Implementation and Compliance Benchmarking of a DGGS-enabled, GeoSPARQL-aware Triplestore

David Habgood[b], Timo Homburg[a], Nicholas J. Car[b,c] and Milos Jovanovik[d,e]

[a]*i3mainz – Institute for Spatial Information & Surveying Technology, Mainz University of Applied Sciences, 55128 Mainz, Germany*

[b]*SURROUND Australia Pty. Ltd, New Acton, Canberra, ACT 2601, Australia*

[c]*Australian National University, Canberra, ACT 2600, Australia*

[d]*OpenLink Software Ltd., Croydon, Surrey, CR0 0XZ, United Kingdom*

[e]*Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje, 1000 Skopje, North Macedonia*

Abstract

We set out to determine the feasibility of implementing Discrete Global Grid System (DGGS) representations of geometry support in a GeoSPARQL-enabled triplestore, and test the GeoSPARQL compliance for it. The implementation is a variant of Apache Jena's existing GeoSPARQL support. Compliance is tested using an adapted implementation of the *GeoSPARQL Compliance Benchmark* testing system developed previously to test for GeoSPARQL 1.0 compliance. The benchmark results confirm that a majority of the functions which were set out to be implemented in the course of this paper were implemented correctly and points out possible future work for full compliance.

Keywords

Geospatial Data, DGGS, GeoSPARQL, Apache Jena Fuseki, Compliance Benchmarking, RDFLib

1. Introduction

This paper presents compliance tests results of extensions to existing GeoSPARQL [1] system implementation, that of Apache's Jena database[1], for Discrete Global Grid System (DGGS) [2] geometry representations. Conformance testing was performed with an updated version of an existing GeoSPARQL compliance benchmark test. [3].

1.1. GeoSPARQL

GeoSPARQL is an Open Geospatial Consortium implementation standard that defines "a core set of classes, properties and datatypes that can be used to construct query patterns" for the

GeoLD 2022: 5th International Workshop on Geospatial Linked Data co-located with ESWC, May 30 2022, Hersonissos, Greece

david.habgood@surroundaustralia.com (D. Habgood); timo.homburg@hs-mainz.de (T. Homburg); nicholas.car@anu.edu.au (N. J. Car); milos.jovanovik@finki.ukim.mk (M. Jovanovik)

https://surroundaustralia.com/about-us (D. Habgood); https://situx.github.io/ (T. Homburg); https://cecs.anu.edu.au/people/nicholas-car (N. J. Car); https://mjovanovik.com (M. Jovanovik)

0000-0002-9499-5840 (T. Homburg); 0000-0002-8742-7730 (N. J. Car); 0000-0001-7360-8015 (M. Jovanovik)

[1]https://jena.apache.org

SPARQL [4] query language used to query Resource Description Framework (RDF) [5] data. In the 10+ years since GeoSPARQL's initial publication, it has become the premier Linked Data and Semantic Web spatial data representation system, as evidenced by widespread deference to GeoSPARQL for spatial concerns in commonly-used Semantic Web data models such as the *Data Catalog Vocabulary* (DCAT) DCAT2 [6] and CIDOC-CRM [7, 8].

In GeoSPARQL, geometry serializations are primitive data types (literals) that describe a geometry's spatial coverage. Version 1.0 of GeoSPARQL, released in 2012, contained descriptions of how to use two geometry serialization types: Geographic Mark-up Language (GML) [9] and Well-Known Text (WKT) [10]. The simple pattern of these two types' implementations lends itself to extension for other geometry types, and the 1.1 version included descriptions of GeoJSON [11], Keyhole Mark-up Language (KML) [12] and a generic DGGS placeholder.

GeoSPARQL currently being updated with a version 1.1, [13] release expected in 2022 which includes new data model elements, query functions and support for the identification and use of DGGS geometry serializations.

1.2. Discrete Global Grid Systems (DGGS)

DGGS are multi-layered grid system representations of surfaces, such as the Earth's. They are an emerging spatial data technology for which great computational efficiency in the performance of tasks such as shortest path analysis, geometry intersections and comparisons is claimed [14]. Their proponents claim that DGGS utilize computing data structures better than traditional spatial data systems, based on coordinate reference systems. This provides motivation to test whether a DGGS implementation can be implemented in a database, and achieve the theoretical high performance expected. The particular type of database chosen, a triplestore, provides the ability to query large amounts of heterogeneous data efficiently without joins. The addition of a DGGS implementation to a triplestore could then provide a platform for efficient queries across spatial and feature data.

