University of Zurich^UZH

LocID - A Unique Object-at-a-Location Identifier; Designing a global hierarchical geographical identifier that accounts for spatial inaccuracy and computational performance

GEO 511 Master's Thesis

Author
Linus Rüegg
17-702-093

Supervised by
Dr. Thomas Phillips (Thomas_Phillips@swissre.com)
Dr. Cheng Fu
Prof. Dr. Robert Weibel

Faculty representative
Prof. Dr. Robert Weibel

30.09.2023
Department of Geography, University of Zurich

Author
Linus Rüegg
Department of Geography
University of Zurich
linus.rueegg@geo.uzh.ch / linus.rueegg@protonmail.ch

Supervisor
Thomas Phillips, PhD
Lead Digital Transformation
Swiss Re Management Ltd.;
Affiliated Research Scientist, Department of Aerospace Engineering and Sciences
University of Colorado at Boulder
Thomas_Phillips@swissre.com / thomas.phillips@colorado.edu

Supervisor
Dr. Cheng Fu
Department of Geography
University of Zurich
cheng.fu@geo.uzh.ch

Faculty Supervisor
Prof. Dr. Robert Weibel
Department of Geography
University of Zurich
robert.weibel@geo.uzh.ch

Abstract

Numerous organizations, such as insurance companies, work with global geospatial data. Global spatial operations, including risk assessment or disaster analysis, take time to compute. Spatial identifiers are a possible solution to speed up important spatial operations. Modern spatial identifiers struggle to encode geometries: Distortion of area, found in the common Mercator projection, poses challenges for precise area calculations and comparisons across latitudes. Complex transformation of areas into local Cartesian coordinate systems is thus needed. The added complexity increases errors. Latitude and longitude coordinates, while precise for location, lack the ability to convey object size or globally consistent location accuracy. Thus, a hierarchical system as outlined in this thesis may well be a solution for performant and accurate object identification. Discrete global grid systems (DGGSs), offer global continuous indexing, maintain consistent spatial resolution, and support data aggregation. This thesis compares different DGGS options based on factors like hierarchical structure, tesselation shape, accuracy, and programming language support. This work demonstrates that triangles, as a tesselation shape, provide consistent area sizes and computational efficiency, making them suitable for LocID's goal of addressing spatial challenges in risk assessment and disaster analysis on the Earth's surface. Consequently, the triangle-based DGGS Quarternary Triangular Mesh (QTM) is selected as the foundation for building LocID. The goal of LocID is to identify, match, recognize contains, and detect changes in objects using only its ID attribute. The process of designing, conceptualizing, and building a reference implementation of LocID follows a circular development loop, where requirements are defined, researched, designed, implemented, and evaluated iteratively. The encoding uses Quaternary Triangular Mesh (QTM), a DGGS based on an octahedron and on triangles as its cell shape. LocID identifiers consist of encoding digits and separators. LocID's geometry part is encoded recursively, with the number of digits indicating the number of levels used for encoding an object. This part is also compressed in a tree-like manner, by not double-encoding branches shared by leaves, leaves being the highest-level QTM cells used for encoding. A reference Python implementation, called LocID.py, is provided. Examples and tests are included to verify the correctness of encoding functions and operations. By encoding geometries within Discrete Global Grid System (DGGS) cells, LocID enables spatial operations without the need for additional spatial computation, providing quick string or byte operations such as matching and containment. LocID extends a DGGS, enabling geometry encoding and processing within cells. DGGSs offer a path to more performant geospatial data processing for points, whereas geometries have not yet been considered. LocID addresses this, offering a solution for encoding geometries in a way that allows a few operations to be performed performantly directly on the ID itself.

Keywords: Disaster Assessment, Discrete Global Grid System (DGGS), Spatial Identifier, Spatial Object Encoding, Polygon Encoding, Quarternary Triangular Mesh (QTM)

Acknowledgments First and foremost I want to thank Allegra Gasser for enduring my numerous elaborations about LocID over the last year and for her meticulous proofreading.
I want to thank Thomas P. Phillips (PhD) for helping me to set out on this journey and for the close mentorship throughout the whole project. Brainstorming, envisioning, and contemplating with him, even before work on this thesis began, provided the groundwork for this project. Further, I would like to extend my gratitude to Dr. Cheng Fu and Prof. Dr. Robert Weibel for guiding me in the scientific process, providing their expertise, and helping structure my work. The use of Geoffrey Dutton's Quarternary Triangular Mesh is also owed to Prof. Dr. Robert Weibel, who provided the idea and literature. I am grateful to all three of the above for their support in scientific writing, which was an immense challenge for me.
My gratitude also goes to Eric Püschel for challenging me to explain the concepts of LocID in a way that non-subject experts can have a chance to comprehend them.
I would like to thank Tim Waldburger, whom I work with at Swiss Re Management Ltd. Through collaboration with him, the idea to build a tool to rapidly match large building footprint datasets while staying in control of the matching accuracy came into existence. This also led to the examples and first visualizations in the introduction.
And thanks to everyone who patiently heard me expound on why triangles are amazing and why everything should be encoded using them.
Finally, I want to express my gratitude to all those who contribute to open technology, data, and science. Especially to the contributors of the projects listed in the 6 section, as well as the OpenStreetMap project for offering geospatial data for use by every person on this planet.
Due to my engagement with Swiss Re Management Ltd., where Thomas Phillips and I work in the same team, this project is a collaboration of the Department of Geography, University of Zurich, and Swiss Re Management Ltd.

Contents

Chapter 1

Introduction

1.1 Motivation

Historically, natural disasters have caused enormous damage to man-made structures. Due to the higher frequencies of events, growing event footprints, and a shift of events into areas where there were formerly fewer or no such perils, these damages are increasing (Banholzer et al., 2014). With this increase, it becomes more critical to analyze these damages but simultaneously more difficult due to the sheer number. Insurance plays a vital role in ensuring people's livelihoods when disasters strike. It is essential for the continuity of a business to receive insurance payouts as quickly as possible to endure a natural disaster. Therefore, large risk knowledge and insurance companies try to assess the outcomes of disasters as quickly and as automatedly as possible. The availability of remote sensing data and geospatial data processing capabilities is very handy for this. Nowadays, it is possible to assess the damage a tornado has done a few hours after the event. This is due to remote sensing data, which are used to predict the track of the tornado. This track can then be overlaid on a set of building footprints, as seen in Figure 1.1.

Until just a few years ago, insurance agents had to field-check hundreds of sites in an area where a disaster had struck. Thanks to recent advancements, agents can now target the places they have to visit in person very precisely. The warehouse in Figure 1.1 was photographed approximately one week after

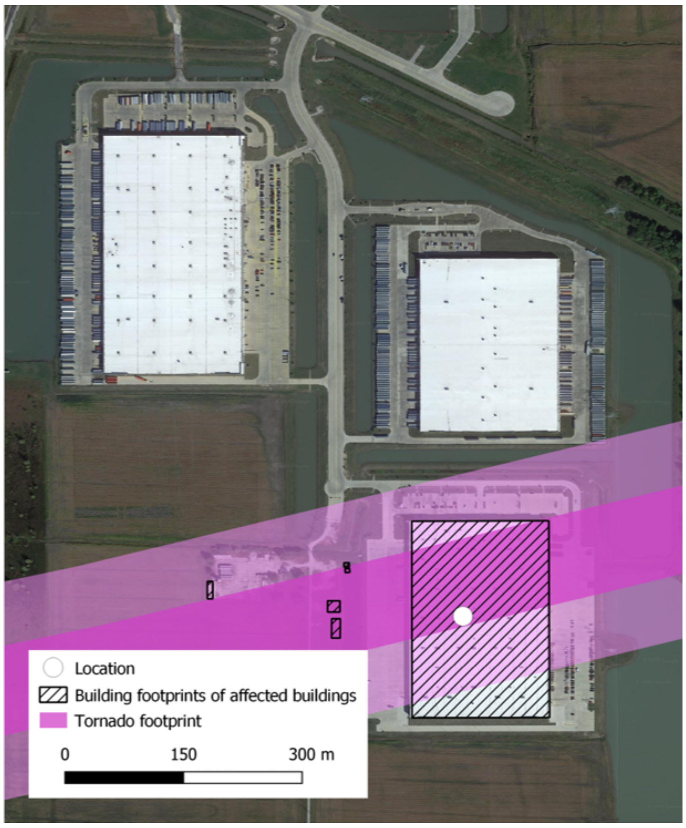


Figure 1.1: Post-event analysis of a tornado track, identifying possibly damaged buildings, done by the Geospatial Risk Insights team at Swiss Re.

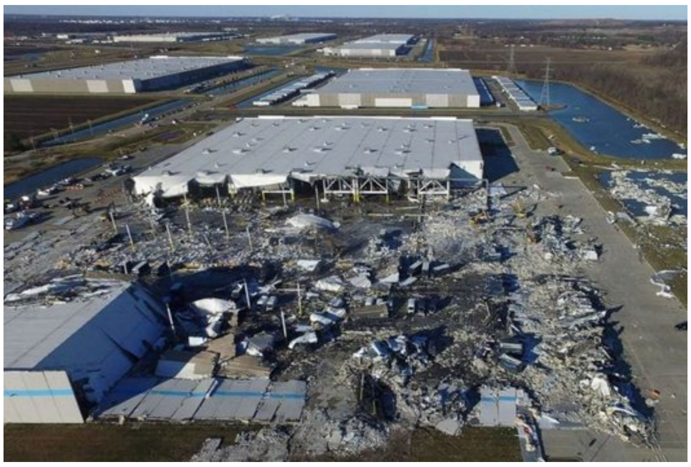


Figure 1.2: Imagery of the actual damage.

the tornado. The photograph is shown in Figure 1.2.

As can be seen, the prediction was very accurate. However, other perils' destruction severity prove more difficult to predict. Tornadoes have a very narrow and confined footprint. Inside the footprint there is mostly total destruction; outside it, there is none at all. Virtually all other perils, such as flooding, tropical cyclones (e.g., hurricanes), storms (hail, thunder, winter), wildfires, and earthquakes do not behave in a black-and-white fashion. All of them devastate much larger areas of land and vary in intensity of damage. The differences in intensity are so large that vector-based intersections, as done with the tornado track example, are more computationally costly and impracticable. Another important analytical tool for larger event footprints is spatial aggregation. When Hurricane Ian hit Florida in September 2022, the world wanted to know how bad the damage was. Insurances needed to assess the damage very accurately down to the building level, including which of the affected buildings belonged to their portfolios. Then, zooming out, they wanted to be presented with an overview that was aggregated and easy to understand, while still representing the actual numbers on the building level. These demands can be achieved more effectively with a hierarchical data model, as hierarchical models inherently aggregate to the next higher level. With a hierarchical grid structure, damages can be calculated on the building level (cell size below 20 m) and then simply aggregated to higher

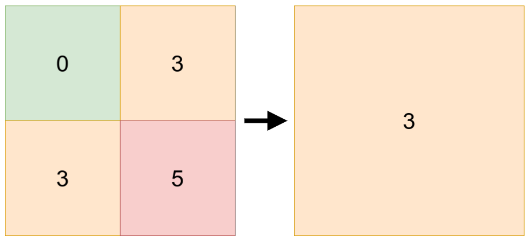


Figure 1.3: One step aggregation of four child cells to one parent cell.

levels (e.g., by taking the median for child cells and giving that value to the parent cell, visualized in Figure 1.3).

Aggregation can also be helpful when trying to match data sets, a task that is also very common when working with spatial risk data, for example, when matching a data set of buildings that have risk scores attached as attributes with the portfolio of an insurer who want to have these attributes attached to the objects in their portfolios. There is no absolute truth in any map. Every map is just a model of reality, and models differ in their degree of abstraction and selective representation of reality. Figure 1.4 shows the attempt to match two freely available building data sets. One originates from the OpenStreetMap project, which contains polygons added by hand by its contributors, and the other one from Microsoft, which contains buildings that were auto-traced from aerial imagery input data by an algorithm. If those building data sets had been available in a hierarchical data structure, the matching could have been accomplished much faster by matching on the cell level, instead of applying three different approaches (distances of the points of inaccessibility (POI), POIs laying in the other polygon and union of polygon areas). The current way of matching buildings also bears another pitfall: areas differ in latitudinal direction, and parameters set (such as the area of union required for matching) do not yield the same results across the world. More on this in Section 1.2. Ideally, in insurance, the matching of objects would be done on the fly and would inform the user about the accuracy of every match instantaneously.

A closely matching related use case is containment. For example, when a contract about building insurance is negotiated, it is significant to know if the

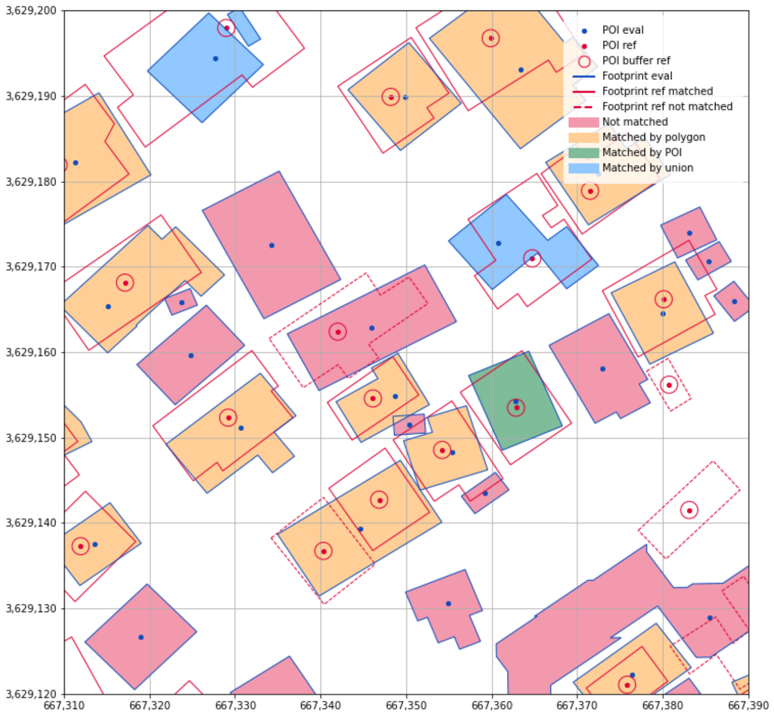


Figure 1.4: Comparison of OpenStreetMap and Microsoft building footprints. Analysis from the Geospatial Risk Insights team at Swiss Re.

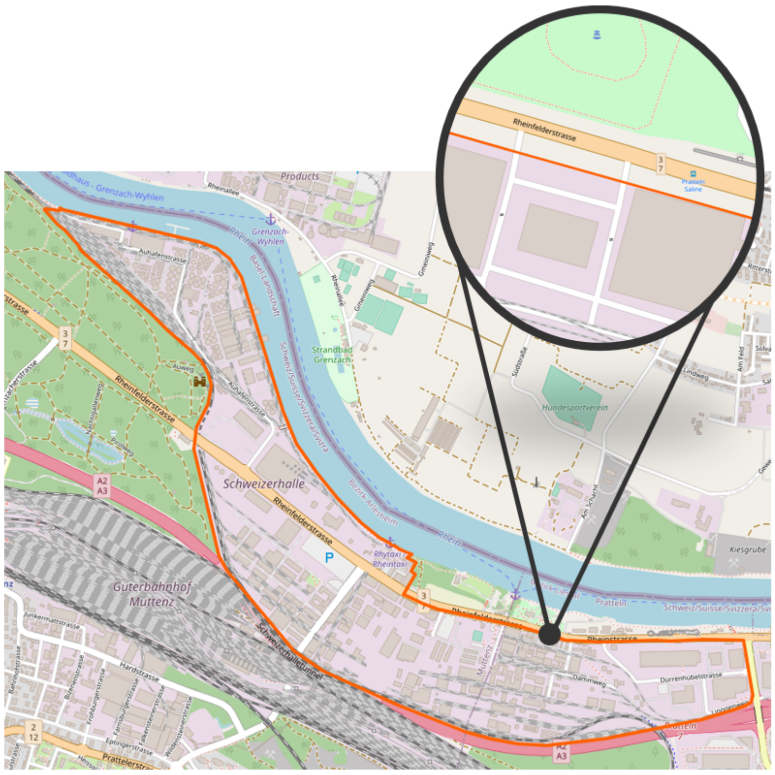


Figure 1.5: Industrial area of Schweizerhalle in OpenStreetMap.

building being insured is located in a zone of increased natural-hazard risk, a certain jurisdictional entity, or, as in Figure 1.5, a campus or industrial zone.

