smart cities

Article

CitySAC: A Query-Able CityGML Compression System

Chengxi Bernad Siew [1] and Pankaj Kumar [2,*,†]

[1] Faculty of Geoinformation and Real Estate, Universiti Teknologi Malaysia, Johor Bahru 81310, Johor, Malaysia; bernad@bernadsiew.com

[2] Geomatics Division, Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), Castelldefels, 08860 Barcelona, Spain

* Correspondence: pankaj.kumar@cttc.es; Tel.: +34-936-452-900

† Current Address: Geomatics Division, Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), Castelldefels, Barcelona, Spain.

Received: 2 January 2019; Accepted: 14 March 2019; Published: 19 March 2019

Abstract: Spatial Data Infrastructures (SDIs) are frequently used to exchange 2D & 3D data, in areas such as city planning, disaster management, urban navigation and many more. City Geography Mark-up Language (CityGML), an Open Geospatial Consortium (OGC) standard has been developed for the storage and exchange of 3D city models. Due to its encoding in XML based format, the data transfer efficiency is reduced which leads to data storage issues. The use of CityGML for analysis purposes is limited due to its inefficiency in terms of file size and bandwidth consumption. This paper introduces XML based compression technique and elaborates how data efficiency can be achieved with the use of schema-aware encoder. We particularly present CityGML Schema Aware Compressor (CitySAC), which is a compression approach for CityGML data transaction within SDI framework. Our test results show that the encoding system produces smaller file size in comparison with existing state-of-the-art compression methods. The encoding process significantly reduces the file size up to 7–10% of the original data.

Keywords: Spatial Data Infrastructure; 3D data; CityGML; encoding; Urban Applications

1. Introduction

The use of 3D city models is becoming increasingly important, as can be seen in various development of Spatial Data Infrastructures (SDIs) [1]. City models are especially important for urban development, identifying business locations and providing the platform that supports the process from civic participation to decision-making and policy-formulation [2]. The use of city models, requires a dedicated data exchange mechanism, due to an additional extension of geometric and semantic dimensions. In order to achieve an efficient decision-making processes involved in spatially-aided city models, an interoperability is highly required which leads to the realization of semantic models such as CityGML [3,4].

CityGML is a semantic-rich model for urban modelling. CityGML is an application profile of GML (Geography Mark-up Language), which itself is XML based. XML has been a popular World Wide Web Consortium (W3C) standard in data exchange and sharing usage in computing systems due to its readable and self-describing benefits. However, transmitting models in CityGML are considered impractical, due to computational costs involved in storage and transfer of schema over distributed networks. This is, for example, problematic when a city model is required for visualization purposes over a distributed environment. Accordingly, large datasets in CityGML should be efficiently compressed before transmission in terms of compression ratio and computational cost involved in it.

This paper presents an approach, called CitySAC (CityGML Schema Aware Compressor), which compresses CityGML and other data models constituting large geometries. It introduces a novel technique for encoding CityGML, which conforms to UTF-8 standards and embeds dictionary compression, compartmentalization and point compression for input geometries. Dynamic dictionary is generated during scanning process, and the dictionary size is fixed and tested at 16-bit size. A fixed reserved characters table is introduced in Table 1 and it is used in this test. The approach is tested and compared with other dictionary and arithmetic compression modules such as Deflate, Fast Info, LZMA and BZIP2. The presented approach is not only limited to CityGML, but can also support other XML formats with large geometries. This paper provides a generic solution to 3D data transaction within SDI framework, where efficiency can be achieved via binary data transactions while maintaining data interoperability. The proposed solution contributes to the development of XML compression technique that enables an efficient handling of spatial data models within CityGML framework. We discuss existing compression techniques and other related works, together with their advantages and limitations in Section 2. We discuss CityGML model and Spatial Data Infrastructure (SDI) framework in Section 3. In Section 4, we present an architecture and data structure of our CitySAC encoding system along with compression and decompression pipelines involved in it. In Section 5, we present the test results of CitySAC system on various CityGML datasets and discuss them. Finally, we conclude the paper and detail the central items of our future work in Section 6.

Table 1. Reserved byte-codes within CityGML.

Byte-Code	**Flag Type**
65500	White Space
65501	Element
65502	Attribute
65503	Attribute Value
65504	Value
65530	Element Close
65001...65499	URI / Face-set Pair

