify gaps where the language itself could be further developed to enable or simplify useful queries. Interoperable semantic data cubes hold great potential both in terms of easier accessibility of remote sensing data and the generation of information from such data. Future research could look at how the whole process chain of semantic enrichment and even the derivation of real-world classes based on models could occur automatically in the data cube when new optical satellite imagery is added. The process chain could even be extended. On the basis of the derived real-world classes, e.g. metrics for landscape structure analysis could be calculated directly in the data cube.

6 Conclusion

This thesis showed the creation of a small prototype of a semantic data cube in the open source rasdaman community version. 13 Sentinel-2 scenes as well as their corresponding semantically enriched SIAM™ products were loaded into the rasdaman array database as collections. These collections form the basis of a 'virtual' data cube which can be queried using OGC WCPS. Several WCPS queries of varying complexity, targeted to work with SIAM™ semi-concepts and outputting zero to two dimensional data were demonstrated and discussed. Various useful constructs could be found. Basic retrieval of categories consisting of one or more SIAM™ semi-concepts as well as the calculation of categorical statistics and basic time series were fast and easy. Even though it can get very verbose, the WCPS switch statement allowing for case distinction was found to be very convenient for color coding different Land Cover or Land Cover Change categories derived from SIAM™ categories or customizing classes shown in a histogram. Another especially powerful construct in WCPS is the possibility to multiply a boolean coverage that is the result of a conditional phrase with another coverage to select AOIs. In combination with SIAM™ data this can be used to build a preselector for Sentinel-2 data (or vice-versa), select areas in a satellite scene of a newer image based on a selection in an older image

or detect edges, e.g. for shoreline extraction. A perceived shortcoming was the lack of an index querying functionality that would allow to return a date, a time interval or a geographic region based on selected pixel values. Furthermore, it is not possible to ad hoc aggregate values for common time intervals such as for a month or a year. To a certain degree, workarounds for this are possible provided the cube has some regular characteristics. Based on the insights gained with the prototypical data cube created in this work, it can be summarized that the rasdaman/SIAM™ combination holds great potential for the creation of accessible semantic EO data cubes and should be further investigated in the future.

References

- Augustin, H. *et al.* (2018) 'A Semantic Earth Observation Data Cube for Monitoring Environmental Changes during the Syrian Conflict', *GI_Forum*, 1, pp. 214–227. doi:10.1553/giscience2018_01_s214.
- Augustin, H. *et al.* (2019) 'Semantic Earth Observation Data Cubes', *Data*, 4(3), p. 102. doi:10.3390/data4030102.
- Baraldi, A. *et al.* (2006) 'Automatic Spectral Rule-Based Preliminary Mapping of Calibrated Landsat TM and ETM+ Images', *IEEE Transactions on Geoscience and Remote Sensing*, 44(9), pp. 2563–2586. doi:10.1109/TGRS.2006.874140.
- Baraldi, A. (2019) 'Satellite Image Automatic Mapper™ - SIAM™ - System and Products Description'. doi:10.13140/RG.2.2.26295.88484.
- Baraldi, A. and Boschetti, L. (2012a) 'Operational Automatic Remote Sensing Image Understanding Systems: Beyond Geographic Object-Based and Object-Oriented Image Analysis (GEOBIA/GEOOIA). Part 1: Introduction', *Remote Sensing*, 4(9), pp. 2694–2735. doi:10.3390/rs4092694.
- Baraldi, A. and Boschetti, L. (2012b) 'Operational Automatic Remote Sensing Image Understanding Systems: Beyond Geographic Object-Based and Object-Oriented Image Analysis (GEOBIA/GEOOIA). Part 2: Novel system Architecture, Information/Knowledge Representation, Algorithm Design and Implementation', *Remote Sensing*, 4(9), pp. 2768–2817. doi:10.3390/rs4092768.
- Baraldi, A., Gironde, M. and Simonetti, D. (2010) 'Operational Two-Stage Stratified Topographic Correction of Spaceborne Multispectral Imagery Employing an Automatic Spectral-Rule-Based Decision-Tree Preliminary Classifier', *IEEE Transactions on Geoscience and Remote Sensing*, 48(1), pp. 112–146. doi:10.1109/TGRS.2009.2028017.
- Baraldi, A. and Humber, M.L. (2014) 'Quality Assessment of Preclassification Maps Generated From Spaceborne/Airborne Multispectral Images by the \textit{Satellite Image Automatic Mapper} and \textit{Atmospheric/Topographic Correction-Spectral Classification} Software Products: Part 1—Theory', *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, pp. 1–23. doi:10.1109/JSTARS.2014.2349932.
- Baumann, P. (1994) 'Management of multidimensional discrete data', *The VLDB Journal*, 3(4), pp. 401–444. doi:10.1007/BF01231603.
- Baumann, P. *et al.* (1998) 'The multidimensional database system RasDaMan', in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data - SIGMOD '98. the 1998 ACM SIGMOD international conference*, Seattle, Washington, United States: ACM Press, pp. 575–577. doi:10.1145/276304.276386.

Baumann, P. (2010) 'The OGC web coverage processing service (WCPS) standard', *Geoinformatica*, 14(4), pp. 447–479. doi:10.1007/s10707-009-0087-2.

Baumann, P. (2017) 'The Datacube Manifesto'. Available at: <https://www.earthserver.eu/tech/datacube-manifesto/The-Datacube-Manifesto.pdf> (Accessed: 28 June 2021).

Baumann, P. *et al.* (2019) 'Datacubes: Towards Space/Time Analysis-Ready Data', in Döllner, J., Jobst, M., and Schmitz, P. (eds) *Service-oriented mapping: changing paradigm in map production and geoinformation management*, pp. 269–299.

Baumann, P. *et al.* (2021) 'Array databases: concepts, standards, implementations', *Journal of Big Data*, 8(1), p. 28. doi:10.1186/s40537-020-00399-2.

Baumann, P. (2022a) 'WCPS grammar', *rasdaman-users (Rasdaman Community Mailing List)*. Available at: <https://groups.google.com/g/rasdaman-users/c/tRTOSQ-FjkM/m/JfBaSaY6CgAJ> (Accessed: 25 April 2022).

Baumann, P. (2022b) 'WCPS grammar', *rasdaman-users (Rasdaman Community Mailing List)*. Available at: <https://groups.google.com/g/rasdaman-users/c/tRTOSQ-FjkM/m/263mNtlUAgAJ> (Accessed: 25 April 2022).

Baumann, P. *et al.* (n.d.) *EarthServer*. Available at: <https://earthserver.eu/wcs/> (Accessed: 26 April 2022).

Baumann, P. and Holsten, S. (2011) 'A Comparative Analysis of Array Models for Databases', in Kim, T. *et al.* (eds) *Database Theory and Application, Bio-Science and Bio-Technology*. Berlin, Heidelberg: Springer Berlin Heidelberg (Communications in Computer and Information Science), pp. 80–89. doi:10.1007/978-3-642-27157-1_9.

Baumann, P. and Merticariu, V. (2015a) *Introduction to WCPS - Part 1: Basic* [Tutorial]. Bremen (EarthServer Webinar). Available at: <https://www.youtube.com/watch?v=MnPABAAQnXM> (Accessed: 25 April 2022).

Baumann, P. and Merticariu, V. (2015b) *Introduction to WCPS - Part 2: Advanced* [Tutorial]. Bremen (EarthServer Webinar). Available at: <https://www.youtube.com/watch?v=m1q2AloQADs> (Accessed: 25 April 2022).

Baumann, P., Schleidt, K. and Escriu, J. (n.d.) *INSPIRE Coverages Demystified*. Available at: <https://inspire.rasdaman.org/> (Accessed: 26 April 2022).

Bradshaw, S., Brazil, E. and Chodorow, K. (2019) *MongoDB: the definitive guide: powerful and scalable data storage*. Third edition. Beijing Boston Farnham: O'Reilly.

Earth Server Federation (n.d.) *EarthServer*. Available at: <https://earthserver.eu/> (Accessed: 25 April 2022).

Esbrí, M.Á. (2021) 'Remote Sensing', in Södergård, C. et al. (eds) *Big Data in Bioeconomy*. Cham: Springer International Publishing, pp. 49–61. doi:10.1007/978-3-030-71069-9_4.

European Commission (2022) *INSPIRE | Welcome to INSPIRE*. Available at: <https://inspire.ec.europa.eu/> (Accessed: 25 April 2022).

Hobona, G. (2021) *CRS Definition Resolver*. Available at: https://external.ogc.org/twiki_public/CRSdefinitionResolver/WebHome (Accessed: 24 March 2022).

Hu, F. et al. (2018) 'Evaluating the Open Source Data Containers for Handling Big Geospatial Raster Data', *ISPRS International Journal of Geo-Information*, 7(4), p. 144. doi:10.3390/ijgi7040144.

ISO/IEC 9075-15 (2019) *ISO/IEC 9075-15:2019 Information technology database languages — SQL — Part 15: Multi-dimensional arrays (SQL/MDA)*. Available at: <https://www.iso.org/standard/67382.html> (Accessed: 24 March 2022).

Jacobs University & rasdaman GmbH (2020) *Big Earth Datacube Standards: Coverages, WCS, WCPS on rasdaman*. Available at: https://standards.rasdaman.com/demo_convolution.html (Accessed: 27 April 2022).

Joshi, A. et al. (2019) 'SciDB Based Framework for Storage and Analysis of Remote Sensing Big Data', *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-5/W3, pp. 43–47. doi:10.5194/isprs-archives-XLII-5-W3-43-2019.

Killough, B. (2019) 'The Impact of Analysis Ready Data in the Africa Regional Data Cube', in *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium. IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, Yokohama, Japan: IEEE, pp. 5646–5649. doi:10.1109/IGARSS.2019.8898321.

Kopp, S. et al. (2019) 'Achieving the Full Vision of Earth Observation Data Cubes', *Data*, 4(3), p. 94. doi:10.3390/data4030094.

Kuhn, W. (2005) 'Geospatial Semantics: Why, of What, and How?', in Spaccapietra, S. and Zimányi, E. (eds) *Journal on Data Semantics III*. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 1–24. doi:10.1007/11496168_1.

Ma, Y. et al. (2015) 'Remote sensing big data computing: Challenges and opportunities', *Future Generation Computer Systems*, 51, pp. 47–60. doi:10.1016/j.future.2014.10.029.

Marr, D. (1982) *Vision: a computational investigation into the human representation and processing of visual information*. San Francisco: W.H. Freeman.

Misev, D. (2020) 'No interpolation method effect when scaling', *rasdaman-users (Rasdaman Community Mailing List)*. Available at: https://groups.google.com/g/rasdaman-users/c/1hZ642H5h_Q/m/dn1jRBPBQAJ (Accessed: 25 April 2022).

Misev, D. (2022) 'WCPS grammar', *rasdaman-users (Rasdaman Community Mailing List)*. Available at: <https://groups.google.com/g/rasdaman-users/c/tRTOSQ-FjkM/m/M5MszcePAQAJ> (Accessed: 25 April 2022).

Misev, D., Rusu, M. and Baumann, P. (2012) 'A Semantic Resolver for Coordinate Reference Systems', in Di Martino, S., Peron, A., and Tezuka, T. (eds) *Web and Wireless Geographical Information Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 47–56. doi:10.1007/978-3-642-29247-7_5.

Morain, S.A. (1998) 'A Brief History of Remote Sensing Applications, with Emphasis on Landsat', in *People and pixels: linking remote sensing and social science*. Washington: D.C. : National academy Press, pp. 28–50.

ODC (2018) *The 'Road to 20' International Data Cube Deployments*. Document published on the ODC Website. Available at: https://www.opendatacube.org/_files/ugd/8959d6_66661f43c6c0461497854700a123cc59.pdf (Accessed: 18 March 2022).

ODC (2021) *Open Data Cube - Architecture and Ecosystem - A High-Level Overview*. Document published on the ODC Website. Available at: https://www.opendatacube.org/_files/ugd/3632b4_269d1d61d7f04677a1d32278042aa51a.pdf (Accessed: 18 March 2022).

OGC (no date) *About OGC*. Available at: <https://www.ogc.org/about> (Accessed: 25 April 2022).

OGC 06-103r4 (2011) *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture*. Available at: <http://www.opengis.net/doc/IS/sfa/1.2.1> (Accessed: 22 February 2022).

OGC 07-011 (2006) *The OpenGIS® Abstract Specification Topic 6: Schema for coverage geometry and functions*. Available at: https://portal.ogc.org/files/?artifact_id=19820 (Accessed: 22 February 2022).

OGC 08-068r3 (2021) *Web Coverage Processing Service (WCPS) Language Interface Standard*. Available at: <https://docs.opengeospatial.org/is/08-068r3/08-068r3.html> (Accessed: 23 February 2022).

OGC 09-048r5 (2019) *OGC Name Type Specification - definitions - part 1 – basic name*. Available at: <https://docs.opengeospatial.org/pol/09-048r5.html> (Accessed: 24 March 2022).

OGC 09-146r2 (2012) *OGC® Coverage Implementation Schema*. Version 1.0.1. Available at: https://portal.ogc.org/files/?artifact_id=48553 (Accessed: 25 March 2022).

OGC 09-146r8 (2019) *OGC Coverage Implementation Schema with Corrigendum OGC*. Version 1.1.1. Available at: <https://docs.opengeospatial.org/is/09-146r8/09-146r8.html> (Accessed: 26 April 2022).

OGC 13-057r1 (2016) *OGC Web Coverage Service Interface Standard – Transaction Extension*. Available at: <https://docs.openeospatial.org/is/13-057r1/13-057r1.html> (Accessed: 23 February 2022).

OGC 17-089r1 (2018) *Web Coverage Service (WCS) 2.1 Interface Standard - Core*. Available at: docs.openeospatial.org/is/17-089r1/17-089r1.html (Accessed: 11 February 2022).

OGC Naming Authority SC (2022) *OGC-NA Group Description*. Available at: <https://www.ogc.org/projects/groups/ogcnasc> (Accessed: 24 March 2022).

Open Access Hub (2022). Available at: <https://scihub.copernicus.eu/> (Accessed: 21 February 2022).

Pham Huu, B. (2022) 'wcst_import resume.json', *rasdaman-users (Rasdaman Community Mailing List)*. Available at: <https://groups.google.com/g/rasdaman-users/c/rqypG8f2Jis/m/ZeqwMmCEKgAJ> (Accessed: 25 April 2022).

Purss, M.B.J. *et al.* (2015) 'Unlocking the Australian Landsat Archive – From dark data to High Performance Data infrastructures', *GeoResJ*, 6, pp. 135–140. doi:10.1016/j.grj.2015.02.010.

rasdaman team (2022) *Rasdaman 10.0.0 documentation*. Available at: <https://doc.rasdaman.org/> (Accessed: 24 March 2022).

rasdaman team (n.d.) *Versions – rasdaman*. Available at: <https://rasdaman.org/wiki/Versions> (Accessed: 26 April 2022).

Rasdaman Workshop (n.d.). Available at: <https://tutorial.rasdaman.org/ogc-and-wcs-tutorial/#exercises-exercise-2> (Accessed: 26 April 2022).

Rowley, J. (2007) 'The wisdom hierarchy: representations of the DIKW hierarchy', *Journal of Information Science*, 33(2), pp. 163–180. doi:10.1177/0165551506070706.

Sen2Cube.at Manual (2022) 'Sen2Cube.at Manual'. Available at: <https://manual.sen2cube.at/> (Accessed: 25 April 2022).

Sentinel-2 User Handbook (2015). ESA Standard Document, p. 64. Available at: https://sentinel.esa.int/documents/247904/685211/Sentinel-2_User_Handbook (Accessed: 21 February 2022).

Sinergise Laboratory for geographical information systems, Ltd. (no date) *Sentinel-hub EO-Browser*. Available at: <https://apps.sentinel-hub.com/eo-browser/> (Accessed: 25 April 2022).

Stonebraker, M. *et al.* (2013) 'SciDB: A Database Management System for Applications with Complex Analytics', *Computing in Science & Engineering*, 15(3), pp. 54–62. doi:10.1109/MCSE.2013.19.

Strobl, P. *et al.* (2017) 'The Six Faces of the Data Cube', in *Proceedings of the 2017 conference on Big Data from Space (BIDS' 2017): 28th 30th November 2017 Toulouse (France)*, pp. 32–35.

Sudmanns, M. *et al.* (2017) 'Automatic Ex-post Flood Assessment Using Long Time Series of Optical Earth Observation Images', *GI_Forum*, 1, pp. 217–227. doi:10.1553/giscience2017_01_s217.

Sudmanns, M. *et al.* (2021) 'One GUI to Rule Them All: Accessing Multiple Semantic EO Data Cubes in One Graphical User Interface', *GI_Forum*, 1, pp. 53–59. doi:10.1553/giscience2021_01_s53.

Sudmanns, M., Lang, S. and Tiede, D. (2018) 'Big Earth Data: From Data to Information', *GI_Forum*, 1, pp. 184–193. doi:10.1553/giscience2018_01_s184.

The Sentinel missions (2021). Available at: https://www.esa.int/Applications/Observing_the_Earth/Copernicus/The_Sentinel_missions (Accessed: 21 February 2022).

Thusoo, A. *et al.* (2009) 'Hive: a warehousing solution over a map-reduce framework', *Proceedings of the VLDB Endowment*, 2(2), pp. 1626–1629. doi:10.14778/1687553.1687609.

