Interactive Web-based Visualization for Accessibility Mapping of Transportation Networks

Alexander Schoedon\textsuperscript{1}, Matthias Trapp\textsuperscript{2}, Henning Hollburg\textsuperscript{3}, and Jürgen Döllner\textsuperscript{2}

\textsuperscript{1} Institute for Geography at the University of Potsdam, \textsuperscript{2} Hasso Plattner Institute at the University of Potsdam, \textsuperscript{3} Motion Intelligence GmbH

Abstract
Accessibility is a fundamental aspect in transportation, routing, and spare-time activity planning concerning traveling in modern cities. In this context, interactive web-based accessibility-map visualization techniques and systems are important tools for provisioning, exploration, analysis, and assessment of multi-modal and location-based travel time data and routing information. To enable their effective application, such interactive visualization techniques demands for flexible mappings with respect to user-adjustable parameters such as maximum travel times, the types of transportation used, or used color schemes. However, traditional approaches for web-based visualization of accessibility-maps do not allow this degree of parametrization without significant latencies introduced by required data processing and transmission between the routing server and the visualization client. This paper presents a novel web-based visualization technique that allows for efficient client-side mapping and rendering of accessibility data onto transportation networks using WebGL and the OpenGL transmission format. A performance evaluation and comparison shows the superior performance of the approach over alternative implementations.

1. Introduction
Today, accessibility analysis and visualization is often performed using Desktop GIS (Geographic Information Systems). Such systems exploit the computation power of desktop PC but possesses limited applicability to everyday life, due to data availability and access, as well as expert domain knowledge required by a user. With respect to these restrictions, performing server-based accessibility analysis in combination with interactive web-based accessibility map visualization (Fig. 1) has various advantages: (1) its usage is not limited to stationary desktop systems but available on a variety of devices (esp. mobile); (2) potentially massive data sources are not required to be completely transmitted, stored, or managed; and (3) implementations based on web-services and WebGL [JG16] can be easily integrated into existing systems and visualization frameworks. In contrast to existing accessibility-map visualization concepts, this paper focuses on visualizing the travel time data directly on the respective transportation network features, rather than (possibility generalized) polygons [GKD10] or specific graph layouts [KSW+S12]. This enables a precise mapping of travel data to the geo-referenced transportation network. However, considering the high geometric complexity (vertices, primitives) introduced by increasing quality of transportation networks [ZZ10], e.g., of massive open data transportation networks (OpenStreetMap (OSM) or General Transit Feed Specification (GTFS)), an implementation of an interactive web-based visualization technique comprises a number of conceptual and technical challenges.

Real-time rendering transportation networks as scenery for data visualization in web-based applications is a performance critical task depending on the geometric complexity of the network and associated travel times. For example, an OSM dataset of the Berlin region comprises approx. 9·10^7 edges (Oct. 2015). Using tradi-
tional visualization approaches using web browsers either faces
users with a predefined, static filtering and mapping (raster data)
or a notable computation-intensive rendering process (vector data).
These two fundamental approaches covering filtering, mapping,
and rendering web-based maps are widely established and have
proven to be effective, but exhibit drawbacks:

**Raster Formats:** Data transmitted in pre-rendered raster data for-
matos does not require any client-side processing prior to render-
ing, can be compressed as well as cached [ESR06], and is used
by major web-mapping services such as Google or Bing maps.
However, one major disadvantage is the lack for client-side filter-
ing and mapping without requesting a complete map tile reload.

**Vector Formats:** In contrast thereto, geodata transmitted using
vector formats enable client-side filtering, mapping, and ren-
dering. This client-side processing however introduces a ma-
or performance impact: both the data processing and rendering
are usually performed on CPU using JavaScript (JS) algo-
rithms. Recent approaches do support hardware-accelerated ren-
dering (GPU) but lack functionality for client side vector data
processing [Gaf12].

Thus, both approach changes in filtering (e.g., selecting a travel
time threshold) or mapping (e.g., color mapping, line styles etc.)
would result in a complete data re-transmission, loading, and
processing. To summarize, an interactive visualization technique
for web-based accessibility maps should adhere to the follow-
ing requirements and challenges: it should support a web-based,
hardware-accelerated implementation using WebGL [Par12] (R1);
a standardized and compact data representation that allows for
decoupling network geometry from temporal data to reduce data
transmission and updates (R2); as well as enable interactive client-
side filtering, mapping, and rendering for visual feedback (R3).