2. Previous Studies

Various compression techniques have been developed in previous decades, which aim to solve particular problems. Graphical compression techniques deal directly with graphic representations, while text compression techniques deal with data of the object. Deering [5] proposed point based geometry compression technique, while Taubin et al. [6] developed a technique for encoding the connectivity of graphical objects. Isenburg and Snoeyink [7] used topology compression on existing graphic representations, which encodes the relationship between points, edges or faces. An efficient compression technique should be associated with features such as wide applicability, high compression ratio, control over lossness, ubiquitous access, minimum latency in reconstruction, multi-resolution representation and selective component-wise compression. The encoder should serve for most popular representation schemes, which in this context is the CityGML system. The encoder should also achieve a high compression ratio, and have manageable lossless over the product. From geometry representation perspective, the use of data type such as float representation for a point does not guarantee precision compared to integer data type representation [8]. Floating point data type achieves better precision when values are closer to zero. Similarly, a mm accuracy of a position represented as 0.209 is often better compared to a nm accuracy of that position represented as 0.209153684. The extra bits are useless and considered as noise. On the other hand, a scaled integer is able to store more precise point information. In addition, the encoder should include schema or header for providing an efficient accessibility. Apart from this, queries should be directly applicable to compressed documents which reduces the latency and computational cost associated with the decompression of original files. For the scaling of geometries in CityGML, a 32-bit integer representation improves the quality of points in comparison with double 64-bit representation. Scaling of integer to represent mm accuracy for geometries is sufficient, while remaining decimal values are mere noise generation. Therefore, 32-bit integer point representation is sufficient unless the points are represented in a very large number where 64-bit integer could be considered.

On the other hand, some of the developed techniques have been specifically targeted for XML, such as InfoSet dictionary method [9] and other general text compressors such as XMill [10], XGrind [11], XPRESS [12], XComp [13], XSC [14] and XCQ [15]. However, none of these methods are suitable for solving the core problem of transmitting data in a Document Object Model (DOM) form, which is embedded for geometric and semantic information. Techniques that deal with semantic and geometric data have been particularly modified for SDI in 3D use cases. XMill technique [10] uses a split and group methodology and then GZip compression module is applied during final stage. Containers are prepared with structure and data domain, and each container is defined via default 8 MB window size. XMill does not support query on compressed document, as all the data is chunked into containers without specifying the exact meaning of data within the containers. In order to improve XMill, XGrind [11] was designed with a capacity to query the compressed document but a structure of the document is left without encoding. While this improvement could solve the query limitation that was not available in the previous compressor, XGrind only adapts Document Type Definition (DTD) of XML which is a conventional representation of markup declaration specifications. Later, XPRESS technique [12] was introduced in which reverse arithmetic encoder is used to divide the elements and content-paths into floating intervals. This concept was also used in other common arithmetic encoders. However, this encoder does not encode data types or value of the elements. XComp method [13] was developed to improve XMill by introducing further containers, while XCQ method [15] was introduced as an indexing method via a Pseudo-Depth-First strategy that builds a DTD tree. XSC [14] used Huffman Coding for detecting DTD data, but this approach could face exponential time complexity if XML document consists of large number of unique elements. On the other hand, Fast-InfoSet (FI) uses ASN.1 notation which is a standard for describing rules and structure of representation, encoding, transmission and decoding of data in the field of telecommunication. However, this technique does not fit well for geometry datasets such as CityGML, as large representation length are required for face-set geometries. In the next section, we discuss CityGML model and various SDI frameworks.

3. CityGML & SDI Framework

CityGML is a semantic-rich model which is encoded in XML schema. The data format is constructed with information such as elements, attributes, values etc. For example, in the test data, <CityModel> has an enclosed tag of </CityModel>, which is a standard XML schema. CityGML provides XML Schema Definition (XSD) which details structural information and data types of the XML documents. CityGML represents building models at different Levels of Detail (LoD), from LoD0 to LoD4, where each level of detail distinguishes different aspects of building. CityGML data are particularly significant in size to be compressed due to voluminous nature of its file size. Currently, data delivery operations for existing 3D SDI does not provide CityGML for analysis purpose due to these drawbacks. Effective compression procedures allow spatial analysis via CityGML directly. In this study, CityGML 1.0 was used as data input for the compression ratio analysis. Although, OGC has released an updated version of CityGML, however, similar results can be achieved with our compression approach. An example of CityGML dataset for Stadt-Etteinhem town in LOD3 is shown in Figure 1.

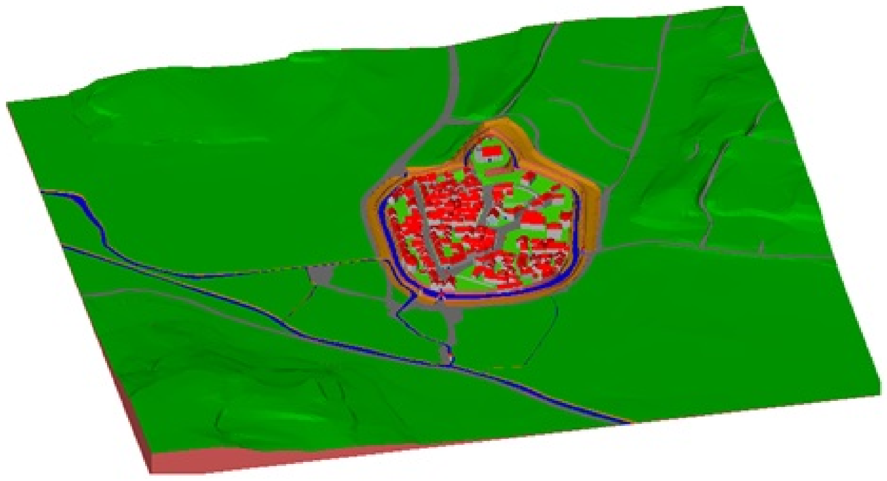

Figure 1. CityGML based building model of Stadt-Etteinhem town in LOD3. (Source: http://citygml.org).