Another touted benefit of DGGSes is their ability to represent both raster and vector spatial information in unified form, for a given spatial accuracy. Commercial companies exist internationally that specilise in raster and vector spatial data integration[2] via DGGS and some large technology companies are known to employ DGGS for large-scale spatial data operations[3].

A recent description of the multiple types of DGGSes, their major characteristics, differences from traditional systems and potential standardized APIs for accessing their unique functions is the Open Geospatial Consortium's second version of its DGGS abstract specification: *Discrete Global Grid Systems - Part 1 Core* [15].

DGGSes exist that use different shapes for their surface-covering grids, and Figure 1 shows some of them.

Figure 2 shows a geometry and successively higher resolution *AusPIX* DGGS [16] approximations of it. There is no theoretical limit to the resolution that a DGGS may attain as smaller

[2]Global Grid Systems, https://www.globalgridsystems.com/, claims such data integration is a major benefit of their use of DGGS

[3]Uber, the ride hailing company, employs its own H3 DGGS for closest vehicle searching: https://eng.uber.com/h3/

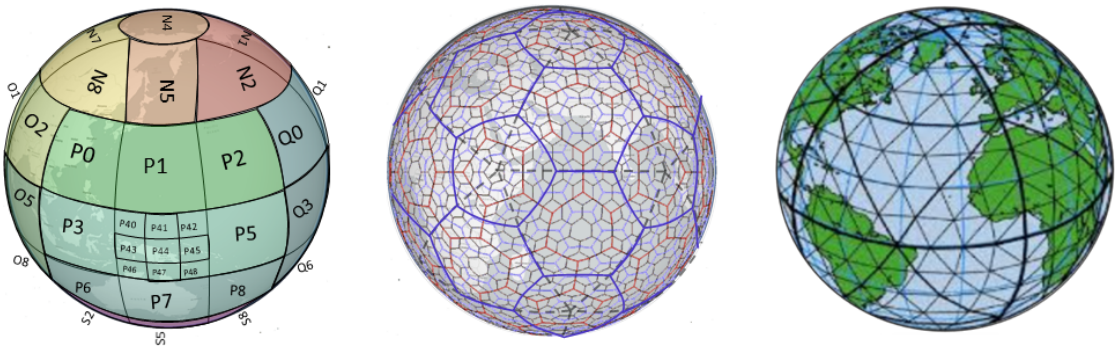

Figure 1: Examples of DGGS based on the mapping of the faces of Platonic solids to the surface model of the Earth, after Figure C.2 from [15]

"cells" may always be defined. Some DGGS, like this one, may use multi-scale cells in one geometry resolution approximation, whereas others use cells only of one size at a time.

2. Implementation

2.1. DGGS Literals Definition

The non-DGGS geometry serializations that GeoSPARQL 1.1 describes define their formats precisely[4], however DGGS is not a single system with a single serialization specification but a class of systems, and members of the class define their geometry data and formats quite differently. For this reason, GeoSPARQL does not refer to an authoritative format reference for DGGS data. To highlight how different the various DGGS formats are: the H3 system[5] uses tessellating hexagons with opaque UUIDs for cell identifiers, whereas the rHealPIX family [17] uses squares with hierarchically-structured identifiers. For this reason, GeoSPARQL 1.1 essentially provides an abstract DGGS description and urges users to implement particular DGGS system descriptions outside the GeoSPARQL standard.

Examples within the GeoSPARQL 1.1 Specification's Annex C show geometry serializations indicated with the `geo:asDGGS` property (`geo:` for GeoSPARQL) and typed with the `ex:auspixDggsLiteral` datatype (`ex:` for the `example.org` demonstration namespace) which informally identifies the *AusPIX* DGGS [16], which is a member of the rHealPIX DGGS family.

The particular DGGS used for compliance benchmarking testing in this paper is also *AusPIX*, however a formal IRI identifier was implemented to indicate AusPIX literals, as opposed to the GeoSPARQL Specification's example IRI: https://w3id.org/dggs/auspixLiteral.

Listings Listing 1 & Listing 2 provide a WKT and an AusPIX DGGS representation of the geometry of the feature shown in Figure 2 for comparison. It can be seen that an AusPIX `CELLLIST` is very similar to a WKT `POLYGON`.

