Interactive 3D Visualization for Monitoring and Analysis of Geographical Traffic Data of Various Domains

Daniil Rodin(✉), Oded Shmueli, and Gershon Elber

Technion, Haifa, Israel
{daniil.rodin,oshmu,gershon}@cs.technion.ac.il

Abstract. Visual interactive tools are of great importance for monitoring and analysis of geographical data, and, in particular, traffic data. Substantial research effort goes into visualization techniques of various kinds of geography-bound traffic data. Unfortunately, such techniques are very domain-specific and often lack useful features. We propose an interactive visualization system for monitoring and analyzing traffic data on a 3D globe. Our system is general and can be transparently used in different domains, which we examplify by two simulated demonstrations of use cases: Logistic Service and Data Communication. Using these examples, we show that our approach is more general than the current state of the art, and that there are significant similarities between several domains in need of interactive visualization, which are mostly treated as completely separate.

Keywords: Geovisualization · Transport visualization · Network visualization · Visual analysis

1 Introduction

The concept of traffic appears in many areas and performing analysis and monitoring of traffic-related data and processes is often an absolute necessity. Visualization is an invaluable tool for analysis and monitoring tasks with visualization of movement data being a research topic of high interest [3].

There are several domains in need of visualization which deal with traffic data that can be naturally mapped onto the globe. These domains include transportation of goods, vehicles, people, data, money, etc. with each such domain requiring visualization tools. While much research in visualization of such domains is performed, it is mostly very domain specific.

In this work, we present a generic visualization system based on primitives that we believe can be applied to many different domains of traffic visualization. Our system visualizes traffic as an interactive animation on a 3D globe where the user can both navigate the globe, interact with various visualization primitives, and control movement on the time scale as well. More importantly, the user can query the data using the provided API, causing the visual primitives to be filtered, highlighted, and/or selected, and/or causing the camera to move towards the primitives representing the resulting data. The primitives include sites (static locations), pipes (fixed routes between sites), and packages (dynamic items representing whatever is being transported), all of which are organized into a spatial hierarchy. In this work, we demonstrate that these primitives can describe use cases from various, potentially hierarchical, domains. We thus further demonstrate that our system is well-suited for visualizing use cases from various, potentially hierarchical, domains, as it is based on the aforementioned primitives.

This work is structured as follows. In Sect. 2, we give an overview of related work. In Sect. 3, we describe the system architecture. We present two examples of use cases of the system, in Sect. 4, and then conclude and discuss future work, in Sect. 5.

2 Related Work

The need in geographical traffic visualization is similar to a need that arises in various fields. The research, however, is often focused on a specific domain. In this section, we discuss works that are relevant to our system, separated into two main domains, namely, vehicle traffic and traffic in computer networks.

2.1 Vehicle Traffic Visualization

With geo-positioning data for transport vehicles becoming increasingly available (with the notable example of GTFS[1]), so are visualization, monitoring, and analysis tools for such data. [8] presents a visual interactive tool for analyzing taxi trips using queries optimized for spatio-temporal data. [10] proposes a technique for optimizing visualization by detecting and utilizing spatial patterns when working with origin-destination type data. [13] provides a visual system for analysis of transportation data aimed at helping municipalities and transport companies. A recent survey of such systems and techniques can be found in [2].

However, all of the above approaches work with aggregated data, while our goal is to provide a system capable of real-time monitoring. Various local public transport information systems, such as [7], provide real-time information about current bus locations to the end users. TRAVIC[2], which is an implementation of [4], is a system that allows monitoring public transport around the world (data provided through GTFS) in real time with playback capabilities. These systems are, however, still heavily domain-specific.

2.2 Visualization of Traffic in Computer Networks

Another domain of geographical traffic visualization is visualization of traffic in computer networks. The benefits of geographical visualization of networking

1 https://developers.google.com/transit/gtfs/.
2 https://tracker.geops.ch/.

data as well as a prototype implementation of a network monitoring system are discussed in [11]. [1] visualizes geo-referenced networks (not necessarily computer networks) on a 3D globe while using surface deformation to reduce data clutter. This approach is expanded by [6], which also adds a method of ‘peeling’ slices of the Earth surface to show occluded regions. [12] unifies the ideas of [11] and [1] in a single system that provides analytical tool for geo-referenced computer networks. Neither of these systems, however, combines the capabilities of real-time monitoring, 3D visualization, and domain-independence.