The progress of implementing SDI in various countries such as Digital China Geospatial Framework (DCGF) [16] and National Land Information System (NLIS) in Poland [17] provides the evidence of growing importance of spatial data sharing for various requirements. The requirements for encoding CityGML in 3D SDI have been thoroughly discussed by Siew and Abdul Rahman [18]. The input must be XML standardized, encoded content must be based on open standard notation and decoder must provide options for lossless output. A tightly coupled encoding and decoding engine provides a framework for an efficient transition of spatial data within web services. Encoding services can also be established to serve the server-to-client and server-to-server scenarios in data transaction mechanism.

Various examples of web orchestration which requires semantic and geometrical information have been discussed [19–22]. Web orchestration has been used to retrieve results and present the 2D geometries via Web Feature Service (WFS) and base map data via Web Map Service (WMS). On the other hand, Web 3D Service (W3DS) has been used to present 3D data in X3D format. This paper does not discuss 3D visualization, as its main aim is to present the innovative capability of our approach to encode 3D geometry and semantic data. The presented approach has been tested and compared with existing encoding packages, which demonstrates its efficiency to retrieve encoded document via direct queries over web services. This paper proposes XML encoder, a schema-aware compressor that encodes data based on dictionary approach while maintaining query-able advantages, reducing storage and bandwidth transmission and providing partial decompression capability. In the next section, we present the CitySAC system.

4. CitySAC Encoding System

We present the CitySAC encoding system which compresses CityGML data transactions constituting large and complex geometries. In Section 4.1, we detail the architecture of CitySAC by describing the data flow from its input to encoded output. In Section 4.2, the data structure of CitySAC is elaborated with respect to data containers.

4.1. Architecture

The architecture of encoder is presented in Figure 2, which shows the compression engine comprising several components such as XMLParser, XSD builder, Entropy Module, Dictionary builder, Parent-Child Identifier (PCI), Scaled Geometry Constructor (SGC) and compression module. The prototype has been developed using C# wrapped with XMLReader and XMLNode-identifier. We discuss these components in detail.

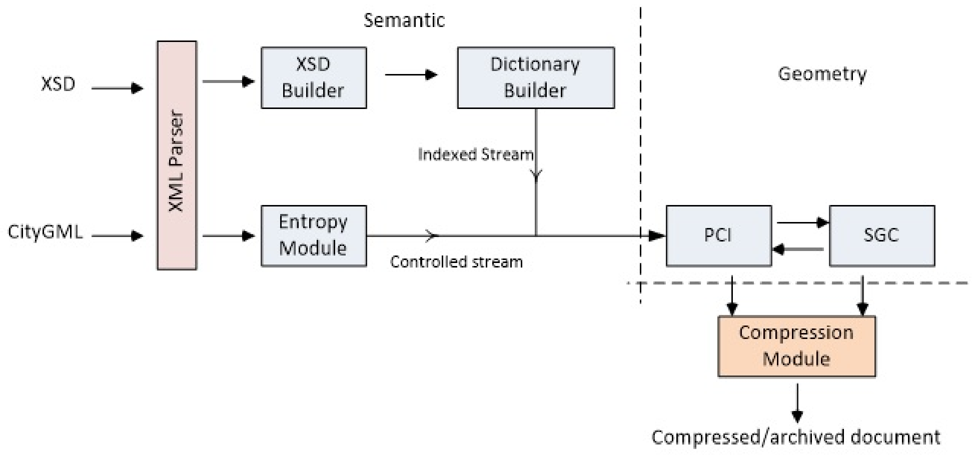

Figure 2. CitySAC Architecture.

4.1.1. Input

The input of this encoder in this test case uses CityGML document based on LOD3. This encoder also accept other types of XML documents, with or without the presence of XSD schema. However, with the XSD schema definition source, the encoder will skip the scanning process to search for data types.

4.1.2. XML Parser

The parser is capable of parsing XML information which is a wrapper of the in-built XMLReader in *C#* library. It parses information such as XML nodes, depths, namespace, value etc. by scanning the input document. It then allows navigation through forward and backward to retrieve intermediate information such as attributes.

4.1.3. XSD Builder

XSD Builder builds a schema reference which serves as main reference coupled with dictionary. The reference includes XML value data types, hierarchical definition of parent and child as well as attributes. This builder only runs if the XML document is accompanied with XSD. If the XSD is not present, then the XML document is directly scanned to parse the relevant information into dictionary.

