Efficient GML-native Processors for Web-based GIS: Techniques and Tools

Chia-Hsin Huang[1,2]
jashing@iis.sinica.edu.tw

Tyng-Ruey Chuang[1]
trc@iis.sinica.edu.tw

Dong-Po Deng[1]
dongpo@iis.sinica.edu.tw

Hahn-Ming Lee[1,3]
hmlee@mail.ntust.edu.tw

[1] Institute of Information Science
Academia Sinica, Taipei 115, Taiwan

[2] Department of Electronic Engineering,
National Taiwan University of Science and Technology, Taipei 106, Taiwan

[3] Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan

ABSTRACT

Geography Markup Language (GML) is an XML-based language for the markup, storage, and exchange of geospatial data. It provides a rich geospatial vocabulary and allows flexible document structure. However, GML documents are usually large and complicated in structure. Existing techniques for XML document processing, either streaming-based or memory-based, may not deal with such GML documents efficiently. There is an urgent need to adapt existing XML techniques to support the processing of large XML/GML documents, as well as to express GML-native geospatial operations.

In this paper, we propose and implement an efficient GML query processor, *GPXQuery*, and a GML-aware streaming parser, *GPSAX*, by extending an XQuery processor and a SAX parser, respectively, to support GML-native geospatial functionalities. In addition to these tools, an XML prefiltering technique is applied to the processors to speed up geospatial operations over large GML documents. Our experiment results show that the XML prefiltering technique significantly improves the performance of both the *GPXQuery* and *GPSAX* processors by reducing either the query execution time or the memory space consumption. Depending on the nature of user queries, the enhanced query processors can achieve a ten-fold performance improvement. These efficient GML-native processors have been used to develop a GML-based Web GIS with a geospatial query interface and a Scalable Vector Graphics (SVG) map navigator.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database Applications – *Spatial database and GIS*; H.3.3 [**INFORMATION SYSTEMS**]: Information Search and Retrieval – *Search process.*

General Terms

Algorithms, Languages, Performance.

Keywords

DOM, GIS, GML, SAX, Web, XML Prefiltering, XPath, XQuery.

1. INTRODUCTION

The nature of geospatial phenomena is complex, diverse, and multi-scaled. In geographic information science, geospatial data models, which are used to abstract physical geospatial objects, are becoming increasingly sophisticated. Often geographic data models are mapped to relational database models for data storage, operation, analysis, and visualization. Many well-known GIS products/databases, such as ESRI's ArcGIS/ArcGIS, Autodesk's MapGuide, Oracle's GIS database, and open source software PostgreSQL/PostGIS, use relational data models. With their robust index and query capabilities, relational-based GIS are able to access geospatial data efficiently. However, problems arise when geospatial data cannot be expressed naturally and efficiently by using the relational data model. We present an example to explain this problem in Section 2. In addition, the data formats used in the above products/databases are usually proprietary and/or specific to the products/databases. Thus, it is more difficult to share, exchange, and manipulate geospatial information across different products/databases.

Geography Markup Language (GML) [4], which provides a rich vocabulary and flexible document structure, is considered an effective approach for expressing increasingly complicated geospatial data, as well as non-geospatial data. GML has been used as a standard markup language for the transportation, storage, and modeling of geospatial data. Intuitively, existing XML processors, such as XPath [16], XQuery [17], and SAX (Simple API for XML) [12], may help process GML documents. However, the large size of documents, the flexible document structures, and the rich vocabulary, increase the difficulty of manipulating GML documents. Moreover, the geographic information contained in GML documents is not well understood by the usual library functions, *e.g.*, string and numeric functions, provided by the existing XML processors. Therefore, there is an urgent need to adapt existing XML techniques, to support the processing of large XML/GML documents and geospatial functionalities.

The XQuery language [17] is sufficiently flexible to query a broad spectrum of XML information sources, including databases and documents. XQuery processors can also be easily extended by calling external function libraries without modifying or recompiling their source codes. In [1][18], XQuery language is extended to support geospatial functionalities. A prototype of a GML query processor has been implemented in [1]. Although extended XQuery processors can process geospatial information, they may not be able to query large GML documents efficiently. Similarly, existing techniques for XML document processing, either streaming-based (*e.g.*, SAX) or memory-based (*e.g.*, DOM, the Document Object Model) [5], may not be suitable for processing GML documents efficiently. The DOM model consumes a large amount of main memory, typically about five times the size of the XML documents; while SAX parses over a document in a linear manner to locate interesting fragments. Thus, both models waste a large amount of computational resources by processing uninteresting fragments. To accomplish GML-based Web GIS, there is a need to adapt existing XML techniques to support the processing of large XML documents, as well as to express GML-native geospatial operations.

In this paper, we propose and implement an efficient GML query processor, *GPXQuery*, and a GML-aware streaming parser, *GPSAX*, by extending an XQuery processor and a SAX parser, respectively, to support GML-native geospatial functionalities. An XML prefiltering technique [8] is also applied to the processors to speed up GML-native geospatial operations over large GML documents. The *GPXQuery* and *GPSAX* processors are then employed to implement a GML-based Web GIS with a geospatial query interface and a Scalable Vector Graphics (SVG) [14] based map navigator. Our experiment results show that the XML prefiltering technique significantly improves the performance of both the *GPXQuery* and *GPSAX* processors by reducing either the query execution time or the memory consumption. Depending on the nature of users' queries, the enhanced query processors can achieve a ten-fold performance improvement.

The contributions of our work are as follows.

