GMZ: A GML COMPRESSION MODEL FOR WEBGIS

Ayush Khandelwal [a], Dr. K. S. Rajan [a]
[a] Lab for Spatial Informatics, International Institute of Information Technology, Hyderabad, India –
ayush.khandelwal@research.iiit.ac.in, rajan@iiit.ac.in

Commission IV, WG IV/4

KEY WORDS: GIS, Lossless compression, GML, Simple features profile, WFS

ABSTRACT:

Geography markup language (GML) is an XML specification for expressing geographical features. Defined by Open Geospatial Consortium (OGC), it is widely used for storage and transmission of maps over the Internet. XML schemas provide the convenience to define custom features profiles in GML for specific needs as seen in widely popular cityGML, simple features profile, coverage, etc. Simple features profile (SFP) is a simpler subset of GML profile with support for point, line and polygon geometries. SFP has been constructed to make sure it covers most commonly used GML geometries. Web Feature Service (WFS) serves query results in SFP by default. But it falls short of being an ideal choice due to its high verbosity and size-heavy nature, which provides immense scope for compression. GMZ is a lossless compression model developed to work for SFP compliant GML files. Our experiments indicate GMZ achieves reasonably good compression ratios and can be useful in WebGIS based applications.

1. INTRODUCTION

1.1 Motivation

GML proves to be a great modelling language for geospatial web due to its advent from XML, a de-facto standard for web, which has the advantages of being human readable, browser friendly, extensible, editable and queryable. It also comes bundled with the two unavoidable drawbacks of XML – verbosity and being text based. XML, as we know is highly verbose in its nature. It is stored as Unicode text forbidding GML to leverage storing coordinates (which make up significant content of a GML document) as floating-point numbers or some combination of integers that can potentially take significantly less space compared to storing coordinates as strings. This bloats the size of GML documents and makes it fall short of being the most favourable choice for current usage patterns that are mostly Internet based. Consequently, we are forced to think of ways to make GML more efficient without the need to do away with the advantages that it comes with.

The rapidly multiplying Internet users put a lot of pressure on Internet services. Smart phones provide enough processing power to users, making mobile GIS feasible. But storage and bandwidth still suffer. Compression is an obvious choice in this direction. With conventional text compression algorithms such as LZ77, Huffman coding, Burrows-Wheeler transform, PPM, etc. already in place, we are inclined to use them everywhere. But these compression algorithms are unbiased towards structure that exists in data and therefore, cannot leverage this towards achieving better compression ratios. Consquently, they produce inferior compression ratios compared to models aimed at XML compression. GML is even more well-structured and predictable indicating that developing GML specific compression models to get better compression ratios make sense. However, we refrain from developing a compression model for the whole of GML. Rather, this work is restricted to anything that falls into GML's Simple Features Profile (GML simple features profile, 2011) because of its high use on the internet through WFS. It's a good first step at realizing what future compression models should be able to achieve.

1.2 Literature survey

Majority of current work on GML compression has focused mostly on just the storage efficiency of data. GPress (Guan and Zhou, 2007) and some other compression models (LI et al, 2008; Weiand Guan, 2010) are based on three principles: separating spatial data, attribute data and file structure and storing in different containers; applying delta encoding on floating point coordinates and finding semantic similarity between attributes. Based on the same idea, GQComp (Dai et al, 2009) uses a custom encoding for coordinates, makes provision for spatial and attribute data querying through the combination of feature-structure tree and R* tree spatial indexing; and achieves good compression. Another compression technique called Gtree (Harshita and Rajan, 2010; Harshita, 2013) restricted to work for only polygon data, uses a tree based structure for managing the coordinate data.

One common issue with most techniques is that they use delta encoding for coordinate compression which leads to loss of precision when calculating the delta. This can lead to errors, slivers, disjoint ends that are highly undesirable. GQComp uses a lossless custom encoding for coordinates and so far, produces best compression ratios. But its query subsystem doesn't make sense as loading the entire data in-memory puts too much pressure on already ladden modern day systems. It is equivalent to decompressing the entire document and then performing query on it.

