LSIVIEWER 2.0 - A CLIENT-ORIENTED ONLINE VISUALIZATION TOOL FOR GEOSPATIAL VECTOR DATA

K. Manikanta[a,*], K.S.Rajan[a]

[a] Lab for Spatial Informatics, International Institute of Information Technology-Hyderabad,
Gachibowli 500032, Hyderabad, India (kondetimanikanta.purushotham@research, rajan@).iiit.ac.in

Commission V, WG V/4

KEY WORDS: Geospatial vector data, Online visualization, Client-based rendering, Web Technologies, WebGIS

ABSTRACT:

Geospatial data visualization systems have been predominantly through applications that are installed and run in a desktop environment. Over the last decade, with the advent of web technologies and its adoption by Geospatial community, the server-client model for data handling, data rendering and visualization respectively has been the most prevalent approach in Web-GIS. While the client devices have become functionally more powerful over the recent years, the above model has largely ignored it and is still in a mode of server-dominant computing paradigm. In this paper, an attempt has been made to develop and demonstrate LSIViewer - a simple, easy-to-use and robust online geospatial data visualisation system for the user's own data that harness the client's capabilities for data rendering and user-interactive styling, with a reduced load on the server. The developed system can support multiple geospatial vector formats and can be integrated with other web-based systems like WMS, WFS, etc. The technology stack used to build this system is Node.js on the server side and HTML5 Canvas and JavaScript on the client side. Various tests run on a range of vector datasets, upto 35MB, showed that the time taken to render the vector data using LSIViewer is comparable to a desktop GIS application, QGIS, over an identical system.

1. INTRODUCTION

With the transition in technologies from desktop to web, the successful transition from Office suite to Google docs(Herrick, 2009), other domains joined the shift too. But for past two decades GIS has been largely desktop centric and web GIS was only used for one way transmission(Steiniger and Bocher, 2009), whereas in other domains it moved to much more interactive level. Earlier, most of the World Wide Web standards are text and document based where we can only view documents on the web browser. But eventually, these standards were extended to allow operational interaction with the rise of eCommerce(Puder, 2004). So, this is the right time for GIS applications to make the shift from desktop oriented models of application software to web oriented approaches which can bring single-installation-instance with single user to single instance handling multiple simultaneous users. In GIS, for 2D datasets the most common data models are Vector and Raster(Winter and Frank, 2000). Each of the data models have multiple number of file formats. The Vector models are largely being handled on desktop applications for functionalities like creation, deletion, modification, analysis and finite visualization(Dangermond, 1988), but most of the Desktop GIS applications are tightly coupled to these formats which raise the problem of interoperability. Hence, to solve this problem, a few open source libraries like Geospatial Data Abstraction Library(GDAL/OGR), translator library for geospatial vector data formats(Steiniger and Bocher, 2009) came into light. The other drawback of using Desktop GIS is its un-usability on multiple computing devices like smartphones, tablets, laptops and personal computers where the users use and switch between the devices while the data and the GIS application resides primarily on one of the these devices. A server based approach with multiple clients accessing it remotely is one potential solution but, the current paradigm of WebGIS gives limited capabilities and functionalities to the user to handle the Geospatial data.

Open GIS consortium(OGC) is an active participant in developing standards that can be adopted by software developers and engineers to develop platform independent interfaces that geo-enable the Web. These specifications empower developers to make complex spatial information and services accessible and useful with all kinds of applications(Wenjue et al., 2004). In 1999, OGC has developed web mapping specifications like Web Map Service(WMS) and Web Feature Service(WFS). WMS is a standard for visualizing Geospatial data over the internet in which, a client requests the WMS server for a map, the server queries its database and responds with necessary information in a raster format(PNG, JPG, SVG). WFS is a service that allows a client to request for retrieving Geospatial vector feature(Peng and Zhang, 2004). WMS can primarily return only an image which can not be edited or spatially analyzed and also the map is pre-rendered by the server and the client is just a dumb display terminal/browser. In the case of WFS, it allows for data manipulation operations like create, delete and update the features but this is also similar to the WMS where the rendering of image happens on the server side. Thus both of the standards are server centric(Lu, 2005), which makes the user to knock server for the simplest of the operations demanding continuous server calls requiring high bandwidth for data transfers. While some of the load has been reduced by the Web Map Tile service(WMTS)(Porta et al., 2013), it still limits the user interaction as far as rendering is concerned.