4.1.4. Entropy Module

Entropy module generates information entropy based on values parsed out by XMLParser. This module ensures the data representation within 16-bit bounding. It also helps to determine if a higher number of representation is required for encoding the data input. The entropy check of the document is done prior to the encoding iterations. Once the byte-representation is confirmed by entropy, then the iterations are proceeded. This entropy check ensures sufficient size of 16-bit denotations for the CityGML input and allows optimization for binary data representation. The first iteration produces the reference point for SGC component, while the second iteration encodes data into UTF-8 string for further compression. In information theory, entropy is used to determine the lowest possible representation in bits for a symbol. Entropy was defined by Shannon [23], with its representation as,

$$H_n = -\frac{1}{n} \cdot \sum_{s \in S_n} \{Pr(s) \cdot log_2[\Pr(s)]\} \tag{1}$$

S_n denotes words having n symbols, $\Pr(s)$ represents the probability that a word $s \in S_n$ occurs, and H_n is the entropy in bits per symbol. In this paper we use a 0-order Markov process, which is first-order Shannon entropy H_1. Markov process models random system which changes states based on transition rules and is only applicable to current state. Examples of entropy of the tags and attributes for some of the CityGML datasets are shown in Tables 2 and 3 respectively.

Table 2. Entropy of the tags (elements) for different CityGML input files.

File Name (.XML)	Bytes	Tags	Unique Tags	Attributes	H_1
Commercial Building	856,064	5375	18	3224	2.368156
National Audit	8,802,304	59,725	18	35,833	2.328374
Putrajaya Convention	987,136	6743	18	4046	2.353207
Putrajaya Mosque	11,190,272	75,749	13	45,446	2.32741
Seri Gemilang Bridge	28,114,944	189,845	12	113,902	2.324867
Putrajaya All	109,928,448	742,902	19	445,507	2.334328

Table 3. Entropy of the attributes for different CityGML input files.

File Name (.XML)	Bytes	Attributes	Unique Attributes	H_1
Commercial Building	856,064	3224	11	0.954407309
National Audit	8,802,304	35,833	11	0.922468795
Putrajaya Convention	987,136	4046	11	0.948378432
Putrajaya Mosque	11,190,272	45,446	9	0.920554196
Seri Gemilang Bridge	28,114,944	113,902	9	0.919246192
Putrajaya All	109,928,448	445,507	11	0.919991184

Unique tags and attributes denote occurrences in entire document, and are used to examine the minimum cost required for bit-representation. As shown in Equation (1), unique tags and attributes contributes to the H_1 output. In the entropy module, pattern encoding is used in which each representation is not limited to one character, but involves the whole pattern while scanning. Predictable outcome denotes the occurrences in a given dataset.

4.1.5. Dictionary Builder

Dictionary builder uses Hash-Set to identify uniqueness parsed by XML Parser into different containers. Hash-Set is a set computed by hashing a value in the list and comparison of uniqueness takes place linearly. Dictionary builder stores data container as shown in Figure 3. This module maps the data input and largely reduces the repetitive occurrences.

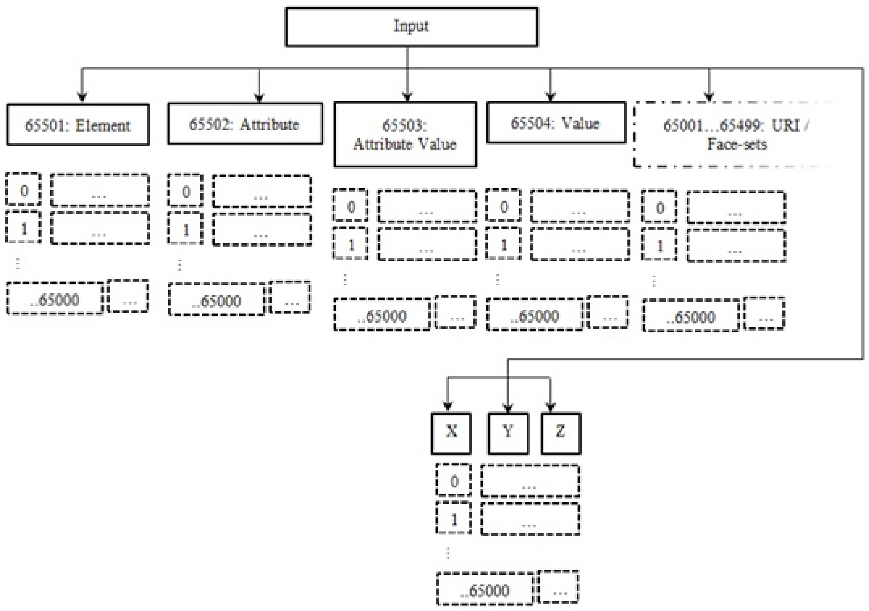

Figure 3. Data container in CitySAC encoder.