- We propose and implement two GML-native processors, *GeoXQuery* and *GeoSAX*, by extending an XQuery processor and a SAX parser, respectively, to support geospatial functionalities, such as intersection, union, buffer, and within.
- We also develop a geospatial indexing plug-in module for an XML prefiltering technique [8] to support the indexing and prefiltering of a GML document by means of *bounded-by* query constraints. As a result, the enhanced technique can produce a smaller GML document fragment, using both non-geospatial and geospatial query constraints.
- We further develop *GPXQuery* and *GPSAX* processors by integrating the enhanced XML prefiltering technique into *GeoXQuery* and *GeoSAX*, respectively, to speed up the processing of queries about large GML documents.
- Finally, the *GPXQuery* processor is used to build a GML-based Web GIS with a geospatial query interface and a SVG map navigator.

The remainder of the paper is organized as follows. In Section 2, we present the data model used in our GML-based Web GIS. The enhanced GML query processors are described in Section 3. The Web GIS is discussed in Section 4. The performance analysis of the GML-native processors is detailed in Section 5. Finally, we present our conclusions in Section 6.

2. GEOSPATIAL DATA MODEL AND GML

A geospatial data model is a collection of geographic feature structures and types, and general integrity constraints. Geospatial data models used in relational databases often provide abstract mechanisms that are carried forward into the programming language and the software engineering environment [6]. However, relational data models usually cannot convey the real structures of geographic features, while the use of different terminologies and different concepts of space may lead to misunderstanding.

GML has three main roles with respect to geographic information: 1), as an encoding for the transport of geographic information from one system to another; 2) as a modeling language for describing the types of geographic information; and 3) as a storage format for geographic information [11]. Furthermore, GML application schemas are used to describe the relations between the domain knowledge of geospatial objects. If the geographic data is very detailed, the relationships in the data are often complicated such that relational databases are unable to process the data. Therefore, mapping a complex geospatial dataset to the schemas of relational databases can be a difficult task.

2.1 Geospatial Data Model

GML allows a great deal of flexibility in the encoding of geospatial datasets because it is possible to design one's own markup tags or document structures to describe geographic features and their geometry, as well as non-geospatial information.

For example, the geographic *footprint* that represents the location is required information in digital gazetteers [7]. Often, the geometric point type is used to represent the footprint of a named place. Since GML 3.0 supports different geographic features by using flexible geometric types, such as points, curves, surfaces, and solids, the footprint of a place can be expressed more clearly though an accurate feature-geometry relationship, such as "extentOf", "centerLineOf" and "position".

We mapped the Alexandria Digital Library (ADL) gazetteer feature type thesaurus to both relational database schemas and GML application schemas, but found that the database schemas were too complicated to be useful. It was also difficult to design a user-friendly interface to access and update the database. However, the GML application schemas were more natural and easier to use with ADL .

There are several advantages of using GML 3.0 to store geospatial datasets. First, GML is a meta-language; therefore, encoding the feature types of named places in GML can include

```
<Streams><streamsMembers>
   <Rivers><riversMembers>
      <FootPrint id="21001000000-11">
        <name> River-11 </name><length> 1603m </length>
        …
        <multiCenterLineOf>
          <MultiCurve><curveMembers><LineString>
               <coordinates>302745.435072,2442803.380075 ...
302882.868766,2442187.96039</coordinates>
             </LineString></curveMembers></MultiCurve>
        </multiCenterLineOf>
      </FootPrint>
    </riversMembers></Rivers>
</streamsMembers></Streams>
<Roadways><roadwaysMembers>
   <GeneralRoad><generalRoadMembers>
      <FootPrint id="4230904000-31">
        <name> Road-31 </name><length> 614m </length>
        …
        <multiCenterLineOf>
          <MultiCurve><curveMembers><LineString>
               <coordinates>301219.608538,2442423.503538 ...
308235.358961,2434884.578507</coordinates>
             </LineString></curveMembers></MultiCurve>
        </multiCenterLineOf>
    </FootPrint> </generalRoadMembers></GeneralRoad>
</roadwaysMembers></Roadways>
```

Figure 1. A fragment of a GML instance.

```
1.   declare namespace my="http://www.sinica.edu.tw/";
2.   declare namespace gml="java:GML.XQGeoExtensions";
3.   declare namespace svg="java:GML.XQSVGExtensions";

4.        declare function my:get_geo1() as element() {
5.           for $var1 in doc("lanyu.xml")//Rivers//FootPrint[@id =
             "21001000000-11"]
6.        return <result1>{$var1}</result1> };
7.        declare function my:get_geo2() as element() {
8.           for $var1 in doc("lanyu.xml")//Roadways//FootPrint[@id
             ="4230904000-31"]
9.        return <result1>{$var1}</result1> };
10.       declare function my:get_building() as element() {
11.       <result>{
12.          for $var1 in doc("lanyu.xml")//Buildings//FootPrint
13.          return $var1
14.       }</result>};

15. svg:GML2SVG( gml:Buffer(
    gml:Intersection(my:get_geo1() ,my:get_geo2() ), 50))

16. svg:GML2SVG(gml:queryByBox(299000,2430000,
    312000,2445000, my:get_buildings()))
```

Figure 2. An XQuery expression with geospatial extensions.

enriched non-spatial information, such as multilingual place names. Second, the ADL feature type is the basis for a catalog of named places that can include diverse information about many places, such as their names, footprints, and inter-relations. This is more suitable for the flexibility of the GML document structure. Third, it is easier to incorporate rich semantics and object-property rules, such as the “is-a” class relationships and the “part-of” object relationships, into GML. For example, by reading a GML application schema, we can understand that the term “schools” is a sub-class of the “buildings” category. Embedding such object-property rules in GML documents may help enhance spatial reasoning applications.

GML 3.0 allows data providers to encode as much geospatial and non-geospatial information in GML documents as they wish. However, the resulting GML documents may be too complicated to be mapped into relational tables in database systems. Therefore, efficient GML-native query processors or processing tools are needed to manipulate such documents. We discuss these issues in Section 3.

