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]

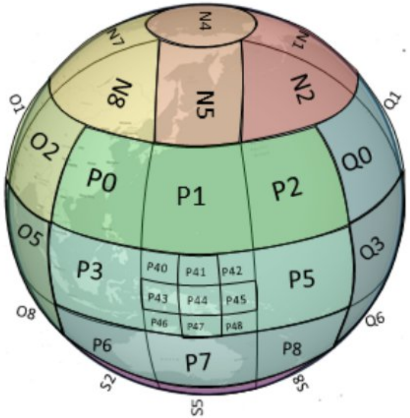
[a]i3mainz – Institute for Spatial Information & Surveying Technology, Mainz University of Applied Sciences, 55128 Mainz, Germany
[b]SURROUND Australia Pty. Ltd, New Acton, Canberra, ACT 2601, Australia
[c]Australian National University, Canberra, ACT 2600, Australia
[d]OpenLink Software Ltd., Croydon, Surrey, CR0 0XZ, United Kingdom
[e]Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje, 1000 Skopje, North Macedonia

Geospatial Semantic Web: Datatypes

- The Geospatial Semantic Web is able to represent different vector data serializations
  - WKT “POINT(1 1)”
  - GML “<…></…>”
- CoverageJSON among others allows for the representation of coverage data
- Missing: Support for Discrete Global Grid Systems (DGGS) in linked data
- One goal of the geospatial semantic web:
  - Enabling semantic descriptions of different geospatial representations
  - DGGS is likely to be adopted by an increasingly larger audience from industry to academia

What is a DGGS?

Think of a set of hierarchical chess boards



- Relationships between parent / child / neighbouring cells can be calculated based on identifiers
- Cells are infinitely divisible - in the example shown, each cell has 9 children - their identifiers are that of the parent, appended with {0..8}
- This allows basic spatial relationships such as parent / child to be calculated based on identifiers
- E.g. “P1” lies within “P”, as does “P3476241”
- In theory, this allows for standard spatial functions (contains, within etc.) to be implemented using deterministic relationships between (sets of) identifiers
- Basic drawing tool to assist in visualising how the rHEALPix DGGS functions: http://dggsdraw.surroundaustralia.com/

Why use a DGGS?

- In theory can utilize computing data structures better than traditional spatial data systems
- ability to represent both raster and vector spatial information in unified form

Why implement DGGS capability *in* a Triplestore?

- Geospatial analysis often involves non-spatial heterogenous feature data
- A triplestore provides an efficient means to query this data
- Use of a DGGS in a triplestore complements this by providing efficient spatial calculations

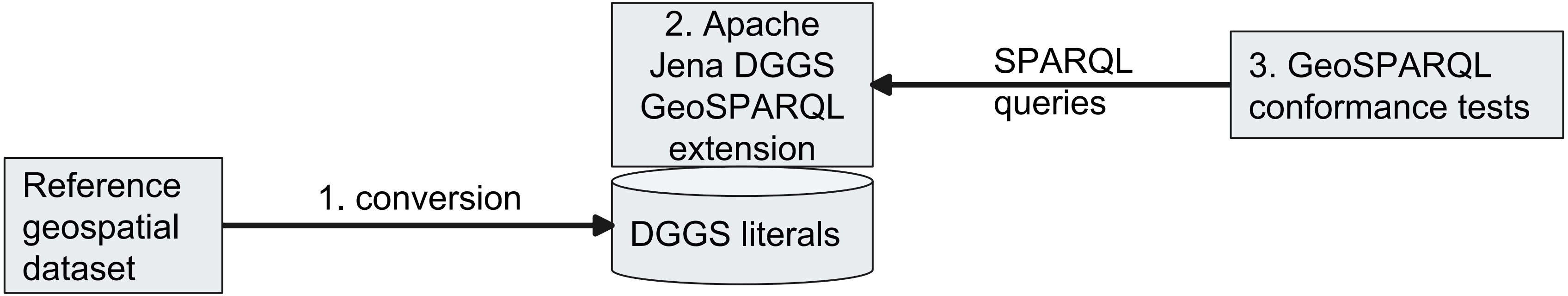
If successful, such an implementation would provide a building block for a powerful spatial analytics platform