4.1.6. PCI & SGC Builder

Parent-child Identifier (PCI) uses the Unique-Resource-Identifier (URI) information from the document and constructs pseudo-tree relation with the nested child. Parent-child is a hierarchical relationship of elements in the XML document. The children are examined by using XML-depth information generated by XMLParser. This information is essential for constructing the topological relationship in between parent and child within the nested polygon. With this PCI tree, a catalog is generated that could be used as client-side retrieval for partial decompression.

In SGC builder, geometry quantization procedure is initiated and then difference of relative values generated using K-Face mean algorithm are stored. The relationship between geometries and face-sets are constructed, which then allows partial decompression for client-side retrieval. Within SGC, the URI of 3D objects is used to identify and cluster group of faces. 3D Object faces consist of minimum three vertices while a group of these faces may form a building shape. The constituting points can be represented in a 3D Euclidean space as

$$\{X_i\}_{i=3}^{n}, X_i \in R^n, \tag{2}$$

where X is a point with set of i points to form a face represented by nth points.

$$K \in \{1.....K\}, \tag{3}$$

where number of faces form Kth cluster.

A finite set of X_i could produce the Z_K cluster centre based on following equation

$$Clus(Z_1, Z_2.....Z_K) = \sum_{i=3} min \parallel X_i - Z_K \parallel^2, \tag{4}$$

The input is finite set of geometries, while the output is the centre of each cluster such that space is partitioned into faces, where each face corresponds to one of the Z_K. This means that each face contains the region of space whose nearest representation is Z_K as shown in Figure 4.

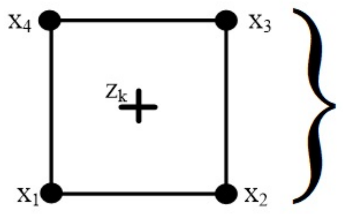

Figure 4. Face-mean cluster centre with Z_K representing cluster centre of the face with vertices X_1, X_2, X_3, X_4.

The cluster centre of each 3D object allows an efficient storage of geometries in integer. This approach is applied to each geometry container. A centre is initialized and then mean of first face is taken into next calculation to produce another mean of that particular set, and this continues until the final set is computed. The cluster centre do not necessarily falls within the polygon. A typical example of this is L-shaped polygon whose cluster centre falls out of the bounded polygon.

4.1.7. Archiving Module

The archiving module uses LZMA, which efficiently reduces the storage of binary data. This module archives data by its container as shown in Figure 3. Each container is then archived in a common archiver such as ZIP so that it could be sent to client-side in one package.

4.2. Data Structure

CitySAC scans the input by following the XML standards rather than geometry. The algorithm assumes the input as topological and geometrical correct by treating polygons, lines, points, solids as geometries and face-sets. In general, the method is a lossless method, with an option to produce near-lossless output. The current encoder uses 16-bit representation with a total number of representation limited to 65,500. A face-set is defined by a face constituted by a set of geometries, and each face is given an URI in accordance with the CityGML standard. The encoder is designed to fit 250 URI face-set pairs, which provides an expansion of face-set representation up to 250 $\times$ 65,500 which leads to around 16.25 million polygons. The byte-codes reserved within CityGML standard are detailed in Table 1.

On the contrary, semantic data is stored based on the XML structure, while the semantic relationship with geometry is defined by URI. The URI is also used to define parent-and-child relationship. A container is defined as a dictionary that stores specific type of data. For each container, a dictionary of 65,000 representation is defined as unique values stored in it. CityGML components do not exceed these large and unique representations, therefore 65,000 representations are considered sufficient. On the other hand, 16-bits representation is taken into consideration in this experiment as it is sufficient for most of the LOD3 usage scenarios.

Our CitySAC system scans the document using XMLReader by identifying XML components along the way. Hash-sets are used to identify unique values, which are then stored in data containers. The encoder again scans the document by matching the indices and then produces encoded paired binary data. The encoder reads the document and identifies <core:CityModel> as an element which is then saved in element dictionary by providing a simulated index 1, if it is found to be unique. Next, xsi:schemaLocation is read and then similar process is performed to save it in attribute dictionary. This continues until the end of document. The encoder uses word matching pattern to recognize particular instances such as gml:posList or gml:pos., Geometries are recognized by retrieving entire position string, which are then splitted into single geometry of X, Y and Z. CityGML standard provides URI in the form of gml:id for each polygon which is then used to construct parent-child relation. When the encoder reads a close element such as </gml:name>, then a flag of close element will be placed in the main encoded document with 65,530 code as detailed in Table 1. When the encoder reads white spaces or line endings, a white-space flag 65,500 will be placed. The data loss here is that only 1 white-space flag is retained while others are ignored, as line-endings or white-space in XML documents are generally used for easy readability.