Tiede, D. *et al.* (2017) 'Architecture and prototypical implementation of a semantic querying system for big Earth observation image bases', *European Journal of Remote Sensing*, 50(1), pp. 452–463. doi:10.1080/22797254.2017.1357432.

USGS (no date) *EarthExplorer*. Available at: <https://earthexplorer.usgs.gov/> (Accessed: 25 April 2022).

Vecera, S.P. and Farah, M.J. (1997) 'Is visual image segmentation a bottom-up or an interactive process?', *Perception & Psychophysics*, 59(8), pp. 1280–1296. doi:10.3758/BF03214214.

Wagemann, J. *et al.* (2018) 'Geospatial web services pave new ways for server-based on-demand access and processing of Big Earth Data', *International Journal of Digital Earth*, 11(1), pp. 7–25. doi:10.1080/17538947.2017.1351583.

Yue, P. *et al.* (2015) 'Towards intelligent GIServices', *Earth Science Informatics*, 8(3), pp. 463–481. doi:10.1007/s12145-015-0229-z.

Zaharia, M. *et al.* (2016) 'Apache Spark: a unified engine for big data processing', *Communications of the ACM*, 59(11), pp. 56–65. doi:10.1145/2934664.

Zhu, Z. *et al.* (2019) 'Benefits of the free and open Landsat data policy', *Remote Sensing of Environment*, 224, pp. 382–385. doi:10.1016/j.rse.2019.02.016.

Appendix A - WCPS Syntax

A.1 Coverage Subsetting

A coverage can be subset by either a trim or a slice operation. While trimming leaves the number of dimensions unchanged, slicing always involves a reduction of dimensions. In the special case that subsetting a coverage leads to a single point, the result is still considered a coverage with the dimension 0 (OGC, 2021, Chapter 7.1.23 Note 1). It is important to keep track of the dimension returned to chose a matching output format, as otherwise, the query will fail. The sequence in which the subsetting axes are specified is not important, as each axis needs to be specified by name.

Syntax	Example
Trimming for \$c in (A) return encode(\$c[axisX(lowerValue : upperValue), axisY(lowerValue : upperValue), axisZ(lowerValue : upperValue)], "MIME Type")	for \$c in (AverageTemperature) return encode(\$c[ansi("2012-12-01T20:07:00.500Z":"2012-12-03T20:07:00.500Z"), Lat(-38:40), Lon(-18:55)] , "netcdf") → Returns a 3D netCDF with changed ranges for each dimension
Slicing on the time axis returning a 2D image for a specified date for \$c in (A) return encode(\$c[axisZ(sliceValue)], "MIME Type")	for \$c in (AverageTemperature) return encode(\$c[ansi("2012-12-01T20:07:00.500Z"), , "image/tiff") → Returns a 2D image that was taken at the requested timestamp
Special case of returning the 2D image for the latest available date for \$c in (A) return encode (for \$c in (AverageTemperature) return encode (\$c[ansi:"CRS:1"(domain(\$c, ansi).hi)], "tiff"))

Syntax	Example
<pre>\$c[axisZ:"CRS:1"(domain(\$c, axisZ).hi)], "MIME Type"))</pre>	<p>→ Caveat: As has been explained in table 2.3 the domain function does return image coordinates for the temporal axis, therefore domain(\$c, ansi).hi will not work to retrieve the latest image without transforming the slice into grid coordinates by specifying "CRS:1".</p>
<p>Slicing on geographical coordinates returning a point</p> <pre>for \$c in (A) return encode(\$c[axisX(sliceValue), axisY(sliceValue), axisZ(sliceValue)], "MIME Type")</pre>	<pre>for \$c in (AverageTemperature) return encode(\$c[Lat(41.716667), Lon(44.791667)] , "text/csv")</pre> <p>→ Returns a 1D value for the requested coordinate</p>
<p>Offsets can be applied within a trim operation but ONLY when the axis values are transformed into grid coordinates</p> <pre>for \$c in (A) return encode(\$c[axisX(lowerValue + offset1 : upperValue + offset2), axisY(lowerValue : upperValue), "MIME Type")</pre>	<pre>for \$c in (AverageTemperature) return encode(\$c[ansi("2012-12-01T20:07:00.500Z"), Lat:"CRS:1"(0+50:80+100), Lon(-18:55)] , "image/tiff")</pre> <p>→ Results in a 2D image with the Latitude range changed</p> <p>CAVEAT: This does not work:</p> <pre>for \$c in (AverageTemperature) return encode(\$c[ansi("2012-12-01T20:07:00.500Z"), Lat(-38:40+10), Lon(-18:55)] , "image/tiff")</pre>

Syntax	Example
	<p>→ <code><ows:Exception exceptionCode="WcpsError" > <ows:ExceptionText>Invalid subset expression: Lat(-38:40 + 10). Expressions inside subsets are only allowed on grid axes. HINT: Try a grid subset instead. E.g. Lat:"CRS:1"(-38:40 + 10) subsets directly on the grid.</ows:ExceptionText> </ows:Exception></code></p>

A.2 Let Clause

The let clause allows binding alias variables to valid WCPS sub-expressions (rasdaman team, 2022, Chapter 5.5.6).

Syntax	Example
<pre>for \$c in (A) let \$alias1 := expression, \$alias2 := anotherExpression, \$aliasN := yetAnotherExpression return encode(\$c[\$alias] , "MIME Type")</pre>	<pre>for \$c in (mean_summer_airtemp) let \$sub1 := \$c[Lat(-20:-10), Lon(130:150)], \$sub2 := 10 return encode (\$sub1* \$sub2 ,"tiff")</pre> <p>→ Returns the subset of a coverage whose values have been multiplied by 10 throughout the image</p>

A.3 Deriving single bands and Multiband Constructor

It is possible to change the band sequence of coverages with multiple bands for example to generate different false color images.

Syntax	Example
for \$c in (A)	for \$c in (AverageTemperature)

<pre>return encode(\$c.BandX , "MIME Type")</pre>	<pre>return encode(\$c[ansi("2012-12-01T20:07:00.500Z")].Blue , "image/tiff")</pre> <p>→ Returns an image with the values of the Blue band only</p>
<pre>for \$c in (A) return encode({ red: \$c.BandX3; green: \$c.BandX2; blue: \$c.BandX1 } , "MIME Type")</pre>	<pre>for \$c in (AverageTemperature) return encode((unsigned char){ red: \$c[ansi("2012-12- 01T20:07:00.500Z")].Blue; green: \$c[ansi("2012-12- 01T20:07:00.500Z")].Green; blue: \$c[ansi("2012-12- 01T20:07:00.500Z")].Red } , "image/tiff")</pre> <p>→ Returns an image where the red and the blue band values are switched</p>

6.1 A.4 Induced Operations

Operations available for a values range type are automatically lifted to the whole coverage. These 'Induced operations' thus apply on all cells of a coverage simultaneously (Baumann, 2010).

Syntax	Example
<pre>for \$c in (A) return function(\$c) , "MIME Type")</pre>	<pre>for \$c in (mean_summer_airtemp) return encode(sin(\$c)+20 , "tiff")</pre> <p>→ Returns an image where the sinus function + 10 was applied to the cell value of each pixel</p>

A.5 Conditional evaluation/Case distinction

It is possible to return apply different functions for pixels of a coverage based on conditional evaluation using the switch statement. As in the underlying rasql implementation, WCPS conditions used in a statement must be specified in order of generality by using the case keyword. The first condition should be the most specific. Each subsequent condition should be more general, the most general being the default condition (rasdaman team, 2022, Chapters 4.10.4.4 and 5.5.10). Either a scalar value or a coverage are returned. The domain of all conditional expressions as well as the result has to be the same. This means, they have to have the same extent, resolution and CRS (rasdaman team, 2022, Chapters 5.5.10 and 11.2.2).

Syntax	Example
<pre>for \$c in (A) return encode(switch case boolCovExpr return covExpr case boolCovExpr return covExpr ... default return covExpr , "MIME Type")</pre>	<pre>for \$c in (mean_summer_airtemp) return encode((unsigned char) switch case \$c < 10 return {red:0; green: 0; blue: 100} case \$c < 30 return {red:255; green: 255; blue: 0} case \$c < 50 return {red:255; green: 150; blue: 0} default return {red: 255; green: 0; blue: 0} , "tiff")</pre> <p>→ Returns an image where pixels have been colored according to the conditions they met</p>

A.6 General Coverage Constructor

Using a coverage constructor is useful when the coverage is either too large to be described as a constant or when the coverage's range values are derived from another source (OGC 08-068r3, 2021, Chapter 7.1.29). The dimensionality of the new coverage is dependent on the number of iteration variables defined in the over clause of the

constructor. 2D coverages constructed in this way are returned in the non geographic coordinate system „Index2D“. This can be tested by using the probing function 'crsSet(\$newCoverage)'.

Syntax	Example
<p>Creating a 1D coverage for \$c in (A) return encode(coverage covName over \$iterVar axis(lo:hi) values scalarExpr , "MIME Type")</p>	<pre>for \$c in (mean_summer_airtemp) return encode (coverage oneDim over \$px x(100:200) values \$px , "csv")</pre> <p>→ returns each value in the range 100 - 200</p>
<p>same as above</p>	<pre>for \$c in (mean_summer_airtemp) return encode (coverage oneDim over \$px x(100:200) values \$c[Lon:"CRS:1"(500), Lat:"CRS:1"(\$px)] , "csv")</pre> <p>→ returns the cell values of pixels that have an index of 500 on the Longitude axis (! in grid values) and an index in the range of 100-200 at the Latitude axis (! grid values as well)</p>
<p>Creating a 2D coverage by inserting values directly using a list for \$c in (A) return encode(coverage covName over \$iterVar1 axis1(lo:hi), \$iterVar2 axis2(lo:hi) values valueList , "MIME Type")</p>	<pre>for \$s in (mean_summer_airtemp) let \$kernel:= coverage exampleKernel over \$x x(-1:1), \$y y (-1:1) value list < 1; 0; -1; 2; 0; -2; 1; 0; -1> return encode (\$kernel, "image/tiff")</pre> <p>→ creates 3x3 coverage with the specified values</p>
<p>Creating a 2D coverage based on another 2D coverage for \$c in (A)</p>	<pre>for \$c in (mean_summer_airtemp) return encode(coverage newCov</pre>

<pre>return encode(coverage covName over \$iterVar1 axis1(lo:hi), \$iterVar2 axis2(lo:hi) values scalarExp , "MIME Type")</pre>	<pre>over \$px x(imageCrsDomain(\$c, Lon)), \$py y(imageCrsDomain(\$c, Lat)) values \$px+\$py , "image/tiff")</pre> <p>→ returns a coverage with the extent (grid coordinates) of 'mean_summer_airtemp' comprising cell values raising from 0 (top-left) to 1,570 (bottom-right).</p>
<p>same as above</p>	<pre>for \$c in (mean_summer_airtemp) return encode(coverage newCov over \$px x(imageCrsDomain(\$c, Lon)), \$py y(imageCrsDomain(\$c, Lat)) values \$c[Lon(\$px+100), Lat(\$py-100)] , "image/tiff")</pre> <p>→ returns a coverage with the extent (grid coordinates) and the values of 'mean_summer_airtemp'. The values of the original coverage are shifted by +100 in the x direction and by -100 in the y direction, so the whole image extent is shifted into the bottom-left direction . New values on the top-right are 0 values.</p>

A.7 Aggregation operations

Coverages can be summarized into a scalar value using aggregation operations, also known as condensers. Multiple types of aggregation can be applied (rasdaman team, 2022, 11.2.1).

Condensing numeric coverages using a shorthand condenser operation

Syntax	Example
<p>Numeric coverages can be summarized with</p>	<pre>for \$c in (AverageTemperature) return max(\$c.Red)</pre>

avg(), add(), min(), max()	
	→ returns the maximum value of the red band

Condensing Boolean coverages using a shorthand condenser operation

Syntax	Example
<p>Count the number of true values</p> <pre>for \$c in (A) return encode(count(booleanExpr), "MIME Type")</pre>	<pre>for \$c in (AverageTemperature) return encode(count(\$c.Red<50), "csv")</pre> <p>→ Returns the number of pixel whose cell values is greater than 50</p>
<p>Return true if some/all values are true</p> <pre>for \$c in (A) return encode(some(booleanExpr), "MIME Type")</pre> <p>or</p> <pre>for \$c in (A) return encode(all(booleanExpr), "MIME Type")</pre>	<pre>for \$c in (AverageTemperature) return encode(some(\$c.Red<50), "csv")</pre> <p>Caveat: Unexpected behavior when using some(), all() instead of count, an empty csv is produced. It is not clear whether this is a bug in the v10.0.0-beta3 version tested or whether the operation has not been applied correctly.</p>

General condenser

Syntax	Example
<pre>for \$c in (A) return condense op over \$iterVar axis(lo:hi), ... using scalarExpr</pre>	<pre>for \$c in (AverageChlorophyll) return condense + over \$p x(imageCrsDomain(\$c[ansi("2015-01-01T00:00:00.000Z":"2015-03-31T00:00:00.000Z")],ansi)) using 1</pre>

Syntax	Example
	<p>→ This query can be used to find how many images are in a cube for a specified timespan as it adds one for every timestamp found in this range. Caveat: The timespan queried is not allowed to exceed the date of the oldest or newest image for the query to work</p>
<pre>for \$c in (A) return encode(condense op over \$iterVar axis(lo:hi), ... using scalarExpr ,"MIME Type")</pre>	<pre>for \$c in (AverageTemperature) return encode(condense max over \$pt t(imageCrsDomain(\$c, ansi)) using \$c.Red[ansi(\$pt)], "tiff")</pre> <p>→ condenses the input coverage along the time axis and returns an image that contains the maximal cell value in the red band for each pixel</p>

A.8 Combining constructor and aggregation queries

Coverage condenser can be combined with coverage constructors.

Syntax	Example
<pre>for \$c in (A) return encode(coverage covName over \$iterVar axis(lo:hi), ... values count(\$c[ansi("2012-12-01T20:07:00.500Z")].Red = \$bucket) ,"MIME Type")</pre>	<pre>for \$c in (AverageTemperature) return encode(coverage histogram over \$bucket x(0 : 255) values count(\$c[ansi("2012-12-01T20:07:00.500Z")].Red = \$bucket), "text/csv")</pre> <p>→ For each value from 0 to 255 the number of times this value has been found in the red band of the AverageTemperature at the specified timestamp is returned</p>

A.9 Coverage Filtering with where

A where clause in the for clause evaluates whether a coverage meets a condition. This is handy for filtering out coverages suitable for further processing. Coverage that don't meet the condition are not further processed (Baumann and Merticariu, 2015a, 5:40).

Syntax	Example
<pre>for \$c in (A) where boolScalarExpr return covExpr</pre>	<pre>for \$c in (AverageChlorophyll, AverageTemperature) where max(\$c.Red)=250 return max(\$c.Red)</pre> <p>Caveat: There is a bug in the v10.0.0-beta3 version which has been discussed in the rasdaman-users group (Misev, 2022). As only AverageTemperature meets the condition and has a maximal value of 250 (AverageChlorophyll a maximal value of 254) this query should return only return 250 once. Instead, it seems like the condition is only evaluated for the first coverage in the list, and this result is returned twice. As noted in the linked discussion, this bug should have been fixed with the stable v10.0.0 release which has not been tested in this thesis.</p>
<pre>for \$c in (A) return encode(condense op over \$iterVar axis(lo:hi), ... where boolScalarExpr using scalarExpr , "MIME Type")</pre>	<pre>for \$c in (AverageTemperature) return encode(condense min over \$pt t(imageCrsDomain(\$c, ansi)) where max(\$c.Red)=250 using \$c.Blue[ansi(\$pt)], "tiff")</pre> <p>→ If max(\$c.Red)=250 is true, a result 2D coverage is returned showing only the minimum values of the blue band that have been observed over the four timesteps.</p>

	Caveat: If max(\$c.Red)=250 is false for all timesteps, a 0 is returned for each timestep. This can not be reflected in a tiff, so another MIME-Type has to be selected for the query to work.
--	--

A.10 Clipping

AOI for WCPS processing can be selected by specifying coordinates in WKT that are used for clipping the image. The examples in the table below show this for a polygon clip of a 2D image. Additional clip functionality like clipping multipolygons, linestrings on a 2D coverage or curtain and corridor clipping on a 3D coverage are implemented in rasdaman's version of WCPS but are not further regarded here.