This paper proposes a new approach for web-based visualization
of accessibility maps based on transportation network data. This
gometry-based approach uses vector data (lines) stored and trans-
mitted using a new standardized glTF file format, which reduces
performance-critical computations in the visualization client (i.e.,
coordinate transformations) and thus facilitates real-time rendering
with high run-time performance yielding low client response times.
To summarize, this work makes the following contributions: (1)
it presents a concept to decouple the visualization geometry and
and data items for interactive web-based accessibility maps based on
WebGL [JG16], and (2) it demonstrates the effectiveness of this
approach by a comparative performance evaluation of different im-
plementation variants.

Related work comprises basically accessibility map visual-
ization, web-based visualization frameworks, and the rendering
of transportation networks using GPUs. Glander et al. present
an accessibility map visualization technique with a focus on
polygon-based approaches [GKD10], and raster-based distance transforms [MG10] which both lack precision in display offered
by our approach. In [YLT’15], a web-based system for visual-
ization of multi-modal accessibility for multiple land-uses is pre-
cluded. However, the visualization technique does not focus on
the specifics of transportation network representations. Altmaier
et al. (2003) was among the first to outline issues in web-based
geovisualization applications [AK03]. In [BS07], challenges, re-
quirements, and concepts of client-based browsing of spatial data
on the World Wide Web are discussed. The presented system
is based on a Java-Applet and does not exploit modern web technolo-
gies for rendering complex spatial data. Vaaraniemi et al. (2011)
as well as Trapp et al. (2015) develop and evaluate approaches
for transport network visualization utilizing modern graphics hard-
ware [VTW11, TSD15]. However, the presented rendering tech-
niques can currently not be implemented using technologies for
browser-based rendering and do no cover specifics of data repre-
sentation and formats.

2. Accessibility-Maps for Transportation Networks

This section briefly discusses design decisions on web-mapping
frameworks, data formats, and required geographic projections. To
evaluate the concepts based on real-world data sets, we choose to
rely on the Route360-JS\(^*\) API for server-side computing of rout-
ing data. Although there are alternatives with advanced WebGL-
integrations available (e.g., OpenLayers 3, Cesium), this work fo-
cuses on the web-mapping framework Leaflet-JS for interoperabil-
ity with the Route360-JS API. However, the glTF-based approach
is not specific to Leaflet and can be integrated into other web-
mapping frameworks.

The exemplary transportation network used in this paper is a part
of an OSM data set of Berlin. It comprises streets, footways, and
data of the transportation infrastructure network. Based on this net-
work a single-source shortest-path accessibility analysis [Mey01]
is performed using the Route360-JS API. It results in the travel
times required by foot, bicycle, car, or public transportation from a
single starting point to all remaining nodes of the network. The geo-
metry, topology, and accessibility data (represented on a per-node
basis) is stored in a database which is used for data conversion to
the file formats of our visualization techniques.

For handling data-intense client/server communication, it is im-
portant to evaluate options on data exchange formats. Both, Cough-
lin and Trevett motivate why a standardized data format close
to hardware devices for applications on the web are required
[Cou14, Tre12]. We considered the following file formats for com-
parison: (1) COLLADA this digital asset exchange format (.dae)
has been established for modeling purposes and exchange of geo-
metry and scene descriptions [BF08]; (2) the Geography Markup
Language (.gml) is based on a XML grammar standardized by
the Open Geospatial Consortium (OGC) to represent geographical
features [GML07]; (3) GeoJSON (.json) a JavaScript object no-
tation that is extended by geographic features comprising geomet-
ries and its properties [BDD’08]; and (4) the OpenGL transmis-
son format (.gltf) was recently released by Khronos Group and
is designed to be a file format close to the requirements of the ren-
dering hardware [CP15]. Table 1 (next page) compares the file for-
mats with respect to their memory footprint and client-side process-
ing requirements. The former is important to evaluate the required
bandwidth, while the processing is the aforementioned bottleneck
in performance of transforming geographic data into GPU array-
buffers.