A last commonly occurring use case in the insurance world is the unambiguous identification of simple geometric objects, such as buildings, up to more complex structures, such as highway infrastructure (e.g., a bridge). Addresses can be inaccurate or ambiguous, and some objects, such as highway infrastructure do not have an address at all. The need for a location identifier, which takes the geometric object at the location into account, arises.

Over sixty years ago, the importance of mathematics in the description and work with geographic data was recognized. Applied by Peter Haggett in his work in the sixties (Barnes, 2019) and later recognized by e.g. Peter Gould (Gould, 1975). Advances in the field of information technology have since led not only to digitization of the field of geographic information science but to entirely new concepts and tools. Allowing one to carry out geographic data operations on a sphere or a sphere-like object approximating the Earth, would not be possible without modern concepts of computation. Nevertheless, computing is still time-consuming. The need for better-performing indexing systems and identifiers is apparent, as mentioned in the above paragraphs.

In order to address the need for faster, large-scale event intersection of spatial data sets and containment queries, it would be beneficial to preprocess and index existing data so that a response can be provided much more quickly when an event or a request occurs. The approach of this thesis involves the creation of a hierarchic location identifier.

Due to my engagement with Swiss Re, a (re)insurance company, this project is a collaboration of the Department of Geography, University of Zurich, and Swiss Re Management Ltd.

1.2 Background

The ancient Greeks realized that the Earth is a sphere. Nevertheless, people are accustomed to viewing maps of the Earth on flat surfaces, such as on computer monitors. Therefore, most map projections and coordinate systems display the globe's surface on a plane. This simplification has huge implications for the accuracy of the topography displayed. The omnipresent Mercator projection is well known to have flaws in almost all use cases outside of navigation (Israel, 2003).

Due to the cylindrical projection, the areas closer to the poles get stretched out, as can be seen in Figure 1.6. This circumstance does not allow accurate area calculations and especially no comparison of areas from different latitudes. To compare area sizes across the globe, every area has to be transformed into a local Cartesian coordinate system to calculate its accurate size. These transformations increase computational time and can lead to rounding errors. Furthermore, plain latitude-longitude coordinate pairs cannot transport any other information but a simple location, as Dutton aptly remarks:

> "While a latitude and longitude can pinpoint where something is on Earth, it cannot say how big that thing is or how accurate the estimate of its location may be." Dutton, 1999, p. 19

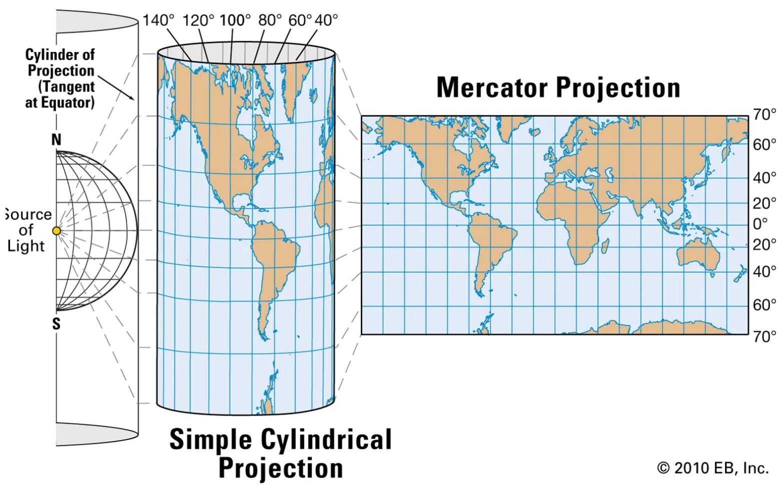
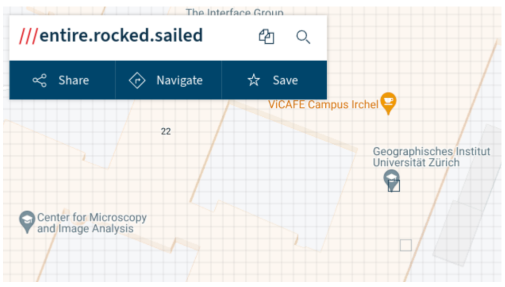


Figure 1.6: Mercator Projection as depicted in the Encyclopedia Britannica.

Hence, latitude and longitude can accurately describe a location, but there could be several objects located at the same location or in proximity. Hence, Dutton and others (addressed in Section 1.3) proposed using a hierarchical system instead of coordinate pairs to address the proximity issue.

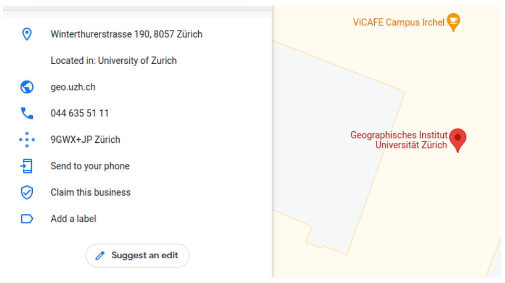
As data sets improve in accuracy due to improved sampling spatial resolution, the risk of mismatching even smaller objects due to differences in labeling or geocoding increases. This uncertainty due to mismatching is difficult to quantify. However, understanding and estimating mismatches is essential in insurance and risk mitigation. The quantification of object-matching precision would expedite the use of a high-resolution geographic information system (GIS) in the industry. As of today, a GIS that aims to answer questions such as "Do these polygons intersect?" as accurately as possible achieves object matching by spatially intersecting datasets. While this approach stands out in accuracy, it is computationally costly, as every data set has to be loaded into computational memory and geometrically checked for intersections. Performance-focused systems, on the other hand, achieve object matching by comparing identifiers only.

Thus, the **overarching objective of this thesis** is to develop a method that combines quantifiable (in)accuracy and performance, by implementing a geographic hierarchy, as Dutton described, while considering geometric



The Interface Group
///entire.rocked.sailed
Share
Navigate
Save
ViCAFE Campus Irchel
22
Geographisches Institut Universität Zürich
Center for Microscopy and Image Analysis

Figure 1.7: Identification of a location on the what3words website.



Winterthurerstrasse 190, 8057 Zürich

Located in: University of Zurich

geo.uzh.ch

044 635 51 11

9GWX+JP Zürich

Send to your phone

Claim this business

Add a label

Suggest an edit

Figure 1.8: Identification of a location on Google Maps, showing the location of OLC '9GWX+JP'.

features other than simple points.

Commonly used location identifiers Modern implementations of location identifiers, such as Google's *Open Location Codes (OLC)* (Rinckes & Bunge, n.d.) or *what3words* (Arthur, 2023) can efficiently pinpoint a location, but not an object. Figure 1.7 shows how a location is identified in *what3words*. The system does not offer the possibility to create an identifier for a building, only for the location of each 3 by 3 meter cell on Earth (Arthur, 2023).

Figure 1.9 shows a tabular comparison of some commonly used modern location identifiers and how they encode objects. OLC uses WGS84 coordinates and encodes them. The first two digits describe a square-like tile of 20° by 20°; the latter digits are hierarchical subunits. A second approach is Uber's H3 model, which is based on hexagons. Using evenly sized hexagons, this approach removes any distortion caused by the curvature of the globe (Uber, n.d.). H3 is especially suitable for distance calculations such as driving distances (shortest path) or convolution. This is due to the property of the chosen grid; every hexagon's centroid has the same distance to its neighbors. No other polygon shares this quality with the hexagon. Both systems cater to the specific needs of the two big tech companies that developed them. However, neither can identify objects and concentrate solely on location-based information.

The U.S. Department of Energy (DoE) developed the Unique Building Identifier (UBID), which is based on Google's OLC approach (Wang et al., 2019). UBID is designed to identify buildings. However, it does this by locating the center of mass of a building and encoding it as an OLC cell. Then add four digits, which represent the number of OLC cells it takes from the center of mass to the bounding box of the building footprint, in the four cardinal directions (Wang et al., 2019, p. 237). UBID is the only approach considered in this study that sets out to solve the problem of identifying objects on a map, as this study intends to focus on. The company QA Locate offers a product called StructureLocator, which achieves unambiguous identification by combining three separate identifiers: One for the geometric object (e.g., a building), one for the closest street and one for the subunit of that object (e.g., flat) ("StructureLocator™ - QA Locate", 2019). Although this commercial system seems very promising, no detailed white paper is available to understand how QA Locator designs the geographic identifier. A comprehensive discussion of related work on coordinate systems, spatial indexing systems, and location identifiers follows in Section 1.4.

Figure 1.9 visualizes the difference between the current UBID as well as H3 (which serves as a placeholder for all considered DGGS) and the LocID approach. This is explained in greater detail in the Methods chapter.

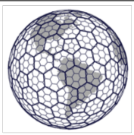
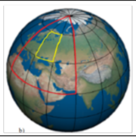
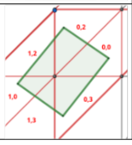
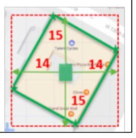
Location Indexing System	LocID		H3		UBID	
Coordinate System used	QTM		H3		OLC	
Geometry Description	Occupied QTM cells		Not implemented, cell only		Centroid & N,E,S,W extent	

Figure 1.9: Comparison of spatial indexing systems. Illustrations used: Purss, 2017; Rinckes and Bunge, n.d.; Uber, n.d.; Wang et al., 2019

1.3 Research context

The differentiation between (geographic) coordinate systems, spatial indexing systems, and location identifiers is not clear-cut. In this thesis, the terms should be understood as follows:

Coordinate systems Location identification methods vary, ranging from basic latitude and longitude pairs to more advanced systems such as UTM (Universal Transverse Mercator), each capable of using different reference systems, all serving the common purpose of pinpointing a specific location in space.

Location indexing systems These methods can either originate from or be converted to various coordinate systems, extending these systems, encoding their coordinates differently, often in a hierarchical or easily indexed manner, or offering an additional indexing system. Their capability to support geometries beyond a point on the Earth's surface may vary, but their fundamental purpose is to establish the spatial relationship between one or more objects in space.

Location identifiers Serve to identify a point or a more complex geometry unambiguously in space.

An index can be used as an identifier and an identifier can be index-based.

Figure 1.9 visualizes the use of location indexing systems and coordinate systems of a few relevant systems in this study.

Discrete Global Grid Systems (DGGSs) fall into the category of such location indexing systems. These and other relevant concepts for this study are introduced in the next paragraphs.

Discrete Global Grid System (DGGS) Some of the mentioned location identifiers can be classified as a so-called discrete global grid system (DGGS).

> "A DGGS is a spatial reference system that uses a hierarchical tessellation of cells to partition and address the globe. DGGSs are characterized by the properties of their cell structure, geo-encoding,

quantization strategy and associated mathematical functions" (Purss, 2017).

This quote comes from the 2017 approved specification on discrete global grid systems by the Open Geospatial Consortium (OGC). The document specifies certain aspects of a system to be considered a DGGS. Following the standard ensures interoperability between systems and allows for some flexibility during the development of such (Purss, 2017). DGGSs have been around since Dutton introduced his mathematical approach to geodesic modeling of planetary relief (Dutton, 1984). He turned these findings into a Discrete Global Grid System (DGGS) that is easy to calculate with. This system is an integral part of this thesis. The later following paragraph, "Computer friendly systems" mentions examples stemming from the world of databases, then again DGGSs are also being adopted in persistent storage (Bondaruk & Roberts, 2019), and in contrast to other "Computer friendly systems", are inherently (geo)spatial. DGGSs have some crucial advantages over traditional GIS, working with not-inherently geospatial database indexing systems.

- DGGSs are continuous (e.g., over longitude 180, at the poles, or across UTM zones).
- The spatial resolution of a DGGS is always explicit and constant over a hierarchic level.
- DGGSs facilitate the aggregation over their hierarchical levels.
- A DGGS facilitates operations across several datasets, as they are all divided in the same hierarchical way.
- No distortion of map projections is introduced and the user cannot accidentally assign the wrong projection (Goodchild, 2019).

Due to these advantages, DGGS-based geospatial information systems (GISs) are considered the foundation of the "next-generation Digital Earth" (Li & Stefanakis, 2020).

In parallel to the evolution of DGGSs, other systems to reference locations evolved, all with their special use cases. The following paragraphs give an overview of those systems and shed some light on what was (technically) needed to develop DGGS.

Human-friendly systems "Human-friendly coordinates" emerged over the last ten to twenty years. Their goal is to communicate geographic locations easily in human language. This is useful in the emergence of location-bound services, such as ride hauling. Furthermore, these systems strive to address places where street addresses (the original "human-friendly coordinates") are not common, such as in a few countries in sub-Saharan Africa (Kotze, 2020). Kotze names Google's OLC, which is known commercially and in Google Maps as *Plus Codes*, as well as *what3words*, as the most important spatial indexing system that fills this gap at the moment. What3words uses a proprietary database to assign every three-by-three-meter cell on the surface of the Earth to a unique combination of three words.

Radio-transmission-friendly systems A noteworthy mention is the Maidenhead Locator (also called the QTH locator), developed in 1970 based on a non-global predecessor from 1959. The Maidenhead Locator was designed as a robust and concise location identifier, even over slightly disturbed Morse or voice connections. Its design principles demanded that a location needs as few characters as possible and that every amateur radio could easily encode and decode the identifier with pen and paper. Maidenhead is nowadays standardized WGS84-based and encodes coordinate pairs into a string of mixed characters and integers. It is, similar to OLC, based on a large-scale rectangular global grid, which is hierarchically divided (Eckersley, 1995).

Computer-friendly systems Computer-friendly systems neither have to be easily understandable in human language nor do they have to be calculable with pen and paper. They are mostly used in databases to find locations faster, thanks to sorting and inherent hierarchy. Notable examples of such systems are R-Tree, B-Tree, Quadtree, Hilbert R-Tree, PM Quadtrees, and S2 (Kamel, 2008; Mao et al., 2023). While R-trees and Quadtrees came out of information technology, Hilbert R-trees and PM Quadtrees are adaptions for spatial data in databases (Kamel & Faloutsos, 1993; Samet, 2006, p. 365ff).

Inherent geospatial systems Besides explicitly for database management designed systems, there are other systems aimed at solving spatial problems with computers. Notable examples are the formerly noted H3 and S2. Like OLC, S2 is developed by Google. It caters to a different need in the geo-landscape of the tech giant. In contrast to OLC, which was developed to allow humans to easily communicate positions on Earth, S2 allows for geometric operations on the sphere, as well as providing a spatial indexing solution. As a base shape, S2 uses a cube which is projected onto the sphere and hierarchically divides each side into four children recursively. Therefore, each cell is a quadrilateral bounded by four geodesics (Veach, n.d.). A third noteworthy system, especially when talking about trying to spatially quantify risk, is OpenEAGGR, developed by risk and incident modeling solutions provider Riskaware (Bush, 2017).