[4]See the GeoSPARQL 1.1 Specification's section on 'Geometry Serializations' for notes on how the systems are used and references to their defining sources

[5]https://eng.uber.com/h3/

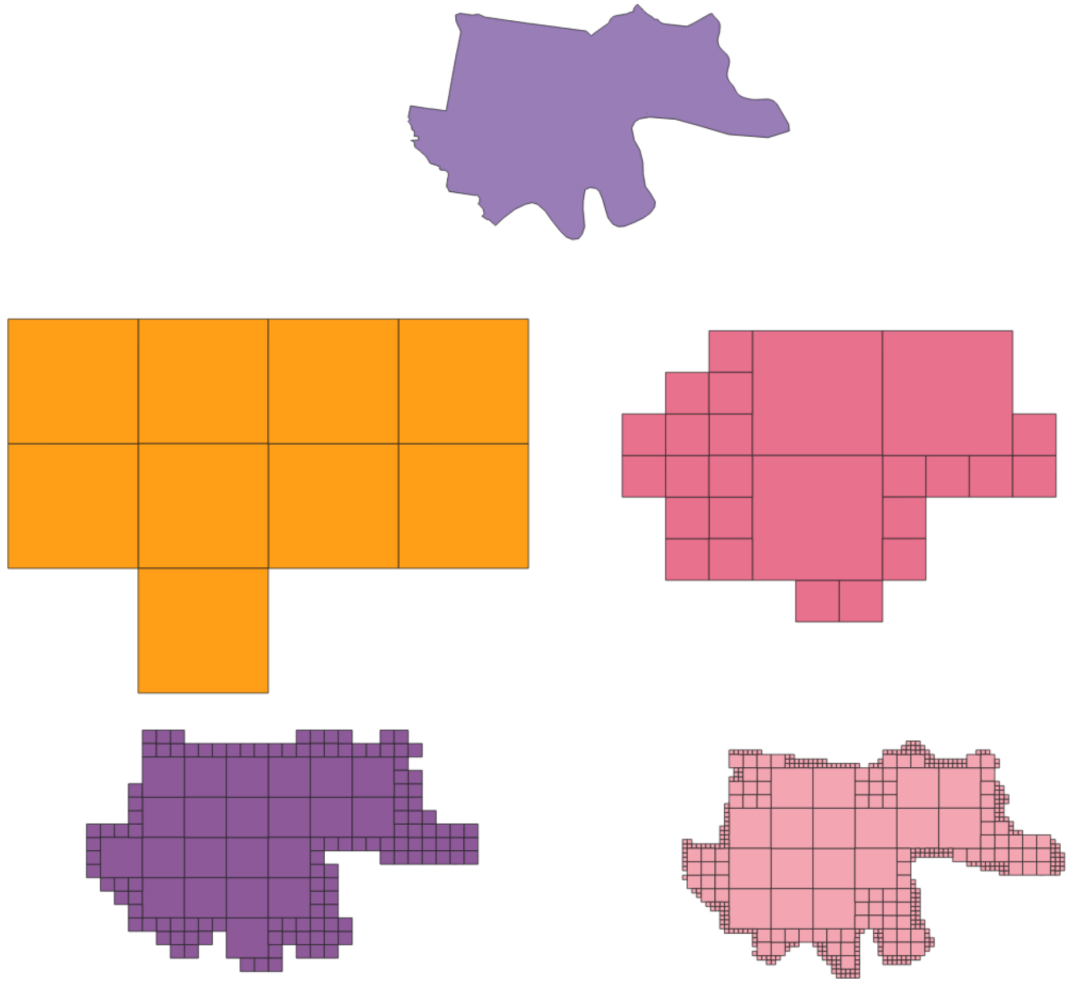

Figure 2: Images of the Australian federal government electorate of Brisbane represented as a polygon (top, center) and then as a series of higher-resolution AusPIX DGGS approximations. The least detailed approximation (in orange) corresponds to the data in Listing Listing 2

Listing 1: A partial representation of the Australian federal government electorate of Brisbane in the Turtle format of RDF using Well-Known Text as the geometry serialization as per GeoSPARQL 1.0

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .

<https://linked.data.gov.au/dataset/asgs2016/commonwealthelectoraldivision/304>
   a geo:Feature ;
   geo:hasGeometry [
      geo:asWKT """
         POLYGON ((
         153.099932 -27.445258, 153.092961 -27.447432, 153.080142 -27.446423,
         ...
         153.053422 -27.439453, 153.053451 -27.439456, 153.053452 -27.439456))
         """^^geo:wktLiteral ;
   ] ;
.
```

Listing 2: A complete but low resolution representation of the Australian federal government electorate of Brisbane using AusPIX DGGS. This representation is shown in Figure 2 as the nine-squared orange shape

```
@prefix dggs: <https://w3id.org/dggs/> .
@prefix geo: <http://www.opengis.net/ont/geosparql#> .