3 System Architecture

In this section, we discuss our system from the technical standpoint. First, we give a brief overview of the architecture in Sect. 3.1. Then we describe what kind of input data the visualizer uses in Sects. 3.2 to 3.4. We mention how the visualizer and the data provider communicate in Sect. 3.5. Finally we discuss the capabilities of the visualizer in Sect. 3.6.

3.1 Architecture Overview

In this subsection, we describe the overall architecture of the system, which is shown in Fig. 1. We discuss each subsystem in detail in the next subsection.

The main part of the system is the visualizer, which is a client-side object responsible for taking the data from the data provider (described next) and generating an interactive 3D visual representation of the data inside the browser. The visualizer in our system is implemented using the Cesium library [5], which gives the end user the ability to navigate on the globe, navigate though the timeline (seeking, playback forward or backward at a given speed), explore the data, etc. The visualizer also provides a client-side API to allow integration with higher level management and analytics systems, which in turn allows operations such as filtering and highlighting the data that meets user-specified criteria.

The visualizer gets the data to visualize from the data provider (Fig. 1(b)). From the point of view of the visualizer, the data provider is also a client-side object, but in real world scenarios this object will be a simple proxy to a remote server that sends the data using a network transfer protocol, like HTTP.

The data is largely separated into a static and a dynamic part. The static data is provided directly as is and does not change over time. The dynamic data, on the other hand, can change over time and is thus provided in the form of events. Each event has a timestamp and information about a piece of dynamic data being created, modified, or deleted. This allows the system to be used for both retrospective analysis and real-time monitoring.

3.2 Static Data

We start our in-depth description of the system with the static data. As stated in the overview, the static data does not change over time. While it can be loaded

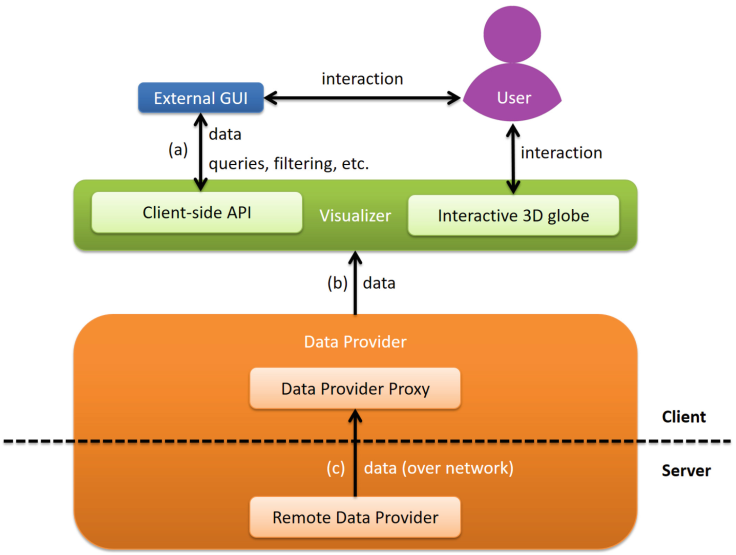


Fig. 1. Architecture overview (Color figure online)

partially on-demand, and its visual representation can change (e.g., collapse or expand, based on the viewing distance), the underlying data is considered unchanged and is provided directly.

Spatial Nodes. All the static data is defined within a hierarchy of what we denote as "spatial nodes", defined as follows.

Definition 1. *A* ***spatial node*** *is a named position and orientation in 3D space that represents a certain location or region. Together, spatial nodes form a tree structure denoted as* ***spatial node tree****. The root spatial node is the Earth itself. Each non-root spatial node can either be defined as an absolute position and orientation (i.e., as latitude, longitude, altitude, heading, pitch, and roll) or using position and orientation relative to its parent spatial node.*

An example of such a hierarchy is Earth, continents, countries, regions, cities, neighborhoods, buildings, floors, and rooms. All the other static data (and most of the dynamic data) is defined over the spatial node hierarchy.