Syntax	Example
for \$c in (A) return encode(clip(coverageExpression, wkt) , "MIME Type")	for \$c in (mean_summer_airtemp) return encode(clip(\$c, POLYGON((-16.497041763341 137.99530162413, -16.497041763341 140.626392111369, -25.9819605568445 138.016956689869, -16.497041763341 137.99530162413))), "tiff") → returns the values for the clipped polygon area
for \$c in (A) return encode(clip(coverageExpression, wkt , subsettingCrs]) , "MIME Type")	for \$c in (mean_summer_airtemp) return encode(clip(\$c, POLYGON((-1116990.22057387 8114947.98756654, - 823682.254419523 8046915.10152991, -771728.418861184,8600745.08905634 - 1014083.79157432 7041755.33238844, -1116990.22057387 8114947.98756654))), "http://localhost:8080/def/crs/EPSG/ 0/28356"

), "tiff")
	→ returns the values for the clipped polygon area (in the same CRS as the input coverage, despite being clipped with coordinates of another CRS)


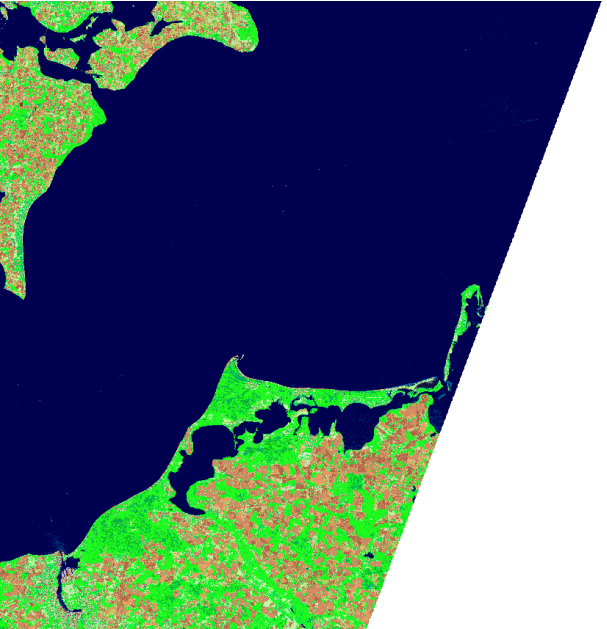

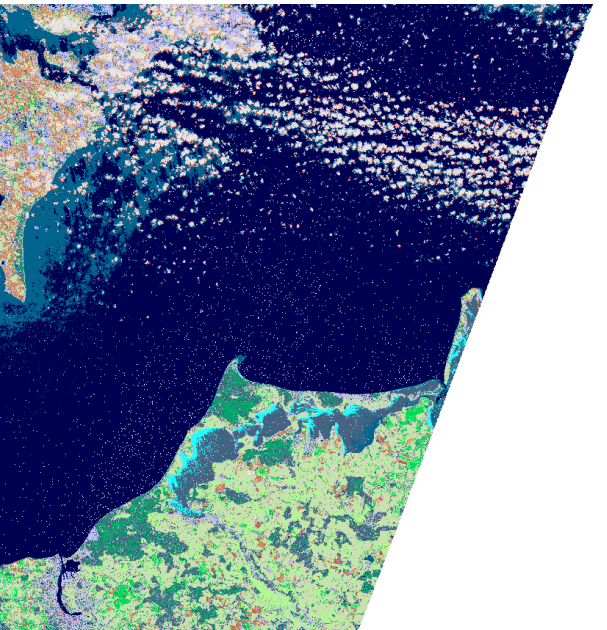
A.11 Additional Functions

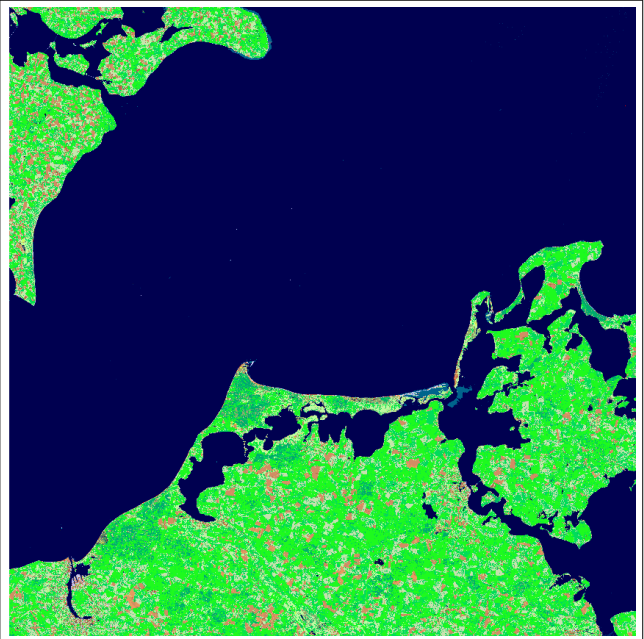
Rasdaman WCPS provides useful functions for extending and scaling a coverage domain as well as to reproject CRS. On top of that it is possible to transform coverage matching the CIS 1.0 standard (OGC 09-146r2, 2012) to a CIS 1.1 (OGC 09-146r8, 2019) conform standard.

Syntax	Example
<pre>for \$c in (A), return encode(extend(covExpr, { axis1(lo:hi), axis2:crs(lo:hi), ... }) , "MIME Type")</pre>	<pre>for \$c in (mean_summer_airtemp) return encode(extend(\$c, { Long(100:170), Lat(-50:-12) }), "tiff")</pre> <p>→ returns an image which the domain of the Longitude axis extended in both East und West directions (original values 111.975:156.275) and extends the domain of the Latitude axis in the South direction while making it smaller in the North directions (compare with original values -44.52499...:-8.974999...). The target extent can be defined as grid CRS as well as geographical CRS</p>
<pre>for \$c in (A), \$d in (B), return scale(\$c[\$c, {imageCrsDomain(\$c)}] , "MIME Type")</pre>	<pre>for \$c in (mean_summer_airtemp), \$d in (AverageChlorophyll) let \$sub := \$c[Lat(-20:-10), Lon(130:150)] return encode(scale(\$c, {imageCrsDomain(\$d[ansi("2015-01-01T00:00:00.000Z")])}) , "tiff")</pre>

	<p>→ Returns an image showing the mean summer air temperature in Australia that is scaled to match the extent of the AverageChlorpophyll image for the selected timestamp</p>
<p>Auto-ratio for spatial scaling for \$c in (A) return encode(scale(\$c, { Long:"CRS:1"(lo:hi) }) , "MIME Type")</p>	<p>for \$c in (mean_summer_airtemp) return encode(scale(\$c, { Long:"CRS:1"(0:100) }), "tiff")</p> <p>→ When specifying just one spatial horizontal axis, the the other spatial horizontal axis will be determined automatically while preserving the original ratio between these axes. The target extent can be defined as grid CRS as well as geographical CRS</p>
<p>Changing the spatial CRS for a coverage for \$c in (A) return encode(crsTransform(covExpr, { axis1:crs1, axis2:crs2, ... }) , "MIME Type")</p>	<p>for \$c in (mean_summer_airtemp) return encode(crsTransform(\$c, { Long:"http://localhost:8080/def/crs/EPSSG/0/28356", Lat:"http://localhost:8080/def/crs/EPSSG/0/28356"}) , "tiff")</p> <p>→ returns an image in the defined CRS</p>
<p>Transforming a CIS 1.0 to a CIS 1.1. coverage encoding for \$c in (A) return encode(\$c, "application/gml+xml", {"outputType":"GeneralGridCoverage"})</p>	<p>for \$c in (AverageTemperature) return encode(\$c, "application/gml+xml", {"outputType":"GeneralGridCoverage"})</p> <p>→ The coverage is returned as CIS 1.1 GeneralGridCoverage.</p>

Appendix B - Images in the data cube

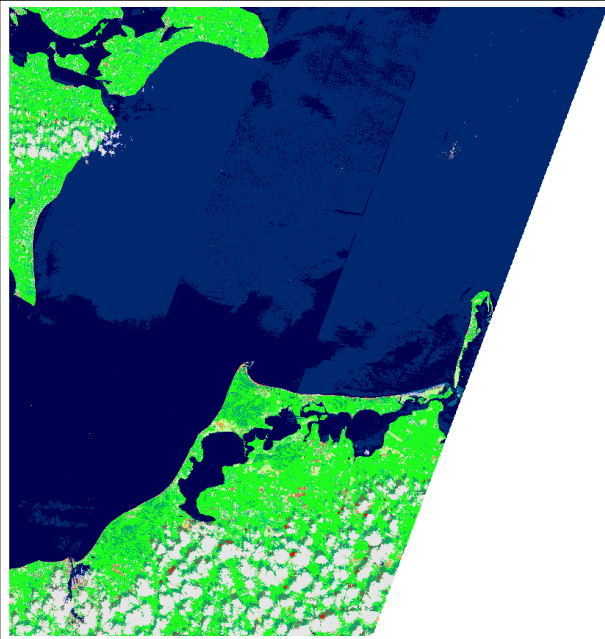
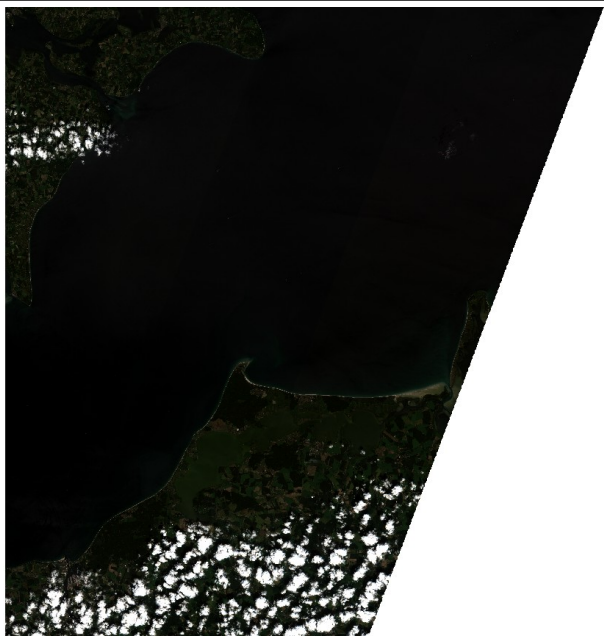
Sentinel-2 RGB	SIAM™ Preclassification (33 categories)
	
2015-08-09T00:00:00.000Z	
	
2016-01-06T00:00:00.000Z	



2016-05-12T00:00:00.000Z



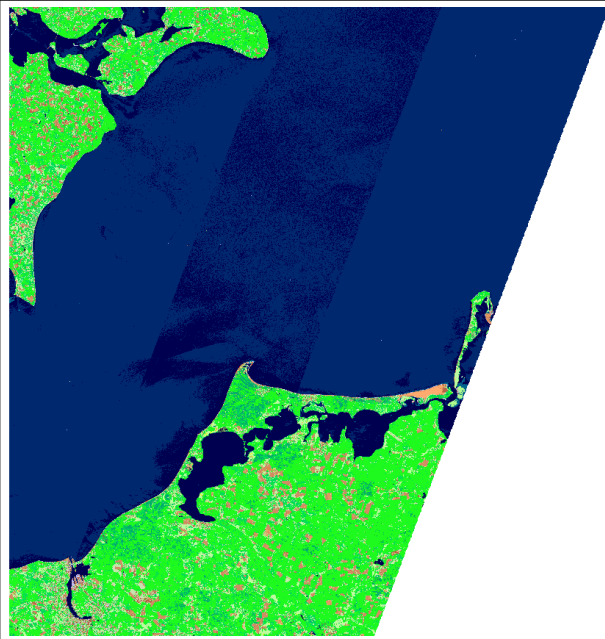
2017-01-10T00:00:00.000Z



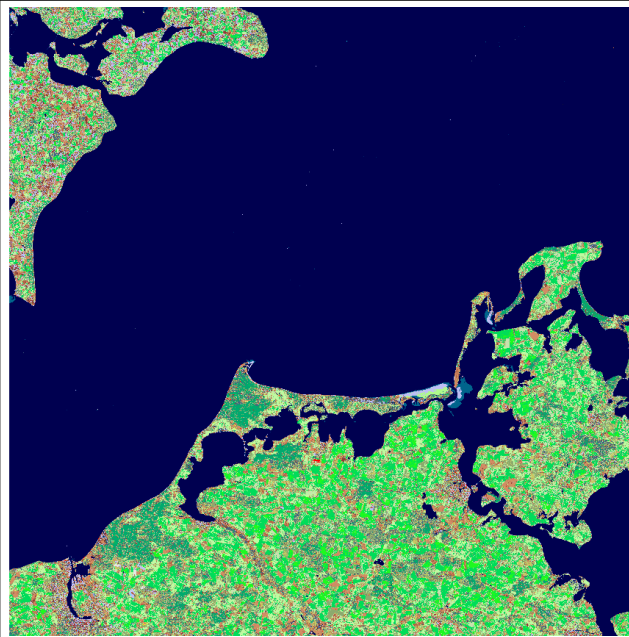
2017-06-19T00:00:00.000Z



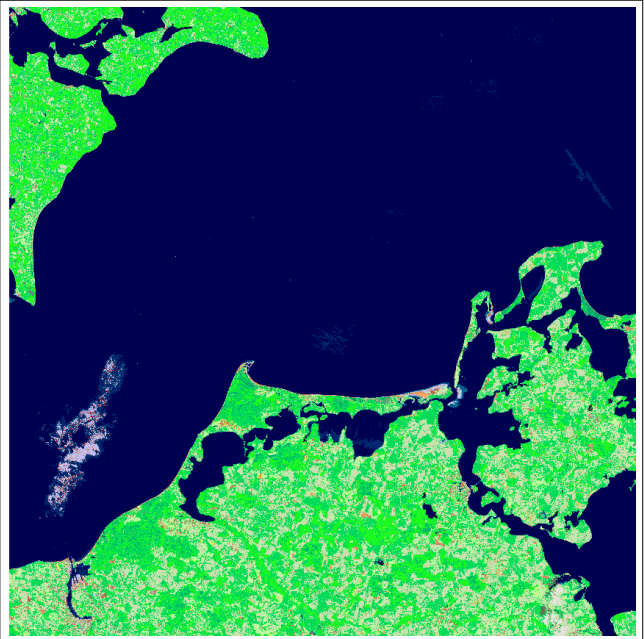
2018-01-07T00:00:00.000Z



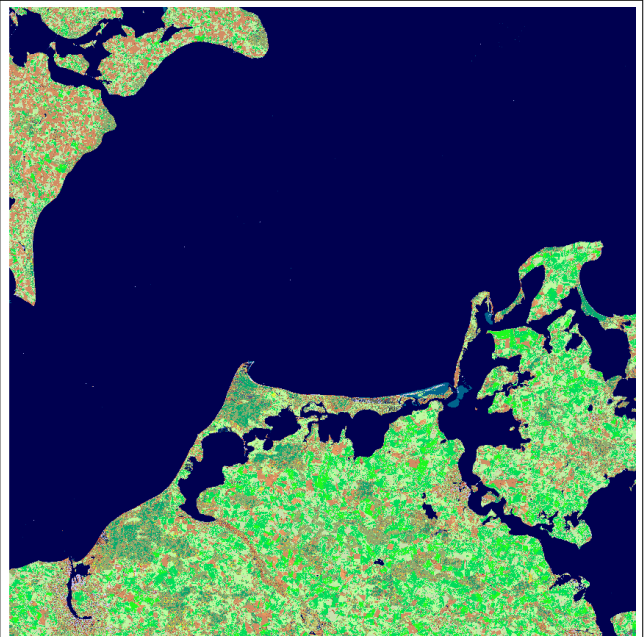
2018-05-30T00:00:00.000Z



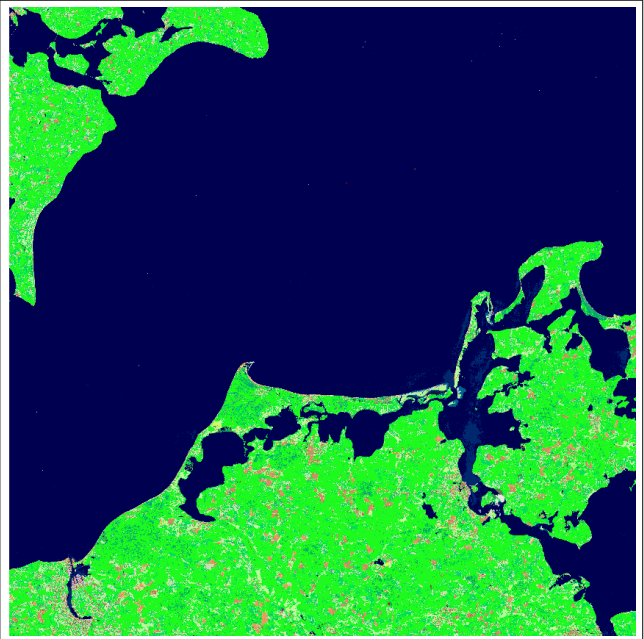
2019-02-16T00:00:00.000Z



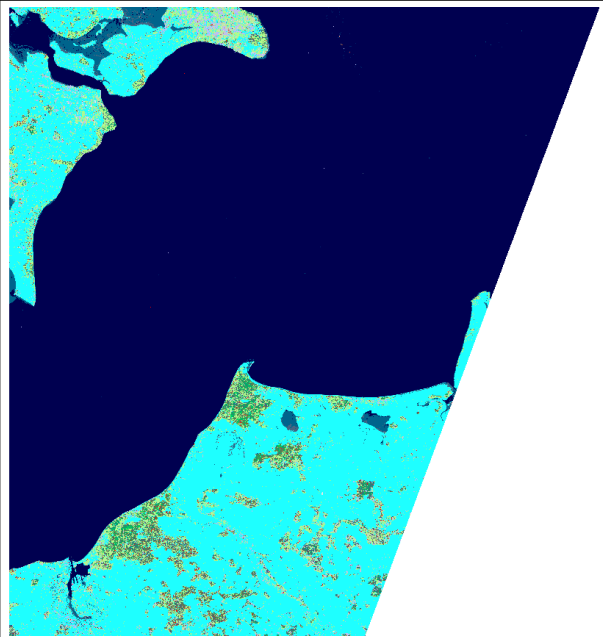
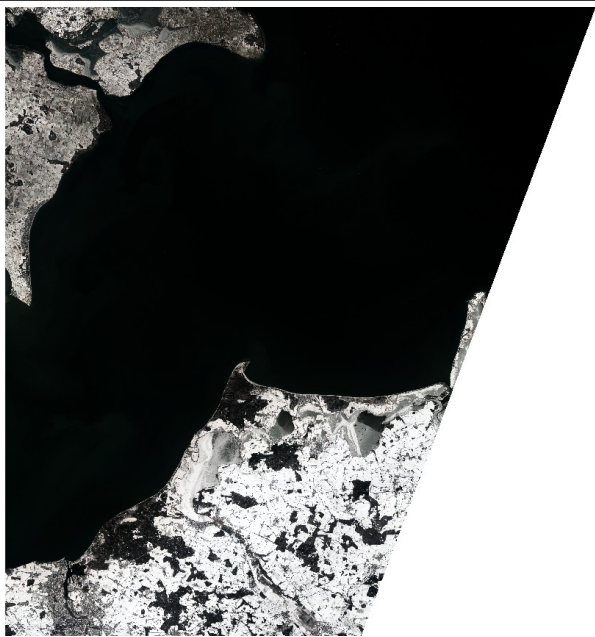
2019-06-26T00:00:00.000Z