2.2 A GML Instance

According to our GML application schema, we encode many types of geographic features, such as rivers, roads, and buildings, in a single GML document. A fragment of a GML instance is shown in Figure 1. Every geographic feature is enclosed by a *FootPrint* element, which can be identified by its id attribute. The ancestor nodes of the feature are its types; for example, the geographic feature *River-11* is a River, which is a sub-class of the Stream class. The descendant nodes of the feature describe its details; for example, the length of *River-11* is 1,603 meters.

3. GML-NATIVE PROCESSORS

This section describes the principle and implementation of four GML-native processors: *GeoXQuery*, *GeoSAX*, *GPXQuery*, and *GPSAX*.

3.1 *GeoXQuery*: A GML Query Engine

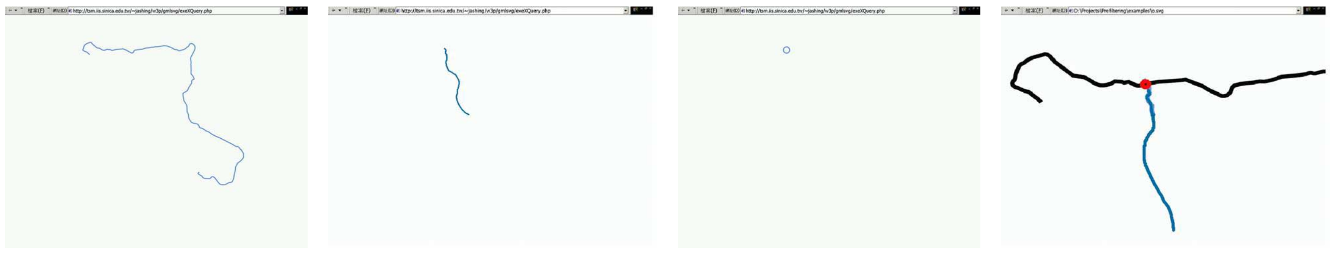
GML data is XML encoded geospatial information. Although XQuery [17], a W3C standard XML query language, provides a rich set of elements, attributes, text data, and alpha-numeric operations to manipulate XML documents, it is still inappropriate for querying and analyzing geospatial information. To fill this gap, we have developed a GML query engine, called *GeoXQuery*. We extend an XQuery processor to support geospatial functionalities by calling external geospatial function libraries.

In our implementation, we extend the Saxon Java XQuery processor [13] by calling geo-extension function libraries that provide all the geospatial operations described in [3]. The libraries are based on the JTS Topology Suite [10], an open source Java API that implements a core set of geospatial data operations, geometric data models, and robust geometric algorithms. In fact, we also implement a GML-to-SVG transformation library for the XQuery processor. Both extensions can be used without modifying or recompiling the source code of the XQuery processor.

Figure 2 shows an XQuery expression for calculating the intersection, buffering, and GML-to-SVG transformation (in line 15) of a river (in lines 4-6) and a road (in lines 7-9) in the document lanyu.xml (Lanyu is a small island close to Taiwan). The results are shown in Figure 3. Note that the operators in line 15 are provided by the geo-extension function libraries declared in lines 2 and 3. A prototype and its source code can be accessed at http://www.iis.sinica.edu.tw/~jashing/geoxquery/.

3.2 *GeoSAX*: A Streaming GML Parser

We also implement *GeoSAX*, an XML/GML streaming parser that supports geospatial functionalities by extending Sun’s SAX parser (provided by Sun’s Java XML API). The *GeoSAX* parser triggers standard SAX-events while parsing a document. Developers need to use those SAX events to store a geographic feature (a GML fragment enclosed by a *FootPrint* element) in an array. They can then use the geo-extension function libraries to transform the array into a geometric object. The libraries support geospatial operations, such as intersection, union, and buffer.



(a) A road. (b) A river. (c) The results of buffering the intersection of the road and the river. (d) Combining and recoloring (a), (b), and (c) in a SVG map. (Note that the map has been enlarged.)

Figure 3. An example of intersection of a road and a river in SVG.

3.3 Syntax of the XQuery Geo-extensions

According to the XQuery grammar [17], a function call should consist of a name and an optional argument list containing one or more expressions separated by commas. The type of a function parameter or return value is either a sequence type or a schema type. Since the geospatial function libraries provided by the JTS Topology Suite use WKT (Well Known Text) as the input format to generate geometric objects, a GML-to-WKT transformation is required before calling the libraries, as shown in line 15 of Figure 2. The parameters typed as a sequence (*i.e.*, GML fragments) in the function *gml:Intersection* are converted into WKT format and used to generate a geometric object. In contrast, the output of the intersection function typed as geometric is first converted into WKT and then transformed into the sequence type. Therefore, the syntax of the XQuery geo-extension function libraries is the same as that of the geospatial operators in [3]; it also complies with the syntax of the function call defined in the XQuery grammar.

3.4 An XML/GML Prefiltering Technique

Ordinary XML query processors (*e.g.*, XPath and XQuery), as well as *GeoXQuery*, generally have difficulty in dealing with large XML documents. Typically, the amount of memory they use to build a DOM tree in the main memory is five times the size of the original document. To reduce the memory consumption by XPath/XQuery processors when answering queries about large documents, we enhance XPath/XQuery processors by employing an XML prefiltering technique (*XPT*). Essentially, the *XPT* uses information retrieval technology to addresses interesting XML fragments by approximately evaluating users' XPath/XQuery expressions. It can refine an XML document into a smaller candidate-set XML document, which is then fed into XPath/XQuery processors to calculate exact answers. In addition, we have developed a geospatial indexing plug-in module (*GIPM*) so that the *XPT* can also perform geospatial filtering functionality. *GIPM* indexes the boundary of each geographic feature in the documents and provides an intersection operator to query indexed features.

