SVGTLib: SVG Tiny Library for a LBS Imaging Server and Non-SVG Mobile Devices

Shivendra Tiwari
shivendra.tiwari@globallogic.com
Lead Engineering, Mobile Business Unit
GlobalLogic India Pvt Ltd, Noida, UP, India 201301

April 2008

Table of Contents

1 Abstract

Imaging is an indispensable feature of any LBS engine. A quality image generation is a mandatory part of an imaging server whereas seamless zooming is an imperative requirement of a client application. Images can be described in two types i.e. Raster Graphics and Vector Graphics. Raster imaging has its own limitations and in other hand Vector graphics gives solutions of the Raster imaging problems. The imaging software and libraries have two aspects - Creating and Rendering. In case of SVGT the challenges are in both of the aspects. There is no perfect SVGT image creation library available in the market and most of the mobile devices don't support SVG image rendering on the other hand. Only the latest devices have SVG support. The sole idea is to develop an effective SVGT image library which is reusable and highly capable of creating quality images in an LBS imaging server. The server technology can be C/C++ and Java/J2EE etc. To solve the image rendering issues on the non-SVG mobile devices, highly reusable vector graphics renderer can be developed on various platforms i.e. J2ME, Symbian C++ and Windows Mobile etc. The underlying content in this paper is not just a tutorial so that one can read and benefit from it; however this is an idea which can be made a complete project and implemented. Once the library has been implemented, it can be reused in the existing projects and can be shown as a technical competency to the prospective customers. This paper talks about SVG Tiny in terms of its strengths, applications, SVG library prototype and limitations. Section 1 through 3 can be skipped if the reader is pretty familiar with SVG and SVGT; however sections 4 and 5 are the essential part of this write-up. For further knowledge and study one can refer to the reference section appended at the end which includes various PDF tutorials and websites.

Keywords: SVG (Scalable Vector Graphics), SVG Tiny, W3C, 2D&3D Graphics, LBS (Location Based Services), and Imaging Server etc.

2 Introduction

Scalable Vector Graphics (SVG) is W3C's non-proprietary alternative to flash and bitmapped graphics. It is a language for describing two-dimensional graphics and graphical applications in XML. SVG enables 2-D resolution and media-independent graphics in a text-based format. This permits integration with XHTML, XSL and XSLT, XLink, SMIL, DOM and other W3 specifications including complete support for CSS, scripting, and animation. There are three object types in Vector Graphics i.e. Vector Graphic Shapes (paths consisting of straight lines and curves), Text, and Images. SVG Tiny 1.2 is the specification currently being developed as the core of the SVG 1.2 language. The SVG Mobile Profiles: SVG Basic and SVG Tiny are targeted to resource-limited devices and are part of the 3GPP platform for third generation mobile phones. SVG images can be generated on the fly from databases and applications. Text in SVG is fully searchable and selectable, as well as accessible. It can be integrates well with server-side technology (C/C++, Servlets, JSP, ASP, PHP, Perl, .NET etc.). Vector graphics objects can be grouped, styled, transformed, and reused. The major features of the SVG include clipping the paths, alpha masking, filter effects, template objects, and extensibility of the images. There are various vendors who support SVG and have implemented image creation/rendering libraries i.e. Adobe, BitFlash, Canon, Corel, CSIRO, Ericsson, Hewlett Packard, ILOG, KDDI, Nokia, Openwave, Schema Software, Sharp, Sun Microsystems, Texas Instruments, and Zoomon etc.

2.1 General advantages of the vector graphics include:

1. Smaller files so faster download times compared to bitmapped graphics (in certain cases)
2. Resolution and device independence
3. Suited for devices with low bandwidth and limited memory
4. Better printing capabilities
5. Permits panning around images
6. Enables zooming in on details not visible when image is initially displayed
7. Grouping, layers, hierarchies, reuse of components
8. Need not come from a static file; can be generated from a database
9. Leverage existing vector graphics investment
10. Various XML inherited capabilities:
    a. XML like validation (DTDs, XML Schema) is possible in SVG.
    b. XML parsers can be used for parsing, manipulating and making the data ready to render.
    c. It is compatible with XHTML and hence it can be well integrated into the Web pages
    d. Scriptable events based on the DOM (hierarchical model) are supported
    e. Text labels and desc (descriptions) are directly searchable (can be indexed by search engines)
    f. Each graphic element may contain a title
    g. Linking from any part of an image based on XLink and a element.
    h. It can reference JPEG or PNG bitmapped graphics via image
    i. It supports complex animations and transformations
    j. SMIL (Synchronized Multimedia Integration Language): conditionals, timing, animation are supported.

k. CSS style information can be used to alter rendering
l. Not limited to fonts available on the target device.

