A Tightly-coupled XML Encoder-decoder For 3D Data Transaction

Commission III – Spatial Information Management

Siew Chengxi Bernad
Khairul Hafiz Sharkawi

3D GIS Research Lab, Faculty of Geoinformation
and Real Estate,
Universiti Teknologi Malaysia

Outline

• Introduction
• The Implementations
• Encoding Results
• Javascript Decoder
• Conclusions
• Acknowledgements
Introduction

• 3D spatial data sharing are getting popular in 3D SDI especially in urban modelling
• Large datasets file size is expected (>=LoD3 ~1Megabytes to 25MB each building)
• 3D data sharing use cases:
  1. Only visualizations?
  2. Visualizations + event-based semantics retrieval?
  3. Semantics + Geometries both presence for analysis

Introduction (cont.)

• Scenario 3 is not practical by sending CityGML data to clients
• Existing compression techniques does not solve the problem well:
  – Non-query-able
  – Non-streaming
  – General usages
  – No partial decompression
Introduction (cont.)

- Encoder-decoder is innovated for this purpose
- This coupling layer allows:
  - Query-able on compressed document
  - Hence allow partial decompression
  - Schema-aware, adapt to large CityGML geometries dataset
- An encoding framework within or with web services for spatial data transfer improves efficiencies

The Implementation

- Encoding based on 7 main components (elements, attributes, attribute values, values, geometries, URI)
- Geometries are encoded based on scaled integer.
- All components are stored in dictionary in uniform 16-bit symbols
- The encoder adapts non-progressive, lossless point, streaming and random access behavior
Scaled Integer

- Scaled Integer (To store geometry information with smaller size)

![Diagram showing scaled integers and stores]

Note: Example by using scaled integer – 100.02 can be stored in integer (4 bytes) instead of double (8 bytes)

Key components

<table>
<thead>
<tr>
<th>Components</th>
<th>Format</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Bit Representation</td>
<td>u_char</td>
<td>1 byte</td>
</tr>
<tr>
<td>2. Close</td>
<td>u_char</td>
<td>1 byte</td>
</tr>
<tr>
<td>3. Tag</td>
<td>u_char</td>
<td>1 byte</td>
</tr>
<tr>
<td>4. Attribute</td>
<td>u_char</td>
<td>1 byte</td>
</tr>
<tr>
<td>5. Attribute Value</td>
<td>u_char</td>
<td>1 byte</td>
</tr>
<tr>
<td>6. Value</td>
<td>u_char</td>
<td>1 byte</td>
</tr>
<tr>
<td>7. URI</td>
<td>u_char</td>
<td>1 byte</td>
</tr>
<tr>
<td>8. X</td>
<td>int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>9. Y</td>
<td>int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>10. Z</td>
<td>int</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>
Encoding workflow

XML Compression with geometry chunking in 16-bit symbol

Encoding results (filesize)
**Time taken (seconds)**

![Graph showing time taken (seconds)](image1)

**Improvements (using deflate as benchmark)**

![Graph showing improvements percentage](image2)
## Comparisons

<table>
<thead>
<tr>
<th>Techniques/Characteristics</th>
<th>Query-able</th>
<th>Partial decompression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deflate alone</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Bzip2 alone</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LZMA alone</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fast-Infoset</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CitySAC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

## Encoding environment

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel Core i7-3610M 2.3Ghz</td>
</tr>
<tr>
<td>Memory</td>
<td>6GB DDR3</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>1 TB HDD with 34.6GB free space on C:</td>
</tr>
<tr>
<td>Operating System</td>
<td>Windows 7 64-bit Professional on DOT.NET Framework 4.5</td>
</tr>
</tbody>
</table>
The Javascript Decoder

- HTML5 and Javascript is becoming popular.
- Javascript provides platform independent solution via interpreter eg. NodeJS and Browser
- Simple binary transformation and decompression could be done.
- Reading binary file is available in WebGL (little-endian and BigEndian)

The Javascript Decoder

- Query-able modules

  Or

- Full decompression to CityGML
  Compressed -> Partial decompressed (binary data) -> CityGML
The Javascript Decoder

Client Side / Recipient Side

Encoded Binary Data Stream

Decoded Binary Data Stream

Decoder Modules

Routing module
Geometry module
URI module
Values module

Elements module
Attributes module

DOM Cached Query

DOM Cached Queried Information

JAVASCRIPT DECODER FOR CITYGAC

PDF Reader

Application snapshots

DOM Cached Queryed Information

DOM Cached Queried Information

Application snapshots
Conclusions

- Schema-aware encoder able to encode CityGML into smaller size (35% - 50%) compare to Deflate (WinZIP)
- Schema-aware encoder able to encode CityGML into smaller size (15% - 30%) and faster compare to LZMA (7-zip)
- Javascript decoder able to provide code-on-demand solution
- Binary representation standard for GML tags?
- XML compression is preferred for large data transaction as partial decompression and query-able compressed document benefits are gained

Acknowledgment

- Financial assistance from Malaysian Peninsular Land Surveyors Board (LJT)
- MyPhD scholarship program from Ministry of Education
Thank you for your attention!

3D GIS Research Lab, FGRE, UTM

Email address:  
cbsiew2@live.utm.my  
or  
bernad@bernadsiew.name