In addition, each spatial node can specify a view distance such that when the camera is further from the node than that distance, the node is considered 'collapsed' and has a different visual representation (we discuss how visual representations are specified in Sect. 3.4). When a spatial node is collapsed, its descendants are not shown at all. This is necessary to reduce visual clutter when looking at complex node trees from afar.

Sites and Pipes. To further help structure the data and thus provide a streamlined API for placing dynamic data, we introduce two additional static data types, defined below.

Definition 2. *A* ***site*** *is a special spatial node that can be used as a storage for dynamic data.*

Examples are airports which can hold airplanes, servers which can hold pieces of data, etc. A site also has a user-visible name and a description in the form of HTML. Each spatial node can either be a site or not.

Definition 3. *A* ***pipe*** *is a bi-directional connection between two sites.*

The pipes serve two main purposes. First, a pipe can be used as a relative position for a piece of dynamic data thus removing the burden of always dealing with absolute positions (see Sect. 3.3). Second, the pipe is visible to the user, which helps to understand at a glance the origin and the destination of dynamic data moving along the pipe. Pipes can be specified in three forms: as geodesic lines, geodesic arcs (i.e., with the middle part elevated), and as explicit sequences of points in space, interpolated with a given degree of continuity. Together, sites and pipes effectively form a graph.

3.3 Dynamic Data

Every piece of dynamic data in our system is represented as a ‘package’, defined as follows:

Definition 4. *A* ***package*** *is a piece of data that has a name, description, position and orientation in space, visual representation (discussed in Sect. 3.4), and any kind of custom properties. All of these properties can change over time.*

The most important dynamic property of a package is its position in space. It can be set in several forms:

- Absolute position on the Earth.
- Absolute position relative to a spatial node.
- In a pipe with a given relative position (interpolation amount between the beginning and the end of the pipe).
- On a site.
- Inside another package (obviously, prohibiting circular nesting).

The last option allows us to dynamically nest packages inside each other and thus enable such use cases as people inside airplanes flying between airports (representing both people and planes with packages) and pieces of data inside data transfers between data centers (again, pieces of data and transfers are represented with packages), etc.

Since the properties of packages may change over time, the data about packages is provided to the visualizer in the form of events. Each event has a timestamp, a package ID, and what happened to that package at that timestamp, i.e., being created, deleted, or having some of its properties changed. This also enables real-time visualization by sending events as they happen in real life.

3.4 Visual Elements

Spatial nodes and packages can have visual representations in the form of lists of visual elements.

Definition 5. *A **visual element** is an atomic piece of visual representation of an object (site, package, etc.).*

We heavily base our visualizer on the Cesium library [5], and thus, visual elements are identical to the visual properties of entities in Cesium. The visual element types include:

- Billboard: a screen-space image with a given pixel size and Z offset (for depth ordering).
- Model: a 3D model (in glTF format[3]) with support for dynamic scaling in the form of minimum pixel size and maximum scale.
- Polygon: a flat shape specified by its border points.
- Wall: a wall-like shape specified as vertical extrusion of a given polyline by a given height.
- Box: an embedded cube model.

All the above visual element types can be colored and textured. 3D models can also be animated.

3.5 Data Provider

As the name implies, the role of the data provider is to provide the relevant data (discussed in the previous sections) to the visualizer.

Definition 6. *The **data provider** is the part of the system that provides the data (spatial nodes, sites, packages, etc.) as shown in Fig. 1(b).*

From the point of view of the visualizer, the data provider is simply a client-side object that provides the capabilities described in the rest of this section.

Providing Metadata. To be able to request from the data provider the actual data to visualize (e.g., sites and packages), the visualizer first has to know some global information about the data. We denote this global information as 'metadata'.

Definition 7. ***Metadata** is the global information, describing the actual data that the data provider provides. The metadata includes the overall timespan of the process being visualized, the ID of the root spatial node, a textual description of the process the data represents, and custom properties to cover the use cases we have not foreseen.*

[3] https://www.khronos.org/gltf/.

Providing Static Data. As discussed in Sect. 3.2, all the static data is bound to the spatial node tree, and thus acquiring the static data simply means acquiring the spatial node tree itself. However, there may be cases when the whole tree contains much more information than what the end user actually needs or can interact with in real time. For example, a spatial node tree might contain detailed information about buildings in 100 different cities, but the end user is going to zoom-in on only one or two of those cities. In such cases, it is useful to support on-demand loading of spatial node subtrees.