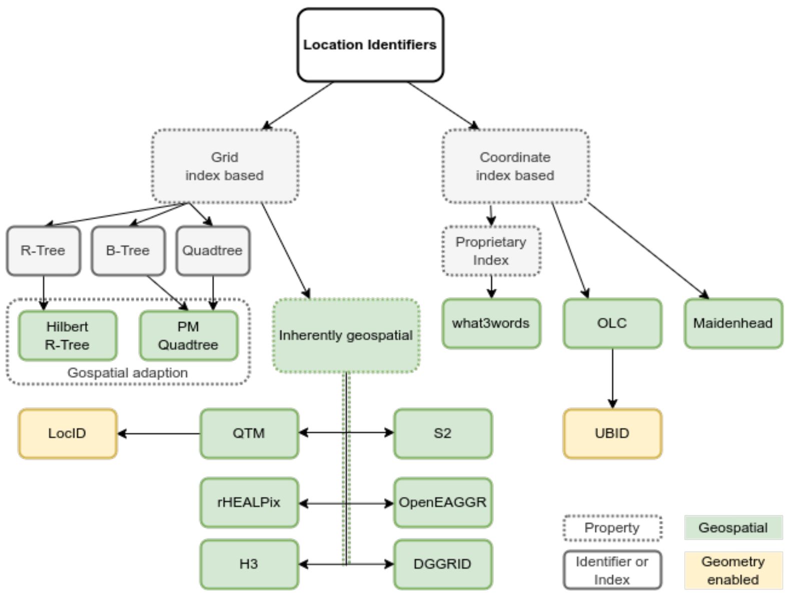


Figure 1.10: Overview of the above-mentioned spatial indices and identifiers.

The need for a performant location identifier As noted above, there are several location identifiers in existence, to meet all kinds of needs. From human-friendly, over radio-transmission-friendly to computational-efficient systems. However, these systems do not satisfy the needs expressed in the opening Section 1.1. The most promising approach is the use of a discrete global grid system, which can aggregate, match, answer contain queries, and unambiguously identify locations on the face of the Earth. Their hierarchical approach, similar to computer-friendly systems but better fit to work with spatial data on a sphere, makes DGGS the concept to be pursued further in this study.

1.4 State of research

State of DGGS

DGGS are applied in a variety of fields. Kmoch, Matsibora, et al., 2022 cite numerous studies from a variety of domains, including crime analysis (Jendryke & McClure, 2019), wildfire modeling (Robertson et al., 2020), coastal environment characterization (Bousquin, 2021), risk analysis for marine traffic (Rawson et al., 2022), or flood mapping (Chaudhuri et al., 2021). Other various large-scale GIS challenges are being re-evaluated in DGGS, including watershed delineation, land use/land cover change statistics, and general earth system modeling (Liao et al., 2020).

Hence, the need arises to compare the different available DGGSs in terms of their capabilities for different use cases. Bondaruk et al. analyzed H3, S2, OpenEAGGR, and DGGRID for usability in the wider research community (Bondaruk et al., 2020). Two years later, Kmoch et al. compared the same four systems with the addition of rHEALPix to their API capabilities as well as area and shape distortions (Kmoch, Matsibora, et al., 2022; Kmoch, Vasilyev, et al., 2022). This study adds the formerly mentioned QTM system by Geoffrey Dutton, which finished development in 1999. The comparison of all systems is done in Table 1.1.

It must be highlighted that some systems shown in Table 1.1 are not

System	**Body**	**Projection Shape**	**Tessellation Shape**	**Area Consistency**	**Indexing Method**	**Implementation Language**
rHEALPix	Ellipsoid	Cube	Varies on the same level	Consistent	Morton space-filling curves	Python
OpenEAGGR	Sphere	Icosahedron	Congruent Triangles or Non-Congruent Hexagons	Triangles yes, Hexagons no	No hierarchical identifiers	C++ with bindings in C, Java, Python
H3	Sphere	Icosahedron	Non-Congruent Hexagons	Not area preserving	Logical exact hierarchy, geometrical approximate	Efficient C implementation with bindings in multiple languages
S2	Sphere	Cube	Congruent Squares	Varies by max factor 1.5	Hilbert space-filling curve	C++ with bindings in Java, Go, Python
DGGRID	Sphere	Icosahedron	User-Choice (Triangle, Diamond, Hexagon)	Varies	User-defined parameters	C++ (bindings to R, Python)
QTM	Sphere	Octahedron	Congruent Triangles	Consistent	Unique Hierarchical IDs	Not available

Table 1.1: Comparison of DGGS

OGC-compliant DGGSs. H3 and OpenEAGGR, for example, do not meet all the OGC criteria (Bondaruk & Roberts, 2019). The authors of H3 argue that the use of some pentagons to form a sphere at the lowest zoom levels of their grid does not affect their use case, as they are elegantly placed in the ocean (where no Uber driver's route will ever lead) (Uber, n.d.). OpenEAGGR was released in the same year as the OGC specification, so they could have met the criteria only by luck. OpenEAGGR heavily overlaps cells in their hexagonal DGGS with aperture three. Since the OGC definition requires positional uniqueness without overlapping cells, this requirement is not met (Bondaruk & Roberts, 2019).

Importance of polygon shape for hierarchical grids As seen in Figure 1.9, spatial indexing systems, and, for that matter, DGGSs, can have different base polygons to divide space. Kmoch et al. remark that there is currently no equal-area congruent hierarchical hexagonal DGGS in a library module comparable to the functionalities of H3, S2, or rHEALPix (Kmoch, Matsibora, et al., 2022).

Dutton described the usage of a triangle-based Quaternary Triangular Mesh (QTM) to set up a congruent, regular, equal area global spatial data model and hierarchical spatial data model (Dutton, 1999, p. 23ff). He proved that an elegant computational solution to precisely identify objects on a map and possibly match them is solvable, due to the hierarchical nature of his approach (Dutton, 1999, p. 41ff). The tessellation of the globe in triangles, unlike other polygonal shapes, such as hexagons, allows one to always use the same shape (the triangle) (Goodchild & Shiren, 1992). This is important for use cases that depend on areas, for example, risk modeling. Furthermore, the triangle is believed to be the most performant option of the tessellation polygon to access cells (Peterson, 2017).

The hexagon also carries a particular advantage: all neighboring cells have exactly the same distance to each other. However, they do not coherently divide into smaller hexagons, like triangles do into smaller triangles. Also, the area size varies more than with triangles. Both can be seen in Figure 1.11.

Although computationally performant DGGSs exist, this study was unable

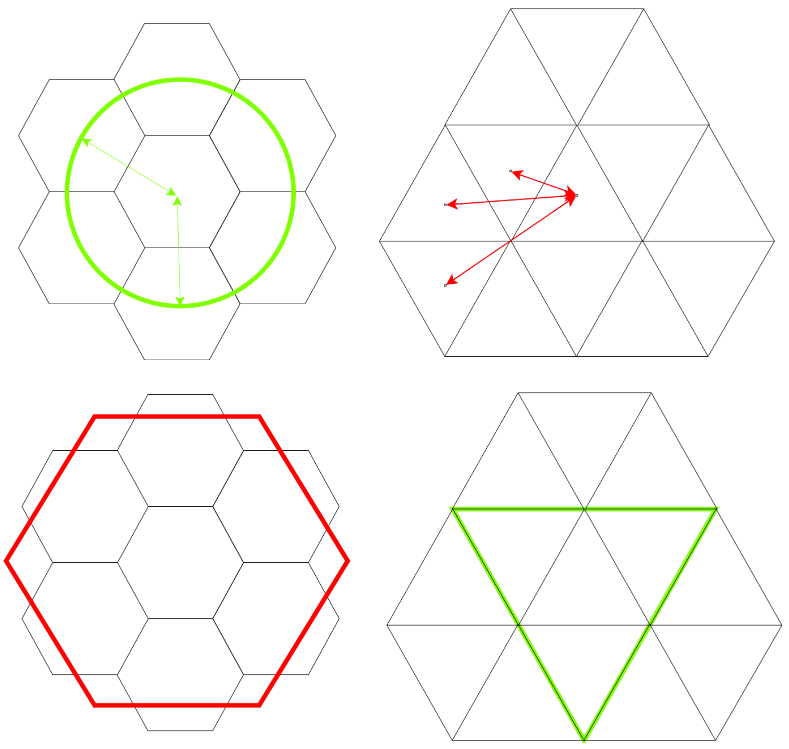


Figure 1.11: Comparison of hexagons and triangles as tessellation shapes.

to identify a system that takes advantage of the unique features of DGGSs to work with large sets of polygons and perform various operations with the index created by a DGGS for such polygons. In the next chapter, the required operations are outlined, and goals are set to reach these objectives.

Chapter 2

Research Goals

2.1 Problem Statement & Use Cases

The only current system identified by this study that can describe the spatial geometry of an object based on an inherent description method of the indexing system is the Unique Building Identifier (UBID). As described in the previous chapter, currently available Discrete Global Grid Systems (DGGSs) do not take advantage of their full potential when it comes to the *identification* (as in *creating an identifier of*) polygon or line features. Although UBID is successful in uniquely identifying a building, it has some distinct weaknesses regarding the mentioned (in Section1.1) use cases:

- Since the Open Location Code (OLC) is WGS84-based, the encoded geometric properties vary in area size across the globe. The base of the OLC grid is the largest level grid cell, set at (20° by 20°) which determines the largest cell size of the hierarchical coordinate system changes latitudinal, making comparisons of objects in different latitudes challenging.

- This leads to the second constraint: The usage of UBID is limited to objects with at least building size ($\tilde{2}$0 x 20m) since it is bound to the precision of the OLC cell, which varies. This becomes an issue with the need to identify smaller parts of a building or just smaller objects on the map in general.

The approach of using a DGGS to build a hierarchical coordinate system and building identifiers from such a system should eliminate those issues and possibly allow an increased computational performance, while also reducing the projection-introduced variation of accuracy.

To encode a building using a DGGS, users would first need to define the size of the cells in the grid. The smaller the cells, the more precise the encoding. Once the user has defined the cell size, a unique identifier can be assigned to each cell. The identifier can be a simple integer or a more complex string that includes the coordinates of the cell, which allows for more operations later on.

Once the buildings have been encoded, the unique identifiers can be used to match the buildings, evaluate their proximity, and perform other operations. For example, string matching can be used to find all the buildings with a certain identifier. The distance between the unique identifiers is determined using the hierarchical cell order represented in the identifier. This can then be applied to find all buildings within a certain radius of a given point.

A geometry-supporting DGGS-based spatial indexing system could solve and speed up certain applications of GIS. One strength is that such a system can unambiguously identify locations. Identifiers created with DGGSs would be hierarchical. The sorting of a data set based on such an identifier results in spatial sorting without having to calculate the distances of objects in a specific projection. This is a unique strength compared to present systems. In addition, the number of cells objects occupy allows comparison of object area size (such as building footprints) around the globe. In the insurance industry, this would have a considerable impact:

1. **Accumulation risk:** When onboarding a new portfolio of buildings to be insured, it would be easy to check if other objects in proximity are already insured, using a company-wide database outfitted with such identifiers. This would minimize the risk of accumulating too many objects at one location.

2. **Data enrichment:** This is useful when rapid matching between buildings and natural hazard layers or event footprints is required. For

example, after a large storm like Hurricane Ian, which brought floods and wind damage. In that case, the portfolios of building footprints have to be matched with information on building materials and intersected with flood footprints to assess the probability of wind and water damage.

Concerning hurricane Ian: Swiss Re had a huge loss caused by the storm and because it had accumulated risks [1]. The accumulation was larger than desired. With a better, company-wide understanding of where Swiss Re has (re)insured objects against which perils, this could have been preemptively mitigated. Also, it took a very long time to determine damages (as mentioned in 2), as spatial data sets from several sources had to be intersected for a huge area.

Besides event processing, there is also the ordinary use case of the yearly renewals of insurance contracts. Every year, insurance policies must be renewed. This is usually done automatically, but it is important to review the policy before renewing to ensure that the coverage is still suitable. All contracts must be concluded again, and this requires considering the ever-changing risks of millions of buildings. Identifiers based on DGGSs could be useful in facilitating this process. Building DGGS-based identifiers can reduce errors in the renewal process, as the code is unique and unambiguous, making it less likely to be entered incorrectly. Additionally, the underlying data sets are used to determine the value and risk of an object and can be updated and matched to the object again. Finally, a change in the building footprint, caused by expansion or demolition would be noticeable in the identifier and inform the insurer about changes in a portfolio.

LocID's Approach This thesis' goal is to explore a new approach to identifying spatial objects, as well as to create a design and a concept to implement such an identifier. Furthermore, a reference implementation in Python is developed and made openly available in an online repository. In the following chapters, the capabilities such an identifier should have, and

[1]bnnbloomberg.ca/swiss-re-posts-loss-on-hurricane-ian-likely-to-miss-targets-1.1838562 and ft.com/content/15cffe34-a19a-4be9-af67-01a444d907ff

the gaps in current concepts are investigated. After taking a deeper dive into use cases and research gaps in the *Research Goals* chapter, the chapters *Methods*, *Results*, and *Discussion* are always divided into the three pillars of this project:

- Design
- Concept
- Reference Implementation

There is no DGGS-based approach to efficiently describe geometries besides points yet. In all DGGSs previously considered, the description of multinode objects like a polygon is done in the "traditional GIS way", by indexing and saving the nodes of such an object or by filling a polygon with DGGS cells of one level, instead of making use of the possibilities a DGGS offers. The only system identified that uses the underlying index to describe polygons is UBID. But UBIDs are designed only for identification and nothing else. UBID's conceptual setup of working with OLC cells does not allow globally coherent operations, due to the underlying Mercator projection.

Encoding the geometry of an object into the identifier itself is needed to make certain operations possible and to make others more accurate. Matching could be made a lot more accurate when the geometry of an object is taken into account in addition to its location. Containment is only possible to assess when the area occupied by an object (its geometry) is known, at least of the enclosing feature. The following chapter outlines which problems there are to solve, considering the state of the research in location identifiers, and which use cases can be served with a DGGS-based object identifier.

2.2 Objectives

This thesis describes the process of designing a new identifier for spatially unique objects at any given location on the Earth's surface. The focus is on laying the requirements for such an identifier, its technical design, and the methodological problems of choosing a discrete global grid system to host the solution. In the final step, the concept is implemented in efficient computer code. In the previous paragraphs, the possible use cases in research and industry of LocID were described. Based on those use cases, the following four **objectives for LocID** are abstracted:

I. LocID **identifies spatial objects on the map without ambiguity.** The created identifier should be able to distinguish between two similarly shaped objects that lie in proximity to each other.

II. **Matching objects** across data sets is possible with LocID, without the need to perform spatial joins. Furthermore, area and latitudinal precision remain the same around the globe, thanks to the application of a discrete global grid system, in contrast to systems based on WGS84 like OLC.

III. When comparing LocIDs, **containing features are recognized** rapidly. This will be achieved by encoding geometric information into the identifier, which other systems, such as UBID lack to do.

IV. **Detecting changes** in spatial objects is possible by comparing LocIDs.

2.3 Goals

To reach these objectives, the following **project goals** were defined:

1. **Design:** Create an identifier design that can satisfy the above-stated objectives.

2. **Concept:** Construct a conceptual setup of a DGGS-based identifier, which explores the possibilities of geometry description with DGGSs.

3. **Reference Implementation:** Build a reference implementation that demonstrates certain capabilities of the concept.

4. **Process:** Conceive and document a process to design, conceptualize, and implement a spatial identifier.

In the following chapters, the methodology for design and concept is explained, the options for underlying systems (such as DGGSs) are analyzed, and the final design, concept, and reference implementation are discussed.

Chapter 3

Methods

3.1 Design Process

The process chosen to develop LocID according to the objectives defined in the previous chapter is circular. It is depicted in Figure 3.1. The idea of the 'development loop' named process is to carefully research, design, implement, and evaluate every single requirement and subrequirement of a project in an iterative manner. In doing so, all components function properly in themselves.