The XML prefiltering framework consists of five major modules: the Indexer, the Query Simplifier (*QS*), the Fast Lightweight Steps-Axes Analyzer (*FLISA*), the Fragment Gatherer (*FG*), and the Micro XML Streaming Parser (*MXSP*). The *Indexer*, a preprocessing module, scans the XML document and constructs an inverted index table that is used by *FLISA* to evaluate users' XPath expressions. Each record in the table has two fields: the name and the position list. The value of the name field is either an element name or an attribute-value string. The value of the position list is an ordered list, in which each element is a triple (*start-tag position*, *end-tag position*, *height*), *i.e.*, the starting- and ending-byte offsets of an element (or an attribute), and the height of the node. In the prefiltering process, the user's application issues an XPath expression to the *QS*. After *QS* simplified the XPath expression by removing some internal XPath steps, *FLISA*, a tiny and efficient search engine, determines the candidate fragments by evaluating the simplified XPath expression. The fragments are either transformed into a series of SAX-events by *MXSP* (an interactive streaming parser) or gathered into a candidate-set XML document by *FG*. Details of the prefiltering technique are given in [8].

Unlike the *XPT* indexes structural information, the *GIPM* indexes text data (*e.g.*, the coordinates of geographic features) followed by certain GML structures (*e.g.*, Point, LineString, MultiCurve, and Polygon). Each geographic feature is simplified as a bounding box, which is defined as two points (minimum X, minimum Y) and (maximum X, maximum Y), to reduce the size of the indexes and rapidly identify a candidate-set of geographic features. In addition, a candidate-set intersection process is used to derive more accurate answers, each of which is selected by both *XPT* and *GIPM*. The refined answers are then proposed as the output of the prefiltering process. A prototype of the XML prefiltering technique is demonstrated in [9]. The prototype and its source codes can be accessed at http://www.iis.sinica.edu.tw/~jashing/prefiltering/.

3.5 Supported Prefiltering Constraints

We observe that users usually request parts of maps that meet certain query constraints. Four commonly used query constraints are: structural constraints (*e.g.*, specifies a feature class), predicate constraints (*e.g.*, specifies the ID of a feature), *bounded-by* constraints (*e.g.*, specifies a viewbox, which is a map selection tool defined by a rectangle), and mixed constraints (*i.e.*, combinations of the above types). Figure 2 shows an XQuery expression (user's query) that includes all of the above constraints. We use it to demonstrate how *XPT* and *GIPM* can help process the constraints.

First, the XPath expression (including predicates, such as [@id="21001000000-11"]) specifying certain geographic feature

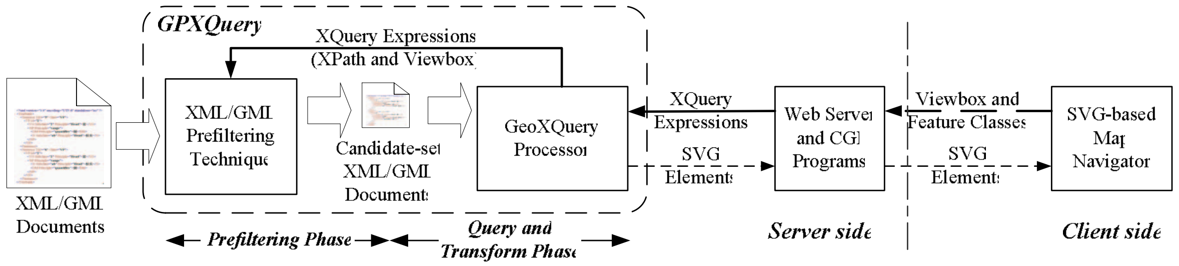

GPXQuery
XQuery Expressions (XPath and Viewbox)
XML/GML Documents
XML/GML Prefiltering Technique
Candidate-set XML/GML Documents
GeoXQuery Processor
XQuery Expressions
SVG Elements
Web Server and CGI Programs
Viewbox and Feature Classes
SVG Elements
SVG-based Map Navigator
Prefiltering Phase
Query and Transform Phase
Server side
Client side

Figure 4. The system architecture of a GML-based Web GIS.