Acquiring static data is thus implemented as a method that returns a subtree of a spatial node tree starting at a node with a given ID. When returning descendants of this subtree root, the data provider may decide to mark some of them as 'details-on-demand' instead of returning them completely. The visualizer then may request them separately or ignore those that the user never gets too close to.

We have also considered an option of letting the visualizer decide the needed level of detail, but decided to let the data provider decide, since it has much more information about the data. It is theoretically possible to let the visualizer provide an abstract "level of detail" numeric value, which the data provider would consider to decide which nodes to mark as details-on-demand, but we leave it for the future work.

Providing Dynamic Data. As discussed in Sect. 3.3, in our system, dynamic data is represented as packages and is provided in the form of events. To enable provision of events in real-time, the event provision is implemented via subscription. The visualizer subscribes to events by giving the data provider a callback function that is called for every event received.

API. The described capabilities of the data provider are exposed as the following narrow API. This API is used to provide the data to the visualizer, as shown in Fig. 1(b).

- **getMeta**(): returns the metadata (asynchronously).
- **getSpatialSubtree**(*subtreeRootId*): returns (asynchronously) a spatial node subtree starting at the node with ID *subtreeRootId*.
- **subscribe**(*eventCallback*): subscribes to dynamic data events.
- **unsubscribe**(*eventCallback*): unsubscribes from dynamic data events.

As discussed in Sect. 3.1, in real life scenarios, this client-side object is likely to be a simple proxy to a remote server that exposes these same capabilities as a web API.

3.6 The Visualizer

Definition 8. *The* ***visualizer*** *is the client-side component responsible for visualizing the data and responding to user input. (Green rectangle in Fig. 1.)*

The visualizer provides two ways of interaction: the 3D globe, with which the user can directly interact, and the client-side API.

Interactive 3D Globe. The main goal of our system is presenting the data to the user in a visual and interactive form, which is one of the responsibilities of the visualizer. We employ the Cesium library to render various data on top of the 3D globe, in a browser. The visualizer is thus responsible for converting the data it received from the data provider to Cesium entities and Cesium primitives that are actually rendered. This is true for both static and dynamic data.

An important feature of the visualizer that is offered directly to the end user is the ability to smoothly navigate through time in the form of playback controls, whereas the user controls the direction and speed of animation. The visualizer is thus also responsible for correctly placing packages in both space and time, and also correctly handling cases of package nesting.

Client-Side API. The necessary capabilities are, however, not limited to what can be done by directly interacting with the 3D globe. (Note that hereafter, we sometimes refer to spatial nodes, sites, pipes, and packages as 'objects' when discussing capabilities not unique to one type only.) There is an obvious need for capabilities such as filtering, highlighting, and tracking objects that satisfy certain criteria. By 'filtering' and 'highlighting' we mean the following.

Definition 9. ***Filtering*** *is hiding and showing objects based on a given criterion (**filter**), usually supplied as a function that takes an object and returns a boolean value indicating whether that object should be visible or not.*

Definition 10. ***Highlighting*** *is temporarily modifying visual properties of some objects to make them visually distinguishable. The exact visual change is defined by custom functions.*

These capabilities are provided through the client-side API of the visualizer, which is shown below. This API is meant to be used by arbitrary external GUI, as shown in Fig. 1(a).

- **spatialNodes**: a filterable collection of spatial nodes.
- **sites**: a filterable collection of sites.
- **pipes**: a filterable collection of pipes.
- **packages**: a filterable collection of packages.
- **defineHighlighting**(*highlight*, *unhighlight*): sets the functions that would be used to highlight objects.
- **registerCustomVisual**(*name*, *visual*): registers a visual representation to use for a visual element named *name*, effectively extending the list of visual elements (Sect. 3.4) that can be provided by the data provider.
- **highlight**(*objects*): highlights a given set of objects using the highlighting method defined with **defineHighlighting**.
- **select**(*object*): selects the given object.
- **navigateTo**(*objects*): navigates the camera to show the given set of objects.