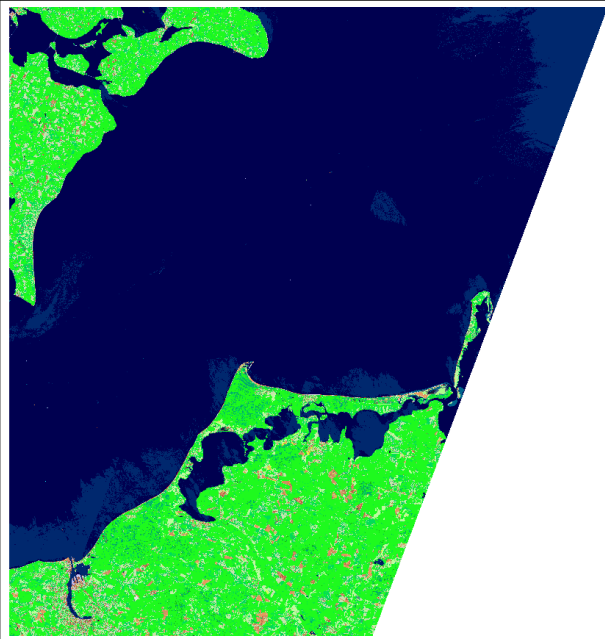
2020-03-27T00:00:00.000Z



2020-06-15T00:00:00.000Z



2021-02-13T00:00:00.000Z



2021-06-18T00:00:00.000Z

Appendix C - Ingredient files for data import

C.1 Ingredients file for importing Sentinel-2 data

```
{
  "config": {
    "service_url": "http://localhost:8080/rasdaman/ows",
    "automated": true
  },
  "input": {
    "coverage_id": "S2_${resolution}",
    "paths": [ "S2*.zip" ],
    // Optional filtering settings
    "resolutions": [ "10m", "60m" ],
    "levels": [ "L1C" ]
  },
  "recipe": {
    "name": "sentinel2",
    "options": {
      "coverage": {
        "metadata": {
          "type": "xml",
          "global": {
            "Title": "'Sentinel-2 data served by rasdaman'"
          }
        }
      }
    },
    "tiling": "ALIGNED [0:0, 0:1999, 0:1999] TILE SIZE 32000000",
    "wms_import": true
  }
}
```

C.2 Ingredient file for importing SIAM™ data

```
{
  "config": {
    "service_url": "http://localhost:8080/rasdaman/ows",
    "tmp_directory": "/tmp/",
    "crs_resolver": "http://localhost:8080/def/",
    "default_crs": "http://localhost:8080/def/OGC/0/Index2D",
    "mock": false,
    "automated": true,
    "track_files": false,
    "subset_correction": false
  },
  "input": {
    "coverage_id": "siam",
    "paths": [
      // Comma-separated lists of paths
    ]
  },
  "recipe": {
    "name": "general_coverage",
    "options": {
      "coverage": {
        "crs": "OGC/0/AnsiDate@EPSG/0/32633",
        "metadata": {
          "type": "json",
          "colorPaletteTable": "auto"
        }
      },
      "slicer": {
        "type": "gdal",
        "axes": {
          "ansi": {
            "min": "datetime(regex_extract('${file:name}', '.*[0-9]+.*$', 1), 'YYYYMMDD')",
            "type": "ansidate",
            "irregular": "true",
            "dataBound": "false",
            "gridOrder": 0
          },
          "E": {
            "min": "${gdal:minX}",
            "max": "${gdal:maxX}",
            "resolution": "${gdal:resolutionX}",
            "gridOrder": 2
          },
          "N": {
            "min": "${gdal:minY}",
            "max": "${gdal:maxY}",
            "resolution": "${gdal:resolutionY}",
            "gridOrder": 1
          }
        }
      }
    }
  },
  "tiling": "ALIGNED [0, 0:1023, 0:1023] TILE SIZE 4194304"
}
```

Appendix D - WCPS queries tested

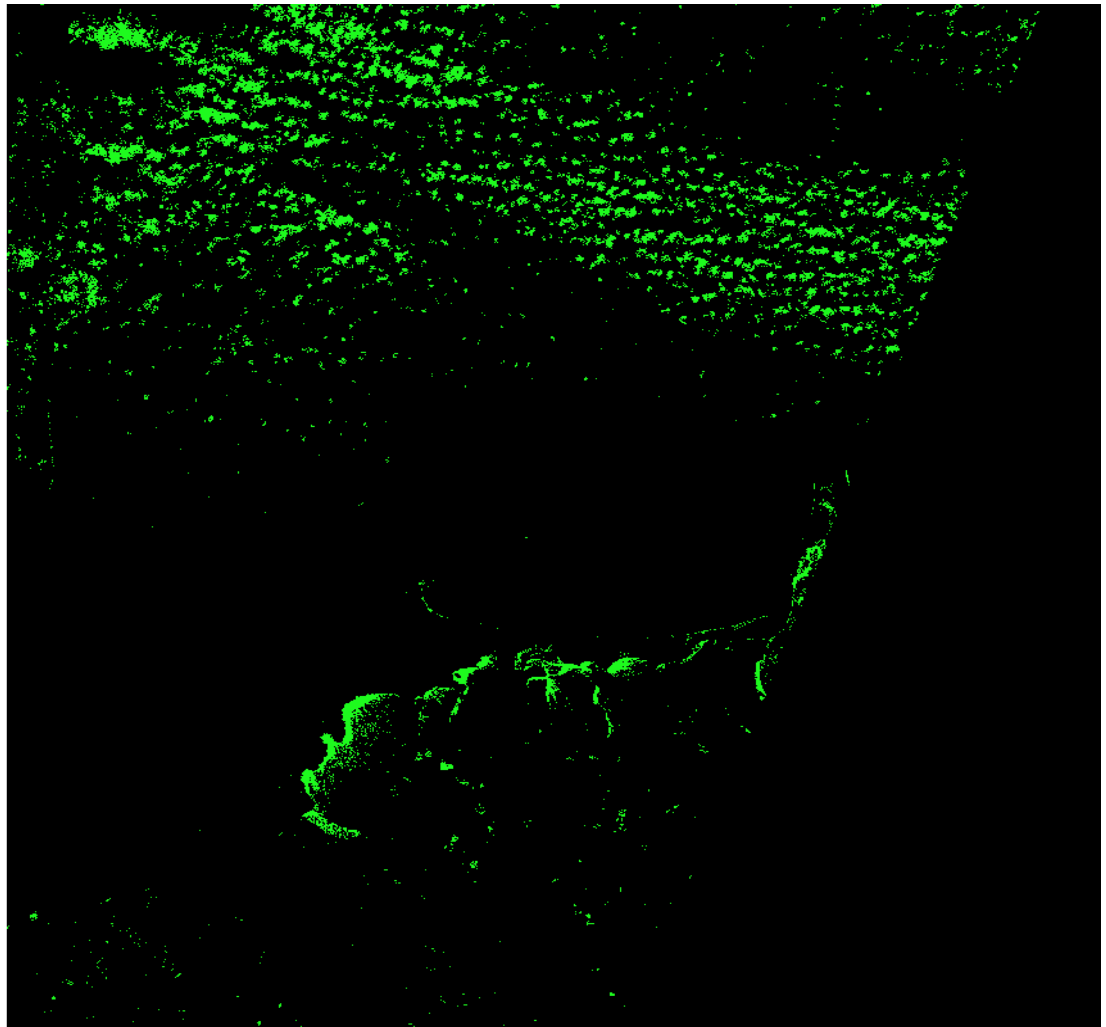
D.1 Descriptive statistics of a 2D image

Single statistical values	
Q1	Categorical statistics of a 2D image: How many m² of the image were covered by Snow or water ice (SN; 29) on 10.01.2017 (Pixel resolution is 10x10m)?
Query	for \$s in (siam) return encode (count((\$s[ansi("2017-01-10T00:00:00.000Z")]=21 or \$s[ansi("2017-01-10T00:00:00.000Z")]=22))*10*10, "csv")
Result	73719300
Q2	Categorical statistics of a 2D image (%): How much of the specified image subset was covered by cloud (CL; 25) on 19.06.2017?
Query	for \$s in (siam) let \$sub := [E(305863:315593), N(5996151:6008781), ansi("2017-06-19T00:00:00.000Z")] return (count(\$s[\$sub]=25)) / (count(\$s[\$sub]>1 and \$s[\$sub]<34)) *100
Result	31.56563562098051
Q3	Categorical statistics of a 2D image for a chosen polygon: How many pixels in the specified AOI were of category 29 (SN) on 10.01.2017?
Query	for \$s in (siam) return count(clip((\$s[ansi("2017-01-10T00:00:00.000Z")]=29), POLYGON((323014.428445768 6018724.6989753, 337518.076649877 6042736.29433543, 337518.076649877 6042736.29433543, 337518.076649877 6042736.29433543, 372326.832339739 6038465.77569756, 375549.865273986 6035967.92517351, 363302.340123849 6022350.61102632, 345736.810632206 6025654.21978393, 333489.285482069 6010344.81334626, 323014.428445768 6018724.6989753))

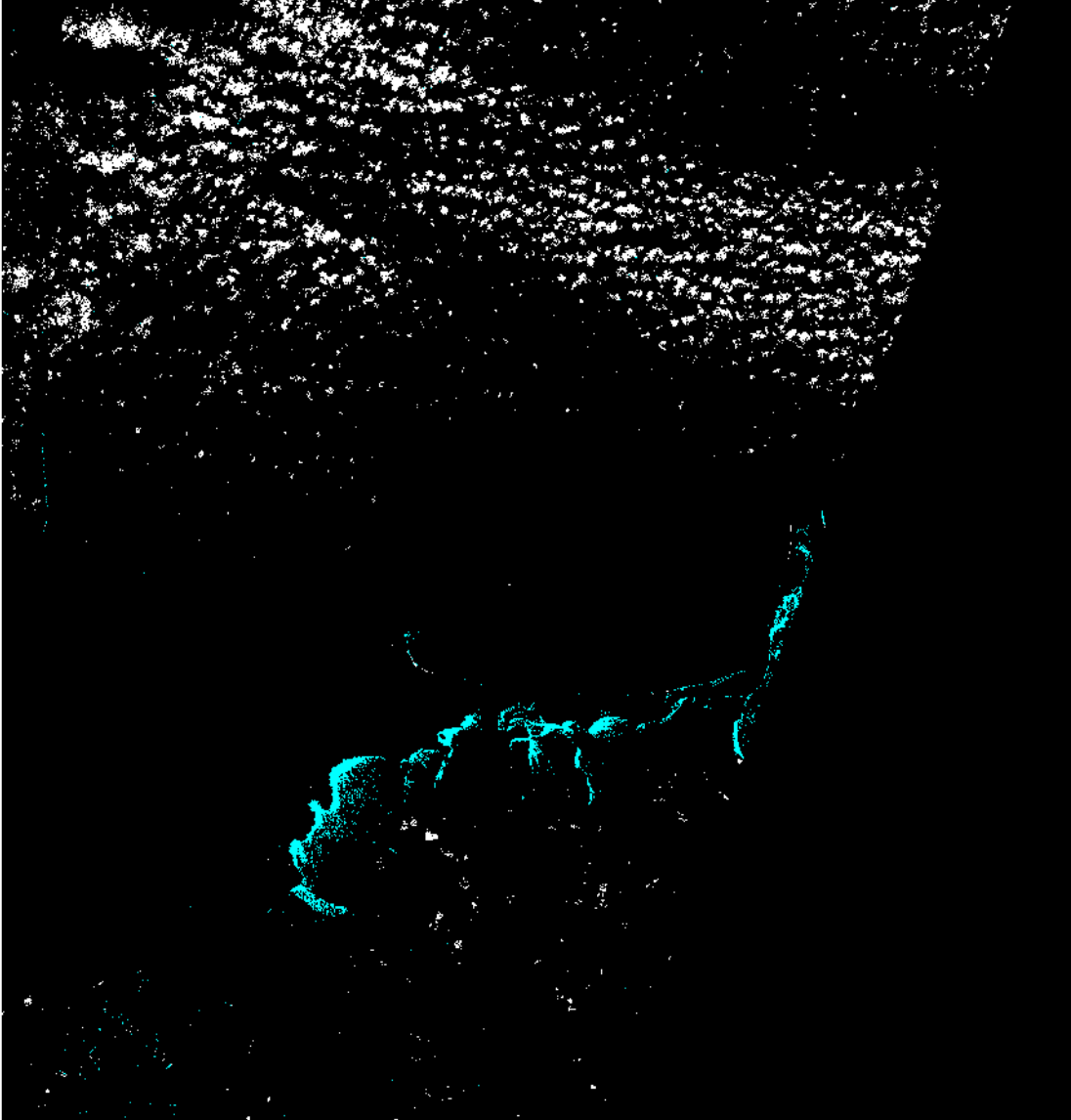
))																																														
Result	82246																																														
	Histograms																																														
Q4	Create values for a histogram with the number of pixels for each category for the 10.01.2017																																														
Query	for \$s in (siam) return encode(coverage histogram over \$bucket x(1 : 33) values count(\$s[ansi("2017-01-10T00:00:00.000Z")] = \$bucket), "text/csv")																																														
Result	<table border="1"> <tr> <td>21</td><td>0</td><td>1540165</td><td>1226635</td><td>2923585</td><td>178058</td><td>115228</td><td>73887</td><td>504</td><td>103813</td><td>6</td><td>1</td> </tr> <tr> <td>17517</td><td>30938</td><td>614776</td><td>483205</td><td>123941</td><td>674</td><td>53</td><td>0</td><td>47847486</td><td>510706</td><td>4597023</td><td></td> </tr> <tr> <td>0</td><td>3390980</td><td>0</td><td>7967283</td><td>10663439</td><td>737193</td><td>4007812</td><td>851273</td><td>0</td><td>184065</td><td></td><td></td> </tr> </table>											21	0	1540165	1226635	2923585	178058	115228	73887	504	103813	6	1	17517	30938	614776	483205	123941	674	53	0	47847486	510706	4597023		0	3390980	0	7967283	10663439	737193	4007812	851273	0	184065		
21	0	1540165	1226635	2923585	178058	115228	73887	504	103813	6	1																																				
17517	30938	614776	483205	123941	674	53	0	47847486	510706	4597023																																					
0	3390980	0	7967283	10663439	737193	4007812	851273	0	184065																																						
Q5	Create values for a histogram with the number of pixels for categories 25 (CL) and 29 (SN) for the 10.01.2017																																														
Query	for \$s in (siam) return encode(coverage histogram over \$bucket x(1 : 2) values switch case \$bucket=1 return count((\$s[ansi("2017-01-10T00:00:00.000Z")] = 25)) default return count((\$s[ansi("2017-01-10T00:00:00.000Z")] = 29)) ,"text/csv")																																														
Result	<table border="1"> <tr> <td>3390980</td> <td>737193</td> </tr> </table>											3390980	737193																																		
3390980	737193																																														
Q6	Create values for a pie chart/stacked bar chart with share of pixels for each category per total amount of pixels (%) for the 10.01.2017																																														
Query	for \$s in (siam)																																														

	<pre>return encode(coverage histogram over \$bucket x(1 : 33) values (count(\$s[ansi("2017-01-10T00:00:00.000Z")] = \$bucket) / count((\$s[ansi("2017-01-10T00:00:00.000Z")]>0) and (\$s[ansi("2017-01-10T00:00:00.000Z")]<34))*100 ,"text/csv")</pre>																																	
<p>Note</p>	<p>Results returned by rasdaman have not been rounded, this has been done with a spreadsheet program</p>																																	
<p>Result</p>	<table border="1" data-bbox="331 707 1353 779"> <tr> <td>0</td><td>0</td><td>2</td><td>1</td><td>3</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td> </tr> </table> <table border="1" data-bbox="331 835 1289 907"> <tr> <td>0</td><td>0</td><td>0</td><td>54</td><td>1</td><td>5</td><td>0</td><td>4</td><td>0</td><td>9</td><td>12</td><td>1</td><td>5</td><td>1</td><td>0</td><td>0</td> </tr> </table>	0	0	2	1	3	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	54	1	5	0	4	0	9	12	1	5	1	0	0
0	0	2	1	3	0	0	0	0	0	0	0	0	0	1	1	0																		
0	0	0	54	1	5	0	4	0	9	12	1	5	1	0	0																			