Abstracted from the four objectives defined in the previous chapter (to be able to identify, match, detect changes, and check for containment), the sub-objectives and requirements to reach them are described and prioritized in the following list:

Requirements

➊ **Initial reading**

➋ **Encoding & identification of a point**

➌ **Polygon matching**

(a) Encoding of polygons, based on ➋

(b) Add the geometric properties

(c) Find or create an efficient way to match two polygon LocIDs

(d) Be able to assess the accuracy of any match / only match above a certain accuracy threshold

❹ **Contain** Goal III. requires LocIDs to be easily queried for the containing of one encoded feature in the other.

(a) Polygon composites (e.g., buildings on a campus)

(b) Point in polygon

❺ **Line features**

(a) Encoding of line features

(b) Matching of line features

(c) Containing of line features (line in polygon)

❻ **Point matching**

❼ **Change detection** of objects with LocIDs is required by research goal IV.

(a) Changes of polygons

(b) Changes of lines

(c) Changes of points (only the position changes, not the geometry)

❽ **Writing & Documenting**

Each requirement is then researched, designed, conceptualized (and implemented), and evaluated after the other. Some requirements or features are implemented in the reference implementation, while others are only conceptualized.

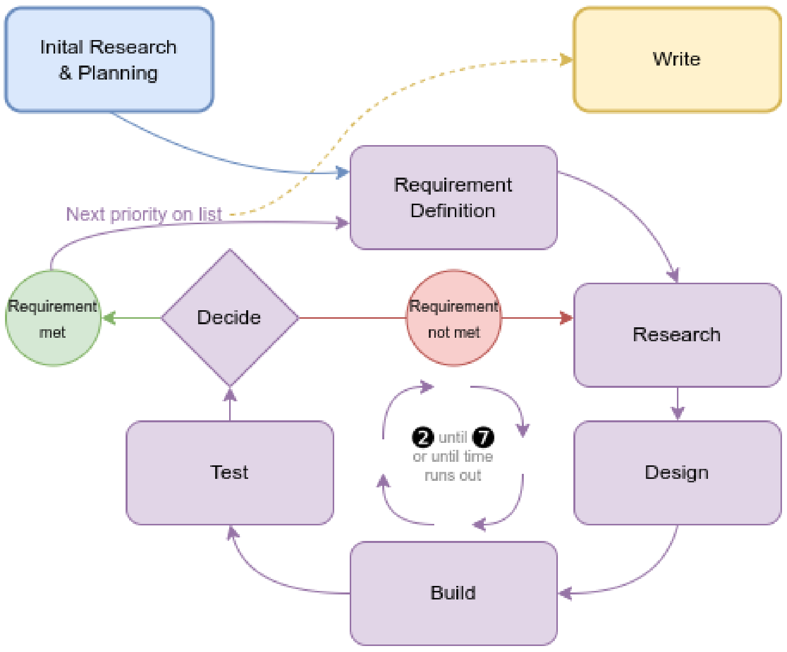


Figure 3.1: Setup of the Development loop

The requirements ❷ through ❼, will each pass through a development loop, consisting of the following steps:

Requirement Definition: This initial phase involves identifying the technical requirements associated with a given use case and evaluating whether they conflict with requirements from previous use cases.

Research: In this phase, the focus is on determining the most appropriate hierarchical system and encoding, as well as identifying, and distinguishing geometric features. It also involves checking for alignment with previously selected systems and feature sets.

Design: During this stage, the chosen encoding method is defined in terms of visual representation, and a methodology for potential matching is established.

Build: The Build phase involves the practical implementation of the encoding and/or matching process using simple Python code.

Test: In the Testing phase, the system's performance is assessed using real data to ensure accuracy and effectiveness.

Decide: Finally, decisions are made regarding whether to retain the current use case within the scope of LocID and whether to progress to the next use case or initiate the writing process.

3.2 Conceptual and Mathematical Model

A variety of DGGSs were presented in the Introduction, some of which can encode shapes other than points, such as lines and polygons. However, none of these systems take advantage of the unique features that a DGGS can offer. DGGSs that support some form of geometry encoding do so by referencing the nodes of a polygon or line, similar to the way vector-based formats do (Kmoch, Matsibora, et al., 2022; Uber, n.d.; Veach, n.d., cf.). This way of encoding does not allow us to perform operations with the help of DGGS grid cells. The only DGGS offering some form of geometry description is H3. It offers the ability to fill a polygon with cells of one fixed resolution (level). The documentation describes the process as follows:

> "Containment is determined by the centroids of the cells. A partitioning using the GeoJSON-like data structure, where polygons cover an area without overlap, will result in a partitioning in the H3 grid, where cells cover the same area without overlap." (Uber, n.d.)

Because H3 does not hierarchically encode its cells, encoding objects using multiple resolution levels is not possible. Hierarchical encoding is a very useful feature for polygon encoding. On the one hand, it can reduce computing complexity because only the necessary cell level is generated, while on the other hand, hierarchical systems can be encoded as (quad-) trees and thus be stored efficiently (Tamminen, 1985)

For this project, the choice of the DGGS is QTM. The reasoning behind this is laid out in the *Design of LocID*, Section (4.1.1) in the *Discussion* chapter. In short, QTM offers coherent parent-child relationships of cells, a flexible representation of the level of detail, flexibility, and a unique projection for conversion. The so-called planar Zenithial OrthoTriangular (ZOT) projection simplifies the reprojection of geometric shapes from other planar representations into the QTM system.

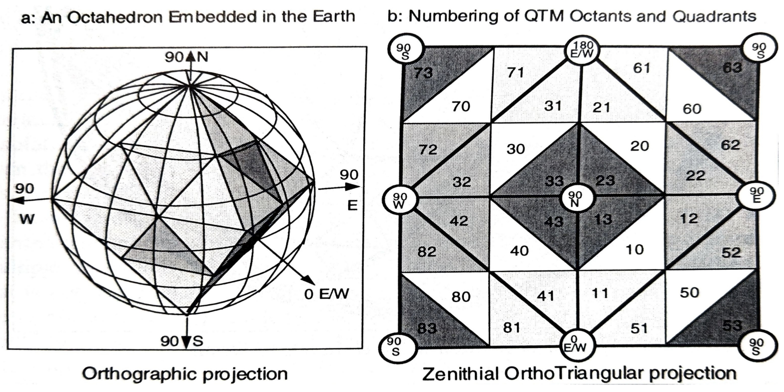

a: An Octahedron Embedded in the Earth
b: Numbering of QTM Octants and Quadrants
90 N
90 W
90 E
0 E/W
90 S
Orthographic projection
Zenithial OrthoTriangular projection

Figure 3.2: Form and orientation of the quaternary triangular mesh (Dutton, 1999, p. 24).

3.2.1 Properties of QTM

Quaternary Triangular Mesh (QTM) has the following properties:

- It is compact. QTM cells are compact and can represent a large area with relatively few nodes. This is because QTM divides space into quadruple triangles and not any higher node polygons.
- It can have (when permitted by the user) a high accuracy: QTM can represent features down to a level of detail that is suitable for the applications of LocID. The QTM triangular cells, called facets, have side lengths of, for example, 9.5 m at level 20 and 60 cm at level 24 on the sphere. QTM supports a maximum of 29 levels, at which level the facet side length is 1 cm.
- Robustness towards spatial inaccuracy: QTM was originally designed for map generalization. Therefore, it is robust to noise and errors and should be suitable for applications where data may be unreliable.
- Connectivity: It is possible to calculate the neighboring facets of any QTM facet. This facilitates operations such as nearby search and path exploration.

QTM uses an intermediary projection to convert from WGS84 coordinates to QTM facets on the sphere. The Zenithial OrthoTriangular (ZOT) projection,

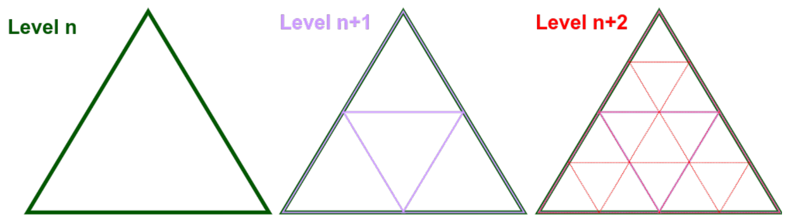


Figure 3.3: Three levels of QTM facets.

is planar. Both projections can be seen in Figure 3.2, which comes from Dutton's book, in which he introduced the latest [1] form of QTM in 1999.

Identifying geometric objects

Disambiguation

- Encode: To convert into a particular form.
  (e.g.: WGS84 $\rightarrow$ QTM)
- Identify: To create an identifier for.
  (e.g.: Building footprint $\rightarrow$ LocID)

Identifying a point in QTM is straightforward and is demonstrated by Dutton. The LocID of a point is the QTM identifier of the facet on which the point lies. This study focuses on the extension of QTM to also encode polygons and lines. QTM offers the ZOT projection in planar space as an intermediary projection. LocID leverages ZOT to encode polygons and lines, as described in the following paragraphs.

Points For the encoding of a single point, the QTM encoding algorithm determines in ZOT space on which of the eight sides of the octahedron the point lies, see Figure 3.2. This side is then recursively divided into 4 triangles of the same area as visualized in Figure 3.3. This recursion goes on until the point to be encoded is closer than a tolerance parameter (tol) passed to the encoding algorithm, to the next node. More on the topic later in Section 4.1.2.

[1]He went through several iterations of QTM. In this last step, the ZOT projection was added as an easy-to-calculate intermediary step while encoding from WGS84 in QTM.

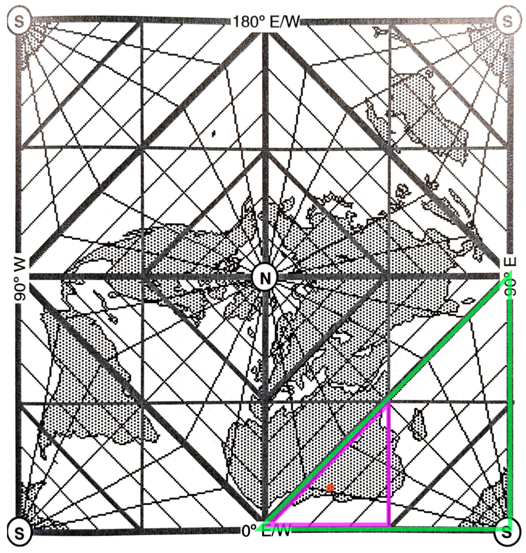


Figure 3.4: Octahedron side 5 is highlighted on the world map as well as its child facet 1 and a point in Angola. The world map shows ZOT as well as WGS84 coordinate grids. Underlying graphics from Dutton, 1999.

The QTM facets are numbered from zero (the middle facet) to three. The orientation of the numbering of the child facets one to three is based on the neighborhood of the parent triangle. Starting with the eight octahedron sides in the Zenithal OrthoTriangular (ZOT) space (again see Figure 3.2 b), the facets '1' face the intersection of the prime meridian, respectively, the 180° meridian with the equator, the '3' facets the poles, and the '2' facets the intersections of the 90° meridian with the equator. The eight octahedron sides are then divided, and the facet closest to one of these points gets the respective number. Then this goes on recursively. All facets have got their QTM ID by lining up the facet numbers of the facets they are in. A point in Angola (marked red in Figure 3.4), just southeast of the intersection of the equator and the prime meridian would lie on the octahedron side number 5 (marked green in Figure 3.4). Because the child facet (marked purple in Figure 3.4) of the octahedron side number five is closest to the said intersection and is number 1, the second digit of the point in Angola will then be 1. So the QTM ID would start with $[5, 1, ...]$.

Besides the first digit, which indicates the octahedral side (1-8), every other

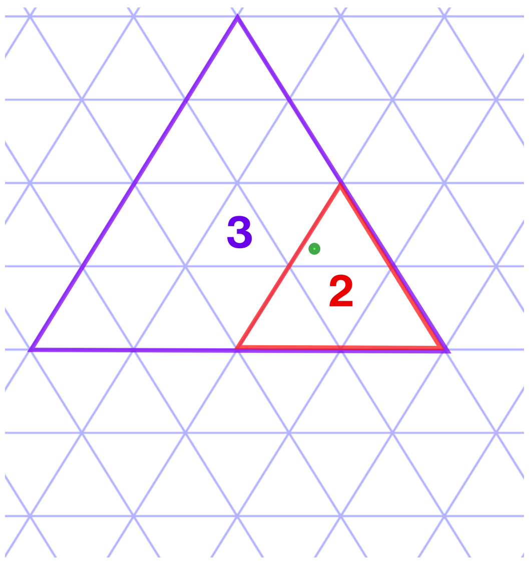


Figure 3.5: Cells needed to encode a point in QTM.

digit of the QTM ID is one of the numbers 0,1,2,3, determined recursively. In case of a point, the QTM ID is the LocID. Regarding Figure 3.5, a point positioned in the parent triangle (purple) number 3 and in the child facet (red) number 2, will get the LocID $[3, 2]$.

Polygons To extend this encoding mechanism to uniquely identify a certain polygon, and to describe a polygon by the facets it occupies, all the facets of one level occupied by a polygon must be identified. By choosing the resolution of the level, the accuracy at which a polygon can be identified is determined. The strategy has to be to calculate as few facets as possible to minimize computational cost. The process chosen in this study attempts to accomplish this as follows: First, only the nodes are encoded. In the example in Figure 3.6 this would mean the facets with the QTM IDs $[3, 2]$, $[3, 0]$, $[1, 0]$ and $[3, 3]$. To cover the entire polygon on the said level, the facets $[1, 2]and[1, 3]$ are missing. Now, the facets one level higher (in purple) with the QTM IDs $[1]$ and $[3]$ are intersected with the polygon in the ZOT space. In the example in3.6 this is enough to cover the whole polygon. Then, the missing facets on the original encoding level (facets in red) are calculated. The red-marked facets are the child facets of the purple-marked ones. These are the facets $[1, 1]$, $[1, 2]$, $[1, 3]$

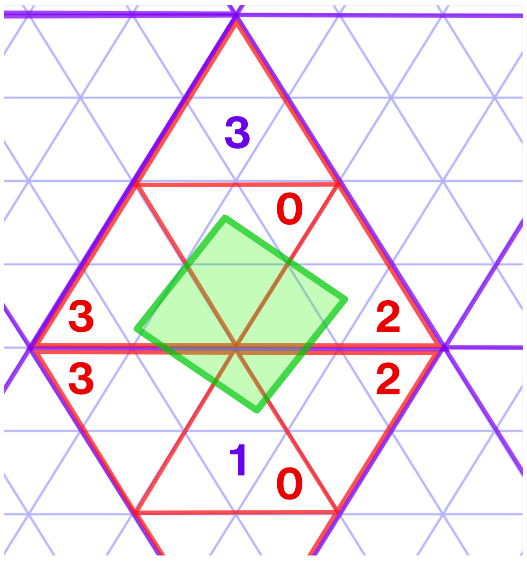


Figure 3.6: Cells needed to encode a polygon in QTM.

and $[3, 1]$. Now these facets are overlayed with the polygon and the ones identified ($[1, 2], [1, 3]$), which are needed in addition to the facets identifying the nodes, to cover the whole polygon.

This algorithm is a mixture of additive and subtractive procedures. Additive in the way that it calculates extra triangles to cover the polygon, and then subtractive, by removing the not needed triangles. In a purely subtractive approach, by, for example, subtracting all not needed facets from the smallest, the whole polygon enclosing QTM facet (later called the smallest enclosing triangle or SeT), is computationally heavy. The smallest enclosing triangle can be quite large, as can be seen in Figure 3.7, and therefore many facets, most probably over several levels, not only over one as in the example in Figure 3.6.

A purely additive approach would be possible, using a region-growing-like method, by just calculating the neighboring facets of the facets where the nodes lay in and grow further from there. But since the nodes have to be encoded anyway to know where to start, and in that process, all the parent facets of the final smallest level facet are calculated already, there is a "head start" for the chosen semi-subtractive process.