Our technique is loosely based on Gtree, specifically designed to work with SFP. We are using a custom encoding which is a mix of delta encoding and dictionary encoding to compress coordinate data. Apart from the fact that it's lossless and produces good compression ratios, our model has provision for query in compressed state. Though the query subsystem is still under development and out of scope of this paper, we would like to emphasize that our model provides access to individual features by decompressing them in isolation. This is an essential requirement for querying and in favour of our claim.

1.3 Dataset

Due to the unavailability of compiled SFP compliant GML 3 datasets, it has largely been prepared by making GML files SFP compliant or by converting shapefiles into SFP compliant GML files. QGIS has been used for the conversion process. We have prepared GML files for 2 countries – India and USA. The India files were downloaded from mapcruzin.com, a provider of region wise shapefiles, and then converted to GML. The USA GML files were downloaded from data.gov, the data portal of the government of US, and then made SFP compliant. The dataset is combination of point, line and polygon GML files. The file size ranges from 20 MB to around 1 GB with most files under 100 MB.

2. MAIN BODY

2.1 Understanding the data

GML is based on an abstract model of geography given by OGC which defines the world in terms of features where each feature has a set of properties. Properties can be grouped into two categories – spatial property, which is the geometry that stores the coordinate data of the feature (point, line or polygon) and non-spatial property, which is the non-spatial description of the feature. A feature is the smallest meaningful unit of GML. It can have any number of spatial and non-spatial properties. Referring to the GML snippet below, feature Road has 1 spatial property and 3 non-spatial properties. All features with the same name compulsorily have the same set of properties. Since, spatial properties are fairly complex compared to non-spatial properties, they have their own GML substructure which is identified using the gml namespace. They are described using a subset of geometry types such as Point, LineString, Curve, Polygon, Surface, etc. The usage of these geometry types is explained in detail in the SFP specification document. Non-spatial properties are restricted to have any structure, a notion imposed by SFP owing to the fact that databases are not designed to handle nested data.

```
<gml:featureMembers>
  <ogr:Road>
    <ogr:GEOM>
      <gml:lineString>
        <gml:posList>34.987644195556605 -
105.217300415038963
          34.987632751464808 -105.217117309570227
          34.987617492675746 -105.216644287109261
        </gml:posList>
      </gml:lineString>
    </ogr:geom>
    <ogr:ID>87687</ogr:ID>
    <ogr:NAME>I 40</ogr:NAME>
    <ogr:TYPE>Interstate</ogr:TYPE>
  </ogr:Road>
  <ogr:Road>
    <ogr:GEOM>
      <gml:lineString>
        <gml:posList>36.809299468994105 -
107.915138244628807
          36.808723449707003 -107.91530609130848
          36.808464050292926 -107.915161132812386
        </gml:posList>
      </gml:lineString>
    </ogr:geom>
    <ogr:ID>87688</ogr:ID>
    <ogr:NAME>NM 575</ogr:NAME>
    <ogr:TYPE>State Highway</ogr:TYPE>
  </ogr:Road>
</gml:featureMembers>
```

Figure 1. GML snippet

This simplification of data into segments - spatial, non-spatial and XML tree structure - groups symantically similar data, which inturn facilitates almost isolated and targeted compression on these data segments. Since GML is predominantly coordinate data like any map data, our focus will be on coordinate data compression with provision for non-spatial data compression and XML structure encoding. Here is a list of characteristics of GML based on which our compression model is based. These characteristics will be referenced in the next section:

2.1.1 Duplication: Coordinates can be duplicated when adjacent polygons share boundaries, when linestrings share end-points or random duplicity among features.

2.1.2 Adjacency: Difference between adjacent coordinates can be very less when data is closely packed which is often the case in polygons and linestrings.