Example SVG image: In the given example these images have been generated from a single SVG image file by just zooming in and zooming out.

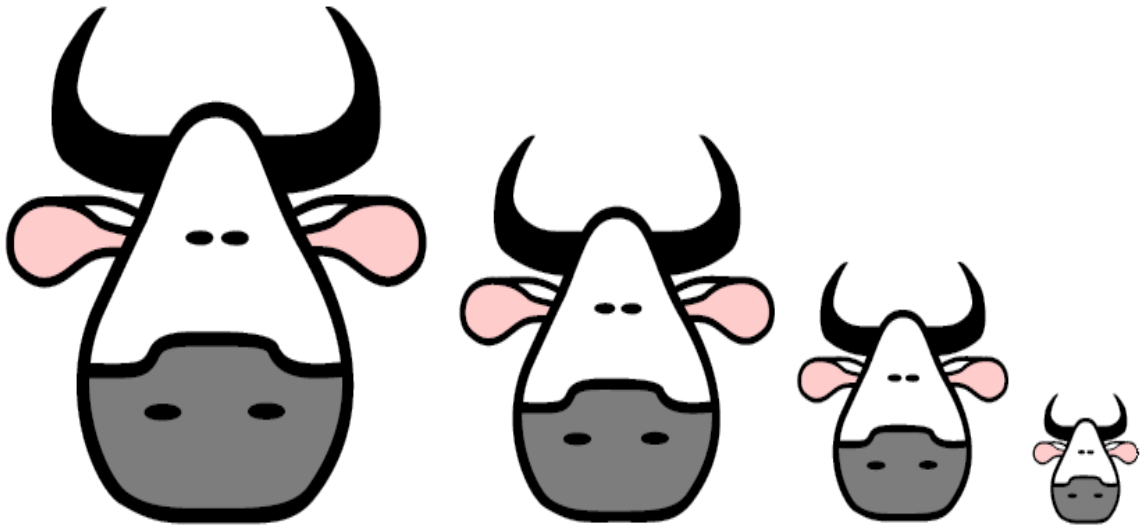

Figure - The SVG supports zooming without data loss

2.2 SVG Converter Applications

There are a number of SVG applications available in the market which can be used for various purposes:-

1. SVG to HTML and Text – extracts text from SVG
2. SVG to PDF – Apache Batik, FOP, FOP Developer Issues
3. SVG inside PDF – kevlindev perl script
4. PDF to SVG - FreeSVG
5. PPT / Visio / Word to SVG – svgMaker
6. SVG to PNG or JPEG - XML_svg2image
7. SVG Slide Toolkit - Sun
8. Graphical user interface to XML data
9. Via declarative transformations such as XSLT
10. Via scripting (loading XML data into the SVG User Agent and transforming using the DOM)

2.3 Recent Trend of SVG

1. SVG with SALT (Speech Application Language Tags) – interactive speech interface; speech input, output, and call controls
2. Database-driven charting using XSLT
3. Dynamic avalanche forecasting
4. SVG Smart Maps - intelligently connects graphic elements to non-graphic elements in a rich semantic web of cooperative communities; see also DCW Smart Maps (IE 5.x)
5. Geometry Markup Language (GML)
6. Chess: Validating moves and commentary using XSLT (IE 5+ required and Adobe SVG extensions)
7. Chess Viewer from RenderX; Animated Chess Game

3 SVG Tiny in Brief

SVG Tiny is a short version of SVG which is targeted to small computing devices. SVG Tiny allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), multimedia (such as raster images and video) and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG content) or via scripting. Because of industry demand, two mobile profiles were introduced with SVG 1.1: SVG Tiny (SVGT) and SVG Basic (SVGB). These are subsets of the full SVG standard, mainly intended for user agents with limited capabilities. In particular, SVG Tiny was defined for highly restricted mobile devices such as cell phones, and SVG Basic was defined for higher-level mobile devices, such as PDAs. In 2003, the 3GPP (3rd Generation Partnership Project) adopted SVG Tiny as the required graphics format for next-generation phones and Multimedia Messaging Services (MMS).

Neither mobile profile includes support for the full DOM, while only SVG Basic has optional support for scripting, but because they are fully compatible subsets of the full standard most SVG graphics can still be rendered by devices which only support the mobile profiles. The latest version of SVG tiny is known as SVGT 1.2 which adds a micro DOM (uDOM), allowing all mobile needs to be met with a single profile.