Each of the filterable collections of objects serves as a sub-API with the following methods:

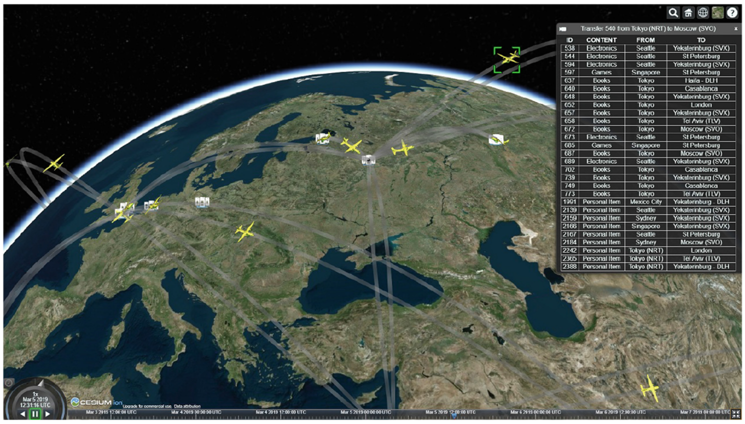
- **visibilityFilters**: a modifiable collection of filters that determine visibility of objects within the 3D view. A filter is a function that takes an object and returns a boolean value indicating whether the object should be visible.
- **queryVisible**(*filter*): returns all visible objects that satisfy the given filter.
- **query**(*filter*): returns all objects (including invisible ones) that satisfy the given filter.

The client-side API also allows the system to be used as an embedded visualizer for higher level data analysis systems such as Elastic Search [9].

4 Examples

In this section, we present two example use cases for our system. First is a logistic service that governs delivery of parcels (Sect. 4.1), and the second is communication between data centers (Sect. 4.2). Note that the only difference between the examples is the data returned by the data provider (spatial nodes, sites, pipes, and packages). Everything else stays the same, which makes our system quite universal.

4.1 Logistic Service


Transfer 540 from Tokyo (NRT) to Moscow (SVO)

ID	CONTENT	FROM	TO
538	Electronics	Seattle	Yekaterinburg (SVX)
544	Electronics	Seattle	St Petersburg
594	Electronics	Seattle	Yekaterinburg (SVX)
597	Games	Singapore	St Petersburg
637	Books	Tokyo	Haifa - DLH
640	Books	Tokyo	Casablanca
648	Books	Tokyo	Yekaterinburg (SVX)
652	Books	Tokyo	London
657	Books	Tokyo	Yekaterinburg (SVX)
658	Books	Tokyo	Tel Aviv (TLV)
672	Books	Tokyo	Moscow (SVO)
673	Electronics	Seattle	St Petersburg
685	Games	Singapore	St Petersburg
687	Books	Tokyo	Moscow (SVO)
689	Electronics	Seattle	Yekaterinburg (SVX)
702	Books	Tokyo	Casablanca
739	Books	Tokyo	Yekaterinburg (SVX)
749	Books	Tokyo	Casablanca
773	Books	Tokyo	Tel Aviv (TLV)
1991	Personal Item	Mexico City	Yekaterinburg - DLH
2139	Personal Item	Seattle	Yekaterinburg (SVX)
2159	Personal Item	Sydney	Yekaterinburg (SVX)
2166	Personal Item	Singapore	Yekaterinburg (SVX)
2167	Personal Item	Seattle	St Petersburg
2184	Personal Item	Sydney	Moscow (SVO)
2242	Personal Item	Tokyo (NRT)	London
2365	Personal Item	Tokyo (NRT)	Tel Aviv (TLV)
2388	Personal Item	Tokyo (NRT)	Yekaterinburg - DLH

Fig. 2. The Logistic Service use case example. On the top right, you can see the contents of the selected airplane, i.e., the parcel packages bound to that airplane package.

The first example that we present is the Logistic Service (Fig. 2). It demonstrates how our system can be used in monitoring and analyzing parcel delivery around the globe. In this example, the sites are storage and sorting centers (including ones in airports) and there are two types of packages: parcels and transportation means for those parcels (e.g., airplanes).