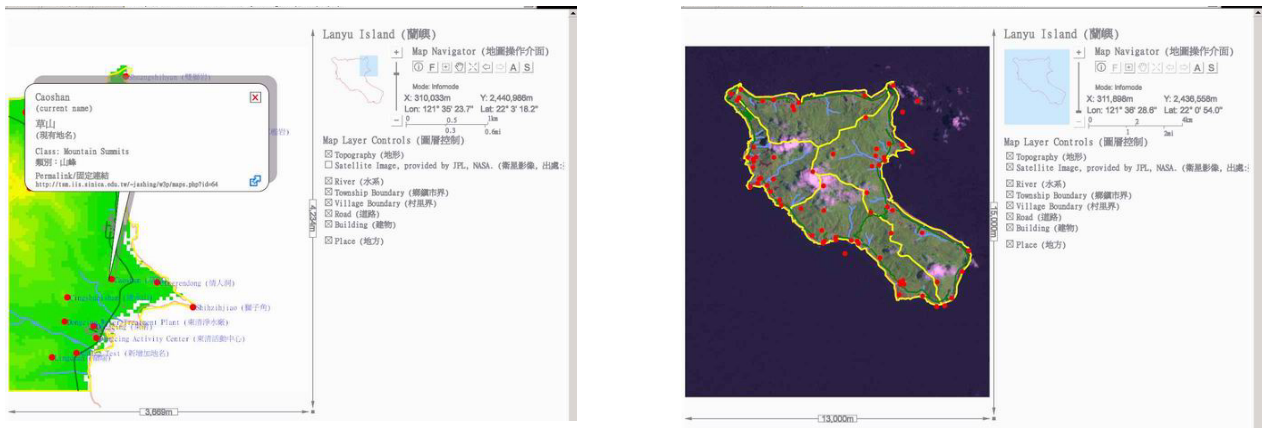

Lanyu Island (蘭嶼)
Map Navigator (地圖操作介面)
Map Layer Controls (圖層控制)
Topography (地形)
Satellite Image, provided by JPL, NASA.
River (水系)
Township Boundary (鄉鎮市界)
Village Boundary (村里界)
Road (道路)
Building (建物)
Place (地方)
Caoshan
草山
Class: Mountain Summits
3,869m
13,000m

Figure 5. Two snapshots of the GML-based Web GIS.

classes (*e.g.*, roads, rivers, and buildings), is evaluated by *XPT*. Second, the *bounded-by* constraint, *i.e.*, the four parameters of the function *queryByBox*, is used by *GIPM* to perform spatial prefiltering. Finally, the mixed constraints can be handled by applying an additional candidate-set intersection process to the outputs of the *XPT* and *GIPM*.

3.6 *GPXQuery*: A Prefiltering GML Query Engine

We have developed *GPXQuery*, an efficient GML query engine that manipulates voluminous GML documents by integrating the *GeoXQuery* processor with the *XPT* and *GIPM*. The integration was finished by adding just a few instructions to the source code of an XQuery processor, because the *XPT* is designed as a non-intrusive software module that can work transparently with existing applications [8]. Therefore, existing XML tools/applications can integrate the *XPT* with minimal modifications and need not consider its mechanisms.

Figure 4 shows the system architecture of a GML-based Web GIS employing the *GPXQuery* processor. We can see that the *GPXQuery* is not only a query engine, but also a GML-to-SVG transformer, in the system. In *GPXQuery*, the *GeoXQuery* processor only sends XQuery expressions to the *XPT* to generate candidate-set GML documents that are actually used to evaluate exactly matching geographic features for the SVG-based map navigator. Note that the *GPXQuery* processor transforms the matched features into SVG elements by calling an external SVG transformation library (*i.e.*, in line 3 of Figure 2).

3.7 *GPSAX*: A Prefiltering GML Streaming Parser

The *XPT* provides an interactive XML streaming parser (*i.e.*, the *MXSP*) that parses XML documents in a random access manner. We extend the parser to support geospatial functionalities. We call the enhanced parser *GPSAX*.

Unlike the *GPXQuery* query engine, *GPSAX* is a GML-aware streaming parser that can help remove some uninteresting GML fragments by evaluating user queries approximately. However, since *GPSAX* cannot answer user queries directly, users need to employ a query algorithm to calculate exact answers. That is, *GPSAX* is more suitable for answering simple queries that contain only XPath expressions with forward only step axes and/or viewbox constraints.

3.8 Cached Results

The candidate-set XML/GML document fragments generated by the *XPT* can be stored in a cache for later XQuery processing. As shown in Figure 4, the *XPT* generates a signature for each XQuery expression and maps that signature to the name of the candidate-set XML/GML documents. The *XPT* can then respond more quickly to the previously received queries.

4. A PROTOTYPE

Figure 5 shows two snapshots of a prototype of a GML-based Web GIS that is accessible at http://www.iis.sinica.edu.tw/~jashing/gmlgis/. Its system architecture is shown in Figure 4. Currently, at the server side, the system uses GML as the language for storing geographic datasets, and employs the *GeoXQuery* processor as its query engine. We also use the *GPXQuery* processor to develop a GML-based Web GIS, which is accessible at http://www.iis.sinica.edu.tw/~jashing/gmlgis_pf/. In addition, a Web GIS of the same interface, but based on a relational database, is accessible at http://www.iis.sinica.edu.tw/~jashing/dbmaps/. The three implementations are made available on the Web so users can freely test their functionality and evaluate their performance. For demonstration purposes, we only use small datasets at the server-side for the above three URLs. However, the performance evaluation and analysis described in Section 5, are conducted using large datasets. The datasets we used are transformed from SEF, a data exchange standard for topographic maps in Taiwan, to GML. The details of the SEF-to-GML and a GML-to-SVG transformation are given in [2].

At the client side, the Web GIS is an SVG-based map browser. SVG (Scalable Vector Graphics) is an XML-based 2D graphics markup language that is being used increasingly for vector, lossless, Web graphics. The client-side map browser renders SVG-marked up 2D geometric shapes, which are transformed from the GML-marked up geographic features retrieved by the GML-native processors. After a user specifies the geographic feature types and the bounding viewbox at the client-side, the GML-native processor at the server-side queries the GML-based geospatial datasets and returns the matched geographic features. The map browser provides map navigation functionalities, such as panning, zooming in/out, and searching by geographic names. The SVG-based geographic features can be overlaid on raster images provided by WMS (Web Map Service). The map in the browser can also be annotated. Note that the user's annotations as well as the geographic names are currently stored in a relational database. Only the geographic features are stored in GML. The client-side implementation of the map browser relies heavily on an existing open source SVG-based web-mapping system [15]. The GML-based server-side implementation, however, is completely new and is the subject of this paper.

5. PERFORMANCE ANALYSIS

We evaluated the performance of both the streaming-based and memory-based GML processors, with and without the prefiltering technique, by applying four types of query constraints to two large GML documents.

