Rendering Technologies Report

May 18, 2004

Submitted To: Program Manager
GeoConnections
Victoria, BC, Canada

Submitted By: Jesse Eichar
Refractions Research Inc.
Suite 400 – 1207 Douglas Street
Victoria, BC V8W 2E7
E-mail: jeichar@refractions.net
Phone: (250) 383-3022
Fax: (250) 383-2140

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

This document compares the available rendering technology that may be used by the User-friendly Desktop Internet GIS (uDig) project. The technologies reviewed and compared in this document are the two renderers that are part of the Geotools project (the j2d renderer and the Lite renderer), the JUMP renderer and the SWT OpenGL plug-in.

All of these technologies have desirable features and areas that need improvement. For example, Geotools j2d renderer has too large of a memory footprint but has very effective zooming and panning functions. This document lists the requirements of the uDig renderer and presents the strengths and weaknesses of each of the reviewed renderers.

2 uDig Renderer Requirements

The purpose of the uDig renderer is, at the most fundamental level, to take, features, coverages and Style Layer Descriptors (SLDs) as input and produce an image for the user to view. A more comprehensive list of requirements is as follows:

- **Feature Collections Must be Renderable**
  The uDig renderer must be capable of rendering collections of GIS Features. See OGC 01-101, Feature Geometry, for an introduction to features.
- **Raster Coverages Must be Renderable**
  Coverages, specifically grid coverages, must be renderable by the uDig renderer. The OGC 00-106 specification introduces Coverages and the related terminology.
- **SLD styling**
  It must be possible for Style Layer Descriptors to be applied to a feature collection for rendering. See OGC 02-070.
- **Output to a Graphics2D object**
  The renderer should be able to display the data graphically on a Java Panel.
- **Instant Feedback**
  A user of an application expects "instant gratification." As soon as a request is made the application should notify the user that the request is being made, and, if possible, some indication of how the operation is progressing should be provided.
- **Scalable for very large datasets**
  Very large datasets, potentially hundreds of megabytes, must be able to be rendered with a typical computer.
- **Reprojection**
  The renderer must have support for transforming between different coordinate systems.
- **Selection Integration**
  Selections should be styled in a different manner from other features and should be raised so that selections are not hidden. Further, the renderer must be able to identify a feature when given a pixel location from the display.
- **Optimized for common tasks**
  The common tasks such as panning and zooming should be efficiently implemented.
- **Accurate Rendered Image**
  The image rendered by the renderer must accurately, at least double accuracy (64 bit representation), represent the data passed to it.

3 RENDERING BACKGROUND

The OGC Style Layer Descriptor specification is a map-styling language for producing georeferenced maps with user-defined styling. SLD specifies how a feature is supposed to be displayed. Some options available are the colour of a feature, how a line should be drawn (solid, dashed, 3 pixels thick, etc...), whether the feature should be represented as a graphic and how features of its type should be shown in the map legend.

4 COMPARISON OF REVIEWED TECHNOLOGIES

	j2d	Lite	JUMP	OpenGL
Feature Rendering	yes	yes	yes	no
Coverage Rendering	yes	no	yes	yes
SLD Styling	yes	yes*	no	no
Graphics2D	yes	yes	yes	no***
Instant Feedback	no	no	no	no
Scalable	no	yes	no	yes
Reprojection	yes	no	no**	no
Selection Integration	no	no	yes	no
Optimized for Common Tasks	yes	no	yes	yes
Double Precision Accuracy	no	yes	yes	yes

* The Lite renderer does not provide SLD support for grid coverage rendering
** A new, and unreleased version of JUMP, provides support for Reprojection
*** SWT OpenGL renders to a SWT panel instead which could be useful

5 GEOTOOLS J2D RENDERER

The Geotools j2d renderer is an efficient rendering implementation, but has a very large memory footprint. The j2d renderer has been optimized for operations such as zooming and panning. In order to make these operations efficient, j2d maintains a cache of off-screen images and also caches the geometries and styling information of the features.