2.1.3 Text-based: Each digit of a coordinate is stored as a byte, which ultimately bloats the size of data.

2.2 Compression Model

The algorithm is a 2-step process and involves 2 passes over the document. The steps are explained in detail:

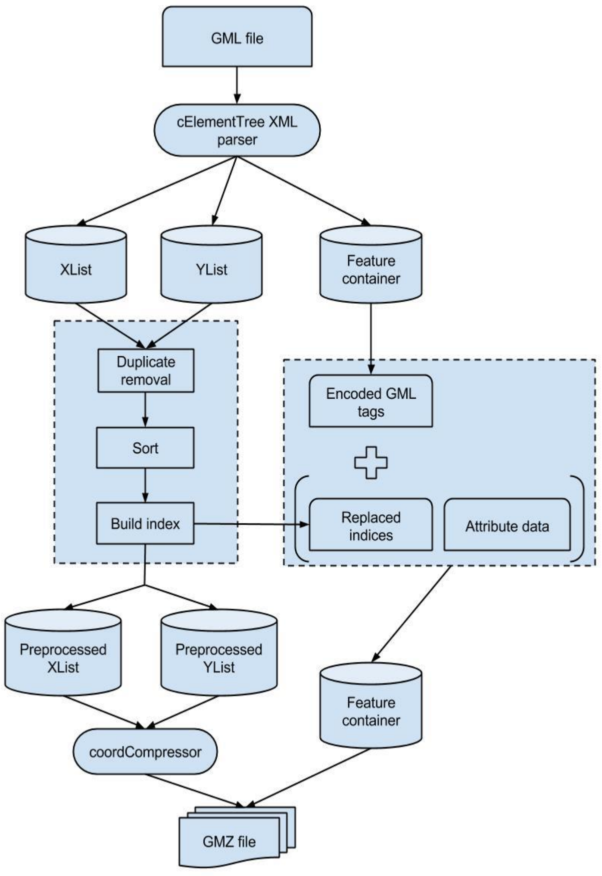

Figure 2. Compression model pipeline

2.2.1 First pass (coordinate compression): In the first pass, the coordinate data is separated from the GML tree as *XList* to store X-coordinates and *YList* to store Y-coordinates. The following steps are performed on each List separately in order – duplicate removal, sorting and index building.

Duplicate removal is done to eliminate data redundancy that we identified in point 2.1.1. This is followed by sorting. The intuition behind sorting can be understood in relation with delta encoding. Delta encoding is a way of storing or transmitting data in the form of differences (deltas) between sequential data rather than complete files; more generally this is known as data differencing (Delta encoding). It is well suited to work with sorted data because sorting brings coordinates with least difference adjacent to each other, which produces minimum values of delta. This is in conjunction with point 2.1.2. The next step is creation of coordinate dictionaries, *XMap* and *YMap*, used for storing the coordinate and its reference as its key-value pair. This reference is just the array index of *XList* for X-coordinate or *YList* for Y-coordinate. These indices will be used in place of the original coordinate during the structure compression step.

The coordinate compression function, *coordCompressor* takes a coordinate list at a time and applies a custom encoding on it. A coordinate is broken down into its integral and decimal parts. The integral parts of successive coordinates exhibit very high repeatability, and therefore, will be stored almost negligibly, only when a new integral part is encountered. Integral part ranges from -90 to 90 for latitude and -180 to 180 for longitude. Hence, it can be stored by a signed 2-byte *short int* datatype. On the other hand, the decimal parts of successive coordinates exhibit high proximity, i.e., mathematical difference between consecutive decimal parts is significantly smaller compared to the decimal parts themselves. The delta of the two consecutive decimal parts is what is stored. It can be stored using any of the unsigned integer datatypes - *byte, short int, int* or *long long int* - depending on its size. This is different from many compression models, which directly apply delta compression on the entire coordinate leading to lossy compression. There is also some metadata that needs to go along with each coordinate – flag to indicate if a new integral part is encountered, length of the decimal part and datatype used for storing delta. We have managed to store all this metadata in just one byte. These operations are applied on each coordinate and help solve the issue identified in point 2.1.3.

2.2.2 Second pass (structure and attribute compression): In the second pass, the structure tags are replaced by the corresponding encodings and attribute data is compressed. The properties of a feature have their own custom namespaces and tags. Nonetheless, they remain the same for all features sharing the same name. Therefore, we store these property names just once per unique feature name. We traverse the feature tree and identify if the property is spatial or non-spatial.

