

Geometry transformations in SLD

Introduction

This page provides some background of the concept of geometry transformations in SLD, a tool that can be used to improve the rendering abilities and address some commonly requested rendering abilities.

Rendering chain and transformations

The idea behind transformations is simple, instead of drawing the geometry directly we plan to support its transformation into a different geometry by means of a filter function declared in the SLD. For example:

```
<PointSymbolizer>
  <Geometry>
    <ogc:Function name="endPoint">

<ogc:PropertyName>aLineGeometry</ogc:Propert
yName>
    </ogc:Function>
  </Geometry>
  ...
</PointSymbolizer>
```

Considering the rendering pipeline there are two steps in which a transformation can be injected. Here is a list of the rendering steps with transformations embedded into them:

- extract the geometry from the feature
- *first transformation operating in native coordinates (meters, degrees)*
- decimation, reprojection and affine transformation to screen coordinates (the coordinates are now expressed in pixels)
- *second transformation operating in screen space (pixels)*
- draw the geometry on screen

Both kinds of transformations are useful:

- real world transformations can be used to extract portions of the geometry, intersect or buffer it, offset and displace it in a way that makes the perceived offset zoom dependent
- pixel transformations are the ones that make the transformation zoom independent and can be used to move apart overlapping lines, dropping shadows and so on

The SE 1.1 specification seems to address, via a finite set of transformations, both of them, depending on the unit of measure used in the symbolizer.

This proposal is about going beyond, and having a pluggable extension point for people to create whatever

transformation they need.

Typical use cases

Geometry transformations can be used in a number of cases. Here are some example:

- drop shadow effect against polygons
- show start/end line decorations. Coupled with a function that extract the current line angle this can be used to draw arrows at the end of the line
- offset overlapping bus lines or offset overlapping points (both of these functions would need to be stateful for the duration of the rendering run)
- linear referencing systems: extract the bus stops out of single bus route, extract a subline representing just a leg in the full bus route
- fake 3d by extruding polygons to an isometric view (which would end up being a multipolygon) resulting in a Google maps like effect: <http://maps.google.com/?q=New+York,+NY&ie=UTF8&om=1&hq=&hnear=New+York,+Manhattan,+New+York&ll=40.748659,-73.98608&spn=0.002979,0.006968&z=18>

A tentative implementation

During the code sprint at FOSS4G 2009 a tentative geometry transformation implementation has been created that addresses only the first type of transformation, the one operating in real world coordinates.

The changes needed to support a simple version of it in the streaming renderer are not big, neither are those required in the parsing. Some API changes were needed so that `Symbolizer.getGeometry()` returns a generic Expression instead of a `PropertyName`.

The [patch to trunk](#) is attached to this page for review.

Besides the changes in the GeoTools Symbolizer interfaces the other new piece of API is the `GeometryTransformation` interface:

```

/**
 * This interface can be implemented by
 geometry transformation functions
 * that wish to be used in the {@link
 Symbolizer} geometry property.
 * <p>It gives the renderer a hint of what
 area should be queried given a certain
 rendering area
 * @author aaime
 *
 */
public interface GeometryTransformation
extends Function {
    /**
     * Returns a query envelope given a
 certain
     * @param renderingEnvelope
     * @return
     */
    ReferencedEnvelope
invert(ReferencedEnvelope
renderingEnvelope);
}

```

The interface should be implemented by all transformation that do change the location of the geometry during the transformation, and it's used by the renderer to guess the query area given the display area. For example, a buffer transformation will require an extended query area to catch those feature that are sitting out of the display area, but whose buffer crosses into the display area.

The renderer uses a visitor to get the full transformation (as transformations can be chained together, for example one could first perform an intersection and then buffer the result).

The patch attached also introduces samples of the above:

- the existing `FilterFunction_buffer` transformation has been modified to implement the

GeometryTransformation interface

- two new functions, FilterFunction_vertices and FilterFunction_pointAt, have been introduced to extract the vertices of a geometry and to get the point at a certain percentage of the line length (actually pointAt is incomplete, works only for perc=0 and perc=1 at this moment).

The patch also includes tests for the transformation machinery.

Looking into the future

The future of this work requires the implementation of more transformations and the ability to perform transformations in pixel space as well. This could be implemented, for example, by using a visitor that takes the transformation and wraps the geometry accesses within a further transformation that applies the generalization, reprojection and world to screen transformations.