First, we establish the spatial node tree in the form of countries, cities, and sorting facilities within cities. Then, demand in various places on the globe for various goods and items from other places on the globe is generated. For each instance of such demand, a parcel which is delivered through the various means of transportation (mainly flights) is created. We also added a case of door-to-door delivery, which, in addition to generated transport, also uses explicit coordinates to simulate GPS tracking of a delivery mini-truck (Fig. 3).

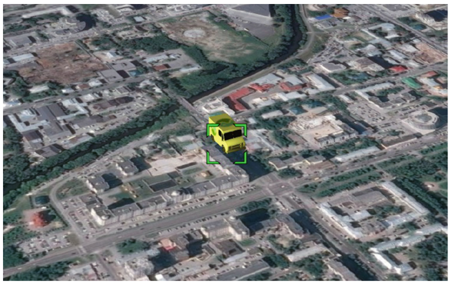
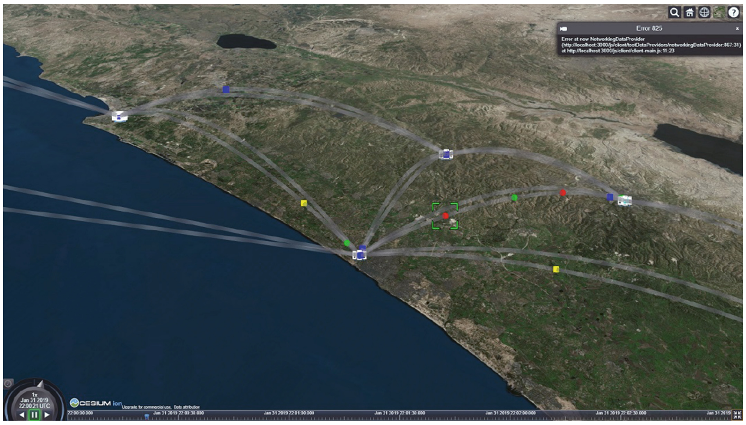


Fig. 3. A package that follows the path specified with absolute coordinates, representing a delivery mini-truck with GPS tracking.

Each parcel is a package and each plane or truck that carries it is also a package since both of these data types are dynamic. Our ability to place packages within other packages allows us to directly model the concept of placing parcels inside airplanes and mini-trucks. This, in turn, allows an intuitive way of executing Client API commands such as "highlight all parcels carrying electronics from Taiwan", which will visually highlight all planes, trucks, and sites that contain such parcels, as well as the parcels themselves.

4.2 Data Center Communication



```
Error 825
Error at new NetworkingDataProvider
(http://localhost:3000/js/client/testDataProviders/networkingDataProvider:867:31)
at http://localhost:3000/js/client/client-main.js:11:23
```

Fig. 4. The Data Center Communication use case example. Different shapes and colors of the items represent different categories of data transfers and network events.

The second example is communication between data centers (Fig. 4). Here, we have a network of data centers which constantly exchange various kinds of data.

In this case, the data centers are sites, direct connections between them are pipes, and the data transfers (possibly consolidated) between them are packages.

Another thing represented as a package in this simulation is the data itself, and an interesting type of such data is cache data for load balancing. Consider a simulated scenario where each data center is supposed to serve data to the closest customers, but the data requested may be stored in another data center. The data centers therefore exchange the data and store it as a cache, thus allowing several instances of the same data to exist.

For monitoring and analyzing the performance of the data centers and of load balancing, we believe it is useful to be able to track data instances and transfers that contain them. Fortunately, it is straightforward to do this within our system. Furthermore, the request to show all the sites and network transfers that contain specific data is conceptually no different from a request to show all the sorting centers and transport vehicles containing parcels with a certain type of product, which was discussed in the previous section.

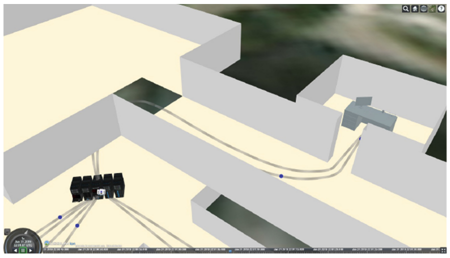


Fig. 5. Zooming in on a building to monitor the communication inside.