Research goal

- See whether a DGGS aware triplestore can be implemented

  To evaluate this, we:

1. Converted a reference dataset of geometries to a DGGS
2. Implemented a set of standard spatial functions in Jena
3. Tested the conformance of the functions applied to the converted data

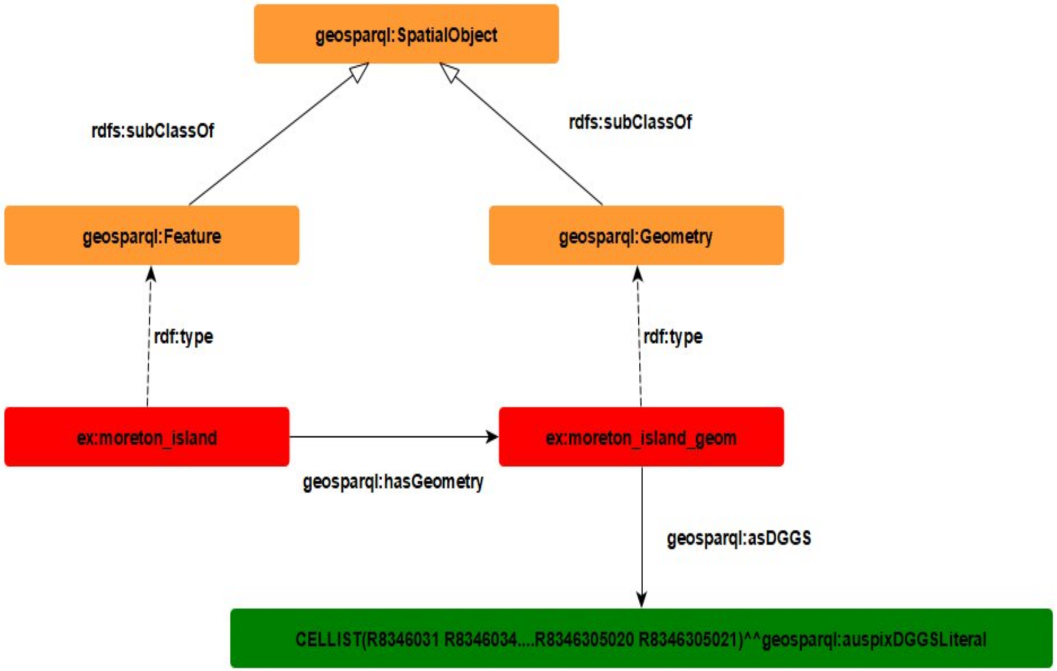


DGGS support in knowledge graphs

- DGGS geometries are described by
  - An identifier of the DGGS grid
  - A list of ordinates describing the area (grid cells) described in the respective DGGS expression
- DGGS identifiers:
  - Identifiers of the actual reference grid can be given in form of a URI
  - Similar to referencing of CRS systems
- Example:
  - ausPIX Grid
- We have defined a URI for our test grid ausPIX

Planned DGGS literal f

- GeoSPARQL 1.1 (in draft) will allow the definition of DGGS literals
- DGGS literals may be used to represent Geometries
- Representation of coverage types is anticipated in a later GeoSPARQL version
- DGGS literals could be one way of representing coverages in GeoSPARQL 1.2 onwards
- First implementation of DGGS literals in GeoSPARQL 1.1 is presented in this presentation



CELLIST(R8346031 R8346034....R8346305020 R8346305021)^^geosparql:auspixDGGSLiteral