<https://linked.data.gov.au/dataset/asgs2016/commonwealthelectoraldivision/304>
   a geo:Feature ;
   geo:hasGeometry [
      geo:asDGGS """
         CELLLIST ((
         R8338506 R8338507 R8338508 R8338516 R8338530
         R8338531 R8338532 R8338534 R8338540))
         """^^dggs:auspixLiteral ;
   ] ;
.
```

Only AusPIX DGGS geometry representations were implemented for this work.

2.2. DGGS Data Production

To test spatial functions' use of DGGS data, traditional geometry data had to be converted to DGGS form. For this, we extended existing traditional-to-DGGS conversion functions present in the *rHEALPixDGGS Python Package*[6]. That package provides many rHEALPix DGGS functions, including geometry visualisation and testing.

The conversion approach we used employs intersecting traditional geometry representations of finer and finer resolution AusPIX cells (square polygons) with traditional representations of the target data formulated as polygons, point, lines etc. Cells selected for the DGGS representation of the target are those that either contain, overlap or are contained by it, depending on the target geometry particulars. Since we progress through cells at different levels of DGGS resolution, any required resolution can be obtained, as per Figure 2. We used the *Shapely*[7] Python library for this task.

Our conversion process was applied to the dataset used by the compliance benchmark tooling for all GeoSPARQL system tests and the resultant data was made available to the testing system online via a customized instance of Apache's Fuseki triplestore interface tool.

2.3. DGGS Functions

The focus of this work was to test whether DGGS geometry serializations could be used with GeoSPARQL's defined set of functions and thus act similarly to non-DGGS or traditional geometries. GeoSPARQL's topological functions are based on the commonly used *Simple Features Access* [10] geometry function specification. Additionally, GeoSPARQL 1.1 defines

[6]https://pypi.org/project/rHEALPixDGGS/
[7]https://pypi.org/project/Shapely/

several other series of functions for geometry aggregation, geometry extent calculation, and so on. These are listed in the GeoSPARQL 1.1 Specification document, Annex B[8].

A set of SPARQL filter functions were written in Java using Apache Jena's Filter Functions extension mechanism[9]. This then allowed instances of the Jena database used by the Fuseki triplestore interface to accept SPARQL queries and to apply them to data containing AusPIX DGGS geometry representations.

While the filter functions implemented data processing logic quite different to that of traditional geometry processing toolkits, given the very different form of AusPIX data to traditional geometry data, the high-level SPARQL functions were intentionally matched to GeoSPARQL's "Simple Features" functions so that, for example, a function for `geof:sfContains` was implemented, as were functions for all of the other `sf` functions except for `geof:sfCrosses`. This omissions was made due to a discrepancy in GeoSPARQL's definition of that function which meant it was un-implementable: its allowed inputs listed differently in GeoSPARQL 1.1's *Simple Features Relation Family* and *Annex B* (normative function definitions). An issue for this in the GeoSPARQL issue tracker has been raised[10].

Currently we have catered for AusPIX geometry representations only and not a combination of DGGS and traditional geometries, thus GeoSPARQL functions which take mixed literal input, for example GeoJSON and DGGS, are not yet implemented. We believe it will be possible to allow both forms of data in a single system in the future but we do not know whether cross-querying of DGGS & non-DGGS data will be possible due to the computational effort of DGGS conversions which would have to be handled *on the fly* or at data load time.

While GeoSPARQL defines three sets of topological query functions - "Simple Features", "Egenhofer" [18] & "Region Connection Calculus" [19] - only "Simple Features" filter functions were implemented. This was due to time constraints and not due to a lack of interest or thought that they would be un-implementable, indeed the GeoSPARQL equivalent topological relations table[11] indicates that most of the "Egenhofer" & "Region Connection Calculus" functions either are or likely could easily be, implemented due to their correspondence with "Simple Features" functions.