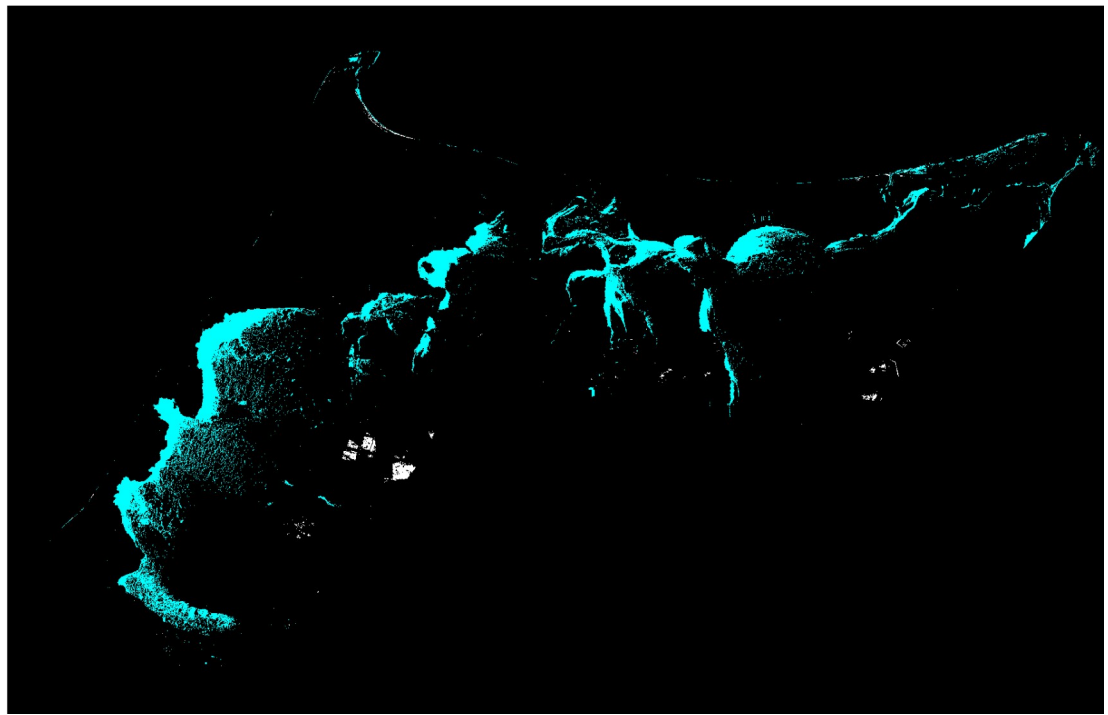
6.2 D.2 Simple selection and display of (composite) categories

Queries that focus at a chosen date (slice on the time axis) and return a 2D map - Selection based on categories	
Q7	Simple selection of a composite category consisting of semi-concepts 25 and 29 (CL and SN) for one scene
Query	<pre> for \$s in (siam) let \$sub := [ansi("2016-01-06T00:00:00.000Z")] return encode((unsigned char) (\$s[\$sub]=25 or \$s[\$sub]=29) , "tiff") </pre>
Result	

Q8	Simple selection of categories 25 (CL) and 29 (SN) in one scene and color coding them (CL = white, SN = blue)
Query	<pre> for \$s in (siam) let \$sub := [ansi("2016-01-06T00:00:00.000Z")] return encode(((unsigned char) switch case \$s[\$sub]=25 return {red:255; green: 255; blue: 255} case \$s[\$sub]=29 return {red:0; green: 255; blue: 255} default return {red: 0; green: 0; blue: 0}) , "tiff") </pre>
Caveat	<p>Potential bug in rasdaman v10.0.0-beta3. It should be noted, that expression optimization did not work as stated in the section 'Induced Operations' found at www.earthserver.eu/wcs (Last accessed: 25.04.2022).</p> <p>The following query did not work but threw an exception:</p> <pre> for \$c in (siam) let \$sub := [ansi("2017-01-10T00:00:00.000Z")] return encode(((unsigned char) switch case \$c=29 return {red:0; green: 0; blue: 255} case \$c=30 return {red:0; green: 255; blue: 0} default return {red: 0; green: 0; blue: 0})[\$sub] , "tiff") </pre> <p>→ <ows:ExceptionText>Failed closing rasdaman db connection: RasManager Error: Could not connect to RasServer .</ows:ExceptionText></p> <p>This might have been fixed already in the stable release of rasdaman v10.0.0.</p>

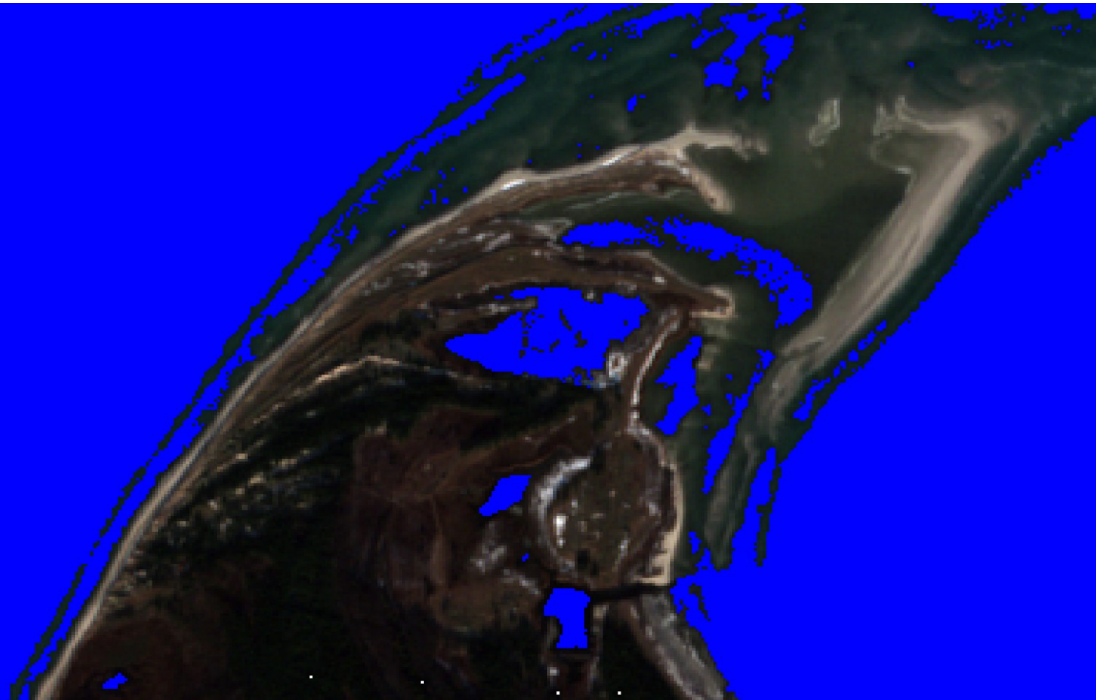
Result	
Q9	<p>Simple selection of categories 25 (CL) and 29 (SN) in one scene and color coding them (CL = white, SN = blue) WITH clipping of an AOI</p>
Query	<pre>for \$s in (siam) let \$sub := [ansi("2016-01-06T00:00:00.000Z")] return encode(clip(((unsigned char) switch case \$s[\$sub]=25 return {red:255; green: 255; blue: 255} case \$s[\$sub]=29 return {red:0; green: 255; blue: 255} default return {red: 0; green: 0; blue: 0}),</pre>

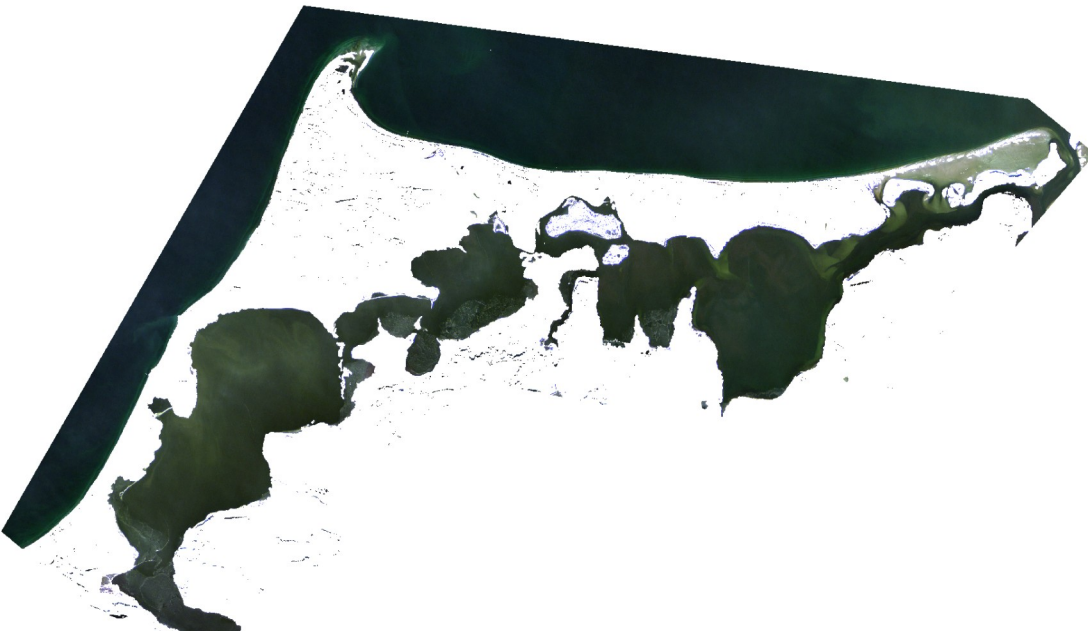
```
POLYGON((  
323014.428445768 6018724.6989753, 337518.076649877  
6042736.29433543, 337518.076649877 6042736.29433543,  
337518.076649877 6042736.29433543, 372326.832339739  
6038465.77569756, 375549.865273986 6035967.92517351,  
363302.340123849 6022350.61102632, 345736.810632206  
6025654.21978393, 333489.285482069 6010344.81334626,  
323014.428445768 6018724.6989753  
))  
)  
, "tiff")
```

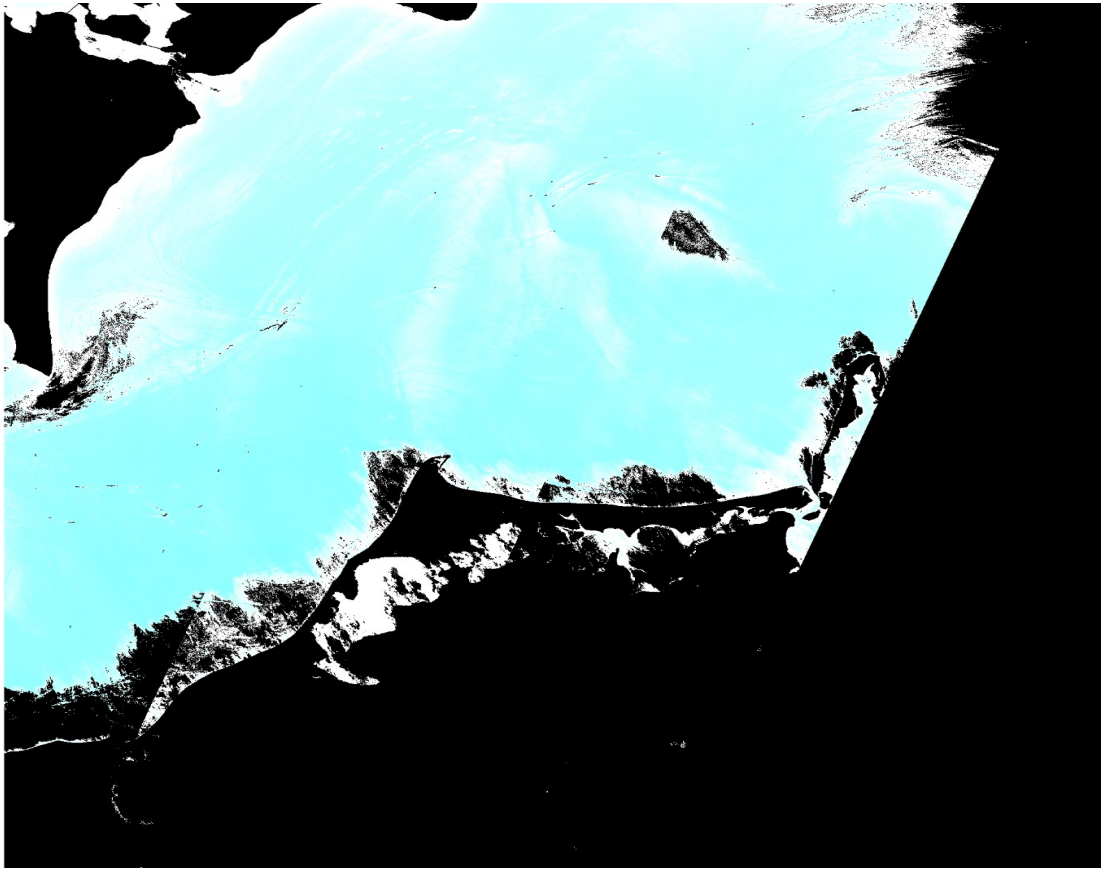


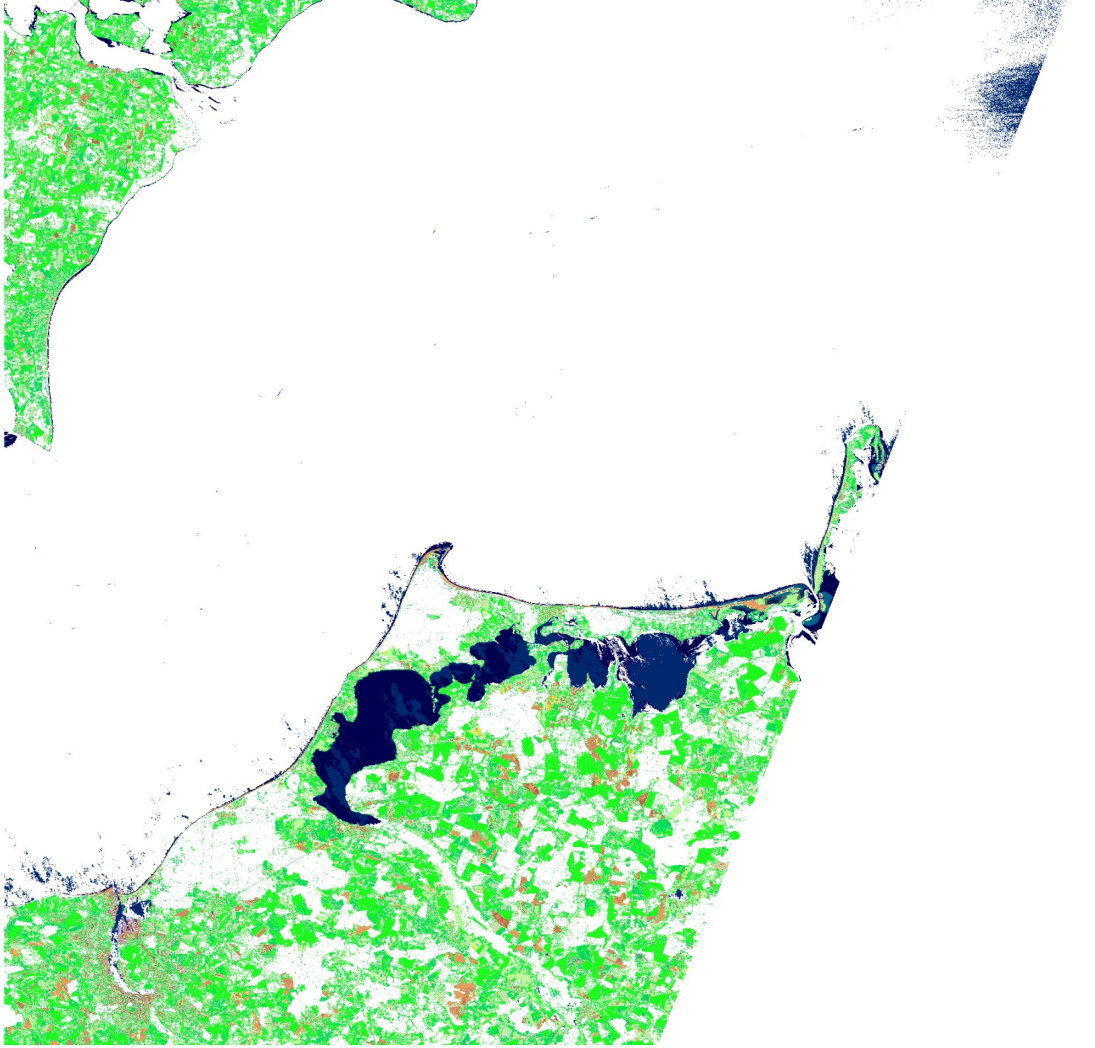
D.3 Fusing Sentinel-2 and SIAM™ data sets


Using the Overlay function to fuse SIAM™ and Sentinel-2 data	
Q10	Select category 'Deep water or shadow' (DPWASH; 21) as a blue are and return it with a Sentinel-2 RGB composite background
Query	<pre> for \$c in (siam), \$s2 in (S2_10m) let \$sub := [E(337580:341342), N(6038490:6041067), ansi("2017-01-10T00:00:00.000Z")] return encode(((unsigned char) switch case \$c[\$sub]=21 return {red:1; green: 1; blue: 255} default return {red: 0; green: 0; blue: 0}) overlay (unsigned char){ red: (\$s2[\$sub].B4 - min(\$s2[\$sub].B4)) / (max(\$s2[\$sub].B4) - min(\$s2[\$sub].B4))*255; green: (\$s2[\$sub].B3 - min(\$s2[\$sub].B3)) / (max(\$s2[\$sub].B3) - min(\$s2[\$sub].B3))*255; blue: (\$s2[\$sub].B2 - min(\$s2[\$sub].B2)) / (max(\$s2[\$sub].B2) - min(\$s2[\$sub].B2))*255 } , "tiff") </pre>
Caveat	<ul style="list-style-type: none"> • The datasets fused with an overlay operator have to have the same data type and are therefore both casted to 8 bit integer (unsigned char). • As the range of the Sentinel-2 values in each band exceed the 256 digits that can be stored with an unsigned char, the band values have to be recalculated to match this requirement. Here, the histogram stretch formula $(\text{Band} - \min(\text{Band})) / (\max(\text{Band}) - \min(\text{Band}))$ is applied to each Sentinel-2 band to solve this issue.