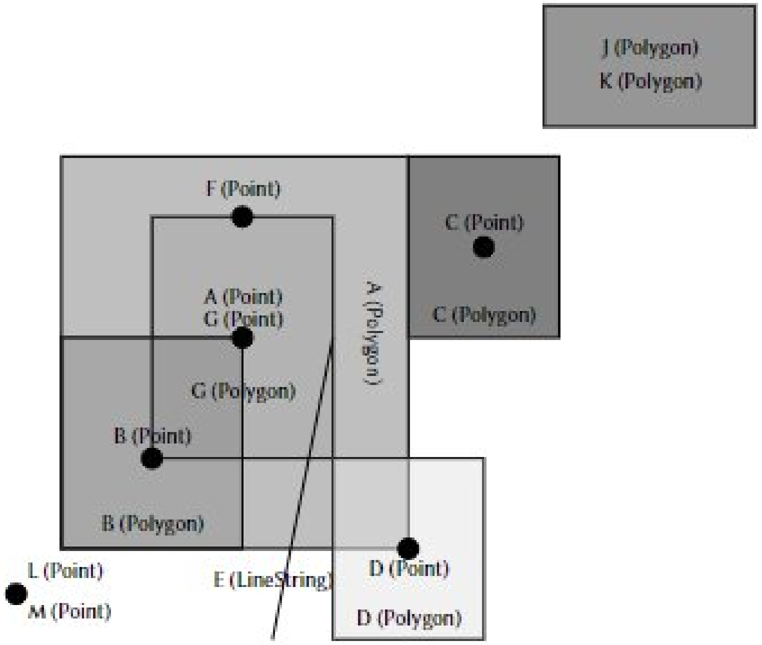
DGGS implementation

- Coded in Java
- Extended Jena's 'FunctionBase2' to allow querying with GeoSPARQL
- Two 'foundational' classes:
  - Cells
  - CellCollections (a collection of… DGGS cells!)
- Functions for low level CellCollection operations
  - e.g. addition, subtraction, equality, ordering, deduplication etc.
- Standard spatial operations:
  - Use set theory and low level functions
  - e.g. for Simple Feature Contains:
  - if A ∪ B = B and A != B, then B contains A

Implementation available here: https://github.com/surroundaustralia/jena-dggs-geosparql