But at the same time, while GIS applications are slow in adapting web technologies, web technologies have suddenly improved which makes the client equally powerful rather than heavy processing being at the server end. Recent evolution of web technologies from server centric models where the client needs to re-

*Corresponding author

quest the server for any functional operations, to client oriented models where the capabilities of client has improved greatly, reducing the data transmissions from client to server over the internet and also being able to handle most of the functionalities by itself, favoring the web applications to become more prevalent(Wood et al., 2014). The extension of HTML(Hypertext Markup Language), HTML5 is the core markup language of the World Wide Web(Anthes, 2012) which diminished the requirement for plugin-in-based technologies such as Microsoft Silverlight, Oracle-Sun JavaFX and Adobe Flash. It has introduced a new element, Canvas which draws graphic contents using JavaScript. This element is useful to GIS in many ways, like easy and fast visualization, seamless interaction with data like scaling, panning the map and applying styling to the objects drawn(Boulos et al., 2010). An alternative to the Desktop GIS application can be implemented with these web technologies that could make it easier for sharing data online and collaborating with other GIS users. For example, in a company there are ten different users to do the basic GIS analysis. If we intend to use Desktop GIS application, we require ten different systems with ten Desktop GIS application installed on each system and requiring a license for each application. Now imagine a client oriented rendering system, which is independent of platform and hardware, and can still work with having one backend server for conversion of file formats whereas the client efficiently handles functions like zooming, panning, styling and labeling. Therefore, a system with these functionalities could make efficient use of resources and also be cost effective. So, there is a need for adapting latest technologies and implementing open source client based rendering GIS application. We develop and demonstrate such a system with the following objectives.

- Building an installation free, platform-independent application which is accessible openly to everyone on the web to visualize the locally-stored[1] vector data anytime they want with a reliable internet connection.
- With the development of the latest web technologies, transferring the server-side functionalities to the client side.
- Data rendering speeds on the developed system to be comparable to the traditional Desktop GIS application.

In this paper, we present LSIViewer *[2], a client-oriented framework for rendering spatial vector data in native HTML5. As part of the working prototype developed and presented here, the scope of this application is limited to a basic set of rendering functionalities. The analytical processing capabilities are currently out of scope of this paper. In section 2, we illustrate the architecture of LSIViewer and show a brief implementation of its components. In section 3, we show the performance comparison of LSIViewer with a typical Desktop GIS application and its performance on multiple web browsers. Finally, in section 4, we conclude and discuss the future applications.

2. ARCHITECTURE OF LSIVIEWER 2.0

LSIViewer is based on a client-server architecture as shown in Figure 1, where the role of the server is to provide an instance of the client application, act as an adapter and convert the input vector data into a client suitable format by encoding, compressing the encoded data and sending it as a response to the client. The client decompresses the encoded response and renders the map on HTML5 canvas using JavaScript. Traditionally, most of the client server architecture are server-centric where the server does all the computations and the client is only used for displaying the response. But here an effort is made to use the client as much as the server, for rendering the vector data as well as for functionalities like zoom, pan, label and apply color. The detailed architecture is delineated in the below subsections.

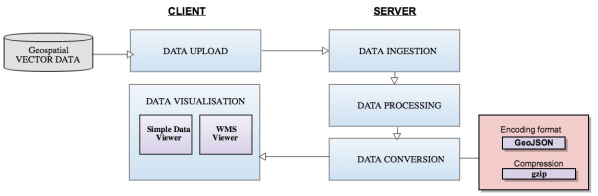

Figure 1. Architecture of LSIViewer 2.0

2.1 Data Upload

The application allows the user to input necessary vector data files through a HTML form and supports five Geospatial vector formats which are ESRI ShapeFile, Geography markup language(GML), Keyhole markup language(KML), GPX and GeoJSON. ESRI Shapefile is the standard de facto format for vector data(Kelso and Patterson, 2009) and is also the most commonly used vector data format, although it is not accepted as an official standard by OGC. GPX(GPS eXchange) is a standardized XML format for GPS data(Morris et al., 2004) whereas GML, KML and GeoJSON are popular vector formats that are used on the web. While selecting the data for upload, the user has to take care of the necessary file formats for successful visualization. In case of ESRI Shapefile, the user has to upload files with Multipurpose Internet Mail Extensions(MIME) type .shp, .shx, and .dbf. For GML, GPX and KML the user has to upload .gml, .gpx and .kml files respectively. When the filled form is submitted, the data is sent to the server using the multipart/form-data algorithm.

