GeoSPARQL 2.0 - Change Requests for OGC Meeting

Timo Homburg

timo.homburg@hs-mainz.de
Mainz University Of Applied Sciences
Lucy-Hillebrand-Strasse 2
55128 Mainz, Germany

Keywords: GeoSPARQL, Raster data, Query capabilities

1 Geospatial Literals

GeoSPARQL currently only supports two Literal types:

– WKT Literal

```
"POLYGON((-77.089005 38.913574,-77.029953 38.913574,
-77.029953 38.886321,-77.089005 38.886321,
-77.089005 38.913574))"^^geo:wktLiteral
```

– GML Literal

```
"<gml:Point srsName=\"http://www.opengis.net/def/crs/OGC/1.3/CRS84\"
 xmlns:gml=\"http://www.opengis.net/gml\">
<gml:posList srsDimension=\"2\">-83.38 33.95</gml:posList>
</gml:Point>"^^ogc:GMLLiteral
```

Considering the large heterogeneity of geospatial data formats, the support of more geospatial data literals should be considered.

1.1 Change Request

The addition and standardization of more literal types. To me the following geospatial literal types should be included with high priority:

– GeoHash Literal[1]

```
"gbsuv"^^geo:geoHashLiteral
```

– GeoJSON Literal[2]
  - Why? - Very common webstandard for maps

```
{\"type\":\"LineString\",
\"coordinates\":[[39.046368900000004,22.2237116],
[39.0462247,22.223842]]}^^geo:geoJSONLiteral
```

– GPX Literal[3]
  - Why? - Useful for capturing and working with GPS data

```
"<gpx version=\"1.0\"><name>Example gpx</name>  <trk>
    <name>Example gpx</name>
    <trkseg>
    <trkpt lat=\"40.133485900000004\" lon=\"22.5475202\"/>
    <trkpt lat=\"40.133099800000004\" lon=\"22.547705500000003\"/>
    </trkseg></trk></gpx>"^^geo:gpxLiteral
```

[1] http://geohash.org
[2] https://geojson.org
[3] https://www.topografix.com/gpx.asp

– KML Literal[4]

```
"<linestring>
<coordinates>23.5530286,38.0473921 23.5554701,38.0484895</coordinates>
</linestring>"^^geo:kmlLiteral
```

– TWKB Literal[5]

```
"\x02000202020808"^^geo:twkbLiteral
```

– (Hex)(E)WKB Literal[6]
  - Why? - Standard format in many other relational database systems

More data literals to be included could be:

– Polyshape(Polyline) Literal[7]

```
"1gwpzF{gr_Da@M"^^geo:polyshapeLiteral
```

– DXF Literal[8]
– Geobuf Literal[9]

```
"GAAiEAoOCgwIBBoIAAAAAgIAAAE="^^geo:geobufLiteral
```

– GeoURI Literal[10]

```
"geo:39.19972000122019,22.763096060395224;crs=EPSG:4326"
^^geo:geoURILiteral
```

– OSM/XML Literal[11]
– SVG Literal[12]
– TopoJSON Literal[13]

```
"{"type": "Topology",
"objects": { "example": {
"type": "GeometryCollection",
"geometries": [{ "type": "Point",
"properties": {"prop0": "value0"},
"coordinates": [102, 0.5]},
{
"type": "LineString",
"properties": {"prop0": "value0","prop1": 0},
```

[4] https://developers.google.com/kml/documentation/kmlreference?csw=1
[5] https://github.com/TWKB/Specification/blob/master/twkb.md
[6] https://mariadb.com/kb/en/library/well-known-binary-wkb-format/
[7] https://developers.google.com/maps/documentation/utilities/polylinealgorithm
[8] http://www.crlf.de/Verlag/DXF-intern/DXF-intern.html
[9] https://github.com/mapbox/geobuf
[10] https://tools.ietf.org/html/rfc5870
[11] https://wiki.openstreetmap.org/wiki/OSM_XML
[12] https://www.w3.org/Graphics/SVG/
[13] https://github.com/topojson/topojson

```
"arcs": [0]
},
{"type": "Polygon",
"properties": {"prop0": "value0","prop1": {"this": "that"}},
"arcs": [[-2]]}]}},
"arcs": [
[[102, 0], [103, 1], [104, 0], [105, 1]],
[[100, 0], [101, 0], [101, 1], [100, 1], [100, 0]]
]
}"^^geo:topoJSONLiteral
```