5.1 The Execution Environment and Datasets

Our setup was an Intel Pentium-4 PC running at 2.53 GHz with 1GB DDR-RAM, a 120GB EIDE hard disk, and the MS Windows 2000 server OS; and our programs were implemented in Java 2 (Standard Edition V.1.4.2). We tested both the streaming-based and memory-based XML processing models, with and without the prefiltering technique. That is, four GML processors were tested: *GeoXQuery*, *GPXQuery*, *GeoSAX*, and *GPSAX*. The datasets we used were a 1.2 GB GML document (the Taipei city map) and a 152 MB GML document (the Sinyi area, a subset of the 1.2GB Taipei city document). Table 1 summarizes the characteristics of these datasets. In addition, four types of query constraints were used to test their performance. As shown in Table 2, they were structural constraints (*e.g.*, specified by an XPath expression), predicate constraints (*e.g.*, specified by a predicate of the XPath expression), *bounded-by* constraints (*e.g.*, specified by a viewbox), and mixed constraints. The experiment results are shown in Table 3 and Table 4. In the tables, we use ***Q***, ***P***, ***D***, ***R***, and ***M*** to refer to the benchmarking ***Q***ueries, ***P***rocessors, ***D***atasets, ***R***unning time (in second), and ***M***emory usage (in Megabyte). The queries are denoted by the identifiers of query constraints in Table 2 separated by "&". The processors are denoted by their prefixes. The *Geo-*, *gipmGP-*, and *GP-*, in Table 3/Table 4, refer to *GeoXQuery*/*GeoSAX*, *GPXQuery*/*GPSAX* enabling the geospatial indexing plug-in module (*GIPM*), and *GPXQuery*/*GPSAX* disenabling the *GIPM*. In the *D* fields, we use S and T to refer to Sinyi and Taipei dataset, respectively. Note that the running time reported in the paper is the average of five runs.

Table 1. Datasets

Datasets	Sinyi (S)	Taipei (T)
Document size (MB)	152	1,147
Document depth	15	15
# of nodes (counting all start- and end-tags and text sections)	9,395,897	53,604,633
# of geometry	211,192	1,893,692
Bounded by (X min, Y min)-(X max, Y max)	(304796, 2766898)-(309577, 2772274)	(294400, 2761500)-(316800, 2789600)
Time for building the DOM tree (s)	38	N/A
Memory usage of the DOM Tree (MB)	1032	N/A
Time for parsing the document using the interactive SAX in the XML prefiltering module (s)	1260	12368
Time for parsing the document using Sun's SAX parser (s)	9	67
Size of the index file (MB)	49	441
Time for indexing (s)	192	1997

Table 2. The benchmarking queries.

Feature Class	**XPath Expression**	**Size of the Fragment**
FC_1	/TGML/ThemeGroup/BuildingGroup/BuildingMembers/Building/FootPrint	In Sinyi: 40MB in Taipei: 206MB
FC_2	/TGML/ThemeGroup/EnergySupplyUtilityGroup/EnergySupplyUtilityMembers/EnergySupplyUtility/FootPrint	In Sinyi: 8KB in Taipei: 28KB
Predicate	**XPath Expression with a Predicate**	
P	//FootPrint [@id="18749"]	
Viewbox	**Coordinates**	**Range** (m^2)
V_1	(294400, 2761500)-(316800, 2789600)	22400 * 28100
V_2	(300000, 2767100)-(309775, 2782575)	9775 * 15475
V_3	(304796, 2766898)-(309577, 2772274)	4781 * 5376
V_4	(305991.25, 2768093.25)-(308233.0, 2770930.0)	2241.75 * 2836.75

The experiments show that the improvement in performance depends to a large extent on the amount of data matched by a test query. If after prefiltering, only a few fragments are selected from a document, *i.e.*, a lot of fragments are filtered out, the performance improvement can reach ten-fold. In other words, the performance improves the most if one is searching for a specific element node (*e.g.*, querying an object by its ID), searching a small area (*e.g.*, querying by a small viewbox), or searching a feature class that only contains a few features.

5.2 Performance Analysis of XQuery-based Processors

The experiment results of testing XQuery-based processors are reported in Table 3. In general, the processors using XML prefiltering technique achieve a better performance than the *GeoXQuery* processor. They can even answer some queries that *GeoXQuery* cannot handle; for example, the queries that were directed to the Taipei dataset. The *GeoXQuery* processor cannot accommodate documents of this size in the main memory, and thus cannot answer the queries completely. For the queries directed to the Sinyi dataset, *gipmGP-* and *GPXQuery* achieve a 20% ~ 1000% performance improvement over *GeoXQuery*. Obviously, for queries that require access to only a small part of a document to get results, the prefiltering technique can greatly improve the performance; the queries with the *P* constraint are such cases.

5.3 Performance Analysis of SAX-based Processors

The experiment results of testing SAX-based processors are reported in Table 4. The results also show that the processors using XML prefiltering technique achieve a better performance than the *GeoSAX* processor if the retrieved document fragments are small. The queries with the *P* or V_2 constraints are such examples. In contrast, because our interactive SAX parser does not perform well (see Table 1), *gipmGP-* and *GPSAX* cannot benefit from prefiltering the GML documents if the candidate-set XML/GML documents are still large; the queries with FC_1 and/or V_1 constraints applying on Taipei dataset are such cases.

5.4 Characteristics Analysis

Table 3 and Table 4 show the characteristics of the compared processors, as well as those of the prefiltering technique. First, SAX-based processors are more suitable for dealing with simple queries against very large documents since they consume relatively less memory compared to XQuery-based processors. However, XQuery-based processors, as mentioned in Section 3.7, are more practical than SAX-based processors because applications can use and extend the XQuery language to query/transform XML documents in flexible ways. In contrast, SAX-based processors can only be used to trigger SAX events while parsing the documents. Therefore, applications have to employ additional query/transformation procedures to perform complicated query/transformations. In other words, using SAX-based processors to manipulate XML/GML documents increases the programmer's workload. Second, the efficiency of the current interactive SAX parser needs to be improved. For queries using the FC_1 constraint, although about 75% of the source documents are filtered out, the overall running time is not reduced. However,

Table 3. The performance results of XQuery-based processors.