Result	
Q11	<p>Hide SIAM™ categories that are not DPWASH (21) or Shadow snow* (SHSN; 30) to get only the relevant Sentinel-2 band combination (Band 4, Band 3, Band 1) for an AOI defined by a clip</p> <p>* Selecting Shadow snow (SHSN) delivered better results when trying to select the lagoon in front of the Darß Peninnsula than selecting a water-related SIAM™ category like SLWASH, TWASH or SASLWA (22, 23 or 24). The reason has not been further investigated in this thesis, as the focus is on WCPS mechanics and not on SIAM™ evaluation</p>
Query	<pre> for \$s in (siam), \$s2 in (S2_10m), \$s2_60m in (S2_20170110_32633_60m_L1C) let \$sub := [ansi("2017-01-10T00:00:00.000Z")], \$scale_used := \$s2[\$sub] return encode(clip((((unsigned char) switch case \$s[\$sub]!=21 and \$s[\$sub]!=30 return {red:1; green: 1; blue: 1} default return {red: 0; green: 0; blue: 0}) overlay ((unsigned char){ red: \$s2[\$sub].B4 / 10; green: \$s2[\$sub].B3 / 10; </pre>

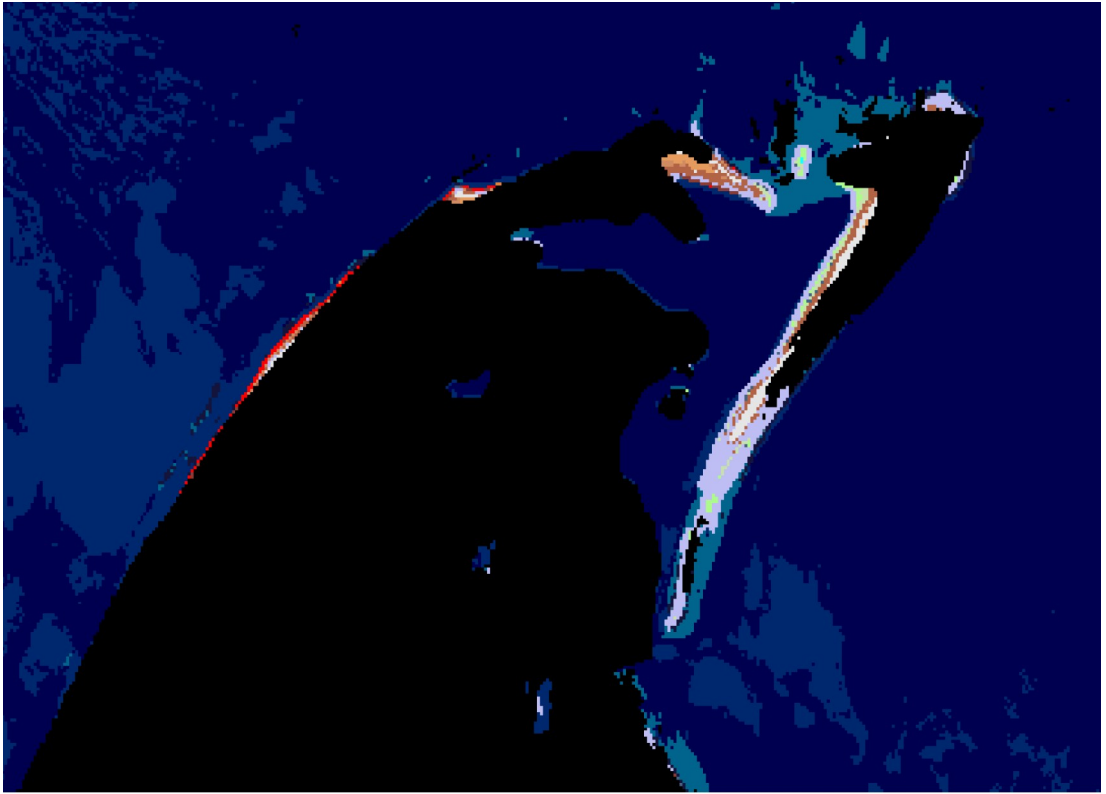
	<pre> blue: scale(\$s2_60m[\$sub], {imageCrsDomain(\$scale_used)}).B1 / 10 })), POLYGON((323014.428445768 6018724.6989753, 337518.076649877 6042736.29433543, 337518.076649877 6042736.29433543, 337518.076649877 6042736.29433543, 372326.832339739 6038465.77569756, 375549.865273986 6035967.92517351, 363302.340123849 6022350.61102632, 345736.810632206 6025654.21978393, 333489.285482069 6010344.81334626, 323014.428445768 6018724.6989753)))), "tiff") </pre>
<p>Note</p>	<ul style="list-style-type: none"> • The overlay function requires the coverages to be fused to have the same resolution. As the coastal aerosol band of Sentinel-2 (Band 1) has a resolution of only 60m compared to Band 3 and Band 4 which have 10m, the scaling function is used to resample the band to match with the other bands. • In the output coverage, the selected SIAM™ categories are displayed as black, as {red:1; green: 1; blue: 1} have been chosen. Note that thus the output coverage has a value of 1 in each band for coverage = 21 or coverage = 30. For the result image here, 1 was classified as additional no data value in QGIS to a) exclude it from being displayed, and b) from having an influence on the min/max settings for calculating the histogram stretch
<p>Result</p>	

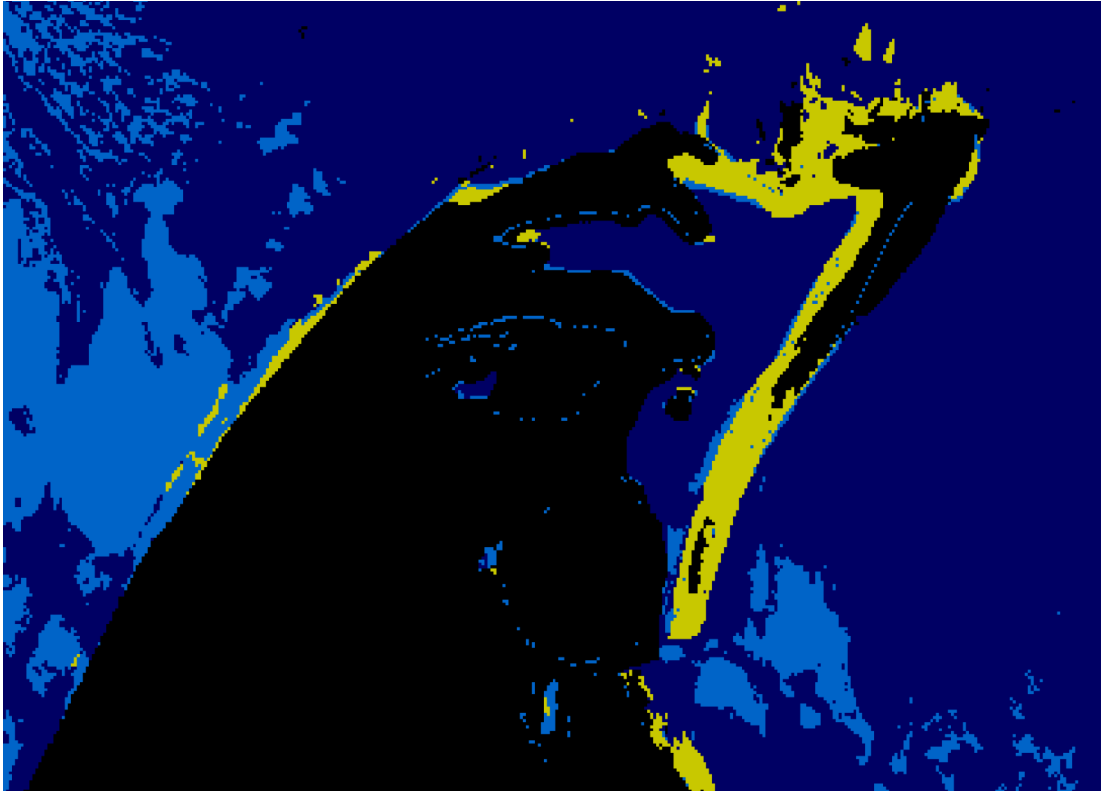
Select pixels of a coverage based on a condition met in another coverage	
Q12	Deriving a coverage with Sentinel Band 2-4 and 8 only for pixels that are DPWASH (21)
Query	for \$s in (siam), \$s2 in (S2_10m) return encode((unsigned char) ((\$s[ansi("2021-06-18T00:00:00.000Z")] = 21)* (\$s2[ansi("2021-06-18T00:00:00.000Z")]))) , "tiff")
Note	<ul style="list-style-type: none"> • Image returned was opened in QGIS and Bands 8, 4, 3 were chosen in this order for the result presented here • The approach is more direct than the overlay approach of Q10 to select Sentinel-2 data only for specific target categories
Result	

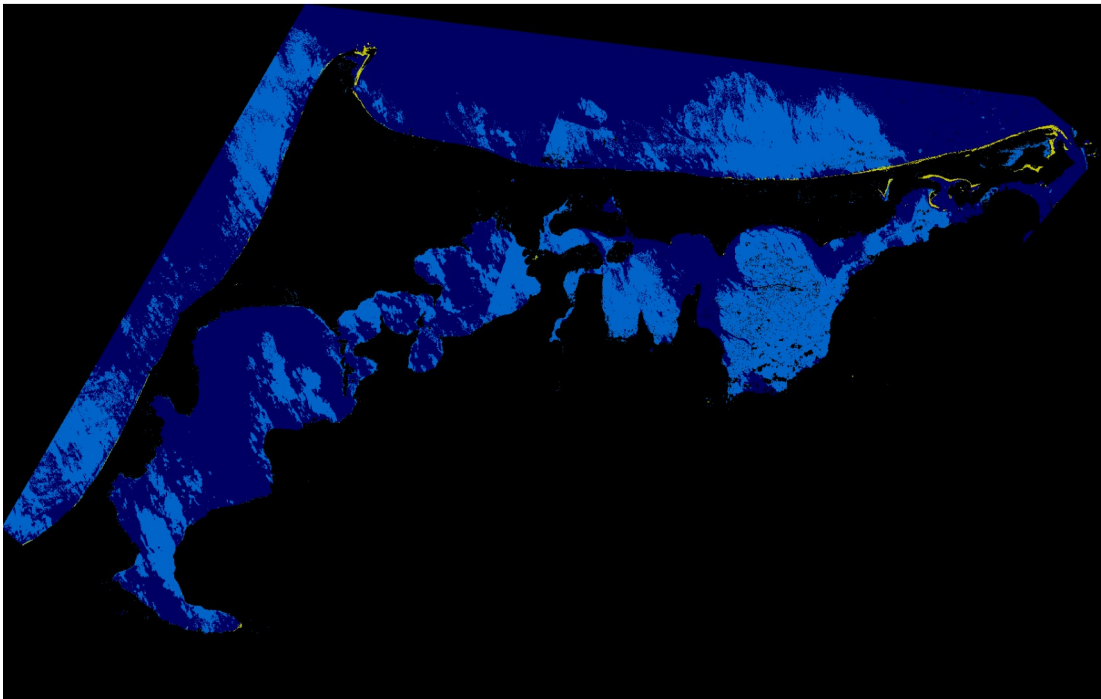
Q13	Deriving a coverage with SIAM™ categories only for pixels that have values in the B4 band that are higher than 500
Query	<pre> for \$s in (siam), \$s2 in (S2_10m) return encode((unsigned char) ((\$s2[ansi("2021-06-18T00:00:00.000Z")].B4 > 500) * \$s[ansi("2021-06-18T00:00:00.000Z")]) , "tiff") </pre>
Note	<p>The coverage gets exported without a color palette applied. For this image, the color palette of the standard SIAM™ layer was copied and pasted to the output image in QGIS.</p>
Result	

Q14	Deriving a coverage with Sentinel Band 2-4 and 8 only for pixels that are DPWASH (21) or Shallow water or shadow (SLWASH; 23) and where Sentinel-2 values in the B4 band are higher than 500
Query	<pre> for \$s in (siam), \$s2 in (S2_10m) return encode((\$s2[ansi("2021-06-18T00:00:00.000Z")].B4 < 500) * ((\$s[ansi("2021-06-18T00:00:00.000Z")]!=21) and (\$s[ansi("2021-06-18T00:00:00.000Z")]!=22)) * \$s2[ansi("2021-06-18T00:00:00.000Z")] , "tiff") </pre>
Note	Image returned was opened in QGIS and Bands 8, 4, 3 were chosen for output image
Result	 <p>The result is a satellite image showing a coastal area. The image is composed of various colored pixels, including green, brown, and white, which represent different land cover types. The image is oriented vertically, with the coastline on the left side. The land area is characterized by a mix of colors, indicating different vegetation and land use patterns. The water area is represented by a light blue color. The image is a mosaic of Sentinel-2 satellite imagery, showing the results of the query for pixels with Sentinel-2 values in the B4 band higher than 500, and pixels that are DPWASH (21) or Shallow water or shadow (SLWASH; 23).</p>

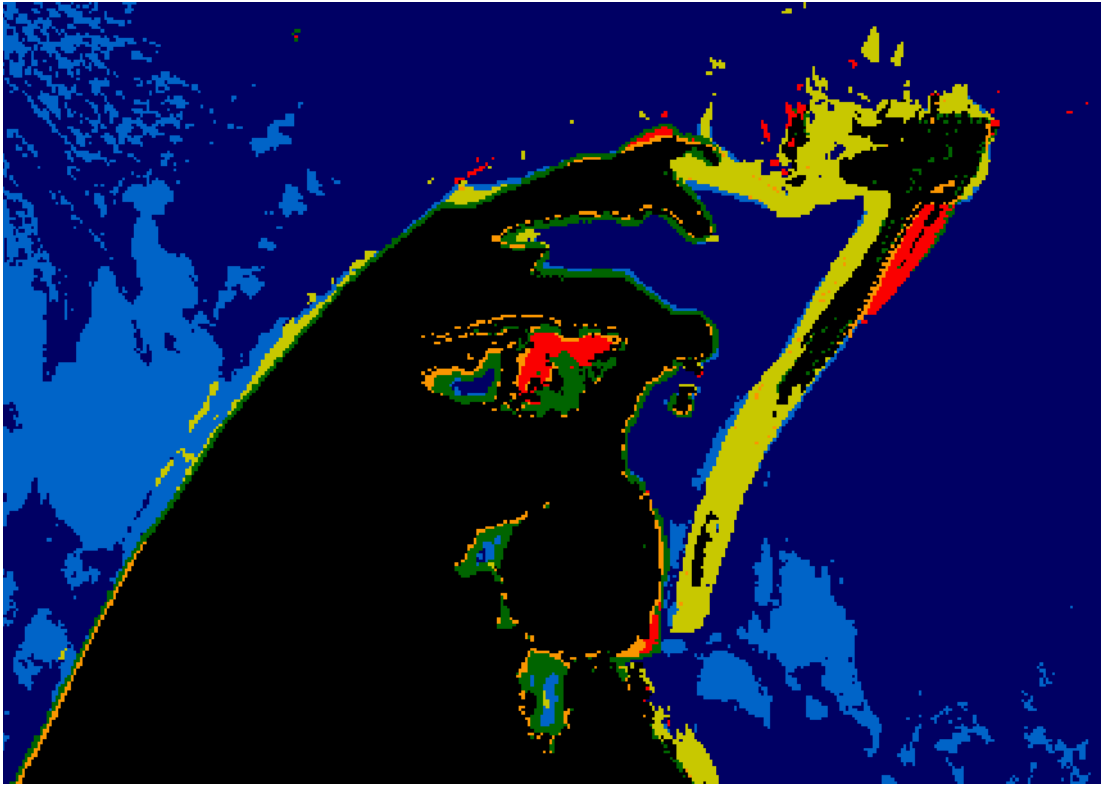
D.4 Investigating categorical trajectories