Example: The following example shows the transparency support in SVG Tiny images.

```
<?xml version="1.0"?>
<!DOCTYPE      svg      PUBLIC      "-//W3C//DTD      SVG      1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg          xmlns="http://www.w3.org/2000/svg"          version="1.1"
baseProfile="tiny" width="467" height="462">

  <!-- This is for the red square -->
  <rect x="80" y="60" width="250" height="250" rx="20" fill="red"
        stroke="black" stroke-width="2px" />
```

```
  <!-- This is for the blue square -->
  <rect x="140" y="120" width="250" height="250" rx="40" fill="blue"
        fill-opacity="0.7" stroke="black" stroke-width="2px" />

</svg>
```

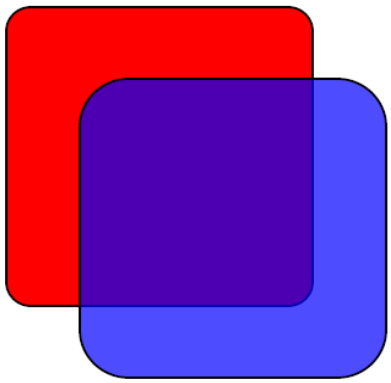

Figure - Transparency support in SVG Tiny images

Experience with SVG Tiny 1.1, which was widely adopted in the industry and shipped as standard on a variety of cell phones, indicated that the profile was a little too restrictive in some areas. Features from SVG 1.1 such as gradients and opacity were seen to have substantial value for creating attractive content, and were shown to be implementable on cell phones. There was also considerable interest in adding audio and video capabilities, building on the SMIL support in SVG Tiny 1.1.

Advances such as DOM Level 3, which introduces namespace support and value normalization, prompted a second look at the use of programming languages and scripting with SVG Tiny. In conjunction with the Java JSR 226 group [JSR226], a lightweight interface called the micro DOM, or uDOM, was developed. This could be, but need not be, implemented on top of DOM Level 3. With this advance, lightweight programmatic control of SVG (for example, for games or user interfaces) and use with scripting languages became feasible on the whole range of platforms from cell phones through to desktops. In consequence, there is only a single Mobile profile for SVG 1.2 - SVG Tiny 1.2.

This specification defines the features and syntax for Scalable Vector Graphics (SVG) Tiny 1.2, the core specification and baseline profile of SVG 1.2. Other SVG specifications will extend this baseline functionality to create supersets (for example, SVG 1.2 Full). The SVG Tiny 1.2 specification adds to SVG Tiny 1.1 features requested by SVG authors, implementers and users; SVG Tiny 1.2 is a superset of SVG Tiny 1.1.

3.1 Modularization

SVG Tiny has a collection of abstract modules that provide specific units of functionality. These modules may be combined with each other and with modules defined in other specifications (such as XHTML) to create SVG subset and extension document types that qualify as members of the SVG family of document types.

3.2 Element and Attribute collections

Modules define a named collection of either elements or attributes. These collections are used as shorthand when describing the set of attributes allowed on a particular element (eg. the "Style" attribute collection) or the set of elements allowed as children of a particular element (eg. the "Shape" element collection). All collections have names that begin with an uppercase character. When defining a profile, it is assumed that all the element and attribute collections are defined to be empty. That way, a module can redefine the collection as it is included in the profile, adding elements or attributes to make them available within the profile. Therefore, it is not a mistake to refer to an element or attribute collection from a module that is not included in the profile; it simply means that collection is empty.

3.3 Profiling the SVG specification

The Tiny profile of SVG 1.2 consists of all of the features defined within this specification. As a modularized baseline specification, it is possible for: superset profiles (e.g., SVG Full 1.2) which include all of the Tiny profile but add other features to the baseline; subset profiles; and special-purpose profiles which incorporate some modules from this specification in combination with other features as needed to meet particular industry requirements.

When applied to conformance, the term "SVG Tiny 1.2" refers to the Tiny profile of SVG 1.2 defined by this specification. If an implementation does not implement the Tiny profile completely, the UA's conformance claims must state either the profile to which it conforms and/or the specific set of features it implements.

3.4 Defining an SVG Tiny 1.2 document

SVG Tiny 1.2 is a backwards compatible upgrade to SVG Tiny 1.1. Backwards compatible means that conformant SVG Tiny 1.1 content will render the same in conformant SVG Tiny 1.2 User Agents as it did in conformant SVG Tiny 1.1 User Agents. A few key differences from SVG Tiny 1.1 should be noted: The value of the version attribute on the root most svg element should be "1.2".

There is no DTD for SVG 1.2, and therefore no need to specify the DOCTYPE for an SVG 1.2 document (unless it is desired to use the internal DTD subset, for purposes of entity definitions for example). Instead, identification is by the SVG namespace, plus the version and baseProfile attributes. In SVG Tiny 1.2, validation is provided by the SVG Tiny 1.2 RelaxNG schema.