Figure 3.7 depicts the encoding of the nodes of a building (Y25 on Campus

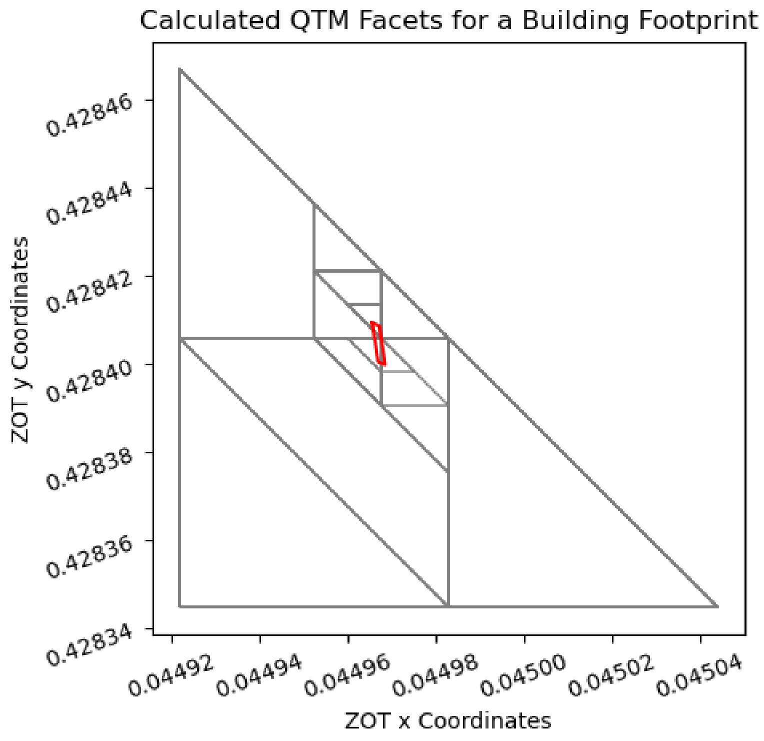


Figure 3.7: Facets calculated to encode nodes of the highlighted polygon. The largest shown triangle is the SeT of the polygon.

Irchel) in the ZOT space and the calculated facets down from the SeT (smallest enclosing triangle). 13 facets have to be encoded, from the SeT down, which are needed to encode the nodes of the polygon. 1 (the SeT) +4 (all children of the SeT) $+2 + 3 + 3$

In the case of the polygon in Figure 3.7, the four nodes fall in the three smallest level facets, while the SeT lies four levels higher. The QTM IDs of the nodes (omitting the second node laying in the same facet) are the following:

$$[1, 1, 3, 3, 0, 1, 3, 1, 3, 0, 3, 1, 2, 3, \mathit{3,\ 1,\ 0,\ 2}]$$

$$[1, 1, 3, 3, 0, 1, 3, 1, 3, 0, 3, 1, 2, 3, \mathit{0,\ 1,\ 0,\ 2}]$$

$$[1, 1, 3, 3, 0, 1, 3, 1, 3, 0, 3, 1, 2, 3, \mathit{0,\ 1,\ 3,\ 2}]$$

Whereby the integers in italics signify the from the SeT diverging nodes.

Lines Lines are encoded in much the same way as polygons are. Instead of intersecting a polygon with the QTM facets in the ZOT space, the line to encode is laid out in ZOT space, its nodes are encoded in QTM, and the missing facets to cover the edges of the line are generated.

Storage of LocIDs QTM identifiers are stored in a PR quadtree (point and region quadtree). Or, more precisely, in a forest of eight PR quadtrees, as every face of the base octahedron has its own quadtree (Dutton, 1999, p. 26). These quadtree identifiers can be compressed when encoding several features in proximity of each other which is done for geometries and lines with LocID.

Operations on LocIDs

Matching For the matching of two LocIDs, the intersection is calculated over the union of facets. By comparing the IDs of the involved facets, there is no need to find geometric intersections, but only matching IDs. The match value is returned as the Jaccard Index and answers the question "How much do A and B match?" with a numeric value between 0 and 1; not at all exactly the same.

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Change Detection The change detection measure is achieved by finding the total number of differing facets between two LocIDs and dividing this number of facets by the number of facets in the first LocID.

$$\Delta = \frac{|(|A1| - |A2|)| + |(|A2| - |A1|)|}{|A1|}$$

Containment Containment is evaluated on a true or false basis. If the intersection of B and A, B being the smaller object, is the same as B, it is true that B lies in A.

$$A \cap B == B$$

3.3 Implementation into Computer Code

The basis of the project serves pseudo-code written by Dutton, who implemented his QTM algorithms in C. The pseudo-code from his book was implemented during the course of this study into Python. Dutton's algorithms serve to encode points from WGS84 coordinates into QTM via ZOT. Additionally, the above-described methods to identify geometric objects as a LocID and methods to perform operations on these LocIDs have been implemented.

All code is available in an online repository [2]. The choice of the programming language, while not being the most performant option, was Python's accessibility. The code written in Python is easy to read, the availability of geospatial tools in Python is very good, and finally, the author speaks Python. Due to the good readability of Python code, this study refrains from using pseudo-code or any code, in this paper.

3.4 Methodical Challenges

Measurability of outcome As there is no other identifier system known to this study that has a matching, contain, or change detection function implemented, it is difficult to compare the performance of this system. The study will try to compare the matching function with QGIS functions. This is achieved by measuring the time of single operation steps: encoding into the used format and actual matching. An overall one-to-one comparison is not possible at the current stage of knowledge about other systems.

Data availability A dataset with pre-matched building footprints is needed to tune the parameters 'tol' and 'lvl' (which will be introduced in the next chapter). To simulate such a dataset, OSM building footprints from all over the world were pulled and for each building, a second, randomized footprint was generated.

[2]https://github.com/thereallinusrg/LocID.py

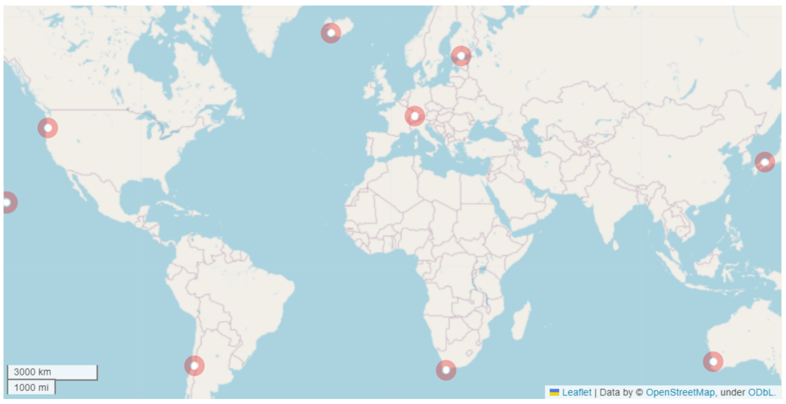


Figure 3.8: Distribution of test set sampling locations over the globe

Nine seed points across the globe were selected by hand, to avoid completely random queries in areas where there are no buildings (see Figure 3.8). Then, random coordinates, plus-minus 0.01 degrees around those points, were generated. For each of these randomized locations, in a radius of 150 m, all building footprints were queried from OSM.

This process was repeated multiple times until a collection of 2000 footprints was created. The footprints were then reprojected to UTM (to perform uniform area transformations during randomization throughout the world). The footprints were randomized in three ways. Each footprint had a:

- 3/5 chance to either be shifted by up to 30 m in any direction
- 1/5 chance of having the convex hull of the original polygon calculated
- 1/5 chance to have performed both

One of these random 150m radius locations is depicted in Figure 3.9.

The randomized footprints were then transformed back to WGS84. Out of these 2000 pairs of original OSM-derived footprints and their randomized peers, a random sample of 200 individual footprint pairs was selected to serve as the final test set.

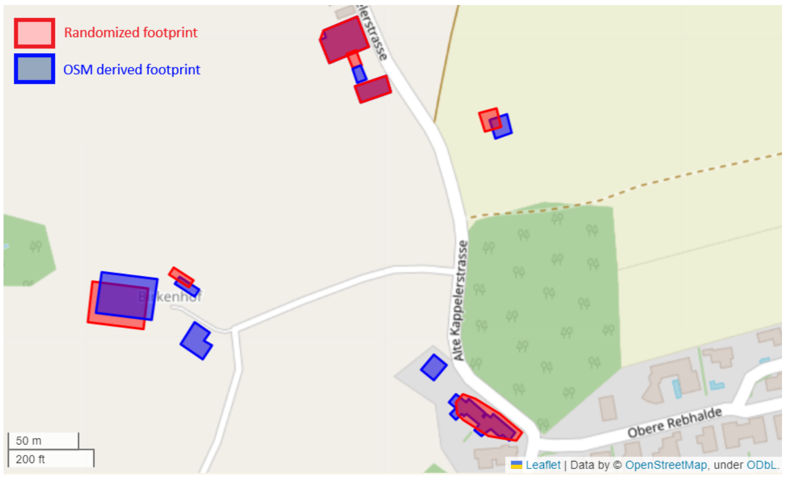


Figure 3.9: Close-up of a part of a test set sampling location

Chapter 4

Results

4.1 Design of LocID

When encoding the coordinates of the Y25 building on the Irchel campus as a WGS84-based polygon with the final LocID encoder, the resulting LocID appears as the following integer list:

$$[\mathbf{1}, 1, 3, 3, 0, 1, 3, 1, 3, 0, 3, 1, 2, 3, \mathbf{8}, 3, 3, 1, 0, 2, 4, 0, 1, 0, 2,$$

$$4, 3, 2, 4, 2, 0, 2, 2, 4, 1, 2, 2, 4, 0, 1, 0, 3, 1, 4, 3, 4, 0, 4, 3, 1, 1, 4, 2, 4, 0]$$

The first digit (1) signifies the octahedral side, and the subsequent integers up to the number 8, identify the SeT (smallest enclosing triangle). Number 8 is both a separator and an indicator. It separates the SeT from the compressed coordinates of the geometry description, and its value, eight, means that this LocID describes a polygon. This can be seen in the above real-world example, and a bit clearer in Figure 4.1, where the geometry indicator is colored red. In the case of a line, this would be the number 7. This number is later termed a geometry indicator.

In addition to the first digit, a LocID is made up of only numbers 0,1,2 and 3, the QTM cell identifiers. Numbers 7 and 8 are used as the just-described separators and geometry indicators. The number 9 is an error code, which comes from the original QTM encoding and is given to uninitialized facets.

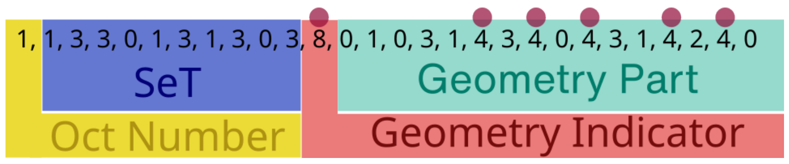


Figure 4.1: The parts of an example LocID marked with different colors. Separators are marked with a ball.

The number 4 is used as a separator in the compressed geometry part, whilst the numbers 5 and 6 are not in use in the current design. They could be used to indicate multi-geometries (multi-polygons and multi-lines), which are not considered in this study.

In the geometry part of LocID, the digits behind the geometry indicator (7 or 8) resemble a recursive encoding scheme. The number of digits between the geometry indicator and the first separator (= the number 4), indicates the number of levels between the SeT and the maximum encoding granularity. If the number of digits between the first and second separators (between 4 and 4) is smaller than the number between the geometry indicator (7 or 8) and the first separator (4), the digits from the beginning of the first set are used to bring the second set to the same length. In the above example, this looks as follows:

$$\ldots\mathbf{8}, 3, 3, 1, 0, 2, \mathbf{4}, 0, 1, 0, 2, \mathbf{4}, \ldots$$

The first set is simple, it consists of the digits between the geometry identifier (8) and the first separator (4):

$$3, 3, 1, 0, 2$$

The length of 5 digits means that there are five levels under the SeT level used to encode the polygon. The second set is only 4 digits long, which means that it shares the first QTM of the first set. Therefore, it refers to the following facet:

$$3, 0, 1, 0, 2$$

This methodology is applied recursively to all branches and leaves of the tree.

4.1.1 The choice of QTM as base DGGS for LocID

QTM offers a way of representing geospatial data that is efficient, accurate, and easy to use with existing tools. It is a Discrete Global Grid System (DGGS) that offers unique benefits. One such benefit is its ability to represent features at any level of detail. This means that QTM can be used to represent both coarse-grained features, such as countries and continents, and fine-grained features, such as buildings and individual trees.

Another advantage is its ability to mix datasets with different levels of detail. This is useful for tasks such as comparing data from different sources or combining data from different scales.

QTM is based on triangles. A triangle-based DGGS has a coherent parent-child relationship, as shown in Figure 1.11, and consistency of the cell area over one level.

Finally, QTM offers a unique conversion projection called the ZOT projection (cf. Figure 3.2). This projection makes it easy to reproject geometric shapes from other planar representations into QTM. This is particularly useful as many geospatial tools still operate only in planar space.

In general, QTM is a flexible and efficient way to represent geospatial data. It is particularly well-suited for applications that require accurate representation of data at different levels of detail and efficient encoding and decoding from and to the WGS84 coordinate system.

4.1.2 Selection of parameters

The QTM encoding function, as envisioned by Dutton (cf. 4.5, takes a tolerance parameter (**tol**) as input. This parameter defines the distance, in the ZOT projection, between the point to be encoded and a node of a QTM facet. The encoding stops when this distance falls below the tol parameter. The smaller the tol parameter, the more accurate the encoding, but also the more computationally expensive.

Dutton's definition of the tol parameter requires it to be between the ZOT distances of 2^{-29} [1] and 1. This maximum tolerance level was chosen by

[1]Approximately 1e-9

Dutton, to keep a QTM identifier of a point below 64 bits.

This LocID introduces a second parameter, lvl, which defines the smallest possible QTM encoding level. This is useful for creating large, uniform data sets of geometries at a fixed maximum level. The lvl parameter must be larger than the smallest enclosing triangle (SeT), as it only affects the geometry part of the LocID, which is defined as at least one level smaller than the SeT.

Since LocID is designed for use cases centered around buildings, the default parameters need to be fine-tuned for buildings. To achieve this, the authors carried out a test with building footprints from around the world to find the tol and lvl parameters best suited to encode buildings in LocID.

In other words, the tol parameter controls the accuracy of the encoding, while the lvl parameter controls the maximum level of detail. The default parameters of these two parameters need to be tuned for different types of data, such as buildings, to achieve optimal results.

To find the parameters best suited for building encoding and matching, a test set of 200 pairs of footprints was created, as explained in Section 3.4. These 200 pairs were then individually encoded with the functions found in `identify.py` [2], more on the encoding functionality later in Subsection 4.2.2, using a range of parameters. For the tol parameter, the range reaches from 0.001 logarithmically to 1e-07. For the lvl parameter from level 17 to 21. The pairs were then matched with the `match` function in `operations.py` [2].

The resulting values, which range from 0 to 1, are the Jaccard indices for the compared pairs of LocIDs, as shown in Subsection 3.2.1. A Jaccard index of 1 means that the two LocIDs share exactly the same QTM facets. A Jaccard index of 0 means that the two LocIDs do not share any QTM facets. In other words, the Jaccard index is a measure of how similar two LocIDs are. The higher the Jaccard index, the more similar the two LocIDs are.

To find the best-fitting parameters for buildings, the tol parameter has to be as small as needed to accurately identify and match buildings but as large as possible to save on encoding time, as encoding is a recursive process halted by tol. For lvl, the same is true: the maximum encoding level has to be as high as possible to accurately describe the geometry of a building, but as low as possible to save on computational cost. To find these tipping points, the deltas between the Jaccard indices from one lvl step respectively tol step have been calculated and plotted, as can be seen in Figures 4.2 and **??**. Each figure has five subplots, with the other parameter fixed in one step.