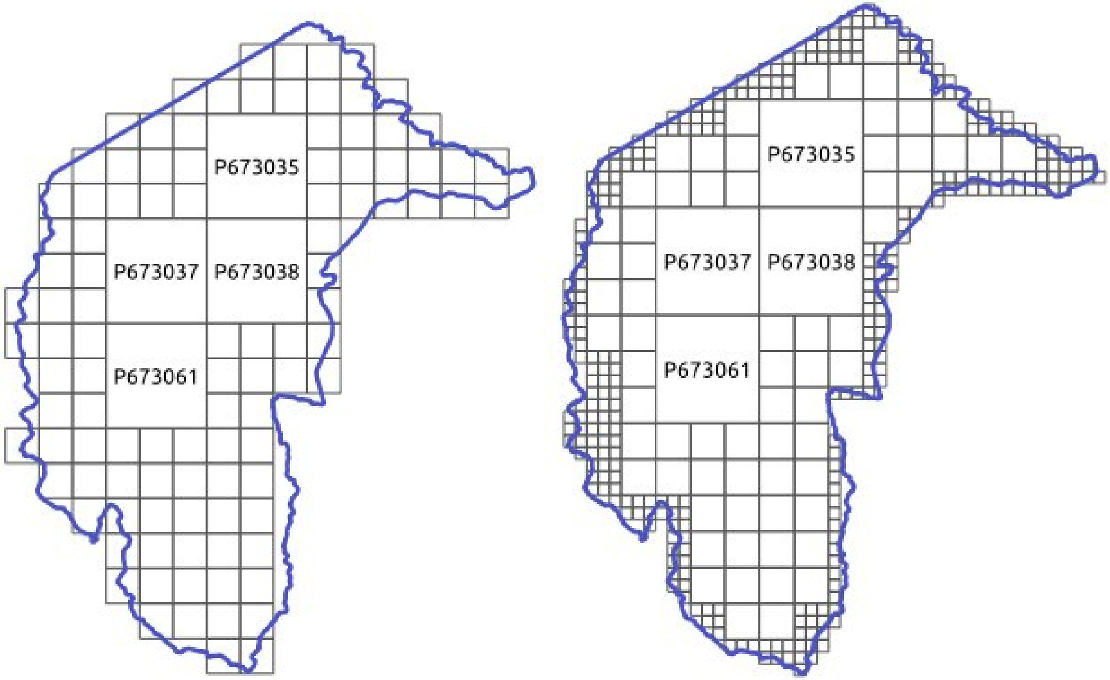
Reference dataset

- Reference dataset reflects the one used in the GeoSPARQL Compliance Benchmark
- Geometries A-M which reflect all Simple Feature relations
- For a compliance test:
    - Conversion of all geometries from WKT to DGGS
    - All geometries now include a WKT, GML and DGGS AUSPIX serialization
    - Tests can only be done on the AUSPIX grid
- Testing further grids requires a possibly more comprehensive reference dataset and test suite
- Conversion code added to rHEALPix DGGS library: https://github.com/manaakiwhenua/rhealpixdggs-py
- Web converter here: https://dggs.surroundaustralia.com/ (not available for public use currently)



Vector geometry to DGGS conversion

- Approximation of the vector geometry using the DGGS grid
- If the approximation is detailed enough:
  - We expect the same behavior as a vector geometry
- Which granularities are necessary for which application case of a DGGS?
- We test with two granularities:
  - AUSPIX Level 5
  - AUSPIX Level 10
- Both granularities are in common use in the AUSPIX grid



Benchmark details

- Extension of the GeoSPARQL Compliance Benchmark (Jovanovik et. al 2021)
- Benchmark idea:
  - Create test queries for all GeoSPARQL extensions (CORE, TOP, GEOEXT, GTOP, RDFSE, QRW)
  - Find out which test queries are concerned by DGGS literals
- CORE, TOP, RDFSE and QRW extension test queries do not test literal contents and are therefore not concerned by adding DGGS literals
- GEOEXT and GTOP extensions work with either one or two geometry literals as input
- Example:
  - geof:sfIntersects(ogc:geomLiteral geom1,ogc:geomLiteral geom2)
  - geof:convexHull(ogc:geomLiteral geom)

Which requirements need to be (re-)tested?

GeoSPARQL 1.0 requirements which are affected by the introduction of DGGS literals

- Requirement 19: Non-topological query functions (e.g. *geof:distance*)
- Requirement 20, 21: *geof:getSRID* and *geof:relate* functions
- Requirement 22: Simple Feature Relations (e.g. *geof:sfEquals*)
- Requirement 23: Egenhofer Relations (e.g. *geof:rcc8eq*)
- Requirement 24: Region Connection Calculus (e.g. *geof:rcc8eq*)

New requirements not yet introduced in GeoSPARQL 1.0 / planned for GeoSPARQL 1.1

- Requirement DGGS Literal: Definition, Empty Literal, CRS, Property *geo:asDGGS*

Extension of GeoSPARQL benchmark: Rationale

GeoSPARQL 1.0 requirements which are affected by the introduction of DGGS literals

- Requirements 22, 23 and 24 are relations between geometries
- Their implementations are similar and sometimes even overlap
- Non-topological query functions are necessary, but likely less often used in GeoSPARQL queries
- A first step should be:
  - Prove that a DGGS implementation can pass all functions of requirements 22
  - This in turn proves that a DGGS implementation of GeoSPARQL literals is feasible
  - Our work: A first implementation with DGGS-DGGS literals only
- Next step:
  - Extend the work to also include function signatures receiving e.g. WKT-DGGS literals
  - Crucial step here: Conversion of literal contents from Vector to DGGS and vice versa

GeoSPARQL Requirement 22

Query Function
Contains(geom1,geom2)
Within(geom1,geom2)
Overlaps(geom1,geom2)
Intersects(geom1,geom2)
Touches(geom1,geom2)
Crosses(geom1,geom2)
Equals(geom1,geom2)
Disjoint(geom1,geom2)

GeoSPARQL Compliance DGGS Benchmark: Query templates

SELECT (xsd:boolean(?sfTouches) as ?touches)

WHERE { my:A geo:hasDefaultGeometry ?aGeom .