4 Problem Definition

SVG image creation and rendering images on the canvas have been an area of research and development specially targeting to small computing devices. There are a number of SVG image libraries are available (i.e. OpenVG, ImageMagic++, CAIRO etc) in the market, however there is NO effective and accurate image library available for SVG Tiny. Since all the navigation applications are being migrated to small computing devices from the PC and the web applications; the image libraries have became important in order to generate effective and accurate images quickly in the Location Engine. There are two major problems have been eyed in this content: developing SVG Tiny image creation library for imaging server and to develop SVG Tiny image rendering libraries for Non-SVG mobile devices.

4.1 SVG Tiny Image Creation in Imaging Server

OpenGL, OpenCV, LibGD and other powerful image libraries are available for Raster image creation in the imaging server. Even for the vector image creation there are some open source libraries available, which can be used to generate the images for PC clients and the web client applications. However there is a lack of vector graphics libraries targeted to small computing device profiles i.e. SVG Tiny. An emerging industry demand for mobile navigation system causes an importance of Tiny image library for the location servers which is generic enough, efficient and robust.

4.2 SVG Image Drawing in Non-SVG Mobile Devices

Sun Microsystems has released JSR 226 i.e. SVG Tiny profile for J2ME devices which can be used to render the SVG files on mobile just by using the provided library. However there are a number of J2ME device which don't support SVG Tiny just because the vendors have not implemented SVG Tiny library in these mobile phones. On the other hand, other mobile platforms i.e. Symbian C++, BREW, Windows Mobile etc don't have SVG support in a large range of devices. It opens a new door of challenges of using SVG images on these non-SVG mobile devices. The idea is to take this opportunity as a challenge for the mobile engineers and provide a solution for it by implementing our own efficient & highly reusable library. There are several different mobile platforms which are still away from SVG Tiny implementation.

5 Proposed Solution: SVGTSLib & SVGTMLib

The proposed solution for the SVG related problems is to develop reusable own library which can be easy used in various LBS Engines and mobile applications. The following block diagram shows how client and server libraries can be used together. The upcoming text has some high level structures and APIs defined but that is not a design. The given structures can be considered to understand the way the APIs should look like. A detailed design is expected while getting into the planning and implementation of the project. Graphics Artist