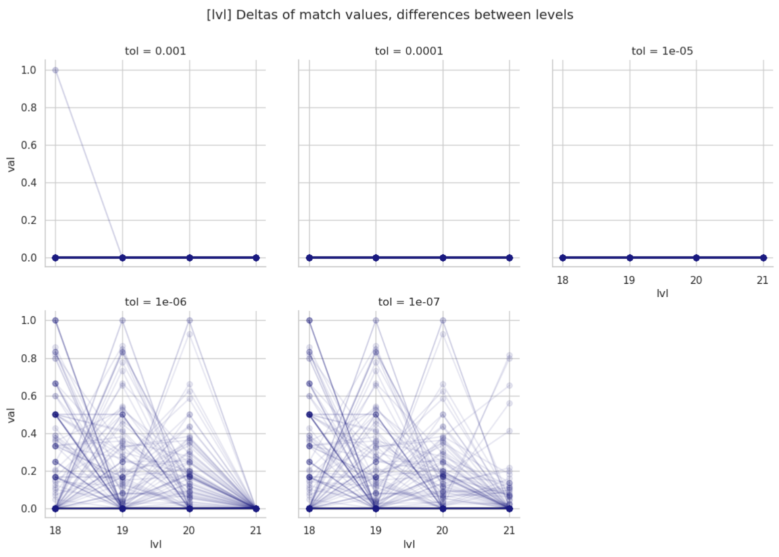


Figure 4.2: Deltas of match values, varying the lvl parameter.

The ideal lvl and tol value for buildings is therefore found, by finding the lvl or tol step from which there is no change to the next, computationally costlier, step. For Figure 4.2, a variation in **lvl** delta values can only be observed for tol (tolerances) smaller than 1e-05. This is most probably due to the fact, that for larger tol values, all nodes of a building get matched to the same QTM facet. E.g.: the tol distance is larger than the distance between the corners of a building. For tol=1e-06, the ideal lvl step is 20, as there is no change from lvl 20 to 21. In the last subplot, tol = 1e-07 a rising variance can be observed for the step lvl (level) 20 to 21. This is most probably due to the fact that tol = 1e-06 is too coarse to detect changes in locations below level 20.

For Figure 4.3, a variation in **tol** delta values can be observed across all five fixed-level subplots. From lvl = 18 on, all of the plots trend towards 0 on the tol = 10e-7. This means that there is little difference between 10e-6 and 10e-7.

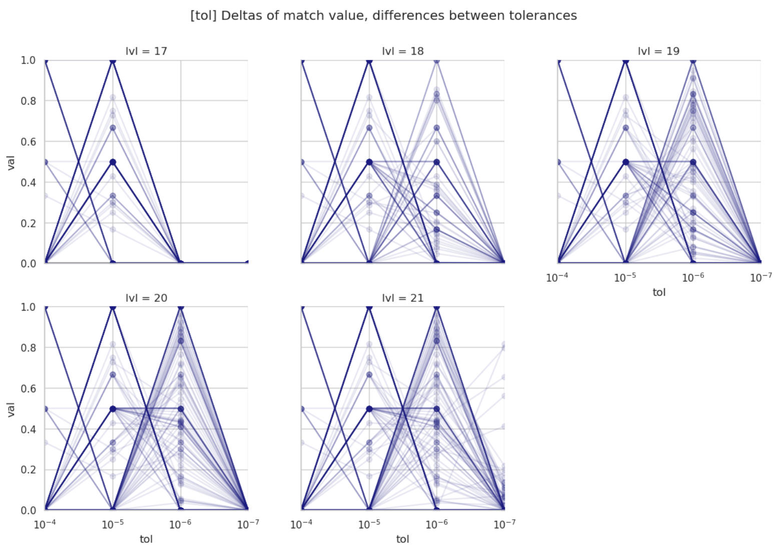


Figure 4.3: Deltas of match values, varying the tol parameter.

The smallest change in deltas can be seen in both figures when the lvl is increased from 20 to 21 and tol is decreased from 10e-6 to 10e-7. Consequently, the parameters lvl = 20 and tol = 10e-6 are the most suitable for identifying and matching the buildings of the test data set, since there is only a slight alteration when using parameters that require more computational effort.

4.2 Conceptual Setup of LocID Components

In this section, the conceptual setup of the LocID components is presented. The different functionalities are divided into three categories: **QTM**; all functions needed to convert the WGS84 coordinates to ZOT and QTM identifiers, the **identification** functionality; all the functions needed to create identifiers for different objects and **operations**; and different functions to perform LocID operations. The reference implementation in Python is also divided into scripts, `QTM.py`, `identify.py`, and `operations.py` [2].

[2]The code can be found in the reference implementation in the open repository found at https://github.com/thereallinusrg/LocID.py .

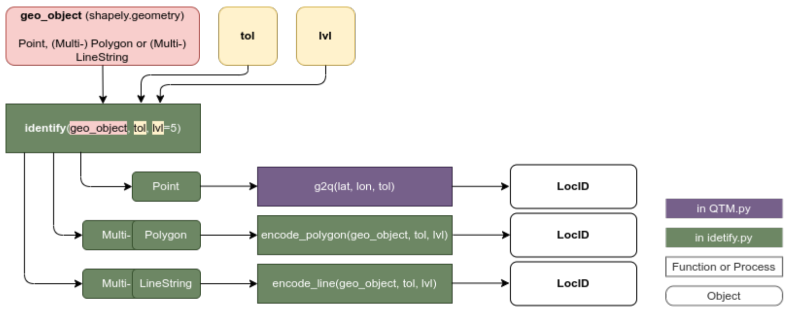

geo_object (shapely.geometry)
Point, (Multi-) Polygon or (Multi-) LineString
tol
lvl
identify(geo_object, tol, lvl=5)
Point
Multi- Polygon
Multi- LineString
g2q(lat, lon, tol)
encode_polygon(geo_object, tol, lvl)
encode_line(geo_object, tol, lvl)
LocID
LocID
LocID
in QTM.py
in idetify.py
Function or Process
Object

Figure 4.4: The `identify` function returns the LocID for shapely.geometry objects with WGS84 coordinates.

4.2.1 QTM

The functions conceptualized and described by Dutton in his book build the first of the three conceptual categories (Dutton, 1999, Appendix A). Dutton created the concept and instructions on how to implement his concept in computer code. The central function envisioned by him is the `QTMencode`. `QTMencode` needs to have a point in the ZOT space as input to work. So the second important function that falls under the QTM category is the `GeoToZot,` which converts geographic coordinates to ZOT coordinates.

LocID extends this functionality by adding a wrapper function, `g2q`, to convert directly from geographic coordinates to QTM ID.

4.2.2 Identification

The identification, the creation of an ID for an object, is different for different types of geometric objects, points, polygons, and lines. Any geometric object encoded, made up of nodes encoded in WGS84 coordinates (EPSG:4326), can be passed to the `identify` function. The reference Python implementation opts for `shapely.geometry` objects as input. Figure 4.4 shows how the identify function distinguishes between the geometry types.

Point

If the geometric object is a point, it gets directly passed to the `g2q` wrapper function. `g2q`, as can be seen in Figure 4.5, creates a geographic point object,

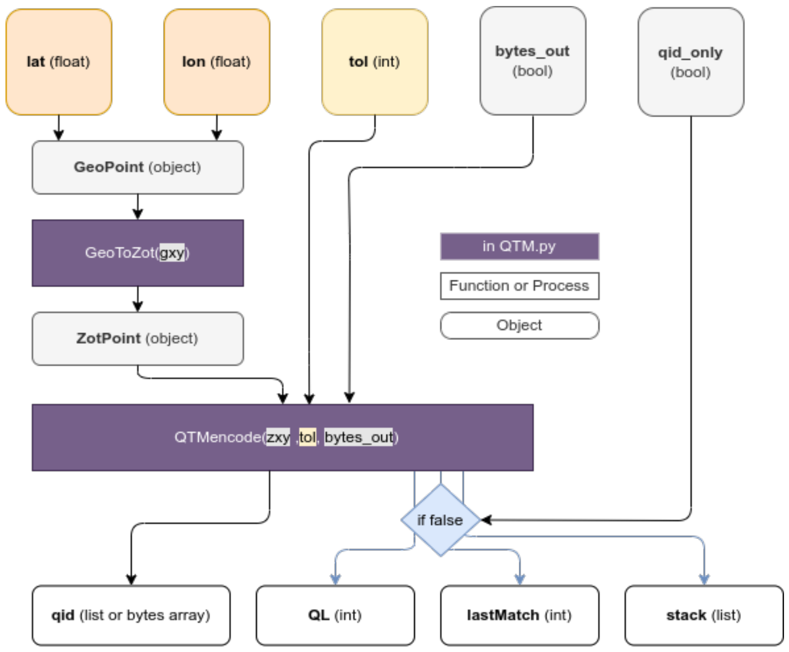


Figure 4.5: The function `g2q` encodes geographic (WGS84) coordinates into a QTM ID.

which gets converted to a ZOT point object by Dutton's `GeoToZot` function. The wrapper also passes the tol parameter, as well as the boolean value (bytes_out) which defines if the returned LocID is in an integer list or byte array format. Finally, the boolean *qid_only* determines if the additional outputs from `QTMencode` are passed on or not. The additional outputs *QL* (QTM level of the encoded point), *lastMatch* (a global variable that saves the QTM level on which the newly encoded point diverged from the one encoded just before), and *stack*. The stack contains ZOT coordinates and QTM identifiers for every facet calculated to identify the encoded point.

Polygon

In contrast to point identification, the `QTMencode` returned stack is essential for polygon encoding. The ZOT coordinates for every facet calculated while encoding the nodes of a polygon are used in polygon encoding. For the example from the beginning of the chapter (the Y25 building on campus Irchel), the QTM facets calculated in the ZOT space are shown in Figure 3.7.

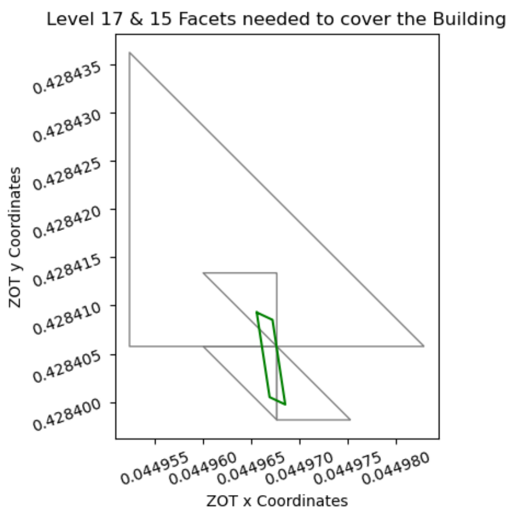


Figure 4.6: Facets needed to cover polygon

At first, only the facets generated while encoding the nodes (the corner points) of the polygon exist. To describe the geometry and perform operations on the ID, a finer grid of facets is needed. This finer grid is created by the function `map_poly`, which maps a shapely EPSG:4326 polygon to the QTM space by first encoding its nodes with `QTMencode` and then turns the resulting list of stacks into a `GeoDataFrame` (cf. Figure 4.8). The lowest-level facets that cover the entire example building in the `GeoDataFrame` are visualized in Figure 4.6.

The resulting facets of the highest level, 17 in this example, do not cover the whole building. To encode a polygon in LocID according to the polygon concept in Subsection 3.2.1, the whole polygon needs to be covered with the same level facets. One part of the example polygon is not covered by Level 17 facets, this part is marked red in Figure 4.7. To create the according level 17 facet(s) to cover the red-marked part, the function `create_missing_facets`, as can be seen in Figure 4.8, recursively splits the facets that are needed to cover the whole building (in the example in Figure 4.6, that would be the larger Level 15 facet), until all maximum level facets (Level 17 in the example) are calculated. The resulting facet grid is then intersected with the original polygon in the ZOT space, and the QIDs of all facets intersecting with the

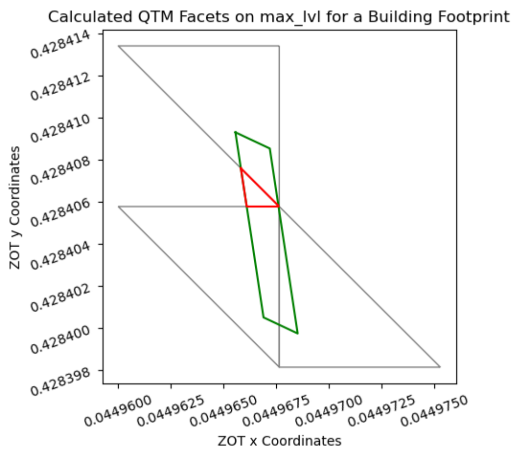


Figure 4.7: Missing Piece

polygon are returned as a list.

The list of QIDs as well as the level of the smallest enclosing triangle (SeT) is then passed to the next function in the polygon identification process (cf. 4.9). This next function takes a list of QIDs and returns a tree-structured list that represents the hierarchy of the items. Each item in the list is assumed to have a unique identifier and a parent identifier, which are used to build the tree structure. The trunk of the tree is the QID of the SeT.

Tree Compression The trunk (SeT QID) is then separated from the branches (facet QIDs), with a separator. The first leave (= one facet) of the tree has the length ($maxlevel - SeTlevel$), which for the example building, is five digits. If the second leave (= second facet) has common parent facets to the first leave, only the diverging facets are encoded. This is visualized in Figure 4.10, where the second leave has the QID $3, 0, 1, 0, 2$, which shares the first QID with the first leave. Therefore, only $0, 1, 0, 2$ is encoded for this leave. Thanks to the known length of the first leave, the now omitted digits can easily be derived. In Figure 4.10, the omitted digits are colored black. Also in the Figure 4.10, a scale-like legend of levels is presented under the resulting compressed LocID. It starts with the octant number ('N') and numbers the corresponding levels from one to nine, ten ('X'), and then 11 to 13, until the first separator ('/'), which also determines the geometry type (8 = polygon).