Spatial properties are made up of one of the 12 geometry types and 10 subtypes provided in SFP. These geometry types are tightly structured due to strict usage specification. These 22 tags have been given a value from 0-21. The tags are replaced by their encoding while traversing the spatial property. Two tags – *pos* and *posList* – are the innermost tags in a geometry tree structure and enclose coordinates for a feature's geometry. We now find the references or indices of these coordinates from *XMap* and *YMap* and replace them with their indices. The value of these indices will be of the order of millions when we have let's say, millions of coordinates. To prevent storing such high values, we store just the delta of indices of successive coordinates. The idea is that coordinates in a feature tend to be very close numerically. Therefore, their position in coordinate List will be close leading to a small value of delta. In our experience, we could notice that this delta could be stored in single signed byte most often.

0 - Point	*13 – Arc*
1 – MultiPoint	*14 – Circle*
2 – LineString	*15 – CircleByCenterPoint*
3 – MultiLineString	*16 - LineStringSegment*
4 – Curve	*17 – LinearRing*
5 – MultiCurve	*18 – Ring*
6 – Polygon	*19 – exterior*
7 – MultiPolygon	*20 - interior*
8 – Surface	*21 – pos*
9 – MultiSurface	*22 – posList*
10 – Geometry	
11 – MultiGeometry	

Figure 3. GML geometry tags available in SFP

Relative to the amount of coordinates, attributes often make up a small part of GML. Developing sophisticated compression technique at the cost of increased complexity is not worthwhile considering the change in overall compression ratios contributed by attribute compression. We start with identifying the type – integer, float or string. Integer and float types are stored as integers and floats, respectively. String types show some amount of duplication across features. They are evaluated for feasibility of dictionary encoding and stored accordingly.

2.3 Software tools

In a typical client-server architecture such as WFS, the server is the producer and provider of GML, and therefore, needs compression support primarily. On the other hand, the client (web browser) is the consumer of GML, and therefore, needs decompression and visualization support. Depending on the need, we have created separate tools for server and client.

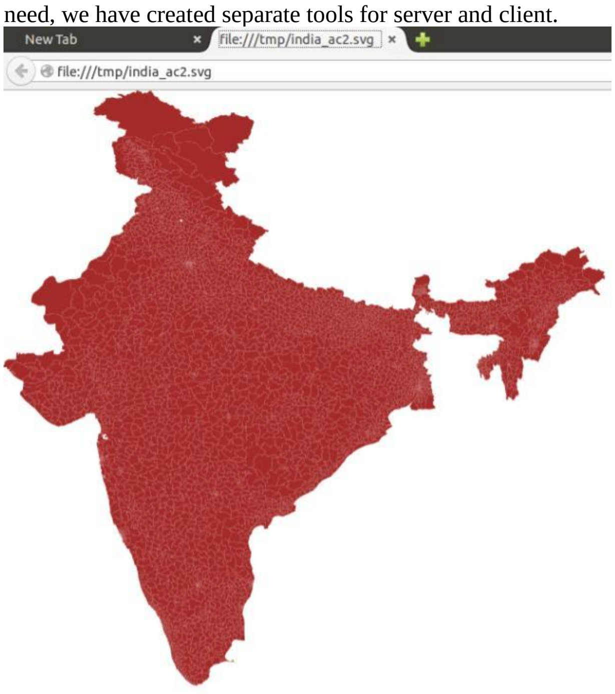

Figure 4. Rendered GMZ in Firefox using an add-on

The compression model has been developed as a python script, with options for compression and decompression given a GML file. The C version of ElementTree known as the cElementTree, which is faster and uses less memory, is being used for parsing XML tree structure. We have constructed functions to handle each geometry type supported in SFP. The compressed data is stored in python's *bytearray*, which is dumbed in a binary file at the end. A compressed binary file is finally returned.

GML, being XML based, has the advantage of being browser readable. It can be parsed natively and rendered in the browser itself. GMZ has no such advantage. To make it browser readable, a lean, cross-platform, client-side and easily installable solution in the form of a Firefox add-on has been implemented to perform decompression and visualization. The files are rendered as SVG in the browser tab.