3. Compliance Benchmarking

We chose an extended version of the GeoSPARQL 1.0 compliance benchmark [3] to test for the compatibility of the given implementations. We added new sub-tests for the existing requirements in order to include the new DGGS literals in the testing. The compliance benchmark consists of a benchmarking dataset and a set of SPARQL queries including its expected answers which are checked for validity in the benchmark execution. Contrary to the original GeoSPARQL Compliance Benchmark publication, this benchmark is not executed on the HOBBIT benchmarking platform, but can instead be executed using a python script which executes

[8] https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html#_annex_b_functions_summary_normative

[9] https://jena.apache.org/documentation/query/writing_functions.html

[10] https://github.com/opengeospatial/ogc-geosparql/issues/304

[11] https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html#_equivalent_rcc8_egenhofer_and_simple_features_topological_relations

the given queries and calculates the compliance score in the way that we describe later on. The benchmark queries and the python script to execute the benchmark have been published on Github[12]

3.1. Compliance Benchmarking Dataset

We chose the same benchmarking dataset as given in the GeoSPARQL compliance benchmark [3] (cf. Figure 3) and extended the dataset with DGGS representations of all of its geometries[13] The dataset is available in two different AusPIX resolutions: Level 7 and Level 10. These are maximum resolutions for representing geometries in the datasets, and two resolutions were chosen to evaluate whether an insufficiently fine resolution could produce erroneous test results.

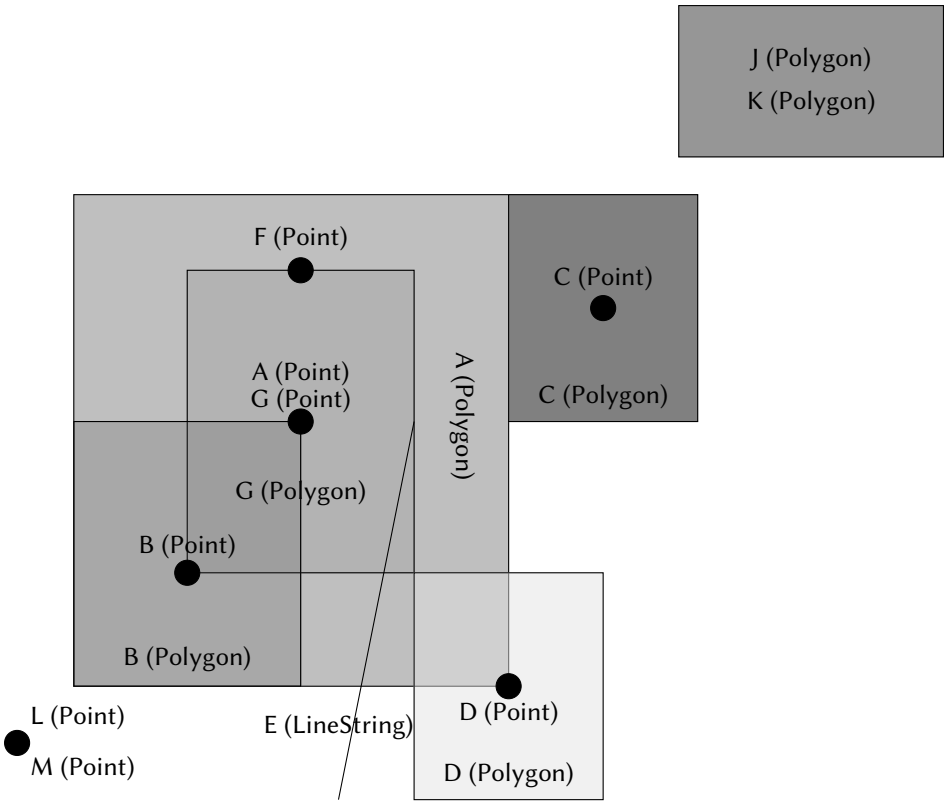

Figure 3: Abstract view of the geometries which are part of the benchmark dataset. Geometries A, B, C, D, G, J and K represent `Polygon` geometries and (aside from J and K) all have a center `Point` geometry, as well. Geometry E represents a `LineString` geometry, while geometries F, L and M represent `Point` geometries. Geometries H and I are empty geometries and not visible in this figure. All geometries are represented in the CRS84 geodetic system, except for geometry M which is represented in EPSG:4326. Each geometry is represented both using WKT and GML literals. After Figure 1 in [3].

We chose two different DGGS resolutions to test the compliance of the implementation in

[12]https://github.com/OpenLinkSoftware/GeoSPARQLBenchmark/tree/dggs
[13]https://github.com/surroundaustralia/geosparql-benchmark-dggs/tree/master/output

these two cases to get an idea if the granularity of DGGS literals has an impact on the compliance benchmark results.

3.2. Compliance Tests

The GeoSPARQL 1.0 Compliance benchmark consisted of a set of 205 test queries for each of the 30 requirements of the GeoSPARQL 1.0 specification.

To extend this set of queries to test for DGGS support, additional queries and appropriate answers which test DGGS support needed to be added.

At first, we extended the GeoSPARQL Compliance benchmark to test for basic DGGS compatibility, defined as test queries for requirements 31-34 containing the following tests for basic DGGS compatibility which are also envisioned as requirements 36-39 for GeoSPARQL 1.1 [13].

- Check that the triple store accepts the `geo:asDGGS` property
- Check that the triple store accepts DGGS literals
- Check that the triple store can cope with empty DGGS literals
- Tests the geof:asDGGS conversion function.

Furthermore, test queries which were used to test GeoSPARQL 1.0 requirements needed to be extended for DGGS inputs. This affected the following requirements:

- **Requirement 19**: Implementations shall support `geof:distance`, `geof:buffer`, `geof:convexHull`, `geof:intersection`, `geof:union`, `geof:difference`, `geof:symDifference`, `geof:envelope` and `geof:boundary` as SPARQL extension functions, consistent with the definitions of the corresponding functions (distance, buffer, convexHull, intersection, difference, symDifference, envelope and boundary respectively) in Simple Features [10]