Concerning memory consumptions, both glTF and COLLADA
perform above average. The comparison uses the Cesium Milk
Memory (Byte)

<table>
<thead>
<tr>
<th>Format</th>
<th>Required, decompress</th>
<th>Required, decompress only</th>
<th>Decompress only</th>
</tr>
</thead>
<tbody>
<tr>
<td>.dae.gz</td>
<td>54,949</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.glb.gz</td>
<td>89,168</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.gltf.gz</td>
<td>212,788</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.json.gz</td>
<td>67,693</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.gml.gz</td>
<td>58,791</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.gml</td>
<td>2,698,953</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparison and rating (dark background means not suitable) of data formats with respect to the amount of memory required for server-to-client data transmission and client-side processing prior to rendering. The data set used for comparison comprises 1840 vertices and 3624 faces (no texture data included).

To display geographic coordinates in Mercator projection (EPSG:3857) on a GPU-rendered map, two major steps have to be performed. First, the geographic projection of each coordinate has to be transformed into a normalized projection by translating the coordinate origin to the top-left corner by the distance of half an equator. Further, the view geometry can be retrieved from Leaflet-JS and used as an offset for the displayed geometries. To display geographic coordinates in Mercator projection (EPSG:3857) on a GPU-rendered map, two major steps have to be performed. First, the geographic projection of each coordinate has to be transformed into a normalized projection by translating the coordinate origin to the top-left corner by the distance of half an equator. Further, the view geometry can be retrieved from Leaflet-JS and used as an offset for the displayed geometries. To display geographic coordinates in Mercator projection (EPSG:3857) on a GPU-rendered map, two major steps have to be performed. First, the geographic projection of each coordinate has to be transformed into a normalized projection by translating the coordinate origin to the top-left corner by the distance of half an equator. Further, the view geometry can be retrieved from Leaflet-JS and used as an offset for the displayed geometries. To display geographic coordinates in Mercator projection (EPSG:3857) on a GPU-rendered map, two major steps have to be performed. First, the geographic projection of each coordinate has to be transformed into a normalized projection by translating the coordinate origin to the top-left corner by the distance of half an equator. Further, the view geometry can be retrieved from Leaflet-JS and used as an offset for the displayed geometries. To display geographic coordinates in Mercator projection (EPSG:3857) on a GPU-rendered map, two major steps have to be performed. First, the geographic projection of each coordinate has to be transformed into a normalized projection by translating the coordinate origin to the top-left corner by the distance of half an equator.

To minimize the client-side data processing and to reduce transformation operations, it’s important to reduce re-projections and avoid spherical units (e.g., degree, latitude/longitude). Leaflet-JS uses the EPSG:4326 standard projection, which is the world geodetic system 1984 (WGS84) and uses a latitude/longitude coordinate format, thus all programming interfaces of Leaflet return values in degree. To remove the need for computing-intense spherical transformation and to simplify coordinates, all geographic references are projected to a normalized range [0,1] with it’s origin in the north-west corner. This moves as close as we can get to hardware-coordinates and conveys device-independence.

3. Browser-based Implementation Variants

This section covers Leaflet-JS integration details and limitations specific to different prototypes of web-based implementations for transportation networks used as scenery for an interactive accessibility-map visualization technique. Subsequently, the performance of the following approaches for Leaflet-JS plug-ins are evaluated (Sec. 3.3):

- **Leaflet line rendering (JavaScript + GeoJSON)** uses software rasterization for rendering. The interactive rendering capability of this approach is limited to small travel time ranges.
- **Canvas overlay (WebGL + GeoJSON)** uses a WebGL integration in Leaflet-JS and renders GeoJSON data representations. It is limited w.r.t. client-side data processing (Sec. 3.1).
- **gltf tiles (WebGL + glTF)** is based on a geometry buffer tiling service that explicitly utilizes the glTF data file format (Sec. 3.2).

During preliminary tests, the native line rendering shows limitations with respect to client-side data processing and rendering performance (above 5 min. travel time). Therefore, we choose not to

© 2016 The Author(s)

current viewport and request travel times respectively by using the
graph correspondence; (3) for visualization, the network tiles are
cached and directly used for mapping and rendering.

Table 2 highlights the advantages of glTF-tiling. It supports
client-side mapping (e.g., color mapping and lines styles) at run-
time on according to user inputs without retransmission of data,
similar to vector tiles. The data processing is transferred from client
to server-side, similar to raster-tiling approaches. This results in
high run-time performance with low latencies and the resulting ge-
ometries can be cached client-side. The rendering is performed on
dedicated GPU using WebGL and thus ensures real-time perfor-
mance for geometrical complex geodata sets.