– X3D Literal[14]

Ontology changes The GeoSPARQL ontology would need to be extended by the described literal types. An extended ontology can be found in https://github.com/i3mainz/geosparql2.0

1.2 Implementation

Implementations of all aforementioned literals exist in the following projects:

– postgis-jena[15] - Extension for Apache Jena
– rdf4j-postgis[16] - Extension for RDF4j

[14] https://www.web3d.org/x3d/what-x3d
[15] https://github.com/i3mainz/postgis-jena
[16] https://github.com/i3mainz/rdf4j-postgis

2 Geometry Exporters

GeoSPARQL does not provide any means to convert geospatial data to other formats despite a huge heterogeneity of geospatial data. Being adopted in a wide variety of other geospatial database system like POSTGIS, such functionality is useful for a variety of usecases in which geodata in different formats is required and should not have to be converted on the client side.

2.1 Change Request

The addition and standardization of geometry converter and exporter functions for all data literals which are defined in the new GeoSPARQL standard. Suggestions can be found in the following list:

- ST_AsGeoJSON(geom1: ogc:GeomLiteral): xsd:string
- ST_AsGeoRSS(geom1: ogc:GeomLiteral): xsd:string
- ST_AsGeoURI(geom1: ogc:GeomLiteral): xsd:string
- ST_AsGML(geom1: ogc:GeomLiteral): xsd:string
- ST_AsKML(geom1: ogc:GeomLiteral): xsd:string
- ST_AsWKT(geom1: ogc:GeomLiteral): xsd:string
- ST_AsWKB(geom1: ogc:GeomLiteral): xsd:string

The aforementioned functions can be given as aggregation functions, property functions or as filter functions for comparisons.

3 Geometry Transformations

The GeoSPARQL standard includes geometric relations according to the RCC8 region connection calculus. It is therefore possible to relate vector geometries to each other, but not to transform geometries before relating them. The demand for this can be seen in a variety of geospatial applications like segmenting LineStrings to identify street segments, reducing the precision of geometries in order to compare geometries with the same precision or to rescale geometries for a better matching.

3.1 Change Request

The addition of geometry manipulations capabilities as relationships for reasoning and filter functions.
Examples:

- ST_Split: Splits a LineString in LineString segments
- Simplification: Simplification of geometries using e.g. Douglas-Peucker algorithm by preserving or optionally non-preserving the geometries' topology
- ST_Normalize: Normalization of a geometry
- ST_PrecisionReducer: Adjusting/Reducing the precision of geometries
- ST_Scale: Scaling a vector geometry to different sizes
- ST_Translate: Geometry translation to other areas
- ST_TransScale: Scaling a vector geometry to different sizes and translating it to a new position

3.2 Implementation

Implementations of all aforementioned literals exist in the following projects:

- postgis-jena[17] - Extension for Apache Jena
- rdf4j-postgis[18] - Extension for RDF4j

[17] https://github.com/i3mainz/postgis-jena
[18] https://github.com/i3mainz/rdf4j-postgis

4 Geometry Attributes

Geometries in GeoSPARQL 1.0 can be described with a variety of attributes. For the next version of GeoSPARQL, these could be extended by further attributes which could be inferred from the geometry itself.

4.1 Change Request

The addition of geometric attributes and functions for filters could be as follows:

Geometry Attributes

- ST_Area(geom1: ogc:GeomLiteral): xsd:double
- ST_Centroid(geom1: ogc:GeomLiteral): ogc:GeomLiteral
- ST_GeometryType(geom1: ogc:GeomLiteral): xsd:string
- ST_MinimumClearance(geom1: ogc:GeomLiteral): xsd:double
- ST_MinimumDiameter(geom1: ogc:GeomLiteral): xsd:double
- ST_MinimumBoundingRadius(geom1: ogc:GeomLiteral): xsd:double
- ST_MinimumDiameter(geom1: ogc:GeomLiteral): xsd:double
- ST_Length(geom1: ogc:GeomLiteral): xsd:double
- ST_PointN(geom1: ogc:GeomLiteral): ogc:GeomLiteral
- ST_GeometryN(geom1: ogc:GeomLiteral): ogc:GeomLiteral
- ST_NumPoints(geom1: ogc:GeomLiteral): xsd:integer
- ST_NumGeometries(geom1: ogc:GeomLiteral): xsd:integer
- ST_X,ST_Y,ST_Z,ST_M,ST_Min{X,Y,Z,M},ST_Max{X,Y,Z,M}(geom1: ogc:GeomLiteral): xsd:double

Geometry Boundaries

- ST_ConcaveHull(geom1: ogc:GeomLiteral): ogc:GeomLiteral
- ST_MinimumBoundingCircle(geom1: ogc:GeomLiteral): ogc:GeomLiteral
- ST_MinimumBoundingCircleCenter(geom1: ogc:GeomLiteral): ogc:GeomLiteral
- ST_MinimumClearanceLine(geom1: ogc:GeomLiteral): ogc:GeomLiteral
- ST_MinimumDiameterLine(geom1: ogc:GeomLiteral): ogc:GeomLiteral
- ST_MinimumRectangle(geom1: ogc:GeomLiteral): ogc:GeomLiteral

5 Geometry Editions

Geometries in GeoSPARQL cannot be modified in-query i.e. to add points or delete points from geometries. This functionality can be useful in many occasions.

5.1 Change Request

The addition of geometric edition functions as follows:

- ST_AddPoint
- ST_AddZ
- ST_AddMeasure
- ST_RemovePoints
- ST_RemoveRepeatedPoints

6 Geometry Relation

GeoSPARQL defines geometric relations of geospatial data by definition of the RCC8 region connection calculus. However, many other functions to relate geometries are common and in use in other relational database management system.

6.1 Change Request

The addition of geometric relation filter functions/relations as follows:

Geometry Similarity Metrics

– ST_AreaSimilarity
– ST_HausdorffDistance
– ST_FrechetDistance
– ST_IntersectionPercentage

Geometry Distance Metrics and Distance Lines

– ST_DistanceSphere
– ST_MaxDistance
– ST_CentroidDistance
– ST_LongestLine
– ST_ShortestLine

7 Support for 3D geometries

GeoSPARQL 1.0 allows to define 3D geometries, as the task of defining geometries is assigned to the formats GML and WKT which both allow the definition of XYZ coordinates. However, GeoSPARQL 1.0 lacks the necessary means of relating 3D geometries to each other. 3D geometries become increasingly important as standards such as CityGML or KMZ put more emphasis on modeling 3D geometries on traditional 3D map spaces.

7.1 Change Request

The addition of 3D-aware geometry relation functions. This includes firstly all functions of the RCC8 calculus to support 3D functions, i.e. the intersection between 3D geometries does not necessarily equal the intersection in a 2D space. In addition, the following functions calculate a different result compared to their 2D counterpart:

- ST_Area3D
- ST_DWithin3D
- ST_Difference3D
- ST_Distance3D
- ST_Intersection3D
- ST_Length3D
- ST_LineLength3D
- ST_LongestLine3D
- ST_MaxDistance3D
- ST_Perimeter3D
- ST_ShortestLine3D

The question whether to define new function names which are specifically defined for 3D applications or if the function definitions of GeoSPARQL 1.0 should just be applied to accomodate 3D data can be discussed.

Ontology changes Support for 3D geometries needs to be reflected in the GeoSPARQL ontology. New classes for 3D geometries need to be defined. A suggestion for the integration for 3D geometries can be seen in the W3C geom ontology https://github.com/w3c-geom-cg/geom which can be integrated into the GeoSPARQL ontology to represent 3D geometries. Another alternative could be the X3D ontology developed by a special interest group: https://www.web3d.org/working-groups/x3d-semantic-web/charter

8 Support for Measurements

Another common occurance is the inclusion of per-point measurements for coordinates, so-called XYZM coordinates whereas M stands for a measurement which has been conducted at the very point. While the serialization format can usually support M coordinates, GeoSPARQL lacks support for the coordinates in terms of functions. Usually, M coordinates define a measurement such as kilometers, which should be preserved even after a geometry has been transformed, i.e. simplified, scaled or moved. Functions to support getting the M coordinate or to filter geometries based on an M coordinate value could be very useful to be supported.