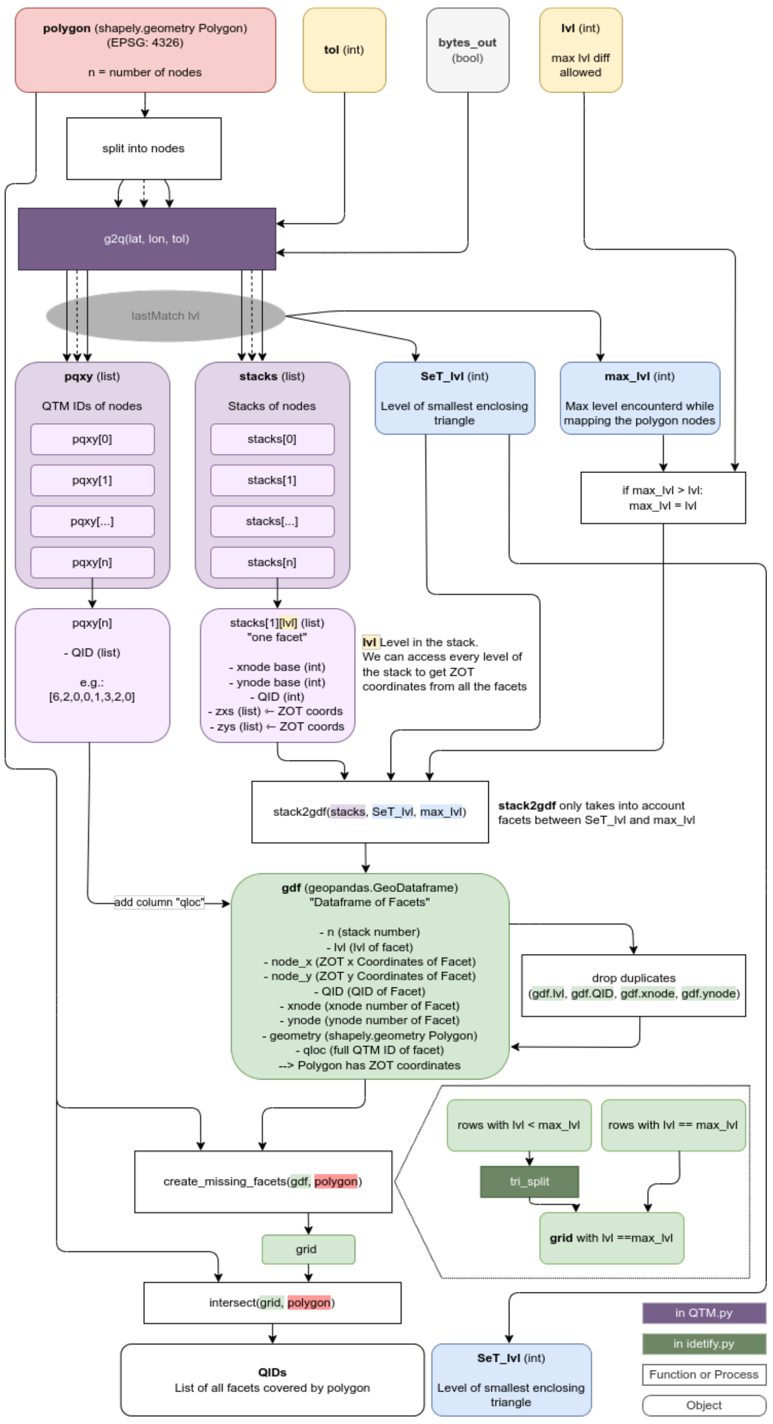

polygon (shapely.geometry Polygon) (EPSG: 4326)
n = number of nodes
tol (int)
bytes_out (bool)
lvl (int)
max lvl diff allowed
split into nodes
g2q(lat, lon, tol)
lastMatch lvl
pqxy (list)
QTM IDs of nodes
pqxy[0]
pqxy[1]
pqxy[...]
pqxy[n]
stacks (list)
Stacks of nodes
stacks[0]
stacks[1]
stacks[...]
stacks[n]
SeT_lvl (int)
Level of smallest enclosing triangle
max_lvl (int)
Max level encounterd while mapping the polygon nodes
if max_lvl > lvl:
max_lvl = lvl
pqxy[n]
- QID (list)
e.g.:
[6,2,0,0,1,3,2,0]
stacks[1][lvl] (list)
"one facet"
- xnode base (int)
- ynode base (int)
- QID (int)
- zxs (list) ← ZOT coords
- zys (list) ← ZOT coords
lvl Level in the stack.
We can access every level of the stack to get ZOT coordinates from all the facets
stack2gdf(stacks, SeT_lvl, max_lvl)
stack2gdf only takes into account facets between SeT_lvl and max_lvl
add column "qloc"
gdf (geopandas.GeoDataframe)
"Dataframe of Facets"
- n (stack number)
- lvl (lvl of facet)
- node_x (ZOT x Coordinates of Facet)
- node_y (ZOT y Coordinates of Facet)
- QID (QID of Facet)
- xnode (xnode number of Facet)
- ynode (ynode number of Facet)
- geometry (shapely.geometry Polygon)
- qloc (full QTM ID of facet)
--> Polygon has ZOT coordinates
drop duplicates
(gdf.lvl, gdf.QID, gdf.xnode, gdf.ynode)
rows with lvl < max_lvl
rows with lvl == max_lvl
tri_split
grid with lvl ==max_lvl
create_missing_facets(gdf, polygon)
grid
intersect(grid, polygon)
QIDs
List of all facets covered by polygon
SeT_lvl (int)
Level of smallest enclosing triangle
in QTM.py
in idetify.py
Function or Process
Object

Figure 4.8: Mapping a polygon in QTM space

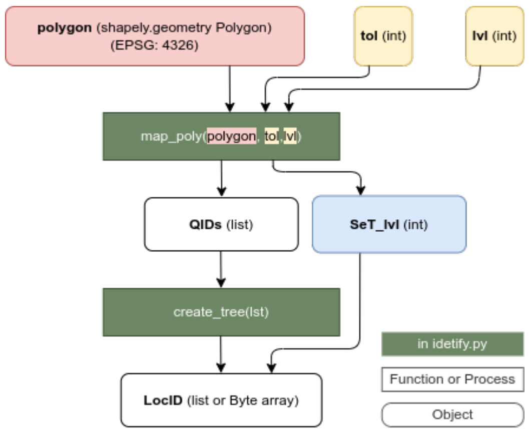


Figure 4.9: Overview of the polygon encoding process

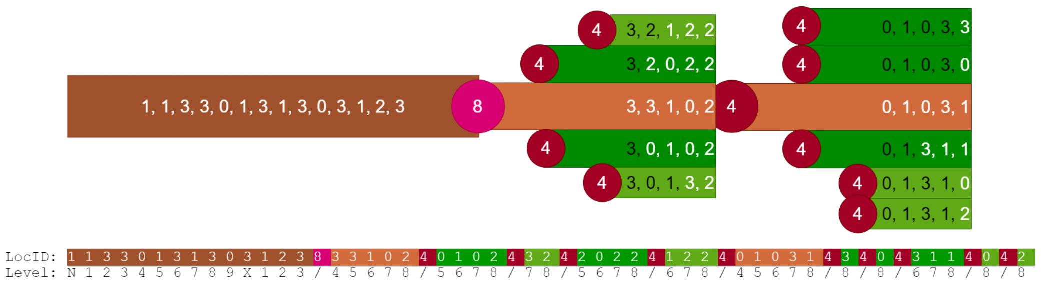

1, 1, 3, 3, 0, 1, 3, 1, 3, 0, 3, 1, 2, 3
8
3, 3, 1, 0, 2
4
3, 2, 1, 2, 2
4
3, 2, 0, 2, 2
4
3, 0, 1, 0, 2
4
3, 0, 1, 3, 2
4
0, 1, 0, 3, 1
4
0, 1, 0, 3, 3
4
0, 1, 0, 3, 0
4
0, 1, 3, 1, 1
4
0, 1, 3, 1, 0
4
0, 1, 3, 1, 2
LocID: 1 1 3 3 0 1 3 1 3 0 3 1 2 3 8 3 3 1 0 2 4 0 1 0 2 4 3 2 4 2 0 2 2 4 1 2 2 4 0 1 0 3 1 4 3 4 0 4 3 1 1 4 0 4 2
Level: N 1 2 3 4 5 6 7 8 9 X 1 2 3 / 4 5 6 7 8 / 5 6 7 8 / 7 8 / 5 6 7 8 / 6 7 8 / 4 5 6 7 8 / 8 / 8 / 6 7 8 / 8 / 8

Figure 4.10: Tree compression of facets into a LocID.

The later branches and leaves all reach the maximum encoding level of 18.

Lines

Lines are encoded in the same way as polygons. Their nodes are encoded in QTM through `g2q` and extra facets are generated to cover all edges connecting the nodes. The function to do so is named differently, although it is just a copy of the polygon function. This is for two reasons: the geometry indicator inserted is the number 7 instead of 8 and it functions as a way of future-proofing. If another project plans to build on LocID and, other than this study, lays its focus more on lines than on polygons and buildings, it should be easily possible to advance on the already established concepts.

4.2.3 Operations

Matching

In order to match two LocIDs, the LocIDs are first divided into SeT, geometry indicator, and geometry part. Then, the index of the first QID discerning is determined. This is, so to speak, the level of the smallest enclosing triangle enclosing both objects. From this level onward, for both objects, the compressed trees are "inflated"; the uncompressed QTM ID of every leaf is restored from this level to the maximum level of encoding. Then, the two objects are represented by two lists of QTM IDs. These two lists containing LocIDs either in integer list or byte array are compared with existing set theory capable, optimized functions (NumPy in the reference implementation). The returned intersection over union (Jaccard index) indicates how good the match of two objects is.

Contains

As with the matching operation, the LocIDs of both objects are both "inflated" and then compared with set-theory operations. For contains, this is done by checking if the intersection of the sets of the two lists (or arrays) is equal to the set of the smaller object. If this is true, the smaller object is contained by the larger.

Change Detection

The change between two versions of an object is quantified by a number expressing the total number of changed facets divided by the number of facets in the LocID of the first version of the object. These numbers are created by again inflating the LocIDs and comparing the sets of the inflated leaves of the two LocIDs.

4.3 Reference Implementation

Accompanying this study is a reference implementation of LocID in Python. It is named LocID.py and can be found on GitHub [3]. The Python version and other system parameters of the system the code was written on, can be found in the Appendix.

The reference implementation follows the logic of categorizing all functionalities into three groups by splitting the code into three Python scripts. **`QTM.py`** contains Dutton's code adapted for the LocID project [4]. The two main functions, `QTMencode` and `GeoToZot` are located here. Incorporating specific Python classes like `GeoPoint` and `ZotPoint`, these additions serve the purpose of holding geographic and ZOT points, respectively, while also ensuring the validity of the coordinates provided when creating these objects. Furthermore, the creation of QTM facets is done object-oriented by implementing a facet class (`Facet`), which creates empty QTM facet objects, which can then be altered by QTMencode. All core functionality is wrapped in the function `g2q`, which converts geographic coordinates directly into a QTM ID, taking tol and lvl as variables and returns the QTM ID either as a list of integers or as a byte array. In addition, tests are implemented in g2q, to

[3]https://github.com/thereallinusrg/LocID.py

[4]In the folder source_material, the file `QTM_original.py` can be found, which is an as close as possible interpretation of Dutton's pseudo-code in Python. Only adding some object classes to make the code Pyhton compatible.

check if the underlying encoding functions work correctly.

`identify.py` only contains one main function, `identify`. All other functions are helpers needed by `identify` to create LocIDs for the different geometry types. `operations.py` contains the actual operation functions (`match`, `contains`, and `change_detection`) as well as two functions to unpack compressed LocIDs. The first of those functions is `split_locid` which splits a LocID into the SeT part (smallest enclosing triangle), geometry indicator, and the tree-compressed geometry part.

Computational speed The `match` function in `operations.py` was tested against a combination of `GeoPandas.sjoin` and `GeoPandas.merge` on a subset of 10 of the 200 example building footprint pairs. The LocID-based procedure was able to match and give a metric about the match quality in a mean run time of 2.25 ms, while it took GeoPandas a mean of 22.2 ms to match the same pairs WGS84 encoded (Computation times reached on the system described in Appendix A). The test setup can be examined in the notebook found under test/speed_test.ipynb found in the online repository [5].

[5]https://github.com/thereallinusrg/LocID.py/blob/main/tests/speed_test.ipynb

Chapter 5

Discussion

The four objectives outlined for LocID in Chapter 2 Section 2.2, namely I. identification, II. matching, III. containment, and IV. change detection, were successfully achieved. It is important to note that these achievements pertain specifically to polygons. However, the design and concepts for identifying lines, as well as matching and comparing lines and points exist, but they are direct adaptations of the polygon-based design and concepts. As polygons were the focus of this thesis (building footprints), there is room for refinement of the handling of line and point features by LocID.

Additionally, the goals set forth in Chapter 2 Section 2.3 were also met. The four goals were: 1. design of an identifier, 2. conceptualization of such, 3. the creation of a reference implementation, and 4. the creation and documentation of the process of achieving the first three goals. The goals were met by designing and conceptualizing an identifier that fulfills the four objectives, as discussed in Chapter 4 Section 4.1 and 4.2. It is important to highlight that while these concepts apply to lines and points, they do not fully consider certain features of these geometries, such as the length of a line. Unlike polygons, where area calculation is addressed by LocID, length estimation for a line is not as straightforward.

Goal 3, which involves creating a reference implementation of the conceptualized setup, was achieved. The identification functionality is implemented for all considered geometry types, including points, lines, and polygons. However, the other operations (matching, containment, and change detection) are currently implemented only for polygons, as shown in Chapter 4 Section 4.3.

Lastly, the fourth goal, which is to formulate and document a process for designing, conceptualizing, and implementing a spatial identifier, has been successfully accomplished through the efforts mentioned above.

5.1 Design of LocID

The LocID design process, the development loop as shown in Figure 3.1, follows an iterative and incremental approach, continuously testing predefined requirements to ensure it meets the objectives formulated in Chapter 2. Although this approach has proven to be effective, it resulted in a greater focus on individual requirements at the expense of the broader context. Therefore, for further iterations of the design process, a broadened view is recommended.

LocID's design departs from the approach of encoding geometric features based on node positions. It encodes geometries within Discrete Global Grid System (DGGS) cells, enabling spatial operations without additional spatial computations. This choice has noteworthy benefits. DGGS cell IDs inherently contain geometry's spatial extent information, allowing some spatial queries without invoking spatial functions explicitly. Matching or intersecting geometries can be quickly determined by comparing their DGGS cell IDs. Identical cell IDs indicate precise overlap, while shared hierarchy levels suggest spatial relationships. Determining containment can be done by examining the hierarchy and relationships between encoded DGGS cell IDs, where prefixes indicate containment or overlap. DGGS cell encoding is efficient, scalable, and supports various resolutions, promoting interoperability across systems and datasets.

Encoding geometries within DGGS cells also has drawbacks. It can significantly increase storage requirements, especially for complex shapes. LocID addresses this with a tree-compressing approach. Encoding and decoding complex geometries into DGGS cell IDs can be challenging and may become more complex with geometry complexity. DGGSs are primarily designed for

2D data, making 3D data encoding more challenging. Maintaining encoding consistency during grid refinement or coarsening can be complex. LocID uses a consistent triangle-based DGGS, QTM, to overcome this. Despite challenges, DGGS cell encoding is suitable for tasks requiring efficient spatial filtering and comparisons, while other data representations may be better for extensive geometric operations.

In summary, LocID's design of encoding geometries into DGGS cells enables rapid spatial filtering and comparisons without complex calculations or additional indexing structures. However, it presents storage and encoding complexity issues. The choice to use LocID should depend on specific application needs and careful consideration of these trade-offs, making it especially useful for applications like natural disaster assessment that require fast spatial assessments.

5.2 Conceptual Setup

Choice of the underlying DGGS Comparing LocID, which is based on a triangle-based global grid, to other DGGS (Discrete Global Grid Systems) that use square, hexagonal, or diamond-shaped cells, reveals distinct advantages and trade-offs in different contexts. First and foremost, triangles maintain a more consistent size-to-area ratio during subdivision, ensuring better accuracy in representing geographic areas compared to squares, hexagons, or diamonds, which all can exhibit area distortions when subdivided.

The advantages of LocID over non-triangle-based DGGS include topological simplicity, as triangles have fewer edges and vertices than every other polygon, making topological operations and algorithms easier. Square-based systems, however, have the advantage of being in alignment. They are well suited for certain applications related to imagery and remote sensing, as square grids align well with the rectilinear nature of pixel-based imagery. The advantages of LocID over hexagon-based DGGS include an inherent hierarchical structure, making it easier to work with tasks that require hierarchical subdivisions, as well as more flexibility, as triangles allow for more adaptive grid resolutions. Hexagon-based systems, however, have the advantage of uniform neighboring distances, as each hexagonal cell is equidistant from its six neighboring cells, and compactness, as hexagons tend to tile the planet more efficiently than triangles. The advantages of LocID over diamond-based DGGS include a more straightforward hierarchical structure compared to diamond grids. Diamond-based systems, however, have the advantage that diagonal movements within a grid of diamond-shaped cells are of the same length as orthogonal movements because the diagonals of a diamond are bisected at right angles, resulting in a consistent distance for both diagonal and orthogonal neighbors.

LocID, based on QTM, has its benefits in its hierarchy, precision in depicting regions, adaptability, and topological straightforwardness, yet it may not be the most suitable option for applications working with distances, for example, convolutions or route calculations.