2.2 Data Ingestion and Validation

Here, the server side implementation of endpoint handlers required to validate the user-uploaded vector files are emphasized. When the server receives the data, it extracts the contents from the client's request, processes it and returns the vector data in a client-suitable format as a response. The server-side application is built on Node.js framework.

Node.js(also termed as Node) is a platform built on Google Chrome V8 JavaScript runtime engine that is event-oriented, non-blocking I/O model framework for coding JavaScript on the server(Jun and Doh, 2013). The main purpose for using Node is to make the application fast, scalable, and lightweight(Nair et al., 2016). Node is implemented in C and C++ focussing on low memory consumption, high performance and allows using JavaScript end to end, both on the server and on the client(Chaniotis et al., 2014). This important aspect helps the system to run the application efficiently without the need of parsers. This has modules that help to process the data passed from the

[1] Locally-stored here implies data stored on your device either mobile, laptop or computer

[2] *LSIViewer in this paper refers to LSIViewer 2.0 http://lsi.iiit.ac.in/lsiviewer/(accessed May 8, 2017)

client. These modules are in the form of plugins, add-ons, and extensions to facilitate the development process of the application. They are broadly divided into the following categories, which are core modules, third-party modules, and local modules.

For this application, we use Express, a Node.js web framework, which provides methods to develop web applications. This application has a POST handler to receive the vector files that are uploaded from the client. Request handling methods in Express application use a callback function whose parameters are request and response objects. This request object contains information passed from the client such as headers, type of encoding and input vector files. Then these vector files are passed to a local module which does the validation of vector data files uploaded from the client. Validation includes determining the type of vector format, file name, and a MIME type, an identifier for file format. The local module returns these parameters after successful execution, from which we know the format of the input files uploaded and its contents.

2.3 Data Conversion

Data interchange format has compelling consequences on data transmission rates and client-side rendering performance. Here we explain the tool used for conversion of the client-uploaded vector files and reasons for choosing GeoJSON as a data interchangeable format between the client and the server. Data conversion is done using a simple, efficient, and popular open source tool, GDAL/OGR, a translator library for raster and vector geospatial data formats. It is an open source, cross platform tool that supports read/write operations for more than 52 vector formats(Bunting et al., 2014). LSIViewer uses a third-party module, ogr2ogr[3], in Node.js. The module converts vector data into GeoJSON format using the GeoJSON OGR driver.

GeoJSON is a JavaScript Object Oriented Notation(JSON) which is simple, human-readable, lightweight format used on the web for transferring geographical information. Each feature in GeoJSON has two parts: geometric and attribute data. Geometric features include point, line, polygon, multi-Point, multi-Line, and multi-Polygon. The other popular data interchange format is GML, a XML-like format introduced by OGC which provides encoding for points, polylines, polygons and other complex spatial data structures. GeoJSON is better at encoding point data while GML is good at encoding polyline and polygon data(Li et al., 2015). Although GML has advantage over GeoJSON in polygon and polyline encoding, we had to choose GeoJSON deliberately for the reasons described in Table 1.

GeoJSON	GML
De-serialization of GeoJSON is very fast	De-serialization of GML is slow
It has simple APIs in Javascript	An external parser is needed to parse the GML data
Key-value pair representation in GeoJSON makes it look simple and human readable	GML is very lengthy due to tags and namespaces

Table 1. Advantages of GeoJSON over GML in a JavaScript environment

[3] ogr2ogr npm module https://www.npmjs.com/package/ogr2ogr/(accessed May 8, 2017)

Data transmission rate depends on the response size. To reduce the size of encoded GeoJSON and make it smaller than its original size without loss of information, a DEFLATE algorithm compression method is used. It achieves faster compression speed with relatively lower compression ratio compared to LZMA(Li et al., 2015). This observation is shown in performance comparison section. After sending the compressed GeoJSON, the client decompresses it, parses it and render on the browser using HTML5 canvas.