Q	***P***	***D***	***R* (s)**	***M* (MB)**	***Q***	***P***	***D***	***R* (s)**	***M* (MB)**
V_3	*Geo-*	S	N/A	N/A	V_4	*Geo-*	S	N/A	N/A
V_3	*gipmGP-*	S	N/A	N/A	V_4	*gipmGP-*	S	196	958
V_3	*GP-*	S	N/A	N/A	V_4	*GP-*	S	N/A	N/A
FC_1&V_3	*Geo-*	S	115	1,033	FC_1V_4	*Geo-*	S	116	1,033
FC_1&V_3	*gipmGP-*	S	115	511	FC_1V_4	*gipmGP-*	S	36	168
FC_1&V_3	*GP-*	S	112	575	FC_1V_4	*GP-*	S	116	575
FC_2&V_3	*Geo-*	S	22	655	FC_2&V_4	*Geo-*	S	21	655
FC_2&V_3	*gipmGP-*	S	2	39	FC_2&V_4	*gipmGP-*	S	2	39
FC_2&V_3	*GP-*	S	2	39	FC_2&V_4	*GP-*	S	2	39
V_1	*Geo-*	T	N/A	N/A	V_2	*Geo-*	T	N/A	N/A
V_1	*gipmGP-*	T	N/A	N/A	V_2	*gipmGP-*	T	N/A	N/A
V_1	*GP-*	T	N/A	N/A	V_2	*GP-*	T	N/A	N/A
FC_1&V_1	*Geo-*	T	N/A	N/A	FC_1&V_2	*Geo-*	T	N/A	N/A
FC_1&V_1	*gipmGP-*	T	N/A	N/A	FC_1&V_2	*gipmGP-*	T	N/A	N/A
FC_1&V_1	*GP-*	T	N/A	N/A	FC_1&V_2	*GP-*	T	N/A	N/A
FC_2&V_1	*Geo-*	T	N/A	N/A	FC_2&V_2	*Geo-*	T	N/A	N/A
FC_2&V_1	*gipmGP-*	T	19	298	FC_2&V_2	*gipmGP-*	T	12	298
FC_2&V_1	*GP-*	T	10	305	FC_2&V_2	*GP-*	T	10	305
P	*Geo-*	S	42	1,065	*P*	*Geo-*	T	N/A	N/A
P	*gipmGP-*	S	6	179	*P*	*gipmGP-*	T	38	179
P	*GP-*	S	5	305	*P*	*GP-*	T	32	305

* The labels ***Q***, ***P***, ***D***, ***R***, and ***M*** refer to the benchmarking ***Q***ueries, ***P***rocessors, ***D***atasets, ***R***unning time, and ***M***emory usage. N/A means that the processor run out of memory and did not finish.

Table 4. The performance results of SAX-based processors.

Q	***P***	***D***	***R* (s)**	***M* (MB)**	***Q***	***P***	***D***	***R* (s)**	***M* (MB)**
V_3	*Geo-*	S	195	12	V_4	*Geo-*	S	158	18
V_3	*gipmGP-*	S	1,420	98	V_4	*gipmGP-*	S	512	57
V_3	*GP-*	S	1,394	76	V_4	*GP-*	S	1,343	57
FC_1&V_3	*Geo-*	S	103	2	FC_1V_4	*Geo-*	S	85	2
FC_1&V_3	*gipmGP-*	S	94	61	FC_1V_4	*gipmGP-*	S	30	37
FC_1&V_3	*GP-*	S	92	38	FC_1V_4	*GP-*	S	81	38
FC_2&V_3	*Geo-*	S	58	2	FC_2&V_4	*Geo-*	S	57	2
FC_2&V_3	*gipmGP-*	S	2	39	FC_2&V_4	*gipmGP-*	S	1	39
FC_2&V_3	*GP-*	S	1	39	FC_2&V_4	*GP-*	S	1	39
V_1	*Geo-*	T	1,183	273	V_2	*Geo-*	T	1,008	243
V_1	*gipmGP-*	T	14,536	770	V_2	*gipmGP-*	T	5,968	767
V_1	*GP-*	T	14,341	692	V_2	*GP-*	T	13,802	489
FC_1&V_1	*Geo-*	T	1,182	273	FC_1&V_2	*Geo-*	T	1,008	243
FC_1&V_1	*gipmGP-*	T	14,534	770	FC_1&V_2	*gipmGP-*	T	5,968	767
FC_1&V_1	*GP-*	T	14,341	692	FC_1&V_2	*GP-*	T	13,802	489
FC_2&V_1	*GXQ*	T	242	2	FC_2&V_2	*GXQ*	T	249	2
FC_2&V_1	*gipmGP-*	T	11	298	FC_2&V_2	*gipmGP-*	T	11	298
FC_2&V_1	*GP-*	T	9	305	FC_2&V_2	*GP-*	T	9	305
P	*Geo-*	S	50	2	*P*	*Geo-*	T	233	2
P	*gipmGP-*	S	5	23	*P*	*gipmGP-*	T	36	298
P	*GP-*	S	4	39	*P*	*GP-*	T	29	305

* In order to verify the query results, all the parsed nodes were saved to a file. Thus, the cost of disk I/O has been included in the running time.

for queries that only select a few nodes from documents, *e.g.*, queries using the *P* or FC_2 constraints, the prefiltering technique can greatly improve performance. Third, the *GIPM* of the prefiltering technique can efficiently filter out uninteresting geographic features with a little additional effort, as shown by the queries with V_4 constraint applying on Sinyi dataset are such cases. However, for queries that only select a few nodes, which are retrieved based on document structural information, using

GIPM is not appropriate, as shown by queries using the *P* constraint. Finally, the prefiltering technique may reduce resource consumption, particularly in memory usage and disk access time. XQuery-based processors need much more memory (often five times the size of the document) to accommodate the DOM tree representation of the document; therefore, prefiltering the document would save a substantial amount of memory. Moreover, because the number of coordinates of a geographic feature may be in the order of tens to hundreds of kilobytes, not assessing unnecessary features would reduce disk access time.