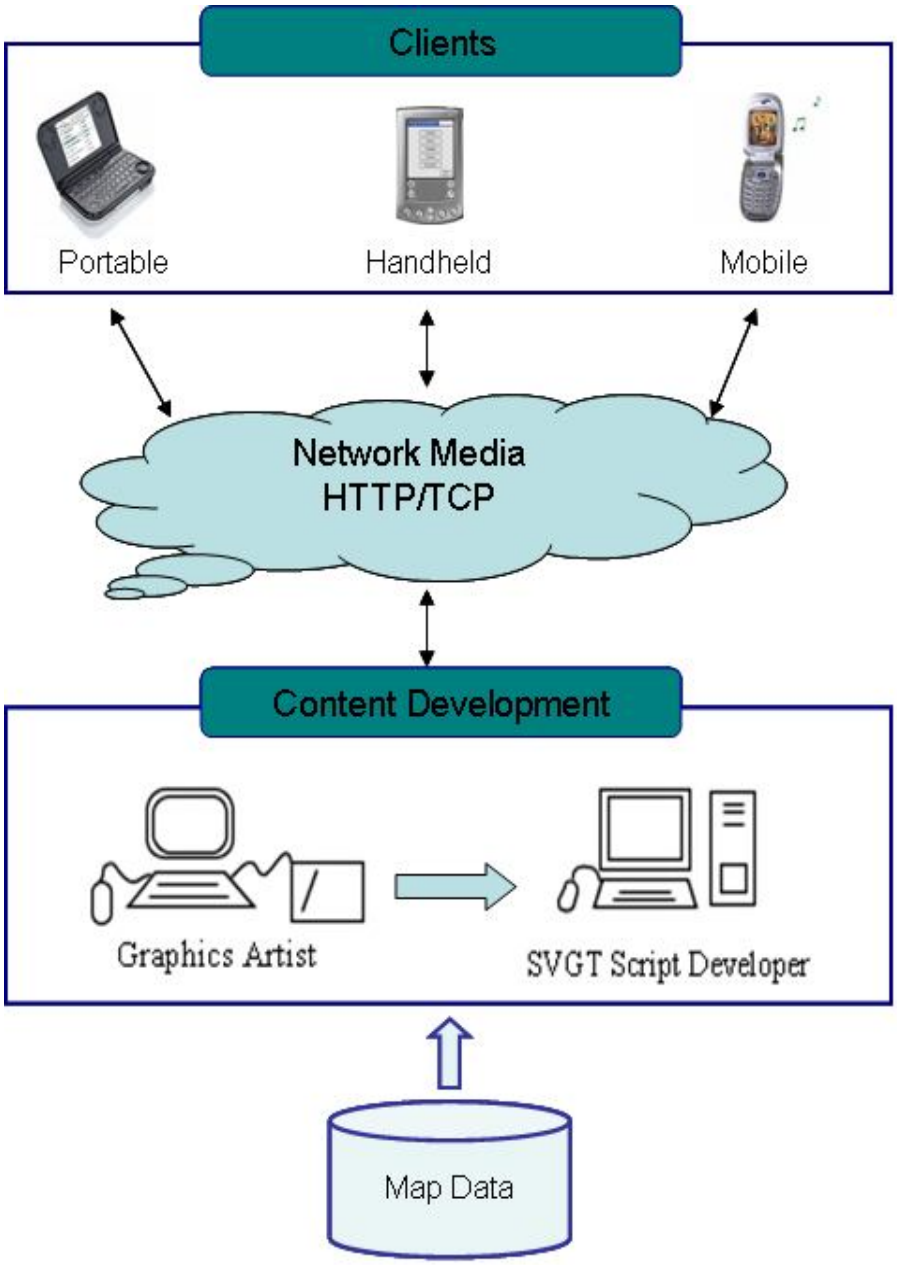

Figure - The basic Architecture of the Proposed Solution

5.1 SVG Tiny Server Library: SVGTSLib

The SVGTSLIb is an SVG Tiny library for Server.

There are various operations which the server library should perform:

1. **Image Creation:** Create new images, or read existing images.
2. **Edit images :** flip, mirror, rotate, scale, transform images, adjust image colors, apply
3. Various special effects, or draw text, lines, polygons, arcs, ellipses and Bezier curves,
4. Compose new images based on other images SVG or Raster images.
5. Save the image in SVG Tiny format.

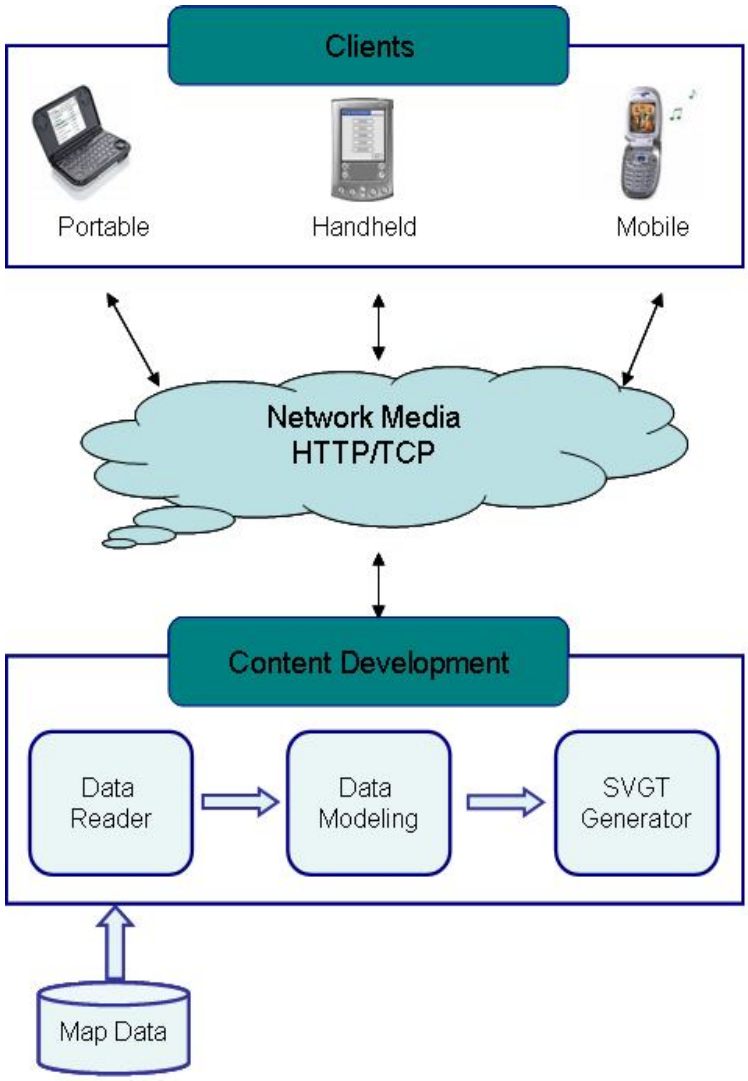

Figure – The SVGT Server Library

The whole data model for this library should be defined while getting into the project actively but the APIs will look like defined in the following section.