2.4 Data Visualization

When the client receives a compressed GeoJSON response from the server it should then be visualized on the client. To visualize data on the client, we have two types of viewers, a Simple Data Viewer(SDV) and WMS viewer (WMSV). Figure 2, shows a 5MB shapefile consisting of data of the roads in South Africa, visualized on SDV and WMSV.

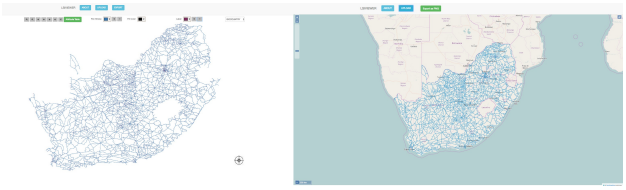

Figure 2. Roads of South Africa on a Simple data viewer(SDV) and WMS viewer (WMSV)

2.4.1 Simple Data Viewer(SDV) The client decompresses the GeoJSON response and sends decompressed data to the JavaScript modules. Web 2.0 has introduced HTML5, a new standard to create interactive, high quality web pages on a PC, tablet and a smartphone(Juntunen et al., 2013). Canvas element, which is introduced in HTML5, is crucial for heavy graphics based applications. Its vector rendering performance is more suitable to GIS applications(Park et al., 2011). It provides Application programming interface(API) which has methods to draw line, arc, circle, text, rectangles and render images. Using these canvas API methods we can render the GIS vector data smoothly on the browser.

The functional modules implemented in the JavaScript which are required to render the map on the canvas are listed below. While GetMapExtent and GetFeatureCoordinates define the canvas size and the feature mapping, the DrawMap and Symbology functions help render the data onto the canvas. Detailed description of the functions implemented are listed below.

(A) GetMapExtent() $\Longrightarrow$ return *(xMin, xMax, yMin, yMax)*
We need extent of the GeoJSON features to draw the map on the canvas, which can be computed with *GetMapExtent()* method. It returns *(xMin,yMin)* the coordinates for the leftmost corner, *(xMax,yMax)*, the coordinates for the right most corner. So, xMax, xMin here are the maximum and minimum longitude coordinates respectively. yMax, yMin are the maximum and minimum latitude coordinates respectively

(B) GetFeatureCoordinates() $\Longrightarrow$ return *(coordinates : [X$\prime$, Y$\prime$])*
This method scales the map with respect to the canvas size. This is done by computing new coordinates X$\prime$ and Y$\prime$. We can scale the size of map by scaling each co-ordinate by multiplying it with a scaling factor. If the original 2D coordinates are *(X, Y)*, for the scaling factor *(xScale, yScale)*, *(X$\prime$,*

Y′) are obtained as the new scaled co-ordinates. This can be mathematically represented as below:

$$xScale = canvas.width \div (xMax - xMin). \quad (1)$$

$$yScale = canvas.height \div (yMax - yMin). \quad (2)$$

$$Scale = xScale < yScale ? xScale : yScale. \quad (3)$$

From (1), (2), (3),

$$X' = (X - xMIN) * Scale. \quad (4)$$

$$Y' = (Y - yMIN) * Scale. \quad (5)$$

(C) DrawMap()
After the coordinates scaling, we use canvas built-in methods *lineTo()*, *moveTo()*, *fillRect()* to render the features on the canvas.

(D) Symbology
Symbology in GIS greatly affects how map can be interpreted. SDV allows the user to perform GIS abilities zoom, pan, color, label and export the map. Table 2 contains the list of functionalities and canvas API methods used to implement them.