- **Requirement 21**: Implementations shall support `geof:relate` as a SPARQL extension function, consistent with the relate operator defined in Simple Features [10]
- **Requirement 22**: Implementations shall support `geof:sfEquals`, `geof:sfDisjoint`, `geof:sfIntersects`, `geof:sfTouches`, `geof:sfCrosses`, `geof:sfWithin`, `geof:sfContains`, `geof:sfOverlaps` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [10]
- **Requirement 23**: Implementations shall support `geof:ehEquals`, `geof:ehDisjoint`, `geof:ehMeet`, `geof:ehOverlap`, `geof:ehCovers`, `geof:ehCoveredBy`, `geof:ehInside`, `geof:ehContains` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [10]
- **Requirement 24**: Implementations shall support `geof:rcc8eq`, `geof:rcc8dc`, `geof:rcc8ec`, `geof:rcc8po`, `geof:rcc8tppi`, `geof:rcc8tpp`, `geof:rcc8ntpp`, `geof:rcc8ntppi` as SPARQL extension functions, consistent with their corresponding DE-9IM intersection patterns, as defined by Simple Features [10]

We tested these requirements for compatibility with all other literal types of GeoSPARQL 1.0 as well, i.e. WKT and GML literals. We tested scenarios using DGGS literals only, as well as scenarios using a mix of DGGS and either WKT or GML literals. In all tests we check for correct results, and assign scores accordingly.

3.3. Compliance Scores

The scoring of the requirements is based on the scoring of the GeoSPARQL 1.0 compliance benchmark [3], but is adjusted because more test queries need to be executed to test DGGS compatibility.

Similar to the GeoSPARQL compliance benchmark, we provide two scores:

- **Correct answers**: The number of correct answers out of all tests.
- **Compliance percentage**: The percentage of compliance with the requirements of the GeoSPARQL 1.0 standard and the additional requirements for DGGS.

The `correct answers` score is rather straightforward – it represents the number of correct answers the tested system provided, out of the 353 total test queries. The `compliance percentage` score tries to provide a uniform overview across all of the requirements included in the benchmark, which are 34 in total. This is achieved by assigning each requirement the same weight and impact in the final score (1/34, or 2.94%), as opposed to the first score in which this uniformity is applied on the level of the test queries. Given that different requirements use a different number of sub-test queries in order to be evaluated (spanning from a single test query, to 64 different sub-test queries per requirement), it is evident that not each test query has the same impact, when viewed from a requirements perspective. Since the GeoSPARQL standard is defined through a set of requirements, we believe providing a score based on the amount of requirements met is of great significance.

In the requirements which are evaluated using multiple test queries, the 2.94% are uniformly distributed among these sub-tests. So, for instance, when a requirement has eight sub-tests, they each contribute with 12.5% to the parent test score, i.e., with 0.3676% (2.94% × 12.5%) to the total benchmark compliance percentage score. With this, a single requirement from the GeoSPARQL standard can be either fully supported, partially supported or not supported at all.

An exception to this rule of uniformly distributing the weights between requirements on one level, and their sub-tests on another level, are the sub-test queries which test the GeoSPARQL functions with different serializations of literals as parameters, i.e., requirements 19–24. When we test a GeoSPARQL function for compliance to the standard while using (a) WKT-only literals, (b) GML-only literals, (c) DGGS-only literals, (d) a combination of WKT and GML literals, (e) a combination of GML and DGGS literals, and (f) a combination of DGGS and WKT literals, the score is uniformly distributed between these six logical groups, each contributing with 16.67% to the parent test score. However, (d), (e) and (f) are practically tested using two queries each: one where one literal type