8.1 Change Request

Adding functions and relations to deal with XYZM and XYM coordinates.

- ST_M: Gets the M coordinate of a point
- ST_FilterByM: Filters a geometry by M coordinate

9 Support for Timepoint Coordinates

Another common occurance is the inclusion of per-point measurements for coordinates, so-called XY(Z)T coordinates whereas T stands for a time measurement (xsd:dateTime) which has been conducted at the very point. A common usecase for such a T coordinate is the tracking and saving of GPS tracks

9.1 Change Request

Adding functions and relations to deal with XYZT,XYT,XYMT coordinates.

- ST_T():xsd:dateTime, ST_MaxT():xsd:dateTime,ST_MinT():xsd:dateTime
- ST_FilterByT(geometry:ogc:GeomLiteral,date: xsd:dateTime,tolerance)ogc:geomLiteral
- ST_PartOfGeometryBefore(geometry: ogc:GeomLiteral,date: xsd:dateTime,tolerance: xsd:double): ogc:GeomLiteral
- ST_PartOfGeometryAfter(geometry: ogc:GeomLiteral,date: xsd:dateTime,tolerance: xsd:double): ogc:GeomLiteral
- ST_PartOfGeometryAt(geometry: ogc:GeomLiteral,date: xsd:dateTime,tolerance: xsd:double): ogc:GeomLiteral

10 Raster data support

Currently, GeoSPARQL is a standard for handling vector data. However, many geospatial datasets are actually given as raster data sets. This change request therefore introduces ideas on how to integrate raster data in the semantic web. The implementation of how this is done can be left to manufacturers of the respective triple stores, as many different variations exist (storage inside the triple store, externally, textual vs. non-textual, hash vs. real data, tiled vs. non-tiled)

10.1 Change Request R1: Raster data literals

Definition of raster literals of the following formats:

- GeoTIFF Literal
- CoverageJSON Literal
- GMLCOV Literal
- RasterWKB Literal

The implementation of said literals is to be discussed as several methods to store raster data can be considered:

- Storage of raster images as literals: A possibility, but may require query optimizations in the triple store as the access of a lot of raster data images might decrease the triple store performance significantly
- Storage of raster data links as literals: Hashes or links to the actual raster data literals can be stored in the triple store. Raster images will be referenced by a URI outside the triple store and only loaded on-demand when a raster function of the triple store actually requires raster data to be loaded. The image is then retrieved, possibly modified and returned

10.2 Change Request R2: Raster data functions

A variety of raster data functions and/or relations to represent:

- Raster (map) algebra operations according to [1] (e.g. raster add raster)
- Raster relation functions, defining RCC8 calculus functions to support raster data
- Vectorization function
- Rasterization function
- Raster modification functions such as:
  - ST_Slope
  - ST_TPI
  - ST_TRI
  - ST_Flip
  - ST_Curvature
- Raster band management:
  - ST_AddBand
  - ST_RemoveBand
- Raster relation functions to vector data (RCC8)

10.3 Change Request R3: Ontology changes

Changes to the GeoSPARQL ontology include:

– Creation of Coverage and GridCoverage classes to represent Coverage data types
– A vocabulary to represent raster data and its meanings

A vocabulary to represent raster data meaning would need to include the possibility to relate every raster bands color configuration with a semantic description. The color configuration can be interpreted in a variety of ranges, e.g. a blue color can be classified as shades of blue indicating a flood altitude. The distinction of flood levels, i.e. color shades can be defined as metadata in the given ontology and then utilized for interpretation and/or vectorization purposes.

10.4 Implementation

A suggestion for integrating raster data using a common vocabulary is given in https://github.com/i3mainz/geosparql2.0. A raster example with interpretations is provided in the same repository.

References

1. Tomlin, C.D.: Map algebra: one perspective. Landscape and Urban Planning 30(1-2), 3–12 (1994) 10.2