5.1 Basic Design

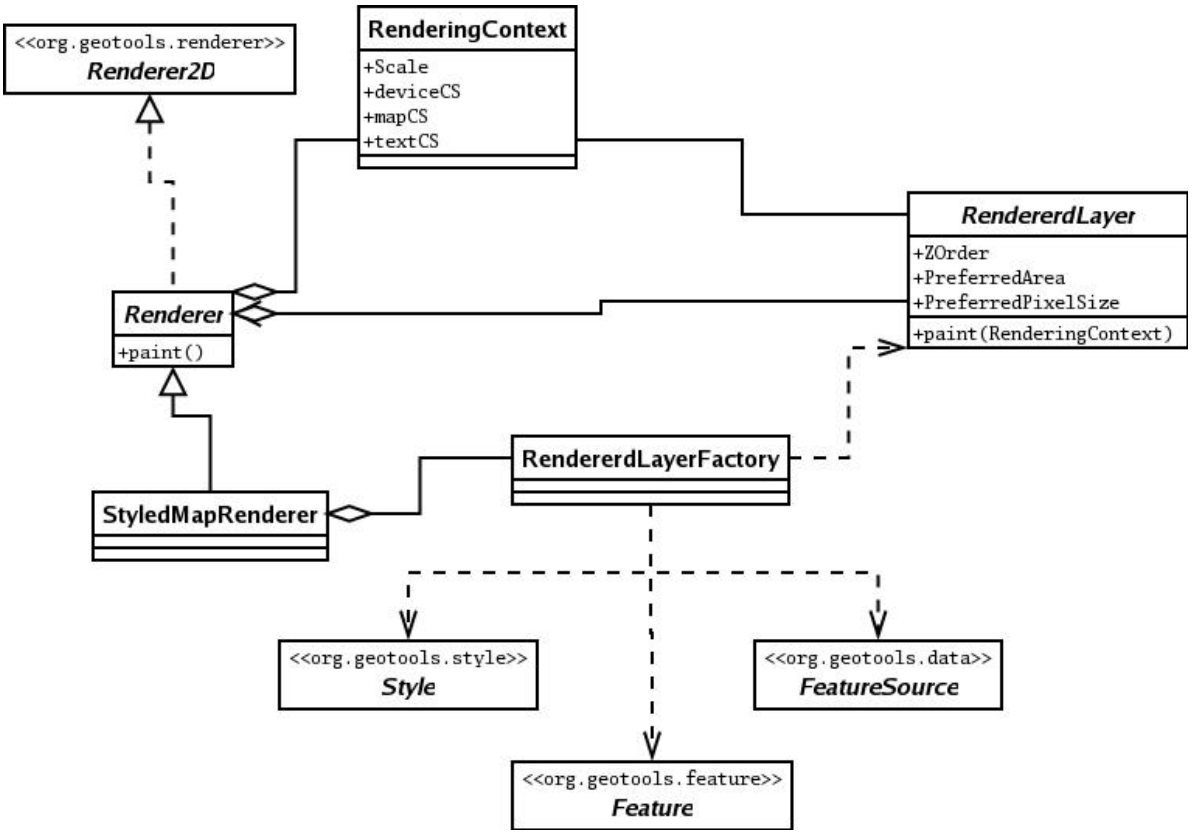

Figure 1: J2D Basic Architecture

The j2d renderer has a RenderingContext that holds the state of the renderer. For example, it holds the different coordinate systems that the rendering takes place in. The mapCS is the coordinate system that each layer is transformed into. The deviceCS is the device coordinate system. Each "unit" is a pixel of device-dependent size. The RenderingContext also encapsulates the clip regions of the view area and the scale that the map is displayed in.

A RenderedLayer contains the functionality for rendering a "layer." A RenderedLayer may be as simple as rendering a scalebar to as complex as rendering a thousand features.

There are four main types of RenderedLayers, see Figure 2: J2D RenderLayer.

- The RenderedGeometries is a collection of j2d geometry objects that are the rendered shapes for a given coordinate system. In addition to the shape data, geometries in a RenderedGeometries object have a style that will be applied to the geometry.
- A RenderedGridCoverage is an object that provides the functionality for rendering grid coverages.
- RenderedLegends and the subclass RenderedMapScale create and render the legend of the features onto the map and display the current scale of the map.
- A RenderedMark is a set of marks and/or labels to be rendered. Marks can have different sizes and orientations (for example, a field of wind arrows).