4.3. Compression & Decompression Pipeline

The encoding schema provides output results as lossless and near-lossless. The lossless output represents geometries in integer64 format, while the representation of geometries in near-lossless is scaled integer. Different stages involved in the compression and decompression pipeline of CitySAC system are show in Figure 5. In the pipeline, an information is transformed with source file represented as $b(t)$, while the transformed output is represented as $\hat{b}(t)$.

The developed encoding schema, CitySAC, employs lossless, non-progressive, streaming and random access behavior. As this encoder do not reduce points that lead to a geometry mesh or shape changes, it is a lossless point encoder. The non-progressive nature of the encoder provides that an output of this encoder is in DOM form, where geometries and semantics are transferred in the compressed document. This is different from the visualization pipeline, where progressive transmissions of low resolution geometries are preferred to high resolution geometries. The encoder employs a non-progressive approach as it consists of only one rate of resolution and decompresses points at full precision. The non-progressive approach also takes the load off a file server. As the use of progressive transmission from lower LOD to higher LOD is always useful therefore, progressive visualization can also be integrated on top of the decoded layer.

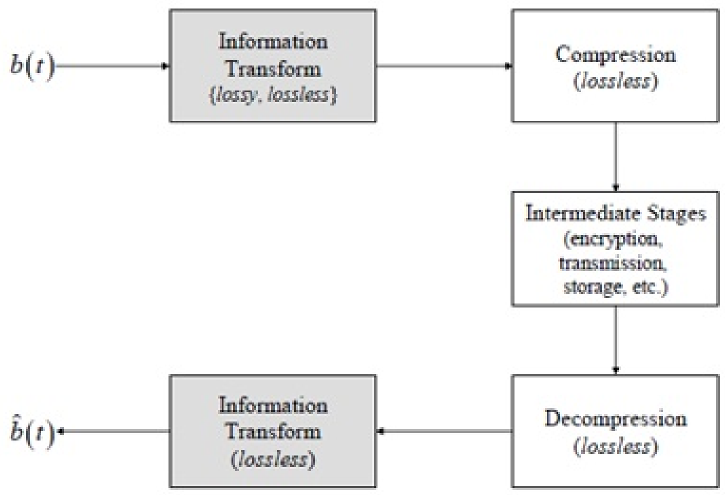
$b(t)$
Information Transform {lossy, lossless}
Compression (lossless)
Intermediate Stages (encryption, transmission, storage, etc.)
Decompression (lossless)
Information Transform (lossless)
$\hat{b}(t)$

Figure 5. Compression and decompression pipeline involved in the CitySAC system.

Main part of the encoder is the dictionary encoding, where all occurrences are indexed and stored. Furthermore, geometries are in-built part of 65,000 face-sets, where each face represents a polygon in CityGML. The main idea of encoder is to follow the XML original structure such that querying capability is retained and compressed content is retrievable. In this research, each representation is defined as a symbol denoted with 16-bits. The core components in the scheme consist of bit representation, close, Tag, attribute, attribute value, value URI and Face-sets (X, Y, Z). Bit representation specifies whether an encoder is 16-bit, 32-bit or 64-bit, close component represents the closing of elements, tag represents an element, attribute and attribute values specify XML attributes and their values while value denotes the content within tags.

The encoder scans the documents on-the-fly with the aid of XSD document at a speed of more than 100,000 lines per second. It provides to access more than 10, 000 face-sets per second or 150,000–200,000 points per second in each polygon. The speed varies depending on the document content and system specifications. This execution speed is calculated before the final stage of lossless compression, which in our case is LZMA. The significant benefit of chunked geometry container is that geometry can be retrieved in a chunk without requiring full decompression, thus improving decompression efficiency. The scanning module is done in parallel programming, where multi-thread is utilized to improve the speed. In the next section, we present the tests of our CitySAC encoder on various data models and discuss the results.

5. Test Results & Discussion

We tested our CitySAC system on 6 CityGML datasets by going through the process, as mentioned in Section 4. The outputs were evaluated by comparing the compression ratio and time taken with existing state-of-the-art techniques. The compression ratio CR_2 is calculated as

$$CR_2 = (1 - \frac{size of compressed file}{size of original file}) \times 100\%. \quad (5)$$

The compression size and ratio results obtained for the tested CityGML datasets after applying CitySAC (with LZMA) and other compression techniques are shown in Table 4. The results for CitySAC encoding system (with LZMA) are shown in the last column. From the results obtained, our CitySAC encoder in integration with LZMA produced better compression results than any of the methods alone. The encoding process provided the compression rate of more than 90% for all the tested datasets, thus reducing the file size up to 7–10% of the original data. The lowest compression rate was found to be for Fast Info technique, while LZMA produced better results in comparison with other techniques used alone, which justifies its integration in our CitySAC encoding system. The compression results for CitySAC (with LZMA) were also found to be better than with the use of LZMA alone.