5.1.1 SVGT Image structures

At high level the ImageObject structure should have a list of Line object to create lines, list of Rectangle object, list of Polylines, list of Arc to create arc shapes like Eclipse, Circle etc.

- **Color**

```
typedef struct {
        int red; //Red component of the color
        int green; //Green component of the color
        int blue; //Blue component of the color
        int opacity; //Transparency of the color
} Color;
```

- **Point**

```
typedef struct {
        int x; //x coordinate
        int y; //y coordinate
} Point;
```

- **Line**

```
typedef struct {
        Point point1;
        Point point2;
        Color color;
} Line;
```

- **Rectangle**

```
typedef struct {
        Point point1;
        Point point2;
        int h; //height
        int w; //width
        Color color;
```


```
} Rectangle;
```

- **Polyline**

```
typedef struct {
        Point* points;//Point list
        int numPoints; //number of points
        int h; //height
        int w; //width
        Color color;

} Polyline;
```

- **Arc**

```
typedef struct {
        int xc; //center x
        int yc; //center y
        int radius; //radius of the arc
        int angle1; //initial angle of the arc
        int angle2; //end angle of the arc
        Color color;
} Arc;
```

5.1.2 APIs for basic graphical shapes

The SVG shapes are the paths consisting of straight lines and curves which form Line, Polyline, Closed Polyline, Arc, Circle, Rectangle, Triangle and Polygons shapes.

- **Rectangles (including optional rounded corners)**

```
public void drawRect(struct ImageObject* iamgeBuffer, int x, int y, int w, int h,
        Color color);
```

- **Circles**

```
public void drawCircle(struct ImageObject* iamgeBuffer, int x, int y, int radius, Color
color);
```

- **Arc**

public void drawArc(struct ImageObject* iamgeBuffer, int x, int y, int radius, int angle1, int angle2, Color color);

- **Ellipses**

public void drawEclipse(struct ImageObject* iamgeBuffer, int x, int y, int xradius, int yradius, Color color);

- **Lines**

public void drawline(struct ImageObject* iamgeBuffer, int x1, int y1, int x2, int y2, Color color);

- **Polylines**

public void drawPoliline(struct ImageObject* iamgeBuffer, struct Point* listOfPoints, int numberOfPoints, int isClose, Color color);

- **Polygons**

public void drawPoliline(struct ImageObject* iamgeBuffer, struct Point* listOfPoints, int numberOfPoints, int isClose, Color color);

5.1.3 Text APIs

- **Text**

void drawText(struct ImageObject* iamgeBuffer, int x, int y, int size, int style, char* text, int strLength, int angle = 0, Color color);

5.2 SVG Tiny Mobile Rendering Library: SVGTMLib

The SVGTMLIb is an SVG Tiny library for Non-SVG Mobile Devices. The whole data model for this library is yet to define but the APIs will look like this