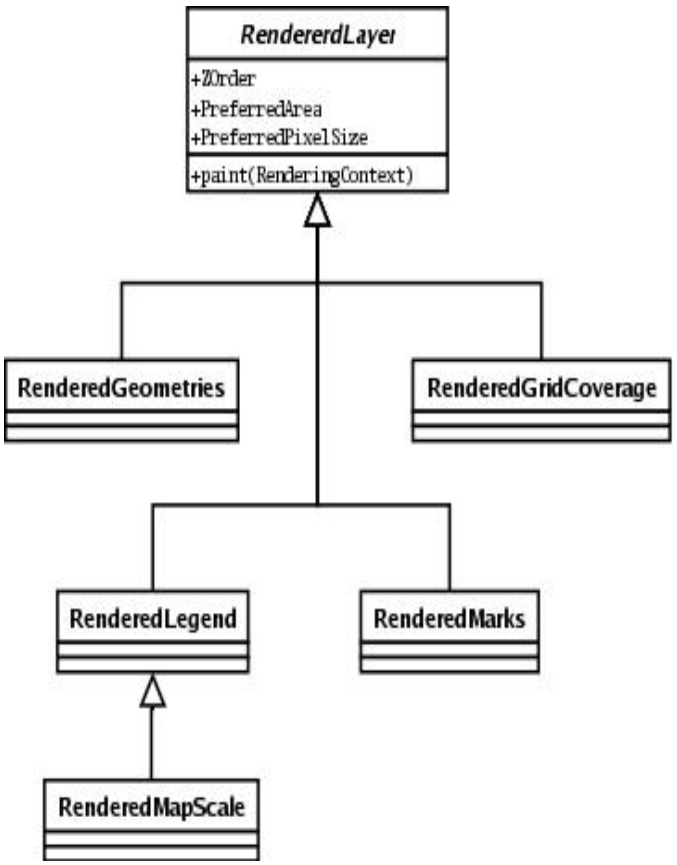

Figure 2: J2D RenderLayer

5.2 Strengths

The j2d renderer is a full-featured efficient renderer. In its current state of development the renderer has the capabilities of Feature and Coverage rendering. It accepts a Map Context and a Graphics2D object as input. The Map Context contains a feature collection and the styling (SLD compatible), to be used to draw the collection. The renderer renders the features in the Map Context; a Coverage is represented as a special Feature, and draws the features to the Graphics2D.

The j2d renderer also provides very quick and optimized performance when performing the common tasks of zooming and panning because of the large cache it maintains. The last requirement j2d satisfies is functionality for reprojection of the feature data. The j2d renderer does not require that all layers in the Map Context be in the same coordinate system. A component of the j2d renderer, the RenderingContext, transforms the features in to one, user-defined, coordinate system before rendering the data to the Graphics2D object.

5.3 Weaknesses

If a map to be drawn consists of 2 million rivers (a very reasonable number), then the j2d renderer must keep in memory 2 million shapes. A simple shape may have four points each of which must have at least two coordinates to identify its location in space. In such a case, the renderer must have more than 128 megabytes of memory allocated for the shapes. This example is an optimistic example that assumes simple shapes and does not take into account the shapes styling.

This example illustrates that the j2d renderer is not scalable enough to meet the rendering requirements of the uDig project. The designers of the j2d renderer acknowledge the need for a large amount of memory, and in order to reduce the amount of memory used by the cache, the coordinates in the shapes are stored as floats. However, another of the uDig requirements specify that double accuracy is required. Another flaw of the j2d Renderer is that it is not threaded, so the entire map must be rendered before anything is displayed and control is returned to the user.

6 GEOTOOLS LITE RENDERER

The Geotools Lite renderer is the mirror image of the j2d renderer's compromises. The Lite Renderer is intended to use a very small memory footprint, and is therefore very scalable, but not efficient with regards to tasks such as zooming and panning.