Table 4. Compression size and ratio for 6 tested CityGML input files after applying CitySAC (with LZMA) and other compression techniques.

File Name (.XML)	Original Size (MB)	Deflate Size (MB)/ CR2 (%)	Fast Info Size (MB)/ CR2 (%)	LZMA Size (MB)/ CR2 (%)	BZIP2 Size (MB)/ CR2 (%)	CitySAC + LZMA Size (MB)/ CR2 (%)
Commercial Building	0.836	0.086/ 89.71	0.469/ 43.90	0.073/ 91.27	0.079/ 90.55	0.06/ 92.82
National Audit	8.594	0.948/ 88.97	4.132/ 51.92	0.696/ 91.90	0.869/ 89.89	0.659/ 92.33
Putrajaya Convention	0.962	0.17/ 82.33	0.43/ 55.30	0.102/ 89.40	0.106/ 88.98	0.090/ 90.64
Putrajaya Mosque	10.928	1.56/ 85.72	4.81/ 55.98	0.992/ 90.92	1.226/ 88.78	0.88/ 91.95
Seri Gemilang Bridge	27.45	3.612/ 86.84	12.352/ 55	2.714/ 90.11	3.519/ 87.18	2.53/ 90.78
Putrajaya All	104.1	13.2/ 87.32	32.1/ 69.16	9.76/ 90.62	12.1/ 88.38	9.8/ 90.59

The time results for CitySAC (with LZMA) along with other techniques to compress input data files are shown in Table 5. In the obtained time results, BZIP2 technique took less time in comparison with other techniques used to compress input CityGML data files. Our CitySAC encoder in integration with LZMA provided better temporal results than the use of LZMA technique alone. Consider the obtained compression ration results, the use of our CitySAC encoding system is efficient in terms of time taken to compress the data files. The developed CitySAC encoding system can be distinguished from existing encoders in following ways:

- The encoding process is done in one-way scan from beginning to end of the document, while maintaining its structure.
- The encoded document can be queried, while geometries can be sliced into chunk sizes to provide partial decompression.
- The encoder employs standard UTF-8 binary format and is built for dynamic representation of large geometric datasets.
- It uses entropy values to check the bit-representation and employs a scaled integer algorithm for double value.
- It is suitable for web transaction, such as Javascript readable byte-code.
- It employs state-of-the-art compressor LZMA, which produces smaller lossless outputs, and is in most cases faster than, LZMA alone.

In the next section, we conclude the paper.

Table 5. Time (in seconds) results for CitySAC (with LZMA) and other compression techniques to compress 6 CityGML input files.

File Name (.XML)	Deflate (s)	Fast Info (s)	LZMA (s)	BZIP2 (s)	CitySAC + LZMA (s)
Commercial Building	0.2	0.5	0.3	0.1	0.4
National Audit	3.8	3.6	4.5	1.2	1.9
Putrajaya Convention	1.1	0.9	1.3	0.3	0.51
Putrajaya Mosque	3.5	4.2	5.8	2.2	4.1
Seri Gemilang Bridge	9.8	11.5	15.2	4.5	11.2
Putrajaya All	14.5	18.5	27.3	11.0	25.5

6. Conclusions

In this paper, we discussed several techniques for data and graphical compression techniques. We highlighted the issues in relation to data transaction and requirement of dedicated SDI for data exchange mechanism. We presented CitySAC encoding system which efficiently compresses data models with large geometries. The encoder was tested on different CityGML datasets to prove that the encoding procedure provides better compression results in comparison with other techniques. The system was able to compress the input files up to 7–10% of the original file size. The decoder can also be applied to generate a full original file or partial compressed data, which takes up only 20–25% of the original file. With face-sets indexing, the retrieval procedure is also able to partially compress the document for particular area-of-interest. This means that partial decompression is supported in the presented approach as one of the advantage of this encoder. The system can be used for various other XML documents with large geometries datasets. This work could be extended to employ spatial coherence in order to improve chunk size compression. A full client-side JavaScript decoder has been created to allow a platform-independent solution. Encoded web services could be established in future as an intermediary services based on service-oriented architecture (SOA) standard. The implementation of document query in the web services can be useful for applications such as urban tourism, tax-collection activities, change-detection, urban management and updating. We also proposed a standardized binary format for CityGML tags, which provides small file transaction benefits, while maintaining interoperability using Javascript decoder. This work could also be implemented in network protocol infrastructure, where all spatial data with geometries could be compressed using a standard procedure or protocol.

Author Contributions: C.B.S. developed the approach and performed experiments. P.K. provided technical advice at some stages. All authors contributed equally in preparing this manuscript.