3. CONCLUSION

3.1 Results

Filename	Original size	Compressed size	Compression percentage
Polygon			
India_adm0	20.1 MB	4.8 MB	76
india_adm1	31.1 MB	7.9 MB	74.6
india_adm2	42.1 MB	10.5 MB	75.06
india_landuse	47.2 MB	7 MB	85.17
india_buildings	76.8 MB	6.8 MB	91.1
india_natural	70.3 MB	14.4 MB	79.5
Linestring			
india_railways	22.7 MB	3.5 MB	84.6
india_waterways	67.7 MB	17.9 MB	73.56
india_roads	840 MB	122 MB	85.5
Point			
india_places	22.6 MB	7 MB	69
india_points	50.3 MB	14.7 MB	71

Table 1. Compression ratios of India GML files

Filename	Original size	Compressed size	Compression percentage
Polygon			
us_adm0	82.2 MB	19.2 MB	76.64
us_adm1	85.1 MB	20 MB	76.5
us_adm2	104 MB	24.2 MB	76.73
us_np_boundaries	55.1 MB	20.5 MB	62.8
us_cities	1.13 GB	242.9 MB	79
Linestring			
us_stateroad	34.7 MB	9.1 MB	73.77
us_trans_road segments	402 MB	87 MB	78.36
Point			
us_places	48.8 MB	19.2 MB	60.65

Table 2. Compression ratios of USA GML files

The average compression percentages for the India and US datasets are 78.64% and 73.05% respectively. This is better than any compression model that we know of, including GQComp. None of the compression models provide compression and decompression speeds for comparison. However, we would like to argue that our compression and decompression speeds would be comparable to others as our model is equally complex in comparison. When comparing with Zip, our model does better in compression ratios but lags behind in speed because Zip is ignorant of structure in data. An observation worth noticing is that the compression ratios for polygon and linestring data are relatively higher than that for point data. This was anticipated because point data has very less amount of spatial data compared to linestring and polygon data.

3.2 Conclusion and future work

The results presented in this work demonstrate the effects that topology and structure of the coordinate data have on the compression potential of a standard GML file. We were able to reduce the original data to its one-fourth/one-fifth without any loss of data. We also showed how GMZ can be used as a data transfer format on WebGIS by making it browser readable through a Firefox add-on. Hence, GMZ can gel with and serve an entire pipeline of a WFS-like architecture.

Future work will mainly focus on implementing an interface for querying and exploring what all spatial and non-spatial queries can be performed on the data without decompressing the file. We would also try to optimize it for speed. Another important thing that we would like to deal with in future is providing native support for GMZ in browser. One more challenge we plan to address is to integrate GMZ as one of the file exchange formats for web services like WFS. We anticipate that this will not only be lighter on the data transmission (bandwidth) rates but also improve some data processing capabilities due to its smaller data size, reducing the memory footprint.

REFERENCES

Dai, Q., Zhang, S. and Wang, Z., 2009, October. GQComp: A Query-Supported Compression Technique for GML. In: *Computer and Information Technology, 2009. CIT'09. Ninth IEEE International Conference on* (Vol. 1, pp. 311-317). IEEE.

Delta encoding. https://en.wikipedia.org/wiki/Delta_encoding. (accessed 1 Mar, 2017)

GML simple features profile, 2011. Open Geospatial Consortium. http://portal.opengeospatial.org/files/?artifact_id=42729. (accessed 15 August, 2016)

Guan, J. and Zhou, S., 2007, April. GPress: Towards effective GML documents compresssion. In: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on* (pp. 1473-1474). IEEE.

Harshita, N.N. & Rajan, K.S., 2010. Effective Topological and Structural Compression of GML coordinate Data. In: *Global Spatial Data Infrastructure (GSDI 12)*. Singapore. GSDI 12.

Harshita, N.N., 2013. *EFFECTIVE TOPOLOGICAL AND STRUCTURAL COMPRESSION OF GML DATA* (Doctoral dissertation, International Institute of Information Technology Hyderabad, India).

Li, Y., Imaizumi, T., Sakata, S., Sekiya, H. and Guan, J., 2008. Spatial Data Compression Techniques for GML. In: *Frontier of Computer Science and Technology, 2008. FCST'08. Japan-China Joint Workshop on* (pp. 79-84). IEEE.

Wei, Q., & Guan, J., 2010. A GML compression approach based on on-line semantic clustering. In: *Geoinformatics, 2010 18th International Conference on*. June 2010. IEEE.