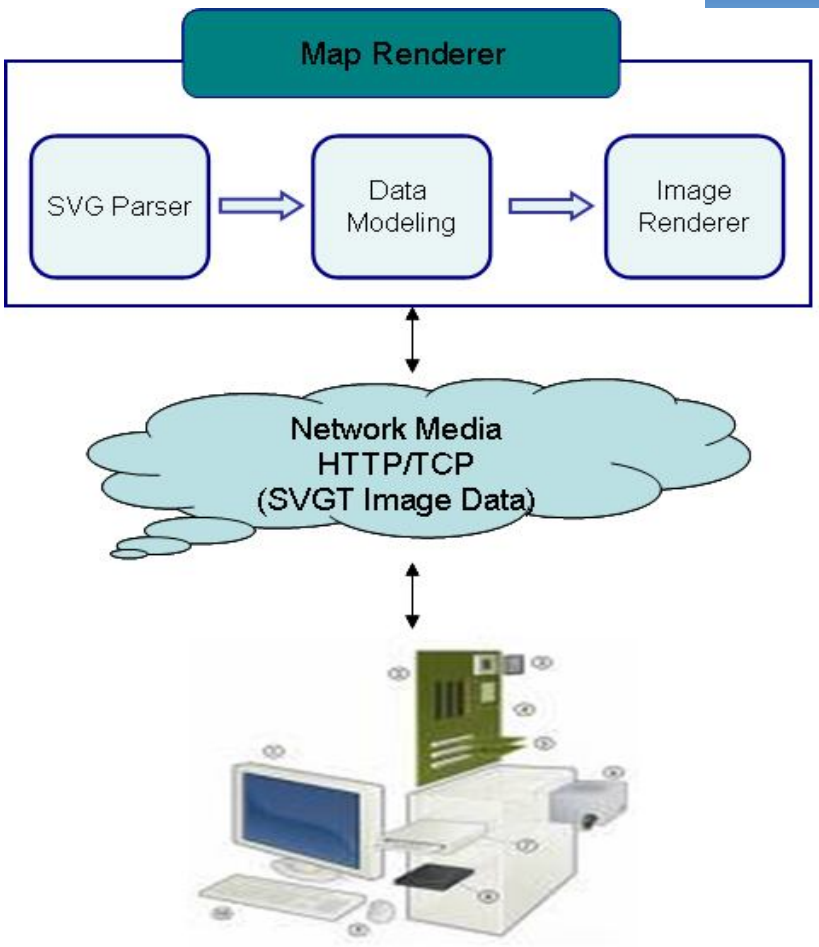

Figure – The SVGT Mobile Renderer Library

5.2.1 SVGT Image structures

The rendering APIs should be able to parse the SVGT XML and store in some data structures so that they can be rendered easily on the canvas. Following are the structures:

- **Color**

```
typedef struct {
        int red; //Red component of the color
        int green; //Green component of the color
        int blue; //Blue component of the color
        int opacity; //Transparency of the color
```


```
} Color;
```

- **Point**

```
typedef struct {
        int x; //x coordinate
        int y; //y coordinate
} Point;
```

- **Line**

```
typedef struct {
        Point point1;
        Point point2;
        Color color;
} Line;
```

- **Rectangle**

```
typedef struct {
        Point point1;
        Point point2;
        int h; //height
        int w; //width
        Color color;
} Rectangle;
```

- **Polyline**

```
typedef struct {
        Point* points;//Point list
        int numPoints; //number of points
        int h; //height
        int w; //width
        Color color;
} Polyline;
```

- **Arc**


```
typedef struct {
        int xc; //center x
        int yc; //center y
        int radius; //radius of the arc
        int angle1; //initial angle of the arc
        int angle2; //end angle of the arc
        Color color;
} Arc;
```

5.2.2 APIs for basic graphical shapes

- **Parse and render the SVGT image**

void renderSVGOnCanvas(char* filePath, GraphicsControl* control);

void renderSVGOnCanvas(char* svgXML, int lengthSvgXML GraphicsControl* control);

- **Rectangles (including optional rounded corners)**

private void drawRect(GraphicsControl* control, int x, int y, int w, int h, Color color);

- **Circles**

private void drawCircle(GraphicsControl* control, int x, int y, int radius, Color color);

- **Arc**

private void drawArc(GraphicsControl* control, int x, int y, int radius, int angle1, int angle2, Color color);

- **Ellipses**

private void drawEclipse(GraphicsControl* control, int x, int y, int xradius, int yradius, Color color);

- **Lines**

private void drawline(GraphicsControl* control, int x1, int y1, int x2, int y2, Color color);

- **Polylines**

private void drawPoliline(GraphicsControl* control, struct Point* listOfPoints, int numberOfPoints, int isClose, Color color);

- **Polygons**

private void drawPoliline(GraphicsControl* control, struct Point* listOfPoints, int numberOfPoints, int isClose, Color color);

5.2.3 Text APIs

- **Text**

private void drawText(GraphicsControl* control, int x, int y, int size, int style, char* text, int strLength, int angle = 0, Color color);

6 Limitations & Scope of Further Improvements

1. The SVGT server libraries can be complex to implement and it needs skilled people to develop the libraries.
2. Data model and the detailed design should be defined before starting development
3. Write efficient code in order to produce an effective and faster library.
4. The existing open source code can be customized and modified for SVG Tiny.
5. The APIs can be richer to incorporate the Raster images into the vector image files.

7 Conclusion

The quality and interactivity of visualizing map data in the World Wide Web is highly dependent on the technologies and formats used. Map data is highly complex and changes according to the use interest by panning and zooming. Because of this complexity, good visualization techniques are required to make very interactive and user friendly. The project described has shown that interactive visualization of dynamically changing map data on Mobile devices can be achieved with SVG Tiny on non-SVG devices too. The highly efficient and reusable SVG Tiny client and server library will surely make a difference in terms of new prospective customers and technical LBS imaging competency in the market.