Concept of the encoding The geometry part of a LocID is encoded in a tree-compressed way while maintaining an uncompressed hierarchical prefix (the SeT) which is important for spatial sorting. Tree compression enables the efficient storage of complex geospatial information within LocIDs. The hierarchical prefix in LocIDs is essential for maintaining an organized representation of geospatial data. It allows users to navigate through the data from coarse-level representations to fine-grained details. The hierarchical prefix also serves as a spatial sorting mechanism, which means that LocIDs with similar prefixes represent spatially adjacent or related features. This simplifies the retrieval of nearby or neighboring spatial data. The hierarchical prefix facilitates the creation of efficient spatial indexing structures, enabling faster retrieval and analysis of spatial data. It also makes it easier to perform spatial analyses, as users can quickly identify and access relevant data subsets based on LocID prefixes. The hierarchical prefix supports scalable geospatial data representation, accommodating applications with diverse spatial requirements and precision levels. It also improves interoperability by providing a standardized way to organize and access geospatial data. Finally, LocIDs with hierarchical prefixes simplify the process of retrieving data for specific geographic regions or areas of interest.

Comparability As LocID is conceptually so different, it is hard to compare to other systems. A comparison to UBID could be made for the uniqueness of the resulting ID, but the significance of such a comparison is limited. A comparison of the performance of the operations would be more meaningful. Of the three operations of matching, change detection, and containment query implemented in the LocID reference, the `match` function was compared against a similar procedure performed with GeoPandas and the same test polygons encoded in WGS84 (cf. Section 4.3). The test came to the result, that the application of LocID sped up the matching process ten times. This result is to be enjoyed with caution, as the two methods in comparison, are not one-to-one drop-in replacements for each other, but rather procedures leading to a comparable result. One thing not considered at all is the time it took to create the LocIDs (the application of the `identify` function) from the WGS84 encoded polygons. While this first result is promising, a setup of a meaningful comparison, will require much more thought and time.

5.3 Reference Implementation

It was initially intended to encode and store LocIDs as byte arrays only and to use integer lists for human-readable representation only. However, the reference implementation was written in Python, which does not offer all the required operations on byte arrays. As a result, some operations in the reference implementation are carried out on integer lists, even when the bytes array output is chosen.

Encoding and saving LocIDs as byte arrays, as originally planned, would have several computational advantages over using integer lists. Byte arrays are more memory efficient than integer lists because they represent data in a more compact form, without extra encoding. In many programming languages, including Python, integers typically require more memory than bytes to store the same information. Byte arrays allow for direct byte-by-byte comparison, which can be significantly faster than comparing integers or other data types. When performing operations that involve comparing LocIDs, byte arrays enable efficient bitwise comparisons, making it easier to check for matches or differences.

Byte arrays are well-suited for bitwise operations, such as bitwise AND, OR, XOR, and shifting. These operations are fundamental in many spatial algorithms, including those that involve spatial queries, such as contains and geometric calculations. Byte arrays provide a more direct and efficient way of performing these operations compared to integer lists. Byte arrays support efficient binary search operations. In geospatial applications, binary search can be crucial for quickly locating and retrieving data within large datasets, as it minimizes the number of comparisons required to find a specific LocID or a range of LocIDs. Byte arrays can be easily serialized and deserialized, making them suitable for efficient data storage and transfer. Serialized byte arrays occupy less space, making them faster to write to and read from disk or transmit over a network. Byte arrays are more compatible with the low-level operations provided by modern CPUs. Many CPUs offer optimized instructions for working with bytes or smaller data types, allowing for efficient parallel processing and optimization of certain algorithms. Byte arrays can take advantage of CPU cache more effectively than larger data types, like integers or integer lists. Cache efficiency can significantly improve the speed of memory access and, consequently, the overall performance of algorithms. When working with byte arrays, there is less need for type conversion between bytes and integers.

Due to the authors' proficiency in Python and the readability of Python code, these drawbacks were taken into account while still implementing the reference implementation in Python.

Chapter 6

Conclusion

The approach demonstrated by LocID represents an advancement in geospatial data representation and analysis by combining the efficiency and hierarchy of a DGGS (Discrete Global Grid System) with the capability to encode complex geometries like polygons and lines directly within its cells. This unique combination allows users to perform spatial operations and queries directly on DGGS cell IDs, simplifying spatial analysis and improving efficiency.

6.1 Contributions

LocID extends an OGC-compliant DGGS by enabling the encoding of geometries and the processing of a few operations on geometric objects within DGGS cells. It takes advantage of the inherent properties of QTM, such as the systems hierarchy and uniformity of the cell area. In this way, computationally heavy spatial operations can be omitted by performing these operations on the IDs through string (or byte) operations. This has implications for the system's speed.

DGGS-based geospatial systems represent the future of geospatial data handling due to their efficiency, scalability, and hierarchical nature (Li & Stefanakis, 2020). However, the challenge of encoding complex geometries within DGGS cells remains. Approaches such as LocID offer a solution by integrating geometries within DGGS cells, providing efficient, scalable, and interoperable systems that can handle the diverse and complex spatial data of the future.

6.2 Limitations

LocID, like any geospatial system, has its own set of restrictions that should be taken into account when deciding whether it is suitable for a particular application. The Python implementation of LocID does not make full use of byte array operations, which slows the execution of certain operations. The underlying DGGS (QTM) of LocID is based on triangles, which have the advantage of being compact and symmetrical but can be problematic when dealing with applications that rely on cell distances, as the distances between cells in a triangular grid are not uniform and can vary depending on direction. This nonuniformity can make distance-based calculations more difficult or even impossible. Additionally, triangles can limit the spatial resolution achievable in certain applications, as they are not suitable for representing fine-grained details and sharp angles, with their area-based description of geometries. While LocID can encode complex geometries such as polygons and lines, the encoding process can become computationally intensive for shapes that are not well-aligned with the underlying triangular grid. Identifying highly detailed geometries with a LocID may therefore require additional computational resources.

6.3 Outlook

LocID demonstrates a possible first step in the direction of extending DGGSs with the encoding of geometric features. While it successfully proves that this is possible, there is room for further research and development in this direction.

A logical next area of exploration is the conceptualization of point and line operations within the LocID framework. Although LocID currently provides concepts for encoding points and lines and performing operations on them, there is potential for further refinement, as the current concept is based on considerations made for polygons.

The fact that six triangles together can form a hexagon is promising. This leads to the idea of the integration of hexagonal layers within the LocID system. These layers could serve various purposes, including convolution, distance calculations, and acting as a conversion layer to facilitate interoperability with other DGGS systems, such as H3. Such integration would expand LocID's versatility and compatibility.

To make LocID more practical and efficient, there is a need for performance enhancement. This involves evaluating and optimizing critical components of the reference implementation. Options include potentially rewriting these components in a more efficient compiled programming language or exploring a transition to a more performance-oriented language.

An exciting prospect on the horizon is the inclusion of 3D capabilities within LocID. Enabling LocID to represent and analyze volumetric data can open up new possibilities for applications that involve three-dimensional geospatial information, further enhancing its flexibility.

Additionally, LocID should consider broadening its suite of operations beyond its existing capabilities such as matching, contain queries, and change detection. Incorporating operations that address various spatial relationships and queries will make LocID a comprehensive and versatile tool for geospatial data processing.

Appendix

Appendix System information

All code was run and tested on the following system:

```
OS: Manjaro Linux x86_64
Host: ThinkPad T14 Gen 1
Kernel: 6.1.44-1-MANJARO
Mem: 30.58 GiB
CPU: AMD Ryzen 7 PRO 4750U with Radeon Graphics
    speed/min/max: 1398/1400/1700 MHz
GPU: AMD ATI 07:00.0 Renoir

Python: 3.11.4 64bit

Python packages:
fiona                     1.9.4
gdal                      3.7.0
geopandas                 0.13.2
geopandas-base            0.13.2
geos                      3.11.2
numpy                     1.25.2
pandas                    2.0.3
proj                      9.2.1
shapely                   2.0.1
```

Any references to computing time, come from running code on said system.

For the writing of this thesis, only open-source software was used. Manjaro Linux as an OS, Obsidian.md for notes and knowledge management, Zotero and BetterBibTeX for bibliography management, VS Code as a code editor, Python with several packages, including, but not exclusively, GeoPandas, NumPy, shapely, GDAL, leaflet, and pyproj, draw.io for most of the graphs, LaTeX and Overleaf for text processing.

Bibliography

Arthur, R. (2023, August 30). *A Critical Analysis of the What3Words Geocoding Algorithm*. arXiv: 2308.16025 [cs]. https://doi.org/10.48550/arXiv.2308.16025

Banholzer, S., Kossin, J., & Donner, S. (2014). The Impact of Climate Change on Natural Disasters. In A. Singh & Z. Zommers (Eds.), *Reducing Disaster: Early Warning Systems For Climate Change* (pp. 21–49). Springer Netherlands. https://doi.org/10.1007/978-94-017-8598-3_2

Barnes, T. (2019). *Haggett, Peter*. SAGE Publications Ltd. https://doi.org/10.4135/9781526421036788331

Bondaruk, B., & Roberts, S. (2019). Discrete Global Grid Systems. Retrieved August 20, 2023, from https://www.semanticscholar.org/paper/Discrete-Global-Grid-Systems-Bondaruk-Roberts

Bondaruk, B., Roberts, S. A., & Robertson, C. (2020). Assessing the state of the art in Discrete Global Grid Systems: OGC criteria and present functionality. *Geomatica*, *74*(1), 9–30. https://doi.org/10.1139/geomat-2019-0015

Bousquin, J. (2021). Discrete Global Grid Systems as scalable geospatial frameworks for characterizing coastal environments. *Environmental Modelling & Software*, *146*, 105210. https://doi.org/10.1016/j.envsoft.2021.105210

Bush, I. (2017, June 16). OpenEAGGR Software Design Document. *Riskaware Ltd*. Retrieved August 22, 2023, from https://github.com/riskaware-ltd/open-eaggr/blob/master/Documents/Software%20Design%20Document.pdf

Chaudhuri, C., Gray, A., & Robertson, C. (2021). InundatEd-v1.0: A height above nearest drainage (HAND)-based flood risk modeling system using a discrete global grid system. *Geoscientific Model Development*, *14*(6), 3295–3315. https://doi.org/10.5194/gmd-14-3295-2021

Dutton, G. H. (1984). Part 4: Mathematical, algorithmic and data structure issues: Geodesic modelling of planetary relief. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *21*(2-3), 188–207.

Dutton, G. H. (1999). *A Hierarchical Coordinate System for Geoprocessing and Cartography* (Vol. 79). Springer-Verlag. https://doi.org/10.1007/BFb0011617

Eckersley, R. J. (1995, December). *Amateur Radio Operating Manual* (4Rev Ed edition). Radio Society of Great Britain
Open Library ID: OL12077315M.

Goodchild, M. F. (2019). Preface. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *54*(1), 1–3. https://doi.org/10.3138/cart.54.1.preface

Goodchild, M. F., & Shiren, Y. (1992). A hierarchical spatial data structure for global geographic information systems. *CVGIP: Graphical Models and Image Processing*, *54*(1), 31–44. https://doi.org/10.1016/1049-9652(92)90032-S

Gould, P. (1975). Mathematics in geography: Conceptual revolution or new tool? *1975*, *27*(2), 303–327. Retrieved August 23, 2023, from https://unesdoc.unesco.org/ark:/48223/pf0000014961

Israel, R. (2003, January 20). *Mercator's Projection.* University of British Columbia Mathematics Department. Retrieved August 18, 2023, from https://personal.math.ubc.ca/~israel/m103/mercator/mercator.html

Jendryke, M., & McClure, S. C. (2019). Mapping crime – Hate crimes and hate groups in the USA: A spatial analysis with gridded data. *Applied Geography*, *111*, 102072. https://doi.org/10.1016/j.apgeog.2019.102072

Kamel, I. (2008). Indexing, Hilbert R-Tree, Spatial Indexing, Multimedia Indexing. In S. Shekhar & H. Xiong (Eds.), *Encyclopedia of GIS* (pp. 507–512). Springer US. https://doi.org/10.1007/978-0-387-35973-1_603

Kamel, I., & Faloutsos, C. (1993). Hilbert R-Tree: An Improved R-Tree Using Fractals. Retrieved August 19, 2023, from http://hdl.handle.net/1903/5366

Kmoch, A., Matsibora, O., Vasilyev, I., & Uuemaa, E. (2022). Applied open-source Discrete Global Grid Systems. *AGILE: GIScience Series*, *3*, 1–6. https://doi.org/10.5194/agile-giss-3-41-2022

Kmoch, A., Vasilyev, I., Virro, H., & Uuemaa, E. (2022). Area and shape distortions in open-source discrete global grid systems. *Big Earth Data*, *6*(3), 256–275. https://doi.org/10.1080/20964471.2022.2094926

Kotze, C. (2020). Addressing the Un-Addressed: Opportunities for Rural-Africa. In A. Froehlich (Ed.), *Space Fostering African Societies: Developing the African Continent through Space, Part 1* (pp. 153–174). Springer International Publishing. https://doi.org/10.1007/978-3-030-32930-3_11

Li, M., & Stefanakis, E. (2020). Geospatial Operations of Discrete Global Grid Systems—a Comparison with Traditional GIS. *Journal of Geovisualization and Spatial Analysis*, *4*(2), 26. https://doi.org/10.1007/s41651-020-00066-3

Liao, C., Tesfa, T., Duan, Z., & Leung, L. R. (2020). Watershed delineation on a hexagonal mesh grid. *Environmental Modelling & Software*, *128*, 104702. https://doi.org/10.1016/j.envsoft.2020.104702

Mao, Q., Qader, M. A., & Hristidis, V. (2023). Comparison of LSM indexing techniques for storing spatial data. *Journal of Big Data*, *10*(1), 1–26. https://doi.org/10.1186/s40537-023-00734-3

Peterson, P. R. (2017). Discrete Global Grid Systems. In *International Encyclopedia of Geography* (pp. 1–10). John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118786352.wbieg1050

Purss, M. (2017, August 1). *Topic 21: Discrete Global Grid Systems Abstract Specification*. Retrieved November 2, 2022, from https://docs.ogc.org/as/15-104r5/15-104r5.html

Rawson, A., Sabeur, Z., & Brito, M. (2022). Intelligent geospatial maritime risk analytics using the Discrete Global Grid System. *Big Earth Data*, *6*(3), 294–322. https://doi.org/10.1080/20964471.2021.1965370

Rinckes, D., & Bunge, P. (n.d.). Open Location Code Definition. Retrieved October 4, 2022, from https://github.com/google/open-location-code/blob/main/docs/olc_definition.adoc

Robertson, C., Chaudhuri, C., Hojati, M., & Roberts, S. A. (2020). An integrated environmental analytics system (IDEAS) based on a DGGS. *ISPRS Journal of Photogrammetry and Remote Sensing*, *162*, 214–228. https://doi.org/10.1016/j.isprsjprs.2020.02.009

Samet, H. (2006, August 8). *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann.

StructureLocator™ - QA Locate. (2019, October 16). Retrieved November 28, 2022, from https://www.qalocate.com/solutions/structurelocator/

Tamminen, M. (1985). Efficient Storage of Quadtrees and Octrees. In H. Freeman & G. G. Pieroni (Eds.), *Computer Architectures for Spatially Distributed Data* (pp. 261–278). Springer. https://doi.org/10.1007/978-3-642-82150-9_15

Uber. (n.d.). *H3*. H3 - Hexagonal hierarchical geosaptial indexing system. Retrieved October 6, 2022, from https://h3geo.org/

Veach, E. (n.d.). *About S2*. S2Geometry. Retrieved August 20, 2023, from http://s2geometry.io/about/

Wang, N., Vlachokostas, A., Borkum, M., Bergmann, H., & Zaleski, S. (2019). Unique Building Identifier: A natural key for building data matching and its energy applications. *Energy and Buildings*, *184*, 230–241. https://doi.org/10.1016/j.enbuild.2018.11.052

Personal declaration

I hereby declare that the submitted thesis is the result of my own, independent work. All external sources are explicitly acknowledged in the thesis.

Zürich, 30.09.2023

[illegible]

Linus Rüegg