Accelerating SVG Transformations with Pipelines

- XML & SVG
- Event Pipelines
- Technologies
- Recommendations

Eric Gropp
Lead Systems Developer, MWH Inc.
SVG Open 2003
XML & SVG In the Enterprise

- SVG can meet a wide variety of presentation and interface needs
- SVG is easily created from XML data streams
- Back-end data providers are then de-coupled from the end uses
- Transformations can quickly meet new needs

What it means to us:
- We can reuse solutions across the breadth of our clients’ platforms
- Open XML design simplifies maintenance for us and our clients
- We can solve many problems with a small toolkit
XML and SVG Transformations

- XSLT is simple, cross-platform, and easy to deploy
- Performance issues haunt XML implementations
Event Pipelines

- Process XML as events
  - “startElement”
  - “characters”
  - “endElement”
- Forward-only processing of XML
- Fast with small memory footprint

XSLT Processing (dramatization)

Pipeline Processing (dramatization)
What Event Pipelines Help

- Large input XML streams
- Speed or concurrency issues
- Computations
- Decoupled pre- or post-processing of XML

Event pipelines almost always use less memory and can be faster than other transformation methods.
Design Example
GML to SVG

• GML is a schema for geographic data and good candidate for pipelines

<table>
<thead>
<tr>
<th>Large Input XML</th>
<th>Shown above is 70k of SVG from 700k of GML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mathematical Operations</td>
<td>Vertex reduction is often needed</td>
</tr>
<tr>
<td>Speed and concurrency issues</td>
<td>Demand for dynamic web mapping</td>
</tr>
</tbody>
</table>
Technologies

SAX

• Forward only, event-driven XML Processing

<comment>
  SVG est potenca
</comment>

• startDocument()
• startElement("","comment","comment", attrs)
• characters([],"SVG est potenca",0,14)
• endElement("","comment","comment")
• endDocument()

XMLReader

Produces SAX events

ContentHandler

Receives SAX events

XMLFilter+

Receives SAX events and produces, removes, or alters SAX events
Sax filters can be strung together to form event pipelines:

- Filters can be reused or swapped out as necessary
- Filters can be used with other producers and consumers of SAX events:

Long or complex pipelines can affect performance
Technologies
SAX

- **Vertex Reduction And Scaling**: Performs Douglas-Peucker vertex reduction and scaling of all GML coordinate sets. Also good for preprocessing XSLT input.

- **Bounds and Property Filter**: Removes features based on bounding box or property criteria.

- **GML Feature Tiler**: Sends XML processing instructions to a file writing content handler for tiling SVG output to multiple files.

- **GML to SVG Transform**: Converts GML content to SVG content while applying pre-set style rules.
public void startElement(string namespaceURI, string localName, string qName, IAttributes atts) {
    if (namespaceURI == GmlUri && localName == "coordinates") {
        buffer = new StringBuilder();
    }
    childHandler.startElement(namespaceURI, localName, qName, atts);
}

public void characters(char[] ch, int start, int end) {
    if (buffer != null) {
        buffer.Append(ch, start, end);
    } else {
        childHandler.characters(ch, start, end);
    }
}
public void endElement(string namespaceURI, string localName, string qName) {

    if (buffer != null) {
        string strCoords = buffer.ToString();
        Point[] workArray = Utility.ParsePointArray(strCoords);
        if (workArray.Length > 2) {
            Point[] reducedPts = Utility.ReduceVertices(workArray, Tolerance);
            strCoords = Utility.PointArrayToString(reducedPts);
        }
        childHandler.characters(strCoords.ToCharArray(), 0, strCoords.Length);
        buffer = null;
    }

    childHandler.endElement(namespaceURI, localName, qName);
}
Technologies
.net’s XmlWriter

- Many XML producers in .net use XmlWriter or XmlReader
  - SqlXml
  - XmlSerializer
  - XslTransform
- Use an adapter that writes to a ContentHandler Interface
- Or derive a filter from XmlWriter/XmlReader directly
Technologies
.net’s XmlWriter

- XML Binding APIs can simplify pipeline assembly
- Use XmlSerializer and XSD.exe to create a schema for your pipelines
- A number of similar JAVA approaches are available

First filter performs vertex reduction

Second, nested filter performs GML to SVG conversion

```
<GmlFilterSet xmlns:xsi="...">
  <Filters>
    <Filter xsi:type="GmlVertexFilter" Tolerance="5">
      <Filter xsi:type="Gml2SvgFilter" Height="500" Width="500">
        <Style>
          .taxParcel {
            stroke:green;
            fill:#d0ffd0;
          }
          .regulatedUse {
            stroke:red;
            stroke-width:10;
            fill:none;
          }
        </Style>
      </Filter>
    </Filter>
  </Filters>
</GmlFilterSet>
```
Technologies

STX

- **Streaming Transformations for XML**
- http://stx.sourceforge.net
- Forward-only, XSLT-like, method for processing XML
- Potentially can deliver many of the advantages of pipelines without the pain

Mapping GML `<Polygon>` to SVG `<path>`

Conversion of GML Coords
## Recommendations

### Data Source

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Why</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Write directly to a serial XML API</td>
<td>• Compatible with XSLT and pipelines</td>
<td>• org.xml.sax.content-Handler interface</td>
</tr>
<tr>
<td></td>
<td>• Prevents unnecessary parsing</td>
<td>• System.Xml.XmlWriter</td>
</tr>
<tr>
<td>• Bias towards forward-only processing</td>
<td>• Caching can reduce pipeline benefits</td>
<td>• Avoid the DOM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• forward-only recordsets</td>
</tr>
<tr>
<td>• Use initial bounding elements</td>
<td>• Simpler filter development</td>
<td>• gml:boundedBy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Min and max values</td>
</tr>
</tbody>
</table>
## Recommendations

### Pipeline Setup

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Why</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filters should have valid XML output</td>
<td>For flexibility and reusability</td>
<td>All filters output valid GML or SVG</td>
</tr>
<tr>
<td>Avoid repeating complex deserializations</td>
<td>They are expensive</td>
<td>GML coordinate list functions are all in one filter</td>
</tr>
<tr>
<td>Avoid complex data structures</td>
<td>You may be better off using the DOM / XSLT</td>
<td>Large amount of data maintained in a filter</td>
</tr>
<tr>
<td>Compare to streamlined XSLT</td>
<td>XSLT may perform just as well</td>
<td>XSLT with simple, forward-only xpath</td>
</tr>
</tbody>
</table>
## Recommendations

### SVG Design

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Why</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use “style” attribute and CSS classes</td>
<td>Simplify filter style rule definition</td>
<td>All style rules can be defined in a single string</td>
</tr>
</tbody>
</table>
| Consider onload event handlers for fixing element order | Element order difficult to handle in filters | function onload_toTop(evt) {
  document.
documentElement.
appendChild(evt.target);
} |
| Take advantage of SVG’s “transform” & ”viewbox” attributes and coordinate representations | Coordinate transformations may be costly on the server, but easy on the client | viewBox="13370000 -425000 3000 5000"
<svg:path d="M100-100 200-100 200-200 100-200 100-100"/> |
Key Points

• XML Pipelines can produce SVG with speed comparable to source-specific SVG writers
• They provide the reuse and flexibility typical of XML based frameworks
• Use them parallel to or in sequence with XSLT and other approaches
• Consider and implement them carefully
• Thank you!

Bird images courtesy of the Nova Scotia Museum of Natural History Website