6.1 Basic Design

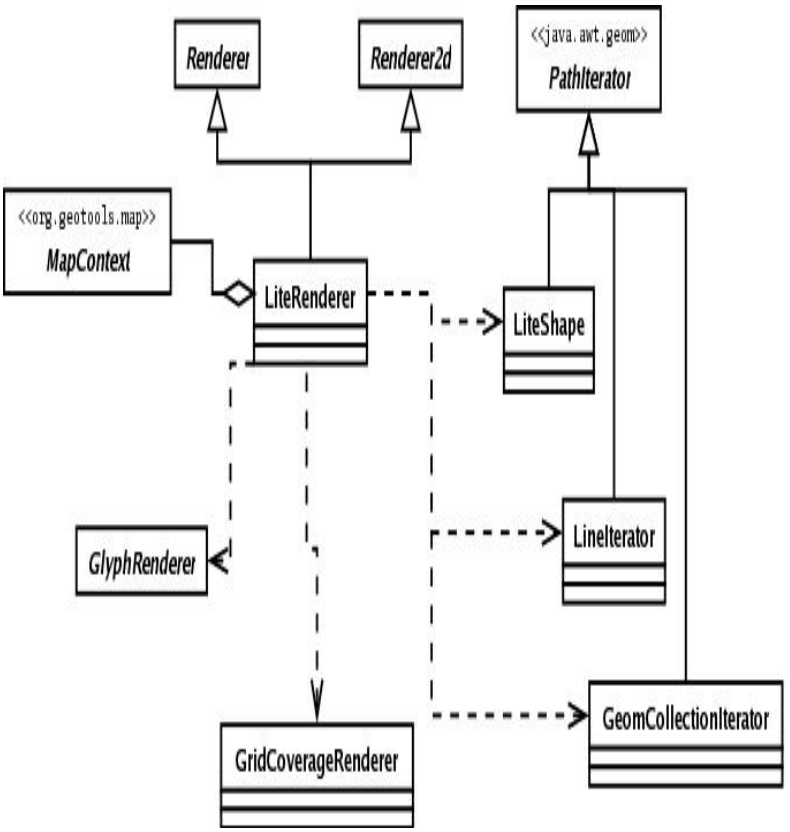

Figure 3: Lite Renderer

The Lite renderer has a very simple architecture. The Lite renderer has a map context providing the data for rendering. There are LiteShapes, LineIterators, GeomCollectionIterators and GlyphRenderers, which are created as needed for rendering the features. The design also has a GridCoverageRenderer that renders grid coverages, although SLD support for grid coverages is not yet implemented.

6.2 Strengths

The strength of the Lite Renderer is primarily is scalability. It renders data as it is read from disk and caches nothing. The accuracy of the data and the results from the transforms and calculations are in double precision until they are drawn to the graphics2D object. As with the j2d renderer, the Lite renderer takes a map context, which has feature layers, grid coverages and SLD styles and renders it all to a Graphics2D. One of the best features of the Lite renderer is that it does not wait for all the features to be loaded into memory to begin displaying the data. As soon as the data begins to arrive the renderer starts rendering the data.

6.3 Weaknesses

The main weakness of the Lite renderer is that it does not cache any information so common zooming and panning tasks are very inefficient. In addition, the Lite Renderer does not have SLD support for rendering grid coverages and finally, it does not provide any support for reprojection.

7 JUMP RENDERER

The JUMP renderer is a well-designed and efficient balancing of requirements. In many ways it is similar to the j2d renderer. It has a cache so that it can zoom and pan efficiently. The rendering is based on something similar to rendered layers but it is a threaded renderer that has double precision accuracy.