3.3. Performance Evaluation and Comparison

The tests are conducted using a Lenovo Thinkpad X230 equipped
with an Intel Core i7-3520M CPU running at 2.90GHz (Quad-
core), 16GB DDR3-RAM @ 1600MHz (SODIMM), and an in-
tegrated Intel HD Graphics 4000 with 256MB RAM (shared).
The implementation variants are evaluated using Firefox 44.0 and
Chrome 48.0 browsers running on an Arch Linux kernel. The tests
are performed under the following conditions: mapping and render-
ning run-time are measured using the same zoom factor and center
location (complete data set is visible) using a full-screen viewport
resolution of 1366 × 768 pixels.

Table 3 shows an overview of the test data used for performance
measurement. The data is a subset of the inner city of Berlin and
comprises car travel times from a single starting point (cf. Fig.1).
Based on the maximum travel time set by the user (required for an
accessibility-map visualization), the geometric complexity varies
with respect to the number of vertices and lines to be displayed.
Figure 4 shows an overview of the performance comparison results
(more precise results are presented in the supplemental material).

Table 2: Comparison and rating (dark background means not suit-
able) of tiling approaches with respect to the requirements of data
processing, mapping, and rendering.

<table>
<thead>
<tr>
<th>Tiling Approach</th>
<th>Processing</th>
<th>Mapping</th>
<th>Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raster</td>
<td>Server</td>
<td>static</td>
<td>Server</td>
</tr>
<tr>
<td>Vector</td>
<td>Client/CPU</td>
<td>dynamic</td>
<td>Client/GPU</td>
</tr>
<tr>
<td>glTF</td>
<td>Server</td>
<td>dynamic</td>
<td>Client/GPU</td>
</tr>
</tbody>
</table>

Table 3: Varying geometric complexity of the test data set.

<table>
<thead>
<tr>
<th>Geometry</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertices</td>
<td>36</td>
<td>162</td>
<td>1,901</td>
<td>14,874</td>
<td>25,821</td>
</tr>
<tr>
<td>Edges</td>
<td>20</td>
<td>88</td>
<td>1,006</td>
<td>7,950</td>
<td>13,829</td>
</tr>
</tbody>
</table>

These indicate superior run-time performance of the glTF approach
over the native line rendering and GeoJSON implementation for
mapping and rendering. Further, they show an increased scalability
of the glTF approach w.r.t. the geometric complexity of the network
and significant differences in mapping and WebGL 1.0 rendering
performance for the different web browsers tested.

4. Conclusions and Future Work

This paper presents a new approach for efficient rendering and
mapping of interactive web-based accessibility-maps. The pre-
sented tiling approach based on glTF file format reduces the
amount of data processing operations required by the client, in-
creases run-time performance for efficient rendering, and simulta-
necessarily reduces the amount of transmitted data for visualization.
With respect to rendering, the work compares this technique to two
alternatives, showing a significant performance increase for mass-
ive data sets. The presented results enable the development of new
interactive techniques for web-based transportation network visu-
alization systems and tools.

Based on these results, future work focuses on developing a glTF
processing back-end with separated tiling and routing server logic.
The presented approach supports the decoupling of network geometry
and accessibility data, which further reduces the amount of travel
data transmission during updates. Further, a complete working
client/server infrastructure is evaluated regarding it’s overall perform-
ances to compare the results with traditional raster or vector
approaches.

Acknowledgments

This work was funded by the Federal Ministry of Education and
Research (BMBF), Germany within the InnoProfile Transfer re-
search group “4DnD-Vis” (www.4dndvis.de) and “MOBIE” (www.
mobie-project.de).

Figure 4: Run-time performance for data mapping and rendering with respect to different travel times (1-10 min.), browser (blue: Chrome,
orange: Firefox), and implementation (dot: lines rendering, dash: canvas overlay, solid: glTF-tiles) in milliseconds on a logarithmic scale.
References


[Cou14] COUGHLIN B.: 3D for the modern web - Declarative 3D and glTF. GMU CS-752 (2014). 2


[ESR06] ESRI: Comparing Vector and Raster Mapping for Internet Applications. ESRI 380 New York St., Redlands, CA 92373-8100 USA, aug 2006. 2


[GML07] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, GENEVA, SWITZERLAND: ISO 19136 (GML 3.2.1), 2007. 2