?aGeom **%%literalrel1%% ?a%%literal1%%** .

my:C geo:hasDefaultGeometry ?cGeom .

?**cGeom %%literalrel2%% ?c%%literal1%%** .

BIND (geof:sfTouches(**?a%%literal1%%, ?c%%literal2%%**) as ?sfTouches)}

GeoSPARQL Compliance DGGS Benchmark: Answers

- Creation of query answer templates for GeoSPARQL 1.0 test queries:
  - Only necessary for functions that return literals
  - Not necessary for SF, Egenhofer, RCC8 which return boolean functions
- 206 queries in theoriginal GeoSPARQL Compliance Benchmark
- 353 queries in GeoSPARQL DGGS Compliance Benchmark

https://github.com/i3mainz/GeoSPARQLBenchmark/tree/dggs/src/main/resources/geosparql10_dggs_compliance

GeoSPARQL Compliance DGGS Benchmark: DGGS Compliance Score calculation

- Scoring approach stays the same for GeoSPARQL 1.0 benchmarking
- More requirements need to be tested because of DGGS literals being added
  - In total 353 test queries as compared to 206 test queries for GeoSPARQL 1.0 w/o DGGS
- Each requirement is weighed 1/34, as 34 requirements need to be tested
- If a requirement requires 8 functions, each function gets a weight of 1/34*1/8
- If a function may receive two literals as input each function call gets a weight of 1/34*1/8*1/12 as 3 literal types and their permutations need to be tested

GeoSPARQL Compliance DGGS Benchmark: Implementation

- Original GeoSPARQL Compliance Benchmark was executable on the HOBBIT benchmarking platform and as an additional Python Script
- The HOBBIT benchmarking platform was envisioned by us to test final standards only
- Therefore we offer a Python script which sends test queries to a DGGS-enabled triple store and collects the results
    - Scoring file
    - Results of each query execution
    - Errorlog during query execution

https://github.com/i3mainz/GeoSPARQLBenchmark/blob/dggs/benchmark_geosparql.py

GeoSPARQL Compliance DGGS Benchmark: Results

- Overall Compliance Score: 54.5%
- For comparison: GeoSPARQL 1.0 enabled triple stores scored between 50% and 70% in the initial GeoSPARQL compliance benchmark
- Compliance Score only for requirement 22:
    - 62.5% and only on DGGS-DGGS literal combinations
- On our given data: No difference between Level 5 and Level 10 granulatity in the AUSPIX grid
- Further scenarios on more grid types need to be tested though

Results for Requirement 22

Relationship	Result	Comment
Contains	Pass	
Within	Pass	
Overlaps	Pass	
Intersects	Fail - may pass now	
Touches	Fail - should pass now	
Crosses	Fail	Not implemented
Equals	Pass	
Disjoint	Pass	

Conclusions

Our research contributed:

- A study on the feasibility of interoperability of DGGS literals and GeoSPARQL
- One implementation of this interoperatibility assumption
- A first application of DGGS literals in a GeoSPARQL knowledge graph
- A definition of a Compliance Benchmark which tests GeoSPARQL DGGS compatibility
- Usage of this DGGS test to verify the implementation for at least one functioning requirement

We this we believe:

- We layed the foundation for testing performance and compliance of DGGS-based triple store implementations
- We contributed a partial reference implementation for GeoSPARQL 1.1 based triple stores
- We also contributed towards the creation of a GeoSPARQL 1.1 benchmark

Future work

Implementation work:

- Implement spatial functions across other relation families (Egenhofer, RCC8)
- Implement non-topological query functions with DGGS compatibility
- Track changes in GeoSPARQL 1.1 which need DGGS support
- Benchmark performance and compare to that using traditional tools and CRSes

Ultimately we want to answer the question:

- Is a DGGS-based triple store faster than a vector literal based triple store?
- What is the performance in a mixed literal environment?
- How to design spatial indices for both geometry representations and which impact do they have on the result?

Thank you

Thank you very much for your attention