6. CONCLUSION AND FUTURE WORK

In the future, we will address a number of issues related to this research. First, in terms of data storage and querying, the GML-based GIS reported in this paper is still impractical for dealing with gigabyte-sized geospatial datasets compared to GIS based on relational databases. Although the index scheme used in the XML prefiltering technique is simple and efficient, the index may still be large. A more efficient index management system or data structure would help reduce the cost of retrieving the index. Second, we currently require that the geographic datasets should not change frequently, because each change requires rebuilding the index. So far, there is no efficient way to update XML documents incrementally. More studies on this topic are needed. It remains to be seen if a two-phase XML updating model could be developed along the lines of the two-phase XML query model. Finally, as GML can represent both geospatial and non-geospatial information in a single document very easily, it raises the issue of how to better process documents of this nature. Are there novel geospatial operations based on the GML data/application models? Are these operators helpful in manipulating maps in Web GIS? For example, as a GML application schema can be quite complicated and, at the same time, reveals a lot of structural information about its instance documents, it may be possible to devise schema-level operators specific to the documents of a particular GML application schema.

7. ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their comment and suggestion which greatly improves the organization and presentation of this paper. Chia-Hsin Huang and Hahn-Ming Lee are both supported, in part, by the National Digital Archive Program, which is funded by the National Science Council of Taiwan (grant no. NSC 95-2422-H-001-007). Chia-Hsin is a PhD candidate at the National Taiwan University of Science and Technology and is supported by a fellowship from the Institute of Information Science, Academia Sinica. This paper outgrows from a joint work of the Open Geospatial Information Team (http://guava.iis.sinica.edu.tw/) at the Institute; the first three authors are indebted to team members Chin-Lung Chang, Yi-Hong Chang, and Andrea Wei-Ching Huang for many stimulating discussions and ideas on geospatial information processing, and they wish to thank Winnie Chang for invaluable logistic support. The prototype as described in Section 4 is built upon the software released by Juliana Williams and Andreas Neumann at www.carto.net; the authors are most grateful for their kindness. Any mistake, omission, or inaccuracy in the paper remains the sole responsibility of the authors.

8. REFERENCES

[1] O. Boucelma and F.M. Colonna, GQuery: a Query Language for GML, In *Proc. of the 24th Urban Data Management Symposium*, 2004, pp. 27-29.

[2] C. L. Chang, Y. H. Chang, T. R. Chuang, S. Ho, and F. T. Lin. Bridging two geography languages: Experience in mapping SEF to GML. *In GML Dev Days: 2nd GML Developers Conference*. 2003.

[3] J. E. Córcoles, P. González, A specification of a spatial query language over GML, *in Proc. of the 9th ACM international symposium on Advances in geographic information systems*, 2001. pp. 112-117.

[4] S. Cox, P. Daisey, R. Lake, C. Portele, and A. Whiteside, editors. OpenGIS© Geography Markup Language (GML) Implementation Specification, Version: 3.00, 2003.

[5] DOM, World Wide Web Consortium. *Document Object Model* (*DOM*), W3C Recommendation.

[6] M. Egenhofer and A. Frank, Object-Oriented Modeling for GIS, *Journal of the Urban and Regional Information Systems Association*, no. 4, 1992, pp. 3-19.

[7] L. L. Hill, Core Elements of Digital Gazetteers: Placenames, Categories, and Footprints, In *Proc. of the 4th European Conference on Research and Advanced Technology for Digital Libraries*, 2000, pp. 280-290.

[8] C. H. Huang, T. R. Chuang, and H. M. Lee. Prefiltering techniques for efficient XML document processing. In *Proc. of the 2005 ACM Symposium on Document Engineering*, 2005, pp. 149-158.

[9] C. H. Huang, T. R. Chuang, J. J. Lu, and H. M. Lee. XML Evolution: Two-phase XML Processing Model Using XML Prefiltering Techniques. to appear in *32nd International Conference on Very Large Data Bases*, 2006.

[10] JTS Topology Suite, http://www.vividsolutions.com/.

[11] R. Lake, The application of geography markup language (GML) to the geological science, *Computers and Geosciences*, no. 31, 2005, pp. 1081-1094.

[12] D. Megginson. *SAX: A Simple API for XML*. http://www.saxproject.org/

[13] Saxon Java XQuery processor, http://saxon.sourceforge.net/

[14] Scalable Vector Graphics (SVG) 1.0 Specification, W3C Recommendation, 2001.

[15] J. Williams and A. Neumann, 2005, http://www.carto.net/papers/svg/ogc_wms_integration/

[16] XML Path Language (XPath), W3C Recommendation, 1999.

[17] XML Query (XQuery), W3C Candidate Recommendation, 2005

[18] G. Xu and X. Tong, GML and XQuery based cadastral spatial object query model description and implementation, in *Proc. of the IEEE International Geoscience and Remote Sensing Symposium*, 2004, pp. 2908-2911.