Client side functionalities	Canvas API methods
Zoom in, Zoom out	scale()
Pan	translate()
Color	fill(), stroke()
Pen width	Uses linewidth property
Labelling	fillText()

Table 2. Shows the list of functionalities available on the toolbar and their respective implementation methods

2.5 WMS Viewer

In SDV, the user can not validate the vector data by comparing with the actual satellite imagery. So, we integrated WMS which helps us to visualize the GeoJSON data by overlaying it on a base map like Google Map or Openstreetmap(OSM). WMS layer provides a set of geo-referenced raster images which renders on the browser viewing area. WMS layer is loaded using Openlayers, an open source JavaScript library that supports OGC standards. It is implemented using AJAX and JavaScript programming language(Han et al., 2009). After a WMS layer is loaded, canvas adds the GeoJSON data as a layer on top of it.

3. PERFORMANCE COMPARISON

LSIViewer's performance can be primarily evaluated on the grounds of its rendering speeds in comparison to the Desktop GIS application viz., QGIS. The datasets used in this comparison study are ESRI Shapefiles downloaded from the United States Census[4] and DivaGIS[5] websites. Each dataset varies in size, between 5KB and 35MB covering the geometries including points, lines and polygons. To achieve this performance we used data compression algorithms like DEFLATE and LZMA on the server side. The data sizes are compared for uncompressed data and compressed data with both the algorithms to delineate the degree to which the data transmission rate has been reduced. We also analyze the time taken for data processing and rendering in each of the algorithms and show why we choose one over the other. Finally for the efficient use of LSIViewer, we compare rendering speeds on different web browsers for datasets of varying size. Table 3 contains information related to the datasets used for analyses and Table 4 contains the system configuration of the client and the server instances used for comparison study.

Name of the Shapefiles downloaded	Layer Selected	File Size	Vertices	Features
Rhode island	Secondary schools	7KB	445	1
Kentucky	Secondary schools	23KB	1385	5
Connecticut	Secondary schools	104KB	6319	8
Arizona	Secondary schools	274KB	17093	15
Massachusetts	Secondary schools	395KB	24192	31
Rhode Island	County subdivision	630KB	37325	40
Virgin islands of the U.S	Estate	897KB	50276	336
Michigan	S L D U	1MB	62574	39
Rhode Island	Roads	1.5MB	82843	1299
Hawaii	Roads	2.5MB	150813	1057
South Africa	Roads	5MB	55858	5559
Mexico	Roads	8MB	103918	10425
India	Roads	15MB	210329	19148
Australia	Roads	23MB	316655	27651
USA	Roads	35MB	431026	46995

Table 3. ESRI Shapefile datasets used for observations

3.1 Compression Method for Efficient Transmission of Data

While choosing a compression method, there are various factors involved like the compression ratio(CR) and the compression time(CT). For our study, we chose DEFLATE and LZMA algorithms due to browser support and their popularity. Figure 3 Compares the difference in size of data transferred back to the client side using GeoJSON encoding by using compression techniques(DEFLATE and LZMA). The x-axis shows the file size of datasets in increasing order. Figure 4 indicates that the overall time taken for a response to reach the client when used DEFLATE compression method is faster than LZMA for all the datasets.

For example, in Figure 3 a 35MB dataset is reduced to 6.4 MB and 7.2 MB with LZMA and DEFLATE respectively. And from Figure 4 for the same file, the time taken to reach the client for LZMA compressed data is 18 seconds while DEFLATE takes 6 seconds. This result clearly indicates that, though the size of

[4]US Census https://www.census.gov/cgi-bin/geo/shapefiles/index.php/(accessed May 8, 2017)
[5]DIVA GIS http://www.diva-gis.org/gdata/(accessed May 8, 2017)

Instance type	Specifications
Server where Node.js is running	Intel(R) Core Duo CPU E8500 @3.16 GHz OS: Ubuntu/Linux SMP-Kernel 3.2.0-64-generic 8 GB DRAM
Client system to test application on Firefox, Chrome, Opera and Safari	Intel Core i5 @1.4 GHz OS : MAC OS X El Capitan RAM : 4 GB DDR3
Client system used for testing on IE	Intel Core i5 @1.80 GHz * 4 OS : Windows 10 RAM : 4 GB DDR3