Funding: This research was partially funded by Ministry of Education, Malaysia, under the MyBrain-MyPhD scheme.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Basanow, J.; Neis, P.; Neubauer, S.; Schilling, A.; Zipf, A. Towards 3D Spatial Data Infrastructures (3D-SDI) based on open standards—Experiences, results and future issues. In *Advances in 3D Geoinformation Systems*; Van Oosterom, P., Zlatanova, S., Penninga, F., Fendel, E.M., Eds.; Springer: Berlin, Germany, 2008; pp. 65–86.
2. Kolbe, T.H.; Konig, G.; Nagel, C.; Stadler, A. *3D-Geo-Database Berlin Version 2.0.1a*; Technical Report; Institute for Geodesy and Geoinformation Science Technische University: Berlin, Germany, 2008.
3. Groger, G.; Kolbe, T.H.; Czerwinski, A.; Na-Gel, C. *OpenGIS R City Geography Markup Language (CityGML) Encoding Standard*; Open Geospatial Consortium: Wayland, MA, USA, 2008.
4. Groger, G.; Kolbe, T.H.; Nagel, C.; Hafele, K.H. *OGC City Geography Markup Language (CityGML) Encoding Standard*; Open Geospatial Consortium: Wayland, MA, USA, 2012; pp. 1–344.
5. Deering, M. Geometry compression. In Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, CA, USA, 6–11 August 1995; pp. 13–20.
6. Taubin, G.; Horn, W.P.; Lazarus, F.; Rossignac, J. Geometry coding and VRML. *Proc. IEEE* **1998**, *86*, 1228–1243. [CrossRef]
7. Isenburg, M.; Snoeyink, J. Spirale Reversi: Reverse decoding of the edgebreaker coding. *Comput. Geom.* **2001**, *20*, 39–52. [CrossRef]
8. Isenburg, M. *LASzip: Lossless Compression of LiDAR Data*; Technical Report; rapidlasso GmbH: Gilching, Germany, 2013.
9. Sandoz, P.; Triglia, A.; Pericas-Geertsen, S. *Fast Infoset*; Technical Report; ITU: Geneva, Switzerland, 2005.
10. Liefke, H.; Suciu, D. XMill: An efficient compressor for XML data. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 15–18 May 2000; Volume 29, pp. 153–164.
11. Tolani, P.M.; Harista, J.R. XGrind: A query-friendly XML compressor. In Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, 26 February–1 March 2002.
12. Min, J.K.; Park, M.J.; Chung, C.W. XPRESS: A queriable compression for XML data. In Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, CA, USA, 9–12 June 2003.
13. Li, W. XCOMP: An XML Compression Tool. Master's Thesis, University of Waterloo, Waterloo, ON, Canada, 2003.

14. Wang, T.J.; Gao, J.; Yang, D.Q.; Tang, S.W.; Liu, Y.F. XPath evaluation oriented XML data stream compression. *J. Softw.* **2005**, *16*, 869–877. [CrossRef]
15. Ng, W.; Lam, W.Y.; Levene, M. XCQ: A queriable XML compression system. *Knowl. Inf. Syst.* **2006**, *10*, 421–452. [CrossRef]
16. Li, P.; Wu, L.; Xiao, X. SDI in China: Progress and issues. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2008**, *37*, 2.
17. Gazdzicki, J.; Linsenbarth, A. GIS in Poland: development towards SDI. In Proceedings of the 10th EC GI & GIS Workshop, ESDI State of the Art, Warsaw, Poland, 23–25 June 2004.
18. Siew, C.B.; Abdul Rahmna, A. A schema-aware encoder for Putrajaya 3D. In *Urban and Regional Data Management*; Laurini, R., Ellu, C., Rumor, M., Zlatanova, S., Eds.; CRC Press: Boca Raton, FL, USA, 2013; pp. 181–190.
19. Stollberg, B.; Zipf, A. OGC web processing service interface for web service orchestration: Aggregating geo-processing services in a bomb threat scenario. In Proceedings of the 7th International Conference on Web and Wireless Geographical Information Systems, Cardiff, UK, 28–29 November 2007; pp. 239–251.
20. Christensen, A.F.; Ostlander, N.; Lutz, M.; Bernard, L. Architectures, Designing service for distributed geoprocessing: challenges and future directions. *Trans. GIS* **2008**, *11*, 799–818. [CrossRef]
21. Walenciak, G.; Stollberg, B.; Neubauer, S.; Zipf, A. Extending spatial data infrastructures 3D by geoprocessing functionality - 3D simulations in disaster management and environmental research. In Proceedings of the International Conference on Advanced Geographic Information Systems & Web Services, Cancun, Mexico, 1–7 February 2009.
22. Lanig, S.; Zipf, A. Proposal for a Web Processing Services (WPS) application profile for 3D processing analysis. In Proceedings of the 2nd International Conference on Advanced Geographic Information Systems, Applications and Services, St. Maarten, Netherlands Antilles, 10–16 February 2010.
23. Shannon, C.E. Communication in the presence of noise. *Proc. IRE* **1949**, *37*, 10–21. [CrossRef]