is the first, and the other literal type is the second parameter in the function (e.g. `function(type1, type2)`), and vice versa (`function(type2, type1)`). These queries technically contribute with 8.33% to the parent test score each, so that the total contribution from the logical group (d) remains 16.67%; the same thing happens for (e) and (f). With this, the technical weight of the queries themselves is 16.67% for the WKT-only query, 16.67% for the GML-only query, 16.67% for the DGGS-only query, 8.33% for the WKT-GML query, 8.33% for the GML-WKT query, 8.33% for the WKT-DGGS query, 8.33% for the DGGS-WKT query, 8.33% for the GML-DGGS query, and 8.33% for the DGGS-GML query. Technically, on a query level, this is an exception of the uniformity rule we practice, but, logically, on a group level, it is upheld.

4. Benchmark Results

We present the results of our benchmark execution for the two implementations we have presented in this publication in Table 1.

Dataset	Benchmark Score	Benchmark Correct Answers	DGGS Score for Req. 22	DGGS Answers for Req. 22
Level 7	54.5%	122/353	62.5%	5/8
Level 10	54.5%	122/353	62.5%	5/8

Table 1
Benchmark results for DGGS compatibility for the Apache Jena GeoSPARQL triple store

The table exhibits two different types of the results for this tested implementation: the overall benchmark score and the score for Requirement 22 only. The latter gives an indication of the successful implementation of the anticipated functionality.

4.1. Discussion of Results

The results of the benchmark show an overall compliance score of 54.5%. This score may also be viewed according to the GeoSPARQL 1.0 extensions and shows a 100% compatibility of the CORE and RDFSE extension, which is akin to the result obtained from GeoSPARQL-Jena in the GeoSPARQL 1.0 compliance benchmark. Also, the query rewrite extension performs comparably with 45.83% compatibility. Differences can be observed in the topology extension (25% compatibility), possibly due to implementation errors and, as expected in the Geometry extension (59.01%) and Geometry Topology extension (39.89%) scores.

The majority of triplestores evaluated utilising the original benchmark achieved scores in the 50%-70% range [3]. This benchmark includes GeoSPARQL extensions which not all vendors attempted to implement, accounting for a significant reduction in the scores. Similarly, we have not attempted to implement all aspects of the GeoSPARQL standard initially, instead focusing on determining the feasibility of implementing DGGS functions in a triplestore. The relevant score for this aspect is Req 22, as shown in Table 1.

The `geof:sfCrosses` function returned an empty result, as expected, as this function has not been implemented. Implementation errors are the likely source of tests for `geof:sfTouches` and `geof:sfIntersects` failing to pass. These are likely easily solvable, given the passing of other, similar, functions. Getting these tests to pass will be taken as immediate *Future Work*.

No difference in compliance has been observed between the two datasets of different resolutions. It is likely that there isn't sufficient polygonal complexity in the test data to highlight differences that we do expect to eventually see, given a large enough difference in resolutions.

5. Conclusions

This publication introduced a first implementation for DGGS handling in a GeoSPARQL 1.0 compatible triple store implementation and a conformance test based on the GeoSPARQL 1.0

compliance benchmark test which checks for the correctness of a DGGS implementation. With this we have contributed two fundamental things:

- A GeoSPARQL implementation using DGGS data in triple stores is feasible
- A low but useful compliance score has been achieved for the important topological query functions

The compliance score overall is low however the GeoSPARQL parts we aimed to implement are as compliant as expected, other than two what seem to be simple implementation errors for `geof:sfTouches` and `geof:sfIntersects`, but we have not yet characterized the error precisely.

5.1. Future Work