Table 4. System specifications of the instances used for testing

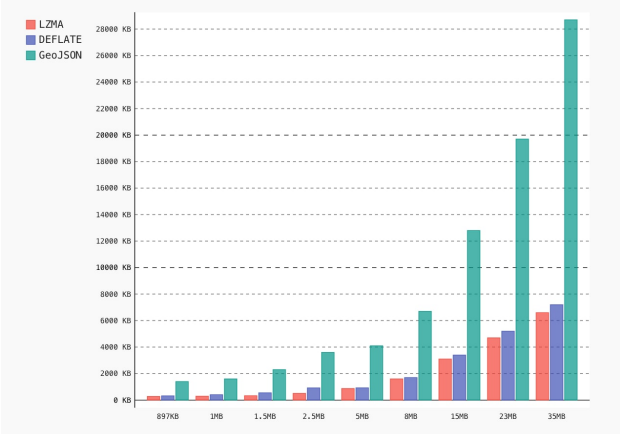

Figure 3. Data size comparison : LZMA v/s DEFLATE v/s Uncompressed GeoJSON

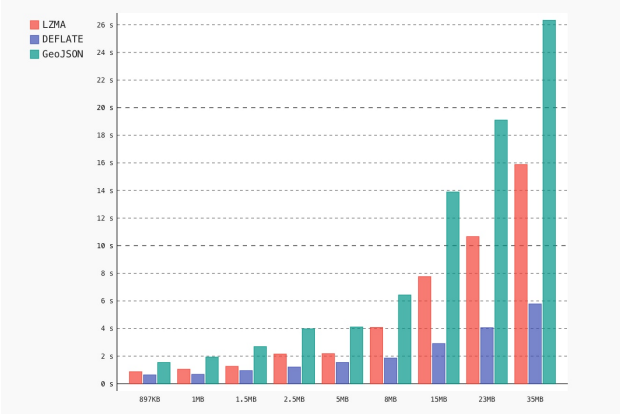

Figure 4. Data processing and transmission time comparison : LZMA v/s DEFLATE v/s Uncompressed GeoJSON

LZMA is lesser than DEFLATE, the time taken to compress the data on the server is more in LZMA method, which complements the conclusions derived by (Li et al., 2015). Based on these observations that the data processing time is equally crucial to the data compression ratios achieved, the application implements the DEFLATE method instead of LZMA.

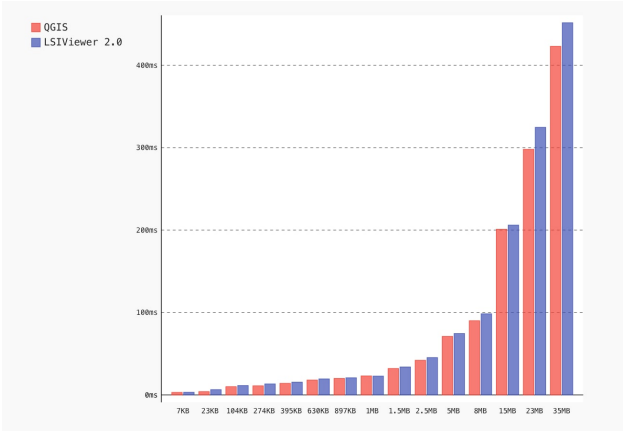

Figure 5. Rendering time comparison between LSIViewer and QGIS

3.2 Is Online Rendering Comparable to Desktop Rendering speeds?

One of the challenges to a data-heavy application like GIS to shift to an Online platform is the response time for rendering these datasets. So, in this study, we compare the time taken to render above datasets in both LSIViewer 2.0 and QGIS, a popular open source Desktop GIS application. As can be seen from Figure 5, across all datasets the performance of the desktop GIS is only marginally better than the online toolkit irrespective of the data-size, for example even a 35 MB shapefile, containing 46995 features with 431026 vertices, can be rendered in time that is very close (8∼ 10% slower) to QGIS rendering time, which is less than a second. This implies that the suite of online technologies that have been implemented as part of LSIViewer 2.0 do show comparable data handling efficiencies.