Queries 2D Land cover change	
Q15	What categories do pixels that were classified as DPWASH (21) in the oldest image belong to in the latest image?
Query	<pre> for \$s in (siam) let \$sub_hi := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).hi)], \$sub_lo := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).lo)] return encode((unsigned char) ((\$s[\$sub_lo] = 21) * \$s[\$sub_hi]) , "tiff") </pre>
Result	
Q16	<p>Observing the trajectory of a category.</p> <p>Selecting pixels, that have been classified as 'Deep water or shadow' (DPWASH; 21) in the oldest satellite scene and either</p> <ul style="list-style-type: none"> • stayed the same (dark blue) • was classified as 'Shallow water or shadow' (SLWASH; 22), 'Turbid water or shadow' (TWASH; 23) or 'Salty shallow water'

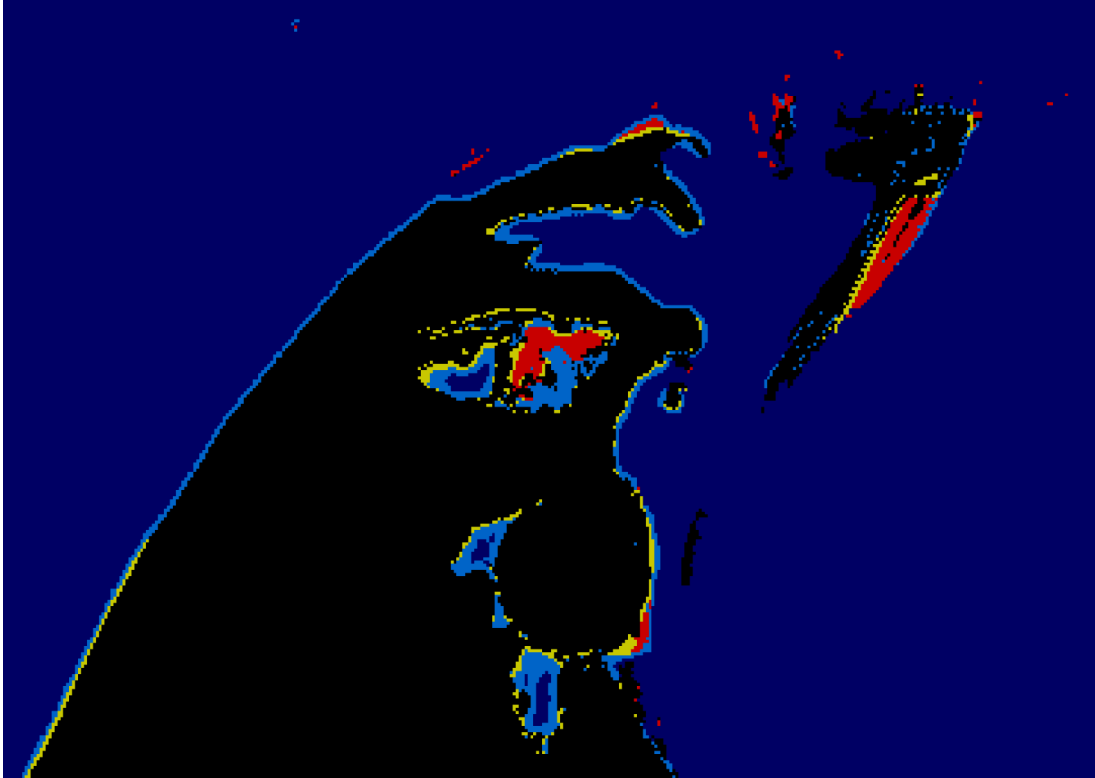
	<p>(SASLWA; 24) (light blue)</p> <ul style="list-style-type: none"> was not classified a water related category anymore (yellow)
Query	<pre> for \$s in (siam) let \$sub_hi := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).hi)], \$sub_lo := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).lo)] return encode(((unsigned char) switch case \$s[\$sub_lo] = 21 and \$s[\$sub_hi] = 21 return {red:0; green: 0; blue: 100} case \$s[\$sub_lo] = 21 and (\$s[\$sub_hi] = 22 or \$s[\$sub_hi] = 23 or \$s[\$sub_hi] = 24 return {red:0; green: 100; blue: 200} case \$s[\$sub_lo] = 21 and \$s[\$sub_hi] != 21 and \$s[\$sub_hi] != 22 and \$s[\$sub_hi] != 23 and \$s[\$sub_hi] != 24 return {red:200; green: 200; blue: 0} default return {red: 0; green: 0; blue: 0}), "tiff") </pre>
Note	<p>The red part is not necessary and only included to be more explicit. The cases excluded should have been caught already in the case statements before as explained in Appendix A.5</p>
Result	 <p>The image shows a satellite-style view of a coastline. A yellow highlight traces a path along the shore, starting from the top right and moving towards the bottom left. The land is shown in shades of brown and green, while the water is dark blue. The yellow highlight appears to be a result of a classification or filtering process applied to the image.</p>

Q17	Observing the trajectory of a category (Same query as Q16 but clipped)
Query	<pre> for \$s in (siam) let \$sub_hi := [ansi:"CRS:1"(imageCrsDomain(\$s, ansi).hi)], \$sub_lo := [ansi:"CRS:1"(imageCrsDomain(\$s, ansi).lo)] return encode(clip((unsigned char) ((unsigned char) switch case \$s[\$sub_lo] = 21 and \$s[\$sub_hi] = 21 return {red:0; green: 0; blue: 100} case \$s[\$sub_lo] = 21 and (\$s[\$sub_hi] = 22 or \$s[\$sub_hi] = 23 or \$s[\$sub_hi] = 24 return {red:0; green: 100; blue: 200} case \$s[\$sub_lo] = 21 and \$s[\$sub_hi] != 21 return {red:200; green: 200; blue: 0} default return {red: 0; green: 0; blue: 0})), POLYGON((323014.428445768 6018724.6989753, 337518.076649877 6042736.29433543, 337518.076649877 6042736.29433543, 337518.076649877 6042736.29433543, 372326.832339739 6038465.77569756, 375549.865273986 6035967.92517351, 363302.340123849 6022350.61102632, 345736.810632206 6025654.21978393, 333489.285482069 6010344.81334626, 323014.428445768 6018724.6989753))), "tiff) </pre>
Result	

<p>Q18</p>	<p>Observing the trajectory of a category.</p> <p>Selecting pixels that</p> <ul style="list-style-type: none"> • were not part of any water related category in the oldest satellite scene and are DPWASH (21) in the most recent scene (red) • were not part of any water related category in the oldest satellite scene and are SLWASH, TWASH or SASLWA (22, 23, 24) in the most recent scene (orange) • were 21 and did not change (dark blue) • were 21 and turned into 22, 23, 24 (light blue) • were 21 and turned into something non water-related (yellow) • were 22, 23, 24 and did not change (green) • were 22, 23, 24 and turned into 21 (dark purple) • were 22, 23, 24 and turned into something non water-related (light purple)
<p>Query</p>	<pre> for \$s in (siam) let \$sub_hi := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).hi)], \$sub_lo := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).lo)] return encode(((unsigned char) switch case \$s[\$sub_lo] != 21 and \$s[\$sub_lo] != 22 and \$s[\$sub_lo] != 23 and \$s[\$sub_lo] != 24 and \$s[\$sub_hi] = 21 return {red:250; green: 0; blue: 0} case \$s[\$sub_lo] != 21 and \$s[\$sub_lo] != 22 and \$s[\$sub_lo] != 23 and \$s[\$sub_lo] != 24 and (\$s[\$sub_hi] = 22 or \$s[\$sub_hi] = 23 or \$s[\$sub_hi] = 24) return {red:250; green: 150; blue: 0} case \$s[\$sub_lo] = 21 and \$s[\$sub_hi] = 21 return {red:0; green: 0; blue: 100} case \$s[\$sub_lo] = 21 and (\$s[\$sub_hi] = 22 or \$s[\$sub_hi] = 23 or \$s[\$sub_hi] = 24 return {red:0; green: 100; blue: 200} case \$s[\$sub_lo] = 21 return {red:200; green: 200; blue: 0} case \$s[\$sub_lo] = 22 or \$s[\$sub_lo] = 23 or \$s[\$sub_lo] = 24 and (\$s[\$sub_hi] = 22 or \$s[\$sub_hi] = 23 or \$s[\$sub_hi] = 24 return {red:0; green: 100; blue: 0} case \$s[\$sub_lo] = 22 or \$s[\$sub_lo] = 23 or \$s[\$sub_lo] = 24 and </pre>

	<pre> \$\$[\$sub_hi] = 21 return {red:100; green: 0; blue: 100} case \$\$[\$sub_lo] = 22 or \$\$[\$sub_lo] = 23 or \$\$[\$sub_lo] = 24 and \$\$[\$sub_hi] != 21 return {red:250; green: 0; blue: 250} default return {red: 0; green: 0; blue: 0}) , "tiff") </pre>
Note	<p>The last two cases</p> <ul style="list-style-type: none"> were 22, 23, 24 and turned into 21 (dark purple) were 22, 23, 24 and turned into something non water-related (light purple) <p>did not occur for the two timestamps queried, so no purple can be seen in the output image</p>
Result	
Q19	<p>Alternative method using overlay</p> <p>Which pixel are DPWASH (21) and which have been SLWASH, TWASH or SASLWA (22, 23, 24) in in the newest images that have not been classified like that in the oldest images</p>
Query	<pre> for \$\$ in (siam) let \$sub_hi := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$\$, ansi).hi)], </pre>

	<pre> \$sub_lo := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).lo)] return encode(((unsigned char) switch case \$s[\$sub_lo] = 21 return {red:0; green: 0; blue: 100} case \$s[\$sub_lo] = 22 or \$s[\$sub_lo] = 23 or \$s[\$sub_lo] = 24 return {red:0; green: 100; blue: 200} default return {red: 0; green: 0; blue: 0}) overlay (unsigned char) switch case \$s[\$sub_hi] = 21 return {red:200; green: 0; blue: 0} case \$s[\$sub_hi] = 22 or \$s[\$sub_hi] = 23 or \$s[\$sub_hi] = 24 return {red:200; green: 200; blue: 0} default return {red: 0; green: 0; blue: 0} , "tiff") </pre>
<p>Note: Query result is equivalent to Query above</p>	<pre> for \$s in (siam) let \$sub_hi := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).hi)], \$sub_lo := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).lo)] return encode(((unsigned char) switch case \$s[\$sub_lo] = 21 return {red:0; green: 0; blue: 100} case \$s[\$sub_lo] = 22 or \$s[\$sub_lo] = 23 or \$s[\$sub_lo] = 24 return {red:0; green: 100; blue: 200} case \$s[\$sub_hi] = 22 or \$s[\$sub_hi] = 23 or \$s[\$sub_hi] = 24 return {red:200; green: 200; blue: 0} case \$s[\$sub_hi] = 21 return {red:200; green: 0; blue: 0} default return {red: 0; green: 0; blue: 0}) , "tiff") </pre>

Result	
Q20	<p>Tracking different trajectories and comparing them: Checking which pixels have gone through the development</p> <p>'Deep Water' (DPWASH; 21) - 'Shallow water' (SLWASH, TWASH, SASLWA; 22, 34, 24) - 'Non-water'</p> <p>compared to those that have gone through</p> <p>'Water' - 'Non-water'- 'Non-water'</p> <p>from the oldest image to 19.06.2017 to the newest image</p>
Query	<pre> for \$s in (siam) let \$sub_hi := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).hi)], \$sub_lo := [E(337580:341342), N(6038490:6041067), ansi:"CRS:1"(imageCrsDomain(\$s, ansi).lo)], \$sub_mi := [E(337580:341342), N(6038490:6041067), ansi("2017-06-19T00:00:00.000Z")] return encode(((unsigned char) switch </pre>

```

case $$[sub_lo] = 21 and
($s[sub_mi] = 22 or $$[sub_mi] = 23 or $$[sub_mi] = 24) and ($s[sub_hi]
!= 21 and $$[sub_hi] != 22 and $$[sub_hi] != 23 and $$[sub_hi] != 24 )
return {red:255; green: 255; blue: 0}

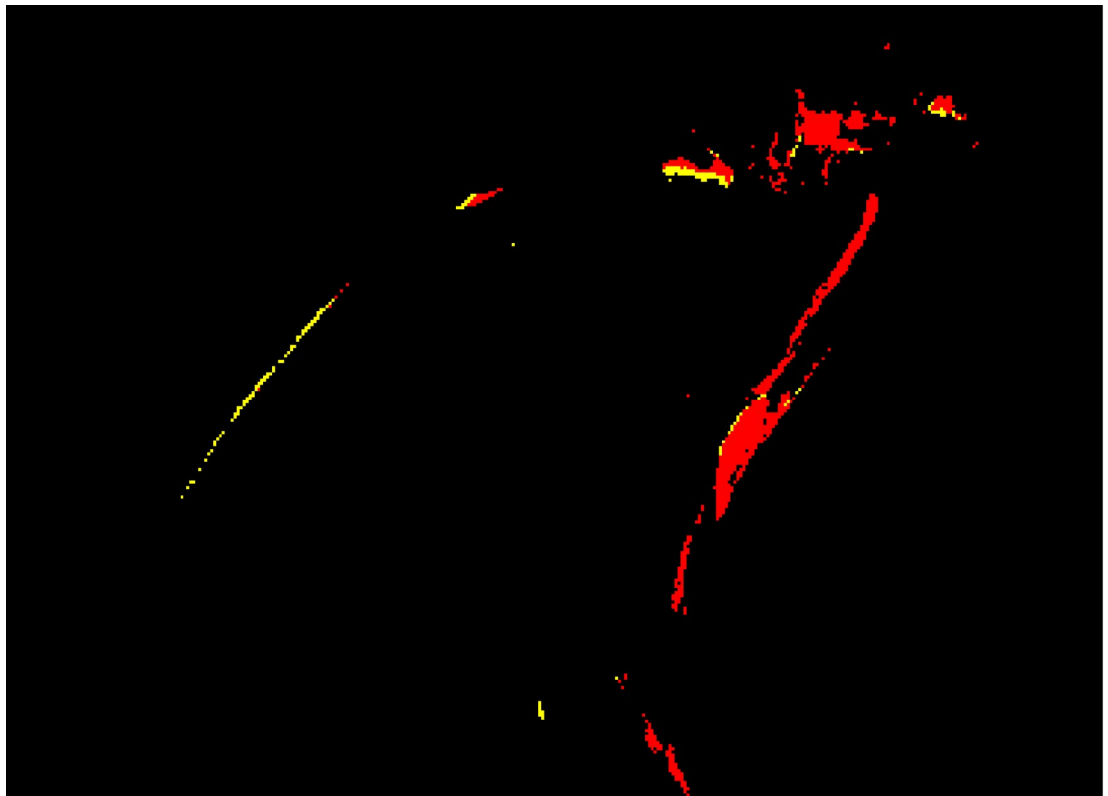
case $$[sub_lo] = 21 and
($s[sub_mi] != 21 and $$[sub_mi] != 22 and $$[sub_mi] != 23 and
$$[sub_mi] != 24) and
($s[sub_hi] != 21 and $$[sub_hi] != 22 and $$[sub_hi] != 23 and
$$[sub_hi] != 24 ) return {red:255; green: 0; blue: 0}

default return {red: 0; green: 0; blue: 0}


)
, "tiff")

```

Result



6.3 D.5 Time series analysis

2D map with values aggregated for a selected period of time	
Q21	Condensing along the time axis by summing up all observations of a (composite) category at a selected location and timespan. Example: Count all water-related pixels (DPWASH, SLWASH, TWASH, SASLWA; 21, 22, 34, 24) for all images between 19.06.2017 and 18.06.2021 (n=9)
Query	<pre>for \$c in (siam) let \$sub := \$c[E(333775:344085), N(6034398:6041229)] return encode(condense + over \$t ansi(imageCrsDomain(\$sub[ansi("2017-06-19T00:00:00.000Z":"2021-06-18T00:00:00.000Z")], ansi)) using \$sub[ansi(\$t)]=21 Or \$sub[ansi(\$t)]=22 Or \$sub[ansi(\$t)]=23 Or \$sub[ansi(\$t)]=24, "tiff")</pre>
Note	The result map shows values from black (0 observations) to white (9 observations)
Result	

Time series (and derived aggregate values) in which each timestamp is taken into account for a period of time

Q22	Follow how a category's occurrence (SN; 29) has changed over time for a selected period of time							
Query	for \$s in (siam) return encode(coverage timeseries over \$p ansi(imageCrsDomain(\$s[ansi("2017-01-01T00:00:00.000Z":"2020-12-31T00:00:00.000Z")], ansi)) values count(\$s[ansi(\$p)] = 29), "text/csv")							
Result	737193	23130	15625	667	2352	1062	702	892
Q23	Follow how the share of clouds of the satellite scene has changed over time for a selected period of time (%)							
Query	for \$s in (siam) return encode(coverage timeseries over \$t ansi(imageCrsDomain(\$s[ansi("2017-01-01T00:00:00.000Z":"2020-12-31T00:00:00.000Z")], ansi)) values count(\$s[ansi(\$t)] = 25) / count((\$s[ansi(\$t)]>0) and (\$s[ansi(\$t)]<34))*100 ,"text/csv")							
Result	3,85	5,64	0,01	0,04	0,04	0,11	0,07	0,07
Q24	Count pixels that belong to SN (29) for the year 2017 in the data cube and derive the maximum value							
Query	for \$s in (siam) return encode(max(coverage timeseries over \$p ansi(imageCrsDomain(\$s[ansi("2017-01-01T00:00:00.000Z":"2017-12-31T00:00:00.000Z")], ansi)) values count(\$s[ansi(\$p)] = 29)),							