7.1 Basic Design

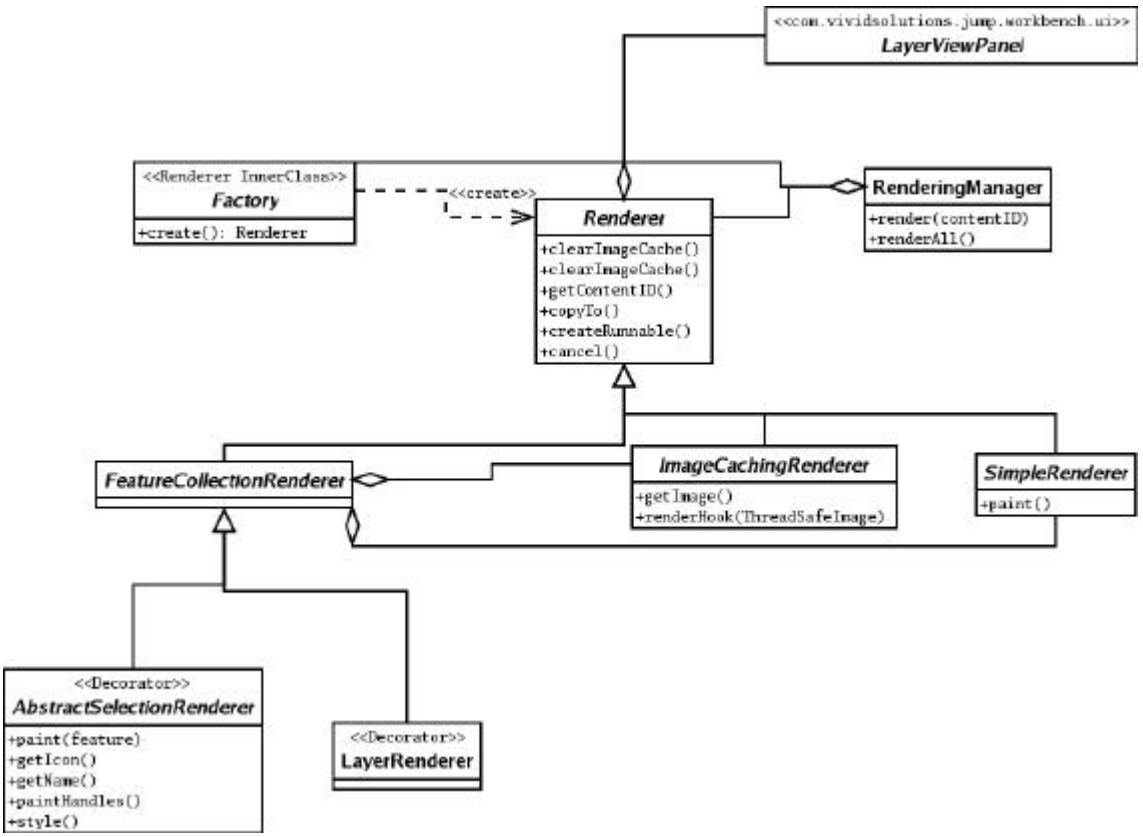

Figure 4: JUMP Basic Architecture

Similar to the j2d renderer, each layer has its own renderer. In JUMP each layer has a LayerRenderer, which usually has an ImageCachingRenderer or a SimpleRenderer. When the number of features in a layer is small (less than 100) the SimpleRenderer is used to render the layer. Layers with more features use the ImageCachingRenderer. The ImageCachingRenderer caches images of the view area so that a layer is not rendered every time the view must be refreshed. For example, if a view is minimized and then later restored, the cached images are used to refresh the view, instead of needlessly rendering the layer.

The child classes of the FeatureCollectionRenderer are “decorators” for the Simple and ImageCaching renderers. In this instance, they literally decorate the layers by determining which style to render the class with. The AbstractSelectionRenderer determines the style for selected features and the LayerRenderer determines the styles of the other layers, usually by looking up the current styling settings in the LayerViewPanel.

The RenderingManager handles refresh events and determines which layers are visible. When layers are modified or the view changes, the RenderingManager determines which layers must be re-rendered and which layers can refresh from their cached images, if they have caches.

All Renderers have access to their LayerViewPanel. The LayerViewPanel contains much of the rendering state. For example the LayerViewPanel provides access to the size of the display area, the images that are selected, the user defined styles, etc...

7.2 Strengths

One of the JUMP renderer's main strengths is the efficiency of zooms and pans, which are optimized for performance. JUMP renders both features and grid coverages. JUMP has double precision accuracy and has support for feature selection.

7.3 Weaknesses

The JUMP renderer is well designed; unfortunately it keeps all the features in memory so the memory footprint is very large. It also doesn't cache rendered shapes so the zooming and panning is less efficient than the j2d renderer, although it is faster than Lite Renderer because the features are kept in memory. In addition to not being scalable, JUMP does not provide support for Style Layer Descriptors. However, given the flexibility of the JUMP styling design, it would not be a major refactoring project to provide SLD support. JUMP also does not provide the reprojection infrastructure that j2d possesses. It should be noted that an unreleased version of JUMP is reported to provide reprojection support. The current JUMP renderer assumes that the features have either been reprojected previously or are already in the required coordinate system. The final weakness that concerns the uDig project is that JUMP does not begin rendering until the entire feature set is loaded into memory. The instant feedback that uDig requires is not satisfied.

8 SWT OPENGL PLUG-IN

The SWT OpenGL plug-in framework was briefly considered but the idea was quickly discarded for a number of reasons. First the plug-in is currently purely experimental, second, most of the requirements needed would have to be developed and finally there are only plug-in implementations for Windows and Linux Motif platforms. The benefit of using the OpenGL plug-in is that it would most likely be the most efficient because OpenGL is closely tied to graphics hardware.

9 CONCLUSION

It can be concluded that each renderer have strengths that can be useful and weaknesses that are unacceptable. The j2d renderer is efficient, well designed and provides many useful features. However, j2d has an unacceptably large memory footprint causing major problems with large datasets. The Lite renderer is the only renderer that meets the scalability requirements but it is also immature when compared to the j2d renderer and the JUMP renderer. The Lite renderer also misses many of the necessary features required by the uDig project. JUMP has a similar set of strengths and weaknesses to the j2d renderer but also has a license associated with it that is prohibitive and therefore is more useful as a source of inspiration and design solutions than a repository of code. Finally, the SWT OpenGL plug-in is immature and has very few of the features required. Therefore it is recommended that the Geotools j2d and Lite renderers form the basis of the implementation and the JUMP renderer be used as a source for design considerations.