3.3 HTML5, Browser type and Rendering performance of LSIViewer 2.0

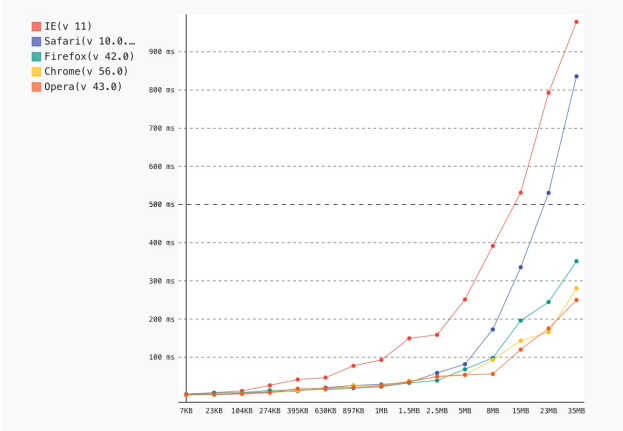

Figure 6. Comparison of rendering time over five popular browsers

The suite of technologies used in LSIViewer 2.0 are compatible with the current web technology standards that supports HTML5. Hence, the application should be agnostic to the user's browser of choice and follow the current development model of not using any additional plugins or software to be installed on the client side. In this subsection, we report the performance of the application across five popular browsers, in alphabetical order, - Chrome, Firefox, Internet Explorer (IE), Opera and Safari. From Figure 6,

Figure 7. LSIViewer application in a browser on a iPad Mini device

Figure 8. LSIViewer application in a browser on Nexus4(Android) device

it can be observed that the rendering of the ESRI Shapefiles over these five popular browsers happens in less than one second.

3.4 Rendering on multiple devices

As the number of smartphones and tablets increasing day by day(Khan and Shah, 2016), it is important to test the application on multiple devices. Figure 7 and Figure 8 shows LSIViewer running on a mobile web browser on both smartphone and a tablet. The application can be opened on any web browser on any device, as shown in the previous section. Hence, we can say that LSIViewer is compatible on multiple devices across different platforms.

4. CONCLUSION AND FUTURE WORK

In current architecture and deployed version of LSIViewer 2.0, it is successfully demonstrated that the server dominant processes which include rendering and other GIS operations, can be migrated to client side with similar performance characteristics as the desktop based versions of GIS tools. This does show that geospatial data handling and rendering can no longer be a platform-specific, operating-system-specific application and can use/share and work with the geospatial data across multiple devices with different operating systems, as long as they do have a browser and connectivity to access a central or enterprise level server where LSIViewer is deployed. The recent developments in web technology standards, primarily HTML5, have brought to the fore some of the capabilities that a core geospatial toolkit looks for and by integrating these with the geospatial data models has enabled the development of an online geospatial rendering application, LSIViewer. As the client level processing is being done in a JavaScript environment, it was found that GeoJSON, as the data model, is more appropriate than the use of GML, the current standard for WebGIS. Although LSIViewer 2.0 follows a client-server architecture, using the DEFLATE algorithm for compression of the GeoJSON data being transferred between client and server, the processing and transmission time is largely improved to an extent of 75%.

The experiments with multiple data size files containing varied spatial features, has shown that the rendering speeds achieved on LSIViewer 2.0 are very similar to a desktop GIS application, which can be considered as a great precursor to the rise of web-based GIS applications. For instance, in a client-server model, a single server with 100 clients where rendering is happening on client's side is far more scaleable than 100 requests with its associated overheads, all being processed by a single server. All the web browsers that supports HTML5 give this capability to the client making this application a platform-independent and installation-free toolkit. LSIViewer 2.0 requires continuous development of its features and performance improvement techniques on both the server side and the client side. While the current implementation takes care of the basic functionalities, there can be an option for other operations like support for multiple vector formats, adding projection support, overlaying multiple layers of vector data and could be extended to support visualization of CityGML data. Since the application is a web-based one, we can always integrate GIS specifications like WFS, WCS. Also, this design can be a starting point to build an enterprise level collaborative platform for developers and users to share and process/work on their data.

REFERENCES

Anthes, G., 2012. HTML5 leads a web revolution. *Communications of the ACM* 55(7), pp. 16.

Boulos, M., Warren, J., Gong, J. and Yue, P., 2010. Web GIS in practice VIII: HTML5 and the canvas element for interactive online mapping. *International Journal of Health Geographics* 9(1), pp. 14.

Bunting, P., Clewley, D., Lucas, R. M. and Gillingham, S., 2014. The remote sensing and GIS software library (RSGISLib). *Computers & Geosciences* 62, pp. 216–226.