	"text/csv")
Caveat	Make sure that the start and the endpoint of the requested interval lies within the time axis domain <ul style="list-style-type: none"> • Unfortunately, we don't know when the maximum happened • When using many or all dates of the data cube, this query becomes naturally slower
Result	737193
Q25	Calculate average cloud cover (CL; 25) for one year
Query	for \$s in (siam) return encode(avg(coverage timeseries over \$t ansi(imageCrsDomain(\$s[ansi("2017-01-01T00:00:00.000Z":"2017-12-31T00:00:00.000Z")], ansi)) values count(\$s[ansi(\$t)] = 25) / count((\$s[ansi(\$t)]>0) and (\$s[ansi(\$t)]<34))*100 ,"text/csv")
Caveat	Make sure that the start and the endpoint of the requested interval lies within the time axis domain
Result	4.74080453174745

Time series in which only selected timestamps are taken into account for a period of time							
Q26a	Following the development of pixels classified as Snow or water ice (SN; 29) over all summers :-) in a selected period of time						
Query	for \$s in (siam) return encode(coverage timeseries over \$t year(0:6) values count(\$s[ansi(\$t*2)] = 29) ,"text/csv")						
Result	996	1277	23130	667	1062	892	1213
Q26b	Following the development of SN (29) over all winters in a selected period of time						

Query	<pre> for \$s in (siam) return encode(coverage timeseries over \$t year(0:6) values count(\$s[ansi(\$t*2+1)] = 29) ,"text/csv") </pre>					
Result	518704	737193	15625	2352	702	22335348
Q28	Deriving the maximum SN (29) of a year or a season					
Caveat	It is important to know how many images are in the target timespan to iterate over an adequate temporal range in the coverage constructor.					
Query	<pre> for \$s in (siam) return encode(coverage timeseries over \$t year(1:6) values max(count(\$s[ansi(\$t*2)] = 29) , count(\$s[ansi(\$t*2+1)] = 29)) ,"text/csv") </pre>					
Result	518704	737193	15625	2352	892	22335348

D.6 Edge detection based on categorical data

Edge detection based on categorical data with the Sobel filter	
Q29	Using the Sobel Operator on just one SIAM™ category by defining a Boolean condition in the where-clause of each coverage condenser
Query	<pre> for \$s in (siam) let \$sub := \$s[E(340230:341230), N(6039800:6040800), ansi("2017-06-19T00:00:00.000Z")], \$kernel1 := coverage kernel1 over \$x x(-1:1), \$y y (-1:1) value list < 1; 0; -1; 2; 0; -2; 1; 0; -1>, \$kernel2 := coverage kernel2 over \$x x(-1:1), \$y y (-1:1) value list < 1; 2; 1; 0; 0; 0; -1; -2; -1> return encode(sqrt(pow(coverage Gx over \$px1 E(imageCrsDomain(\$sub, E)), \$py1 N(imageCrsDomain(\$sub, N)) values condense + over \$kx1 x(imageCrsDomain(\$kernel1, x)), \$ky1 y(imageCrsDomain(\$kernel1, y)) where \$sub[E(\$px1), N(\$py1)] =21 using \$kernel1[x(\$kx1), y(\$ky1)] * \$sub[E(\$px1+\$kx1), N(\$py1+\$ky1)], 2.0) + pow(coverage Gy over \$px2 E(imageCrsDomain(\$sub, E)), \$py2 N(imageCrsDomain(\$sub, N)) values condense + over \$kx2 x(imageCrsDomain(\$kernel2, x)), \$ky2 y(imageCrsDomain(\$kernel2, y)) where \$sub[E(\$px2), N(\$py2)] =21 using \$kernel1[x(\$kx2), y(\$ky2)] * \$sub[E(\$px2+\$kx2), N(\$py2+\$ky2)], 2.0)),"image/tiff") </pre>



Q30 Using the Sobel Operator on two SIAM™ categories (forming a composite category and potentially a real-life class) by defining two Boolean conditions in the where-clause of each coverage condenser

Query

```

for $s in (siam)
let $sub := $s[ E(340230:341230), N(6039800:6040800), ansi("2017-06-19T00:00:00.000Z")],
$kernel1 := coverage kernel1
over $x x(-1:1), $y y (-1:1)
value list < 1; 0; -1; 2; 0; -2; 1; 0; -1>,
$kernel2 := coverage kernel2
over $x x(-1:1), $y y (-1:1)
value list < 1; 2; 1; 0; 0; 0; -1; -2; -1>
return encode( sqrt(
pow( coverage Gx
over $px1 E(imageCrsDomain($sub, E)),
$py1 N(imageCrsDomain($sub, N))

```

```

values
condense +
over $kx1 x(imageCrsDomain($kernel1, x)),
$ky1 y(imageCrsDomain($kernel1, y))
where $sub[E($px1), N($py1)] =21 or $sub[E($px1), N($py1)] =30
using $kernel1[x($kx1), y($ky1)] * $sub[E($px1+$kx1), N($py1+$ky1)], 2.0)
+
pow( coverage Gy
over $px2 E(imageCrsDomain($sub, E)),
$py2 N(imageCrsDomain($sub, N))
values
condense +
over $kx2 x(imageCrsDomain($kernel2, x)),
$ky2 y(imageCrsDomain($kernel2, y))
where $sub[E($px2), N($py2)] =21 or $sub[E($px2), N($py2)] =30
using $kernel1[x($kx2), y($ky2)] * $sub[E($px2+$kx2), N($py2+$ky2)], 2.0)
),"image/tiff")

```



Q31	Using the Sobel Operator on just one SIAM™ category by defining a Boolean condition in the using-clause of each coverage condenser
Query	<pre> for \$s in (siam) let \$sub := \$s[E(340230:341230), N(6039800:6040800), ansi("2017-06-19T00:00:00.000Z")], \$kernel1 := coverage kernel1 over \$x x(-1:1), \$y y (-1:1) value list < 1; 0; -1; 2; 0; -2; 1; 0; -1>, \$kernel2 := coverage kernel2 over \$x x(-1:1), \$y y (-1:1) value list < 1; 2; 1; 0; 0; 0; -1; -2; -1> return encode(sqrt(pow(coverage Gx over \$px1 E(imageCrsDomain(\$sub, E)), \$py1 N(imageCrsDomain(\$sub, N)) values condense + over \$kx1 x(imageCrsDomain(\$kernel1, x)), \$ky1 y(imageCrsDomain(\$kernel1, y)) using \$kernel1[x(\$kx1), y(\$ky1)] * (\$sub[E(\$px1+\$kx1), N(\$py1+\$ky1)]=21) , 2.0) + pow(coverage Gy over \$px2 E(imageCrsDomain(\$sub, E)), \$py2 N(imageCrsDomain(\$sub, N)) values condense + over \$kx2 x(imageCrsDomain(\$kernel2, x)), \$ky2 y(imageCrsDomain(\$kernel2, y)) using \$kernel1[x(\$kx2), y(\$ky2)] * (\$sub[E(\$px2+\$kx2), N(\$py2+\$ky2)]=21) , 2.0))) ,"image/tiff") </pre>



Q32 Using the Sobel Operator on two SIAM™ categories (forming a composite category and potentially a real-life class) by defining two Boolean conditions in the using-clause of each coverage condenser

Query

```

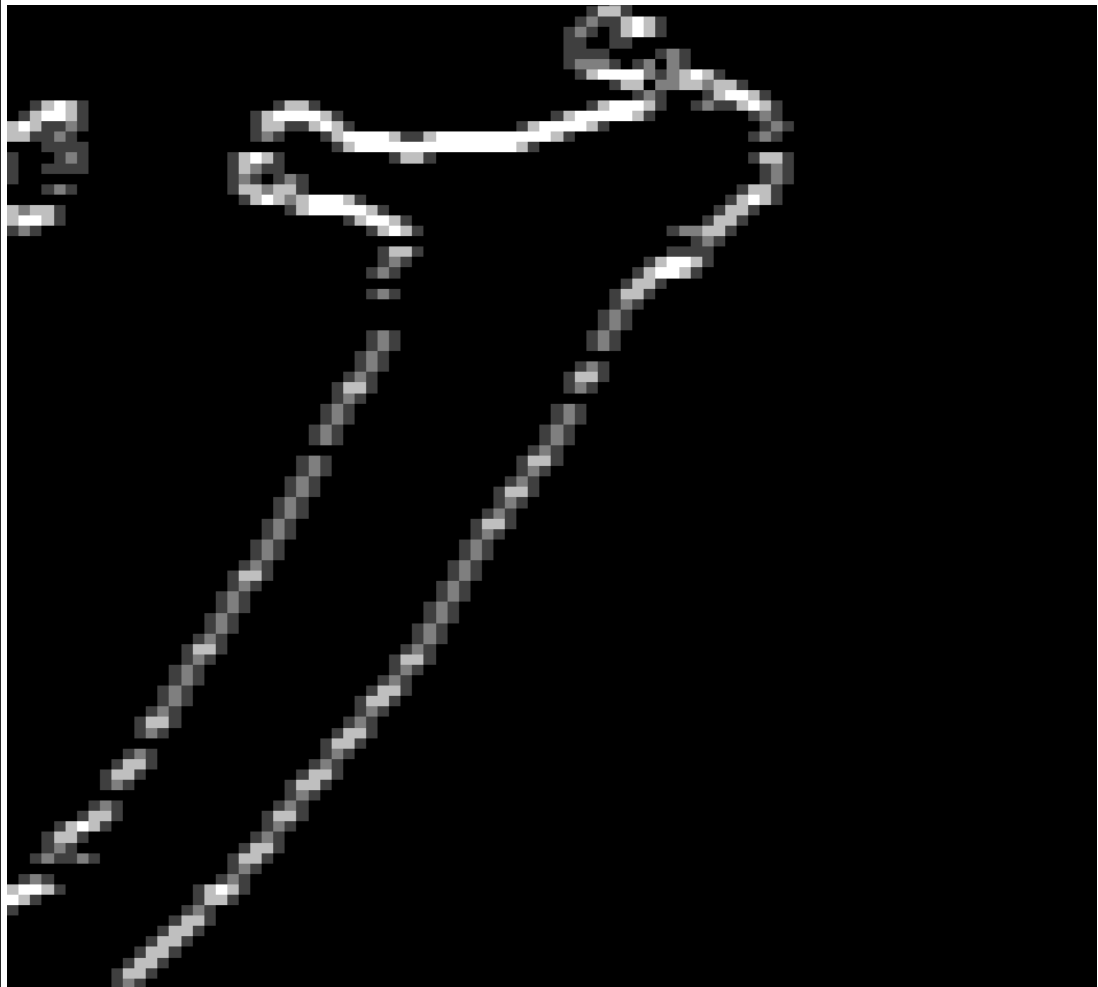
for $s in (siam)
let $sub := $s[ E(340230:341230), N(6039800:6040800), ansi("2017-06-19T00:00:00.000Z")],
$kernel1 := coverage kernel1
over $x x(-1:1), $y y (-1:1)
value list < 1; 0; -1; 2; 0; -2; 1; 0; -1>,
$kernel2 := coverage kernel2
over $x x(-1:1), $y y (-1:1)
value list < 1; 2; 1; 0; 0; 0; -1; -2; -1>
return encode( sqrt(
pow( coverage Gx
over $px1 E(imageCrsDomain($sub, E)),
$py1 N(imageCrsDomain($sub, N))


```

```

values
condense +
over $kx1 x(imageCrsDomain($kernel1, x)),
$ky1 y(imageCrsDomain($kernel1, y))
using $kernel1[x($kx1), y($ky1)] * ($sub[E($px1+$kx1), N($py1+$ky1)]=21 or
$sub[E($px1+$kx1), N($py1+$ky1)]=30), 2.0)
+
pow( coverage Gy
over $px2 E(imageCrsDomain($sub, E)),
$py2 N(imageCrsDomain($sub, N))
values
condense +
over $kx2 x(imageCrsDomain($kernel2, x)),
$ky2 y(imageCrsDomain($kernel2, y))
using $kernel1[x($kx2), y($ky2)] * ($sub[E($px2+$kx2), N($py2+$ky2)]=21 or
$sub[E($px2+$kx2), N($py2+$ky2)]=30) , 2.0)
),"image/tiff")

```



Q33a	Creating the mask coverage on which the Sobel filter should be applied by using a switch statement
Query	<pre> for \$s in (siam) let \$sub := \$s[E(340230:341230), N(6039800:6040800), ansi("2017-06-19T00:00:00.000Z")], \$mask := coverage mask over \$mx1 E(imageCrsDomain(\$sub, E)), \$my1 N(imageCrsDomain(\$sub, N)) values switch case (\$sub[E(\$mx1), N(\$my1)] = 21 or \$sub[E(\$mx1), N(\$my1)] = 24) return 1 default return 0 return encode(\$mask ,"image/tiff") </pre>
Note	<ul style="list-style-type: none"> • In the output image, the white areas have a value of 1, the black areas a value of 0 • The coverage constructor requires that axes are given in image coordinates. Currently it is not possible to retain or define a geographic CRS for a coverage produced in such a way.
Result	

Q33b	<p>Testing the Sobel filter on the mask created by a coverage constructor using a switch statement in the values clause. Note that to speed up the processing time, applying only the horizontal edge detector kernel 1 was tried.</p>
Query	<pre> for \$s in (siam) let \$sub := \$s[E(340230:341230), N(6039800:6040800), ansi("2017-06-19T00:00:00.000Z")], \$kernel1 := coverage kernel1 over \$x x(-1:1), \$y y (-1:1) value list < 1; 0; -1; 2; 0; -2; 1; 0; -1>, \$mask := coverage mask over \$mx1 E(imageCrsDomain(\$sub, E)), \$my1 N(imageCrsDomain(\$sub, N)) values switch case (\$sub[E(\$mx1), N(\$my1)] = 21 or \$sub[E(\$mx1), N(\$my1)] = 24) return 1 default return 0 return encode(sqrt(pow(coverage Gx over \$px1 E(imageCrsDomain(\$mask, E)), \$py1 N(imageCrsDomain(\$mask, N)) values condense + over \$kx1 x(imageCrsDomain(\$kernel1, x)), \$ky1 y(imageCrsDomain(\$kernel1, y)) using (\$kernel1)[x(\$kx1), y(\$ky1)] * (\$mask)[E(\$px1+\$kx1), N(\$py1+\$ky1)], 2.0)) ,"image/tiff") </pre>
Result	<p>The query run for more than an hour and returned the following exception:</p> <pre> <ows:Exception exceptionCode="RasdamanError"> <ows:ExceptionText>Failed closing rasdaman db connection: RasManager Error: Could not connect to RasServer .</ows:ExceptionText> </ows:Exception> <ows:Exception exceptionCode="RasdamanError"> <ows:ExceptionText>Failed internal rasql query: </pre>

	<code>%query%</ows:ExceptionText> </ows:Exception></code>
Q34a	Creating a mask coverage on which the Sobel filter should be applied by only using conditions in the values clause of the coverage constructor
Query	<pre> for \$s in (siam) let \$sub := \$s[E(340230:341230), N(6039800:6040800), ansi("2017-06-19T00:00:00.000Z")], \$mask := coverage mask over \$mx1 E(imageCrsDomain(\$sub, E)), \$my1 N(imageCrsDomain(\$sub, N)) values \$sub[E(\$mx1), N(\$my1)] = 21 or \$sub[E(\$mx1), N(\$my1)] = 24 return encode(\$mask ,"image/tiff") </pre>
Result	The result image looks the same as in Q33a, but while black values have also a value of 0, the white areas have a value of 255.
Q34b	Testing the Sobel filter on a mask created based on a coverage constructor with conditions in the value clause. Note that to speed up the processing time, applying only the horizontal edge detector kernel 1 was tried.
Query	<pre> for \$s in (siam) let \$sub := \$s[E(340230:341230), N(6039800:6040800), ansi("2017-06-19T00:00:00.000Z")], \$kernel1 := coverage kernel1 over \$x x(-1:1), \$y y (-1:1) value list < 1; 0; -1; 2; 0; -2; 1; 0; -1>, \$mask := coverage mask over \$mx1 E(imageCrsDomain(\$sub, E)), \$my1 N(imageCrsDomain(\$sub, N)) values \$sub[E(\$mx1), N(\$my1)] = 21 or \$sub[E(\$mx1), N(\$my1)] = 24 </pre>

	<pre> return encode(sqrt(pow(coverage Gx over \$px1 E(imageCrsDomain(\$mask, E)), \$py1 N(imageCrsDomain(\$mask, N)) values condense + over \$kx1 x(imageCrsDomain(\$kernel1, x)), \$ky1 y(imageCrsDomain(\$kernel1, y)) using \$kernel1[x(\$kx1), y(\$ky1)] * (\$mask)[E(\$px1+\$kx1), N(\$py1+\$ky1)], 2.0)) ,"image/tiff") </pre>
Result	Resulted in the same exception as Q33b.