8 References

8.1 Base References:

1. http://www.w3.org/TR/SVG/intro.html
2. http://www.adobe.com/svg/demos/devtrack/svgdraw.html
3. http://www.xml.com/pub/a/2004/09/08/tree.html
4. http://jcp.org/en/jsr/detail?id=226

5. http://svg.org/story/2006/7/11/15617/9573
6. CAIRO: A Vector Graphics Library - http://www.cairographics.org/manual/
7. List of SVG Software - http://www.scale-a-vector.de/soft-e.htm
8. SVG & XForms accessibility issues report http://webcc.fit.fraunhofer.de/downloads/projects/bentoweb/deliverables/BenToWeb_D4.5_rev.pdf
9. SVG Open 2008 - http://www.svgopen.org/2008/?section=abstracts_and_proceedings
10. OpenCV Reference Manual: http://www.cs.unc.edu/Research/stc/FAQs/OpenCV/OpenCVReferenceManual.pdf
11. The ImageMagick graphics library : http://www.imagemagick.org/Magick++/tutorial/Magick++_tutorial.pdf

8.2 Reference Forums and Discussions

1. C APIs/library to create SVG Tiny (Mobile SVG) http://www.nabble.com/C-APIs-library-to-create-SVG-Tiny-(Mobile-SVG)-td14832604.html
2. How to use SVG plugin for rendering SVGT in c++ in Symbian http://discussion.forum.nokia.com/forum/showthread.php?t=92980
3. http://www.imagemagick.org/discourse-server/viewtopic.php?f=2&t=12323

8.3 Definitions and Jargons

- **SVG (Scalable Vector Graphics):** SVG is a language for describing two-dimensional graphics and graphical applications in XML. SVG 1.1 is a W3C Recommendation and forms the core of the current SVG developments.
- **W3C:** The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding.
- **LBS (Location Based Services):** A location-based service (LBS) is an information and entertainment service, accessible with mobile devices through the mobile network and utilizing the ability to make use of the geographical position of the mobile device.[1][2][3] LBS services include services to identify a location of a person or object, such as discovering the nearest banking cash machine or the whereabouts of a friend or employee. LBS services include parcel tracking and vehicle tracking services. They include personalized weather services and even location-based games. They are an example of telecommunication convergence.
- **Imaging Server:** A component of LBS Engine which creates images from the digitized vector data provided by different vendors.
- **Seamless zooming:** A feature of imaging to get a feel of highly smooth zoom-in and zoom-out without flickering and distortion in the images is known as seamless zooming.
- **DTD:** Document Type Definition (DTD) is one of several SGML and XML schema languages, and is also the term used to describe a document or portion thereof that is authored in the DTD language. A DTD is primarily used for the expression of a schema via a set of declarations that conform to a particular markup syntax and that describe a class, or type, of document, in terms of constraints on the structure of that document.

- **XHTML:** The Extensible Hypertext Markup Language, or XHTML, is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax. While HTML is an application of Standard Generalized Markup Language (SGML), a very flexible markup language, XHTML is an application of XML, a more restrictive subset of SGML.
- **XSL:** In computing, the Extensible Stylesheet Language (XSL), a family of transformation languages, allows one to describe how to format or transform files encoded in the XML standard.
- **XSLT:** Extensible Stylesheet Language Transformations (XSLT) is an XML-based language used for the transformation of XML documents into other XML or "human-readable" documents. The original document is not changed; rather, a new document is created based on the content of an existing one.[2] The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text.[3] XSLT is most often used to convert data between different XML schemas or to convert XML data into HTML or XHTML documents for web pages, creating a dynamic web page, or into an intermediate XML format that can be converted to PDF documents.
- **XLink:** The XML Linking Language, or XLink, is an XML markup language used for creating hyperlinks in XML documents. XLink is a W3C specification that outlines methods of describing links between resources in XML documents, whether internal or external to the original document
- **SMIL:** The Synchronized Multimedia Integration Language, is a W3C recommended XML markup language for describing multimedia presentations. It defines markup for timing, layout, animations, visual transitions, and media embedding, among other things. SMIL allows the presentation of media items such as text, images, video, and audio, as well as links to other SMIL presentations, and files from multiple web servers. SMIL markup is written in XML, and has similarities to HTML.
- **DOM:** The Document Object Model (DOM) is a platform- and language-independent standard object model for representing HTML or XML and related formats. A web browser is not obliged to use DOM in order to render an HTML document. However, the DOM is required by JavaScript scripts that wish to inspect or modify a web page dynamically. In other words, the Document Object Model is the way JavaScript sees its containing HTML page and browser state.
- **CSS:** Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML, but the language can be applied to any kind of XML document, including SVG and XUL.
- **3GPP platform:** The 3rd Generation Partnership Project (3GPP) is a collaboration between groups of telecommunications associations, to make a globally applicable third generation (3G) mobile phone system specification within the scope of the International Mobile Telecommunications-2000 project of the International Telecommunication Union (ITU). 3GPP specifications are based on evolved Global System for Mobile Communications (GSM) specifications. 3GPP standardization encompasses Radio, Core Network and Service architecture.
- **Geography Markup Language (GML):** The Geography Markup Language (GML) is the XML grammar defined by the Open Geospatial Consortium (OGC) to express geographical features. GML serves as a modeling language for geographic systems as well as an open interchange format for geographic transactions on the Internet.