Chaniotis, I. K., Kyriakou, K.-I. D. and Tselikas, N. D., 2014. Is node.js a viable option for building modern web applications? a performance evaluation study. *Computing* 97(10), pp. 1023–1044.

Dangermond, J., 1988. Trends in GIS and comments. *Computers, Environment and Urban Systems* 12(3), pp. 137–159.

Han, W., Di, L., Zhao, P. and Li, X., 2009. Using ajax for desktop-like geospatial web application development. In: *2009 17th International Conference on Geoinformatics*, Institute of Electrical and Electronics Engineers (IEEE).

Herrick, D. R., 2009. Google this! In: *Proceedings of the ACM SIGUCCS fall conference on User services conference - SIGUCCS'09*, Association for Computing Machinery (ACM).

Jun, S. H. and Doh, K. T., 2013. Design and implementation of web GIS server using node.js. *Journal of Korea Spatial Information Society* 21(3), pp. 45–53.

Juntunen, A., Jalonen, E. and Luukkainen, S., 2013. HTML 5 in mobile devices – drivers and restraints. In: *2013 46th Hawaii International Conference on System Sciences*, Institute of Electrical and Electronics Engineers (IEEE).

Kelso, N. V. and Patterson, T., 2009. Natural earth vector. *Cartographic Perspectives* 64, pp. 45–50.

Khan, M. H. and Shah, M. A., 2016. Survey on security threats of smartphones in internet of things. In: *2016 22nd International Conference on Automation and Computing (ICAC)*, Institute of Electrical and Electronics Engineers (IEEE).

Li, W., Song, M., Zhou, B., Cao, K. and Gao, S., 2015. Performance improvement techniques for geospatial web services in a cyberinfrastructure environment – a case study with a disaster management portal. *Computers, Environment and Urban Systems* 54, pp. 314–325.

Lu, X., 2005. An investigation on service-oriented architecture for constructing distributed web GIS application. In: *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, Institute of Electrical and Electronics Engineers (IEEE).

Morris, S., Morris, A. and Barnard, K., 2004. Digital trail libraries. In: *Proceedings of the 2004 joint ACM/IEEE conference on Digital libraries - JCDL'04*, Association for Computing Machinery (ACM).

Nair, L. R., Saleem, S. and Shetty, S. D., 2016. Scalable interactive geo visualization platform for GIS data analysis. In: *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, Institute of Electrical and Electronics Engineers (IEEE).

Park, M.-R., Park, K.-H. and Ahn, J.-S., 2011. Design and implementation of a computing environment for geovisual analytics using HTML5 canvas. *Journal of the Korean Association of Geographic Information Studies* 14(4), pp. 44–53.

Peng, Z.-R. and Zhang, C., 2004. The roles of geography markup language (GML), scalable vector graphics (SVG), and web feature service (WFS) specifications in the development of internet geographic information systems (GIS). *Journal of Geographical Systems*.

Porta, J., Parapar, J., García, P., Fernández, G., Touriño, J., Doallo, R., Ónega, F., Santé, I., Díaz, P., Miranda, D. and Crecente, R., 2013. Web-GIS tool for the management of rural land markets. *Earth Science Informatics* 6(4), pp. 209–226.

Puder, A., 2004. Extending desktop applications to the web. *ISICT '04 Proceedings of the 2004 international symposium on Information and communication technologies* pp. 8–13.

Steiniger, S. and Bocher, E., 2009. An overview on current free and open source desktop GIS developments. *International Journal of Geographical Information Science* 23(10), pp. 1345–1370.

Wenjue, J., Yumin, C. and Jianya, G., 2004. Implementation of OGC web map service based on web service. *Geo-spatial Information Science* 7(2), pp. 148–152.

Winter, S. and Frank, A. U., 2000. Topology in raster and vector representation. *GeoInformatica* 4(1), pp. 35–65.

Wood, D., King, M., Landis, D., Courtney, W., Wang, R., Kelly, R., Turner, J. A. and Calhoun, V. D., 2014. Harnessing modern web application technology to create intuitive and efficient data visualization and sharing tools. *Frontiers in Neuroinformatics*.