We expect that, after this initial work, extensions to this implementation to handle other topological query functions (Egenhofer, RCC8) will be straightforward, given their similarity to "Simple Features" and that extensions for many other GeoSPARQL functions will be feasible too. We cannot yet predict exactly how DGGS/non-DGGS conversions can be sensibly implemented, given the need to provide common spatial indexing across all data, other than by dual-storing DGGS/non-DGGS geometry data with a conversion-on-load approach. New conversion tooling willbe needed for this as we currently do not have integrated conversion capability.

We envision that once the GeoSPARQL 1.1 standard has been approved, a successor to the GeoSPARQL compliance benchmark which is able to test for GeoSPARQL 1.1 compatibility should be created and also be executed in a reproducible environment such as the HOBBIT benchmarking platform.

References

[1] M. Perry, J. Herring, OGC GeoSPARQL - A Geographic Query Language for RDF Data, OGC Implementation Standard, 2011.

[2] K. Sahr, D. White, Discrete Global Grid Systems, Computing Science and Statistics (1998) 269–278.

[3] M. Jovanovik, T. Homburg, M. Spasić, A GeoSPARQL Compliance Benchmark, ISPRS International Journal of Geo-Information 10 (2021) 487. doi:10.3390/ijgi10070487.

[4] A. Seaborne, S. Harris, SPARQL 1.1 Query Language, W3C Recommendation, W3C, 2013. URL: https://www.w3.org/TR/2013/REC-sparql11-query-20130321/, accessed on 2022-01-15.

[5] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, World Wide Web Consortium, 2014. URL: https://www.w3.org/TR/rdf11-concepts/, accessed on 2022-01-15.

[6] A. Gonzalez-Beltran, D. Browning, S. Cox, P. Winstanley, Data Catalog Vocabulary (DCAT) - revised edition, W3C Recommendation, World Wide Web Consortium, 2018. URL: https://www.w3.org/TR/vocab-dcat-2/, accessed on 2022-01-15.

[7] M. Doerr, G. Hiebel, Ø. Eide, Crmgeo: Linking the cidoc crm to geosparql through a spatiotemporal refinement, Int J Digit Libr 18 (2017) 271–279. doi:10.1007/s00799-016-0192-4.
[8] G. Hiebel, M. Doerr, Ø. Eide, Crmgeo: A spatiotemporal extension of cidoc-crm, International Journal on Digital Libraries 18 (2017) 271–279.
[9] C. Portele, OpenGIS® Geography Markup Language (GML) Encoding Standard. Version 3.2.1., Technical Report, Open Geospatial Consortium, 2007.
[10] John R. Herring, OGC 06-103r4 Simple feature access - Part 1: Common architecture, OpenGIS® Implementation Standard, Open Geospatial Consortium, 2011. URL: http://www.opengis.net/doc/is/sfa/1.2.1.
[11] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, S. Hagen, The GeoJSON Format, RFC 7946, 2016. URL: https://www.rfc-editor.org/info/rfc7946. doi:10.17487/RFC7946.
[12] D. Nolan, D. T. Lang, Keyhole markup language, in: XML and Web Technologies for Data Sciences with R, Springer, 2014, pp. 581–618.
[13] N. J. Car, T. Homburg, GeoSPARQL 1.1: Motivations, Details and Applications of the Decadal Update to the Most Important Geospatial LOD Standard, ISPRS International Journal of Geo-Information 11 (2022). URL: https://www.mdpi.com/2220-9964/11/2/117. doi:10.3390/ijgi11020117.
[14] M. B. J. Purss, R. Gibb, F. Samavati, P. Peterson, J. Ben, The ogc® discrete global grid system core standard: A framework for rapid geospatial integration, in: 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), 2016, pp. 3610–3613. doi:10.1109/IGARSS.2016.7729935.
[15] R. Gibb, Topic 21 - Discrete Global Grid Systems - Part 1 Core Reference system and Operations and Equal Area Earth Reference System, Report, Open Geospatial Consortium, 2021. URL: http://www.opengis.net/doc/AS/dggs/2.0.
[16] J. Bell, AusPIX Conceptual Framework for Data Integration based on DGGS Location, Report, Geoscience Australia, 2020. URL: https://doi.org/10.26186/140152. doi:10.26186/140152.
[17] R. Gibb, A. Raichev, M. Speth, The rHEALPix Discrete Global Grid System , Unpublished paper, Landcare Research New Zealand, 2016. doi:10.7931/J2D21VHM.
[18] M. J. Egenhofer, A formal definition of binary topological relationships, in: W. Litwin, H.-J. Schek (Eds.), Foundations of Data Organization and Algorithms, Springer Berlin Heidelberg, Berlin, Heidelberg, 1989, pp. 457–472.
[19] D. A. Randell, Z. Cui, A. G. Cohn, A spatial logic based on regions and connection., KR 92 (1992) 165–176.