This example also demonstrates the benefit of 3D and the hierarchical structure by allowing the visualization of some buildings from the inside as 3D primitives and models (Fig. 5). This allows to monitor and investigate what is happening on the building level in the most intuitive way.

5 Conclusions and Future Work

We have presented a generic visualization system that allows monitoring and data analysis for geographical traffic data of various domains on a 3D globe with the ability to navigate in both space and time in a continuous (and thus visually coherent) way. Our infrastructure manages all static and dynamic hierarchical data, and exploits the Cesium library for rendering 3D geometry on a 3D globe, time controls, mouse-based spatial navigation, and basic GUI.

We have presented two simulated examples from two completely different domains (a logistic service and a data center communication) implemented in a very similar way based on the same set of abstractions. We, thus, demonstrate that there are significant similarities between these, and potentially more, domains in need of interactive visualization that are mostly treated as unique.

Needless to say, there is a lot of room for improvement. First and foremost, there is a potential for more use cases for traffic visualization. Examples include visualization of traffic of money, electricity, natural resources (e.g., oil), etc.

We define pipes as static data that connects sites. However, one can easily envision scenarios where allowing pipes to have dynamic properties is desirable, such as showing current availability, throughput, or other properties of a pipe. Having unidirectional pipes is also a possible future extension.

Another aspect that can be improved is additional GUI. We provide the default GUI supported by Cesium for spatio-temporal navigation and viewing properties. We also support a client-side API of the visualizer to attach any additional GUI the specific use case requires. Nevertheless, the burden of creating additional GUI can be lifted if one designs and implements a large enough set of universal GUI widgets that can be used with our system.

References

1. Alper, B., Sümengen, S., Balcisoy, S.: Dynamic visualization of geographic networks using surface deformations with constraints. In: Proceedings of the Computer Graphics International Conference (CGI). Computer Graphics Society, Petrópolis, Brazil (2007)
2. Andrienko, G., Andrienko, N., Chen, W., Maciejewski, R., Zhao, Y.: Visual analytics of mobility and transportation: state of the art and further research directions. IEEE Trans. Intell. Transp. Syst. **18**(8), 2232–2249 (2017)
3. Andrienko, N., Andrienko, G.: Visual analytics of movement: an overview of methods, tools and procedures. Inf. Vis. **12**(1), 3–24 (2013)
4. Bast, H., Brosi, P., Storandt, S.: Real-time movement visualization of public transit data. In: Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 331–340. ACM (2014)
5. Cozzi, P., Bagnell, D.: A WebGL globe rendering pipeline. GPU Pro 4: Advanced Rendering Techniques, vol. 4, pp. 39–48 (2013)
6. Debiasi, A., Simões, B., De Amicis, R.: GeoPeels: deformation-based technique for exploration of geo-referenced networks (2015)
7. Farkas, K., Nagy, A.Z., Tomás, T., Szabó, R.: Participatory sensing based real-time public transport information service. In: 2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS), pp. 141–144. IEEE (2014)
8. Ferreira, N., Poco, J., Vo, H.T., Freire, J., Silva, C.T.: Visual exploration of big spatio-temporal urban data: a study of new york city taxi trips. IEEE Trans. Vis. Comput. Graph. **19**(12), 2149–2158 (2013)
9. Gormley, C., Tong, Z.: Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine. O'Reilly Media Inc., Sebastopol (2015)
10. Guo, D., Zhu, X., Jin, H., Gao, P., Andris, C.: Discovering spatial patterns in origin-destination mobility data. Trans. GIS **16**(3), 411–429 (2012)
11. Hofstede, R., Fioreze, T.: SURFmap: a network monitoring tool based on the Google Maps API. In: 2009 IFIP/IEEE International Symposium on Integrated Network Management, pp. 676–690. IEEE (2009)
12. Hu, H., Zhang, H., Li, W.: Visualizing network communication in geographic environment. In: 2013 International Conference on Virtual Reality and Visualization, pp. 206–212. IEEE (2013)
13. Liu, S., Pu, J., Luo, Q., Qu, H., Ni, L.M., Krishnan, R.: VAIT: a visual analytics system for metropolitan transportation. IEEE Trans. Intell. Transp. Syst. **14**(4), 1586–1596 (2013)