Master 2 Research - Interaction Specialty

Interactive Illustrative Map Visualization

Author:
Seren Thompson

Supervisor:
Tobias Isenberg

Hosting lab/enterprise:
INRIA AVIZ

1 April, 2015 – 30 September, 2015

Secrétariat - tel: 01 69 15 66 36 Fax: 01 69 15 42 72
email: Murielle.Benard@u-psud.fr

Contents

Summary

Contemporary maps are designed for utility, but neglect the traditional artistic aspects of cartography. We create a web-based tool to allow users to play with map data in creative ways and explore the aesthetic aspects of abstracting and stylizing map presentations. We build on previous work in this area and integrate pan-and-zoom map navigation with localized and global data-abstraction techniques, including force-direction, Progressive Meshes, and modified Ramer-Douglas-Peucker line simplification. Using this application and these techniques, users can interact with map information and explore the artistic aspects of cartography, bringing creativity to an area which is normally focused on utility.

Keywords

Cartography, Maps, Web, Javascript, Abstraction

CHAPTER 1

INTRODUCTION

Maps are a common and powerful tool in modern society. With the advent of mobile devices, wireless networks, and accurate geo-location, the availability and utility of maps has never been greater. Because maps are representations of geography and not photographic reproductions, map information is neccessarily an abstracted model of reality. Cartographers often modify these representations for utilitarian reasons, however maps have many uses besides way-finding and navigation. In this thesis we discuss our project to enable users to customize map data for their own purposes. Our tool allows users to create and visualize embellished and abstracted representations of map information.

Map designers alter the represtation of map data for a number of reasons. Projecting a curved surface (e. g. a globe) onto a flat plane introduces distortion and map designers must choose which aspects to distort. Because of this, there exist dozens of map projections, each one with different trade-offs in surface continuity and distortion to area, bearing, and distance [26] (see examples in Figure 1.1).

Cartographers also commonly employ abstraction to increase a map's clarity and readability. Grouping, eliminating, and highlighting map elements are techniques that help users understand the dense information being presented. For example, a useful grouping abstraction is the practice of dividing map features into a limited set of visually-distinct feature classes such as roads (highway, state-road, unpaved, etc.) and land (building, park, forest, etc). Map features of the same class are then represented identically (except for shape) despite the real-life differences between class members. Another commonly used abstraction technique is eliminating small or trivial features to reduce clutter, while emphasizing or enlarging important small-scale

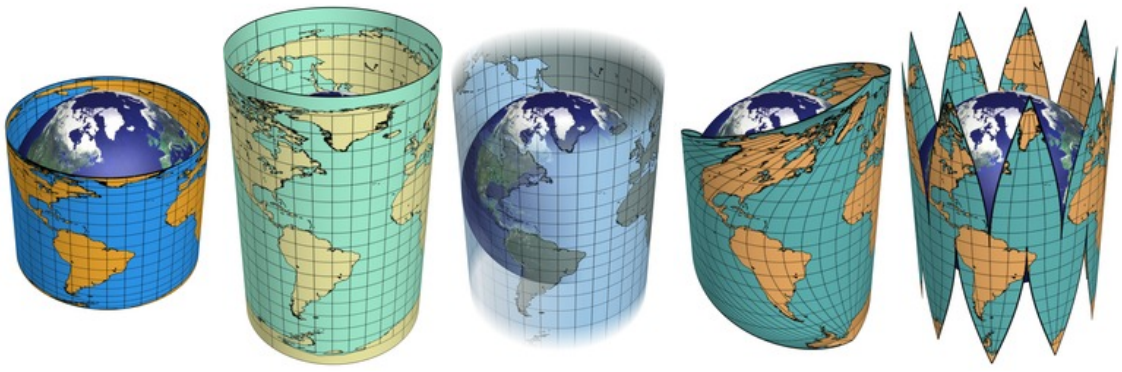

Figure 1.1: Projections from left to right: Lambert's cylindrical, Equidistant cylindrical, Mercator, ordinary sinusoidal, symmetrically interrupted sinusoidal. [9].

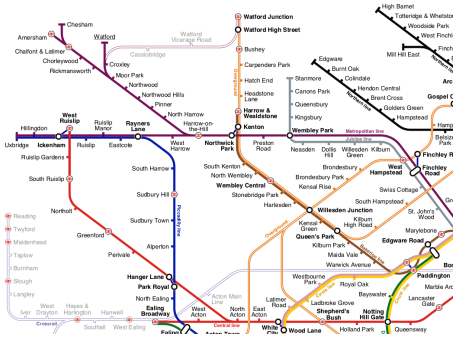

Figure 1.2: Map of London Underground system, showing how most line angles have been reduced to horizontal, vertical, and diagonal [24].

Figure 1.3: Celestial map showing attention to artistry and decorative qualities, "Celeste-Septentrional", Nicolas de Fer, 1705.

features (e. g. bathrooms, hiking-paths, and points-of-interest). Even core elements such as roads are typically abstracted since, for large scale maps, they would be too thin to see without enhancing their displayed width.

Adherence to geographical truth can obscure the information that a map needs to convey and creative representations can help make the important aspects more salient. In the 1930s, the London Underground switched over to a simplified schematic map of stations and connecting lines (see Figure 1.2), conceived by Harry Beck [10]. The design was initially rejected as being too radical a departure from the traditional representation, but was adopted after a trial run of the map was wildly popular [23].

Cartography has evolved into a science in which accuracy, utility, and information content are normally the focus. Even after the Middle Ages, map-makers straddled the boundary between faithful representation and creative expression. In France, during the Age of Enlightenment, cartographer Nicolas de Fer produced hundreds of well-received works which focused more on artistic effect than accuracy [18] (see the example in Figure 1.3).

The style and presentation of map data strongly influences user experience, both in terms of usability and pleasure [17]. Map appearance emerges from a combination of the aesthetics of the era, and the technology available. The most modern electronic maps have a flat appearance with small set of discrete colors. By contrast, early western maps from the 1600s have much more detail, colors (as permitted by the printing technology), and embellishment. The embellishments are particularly interesting as they add artistic and cultural content that lends flavor and context to the geographic information. Examples include Henricus Hondius's double-hemisphere map of the world which includes depictions of the four elements

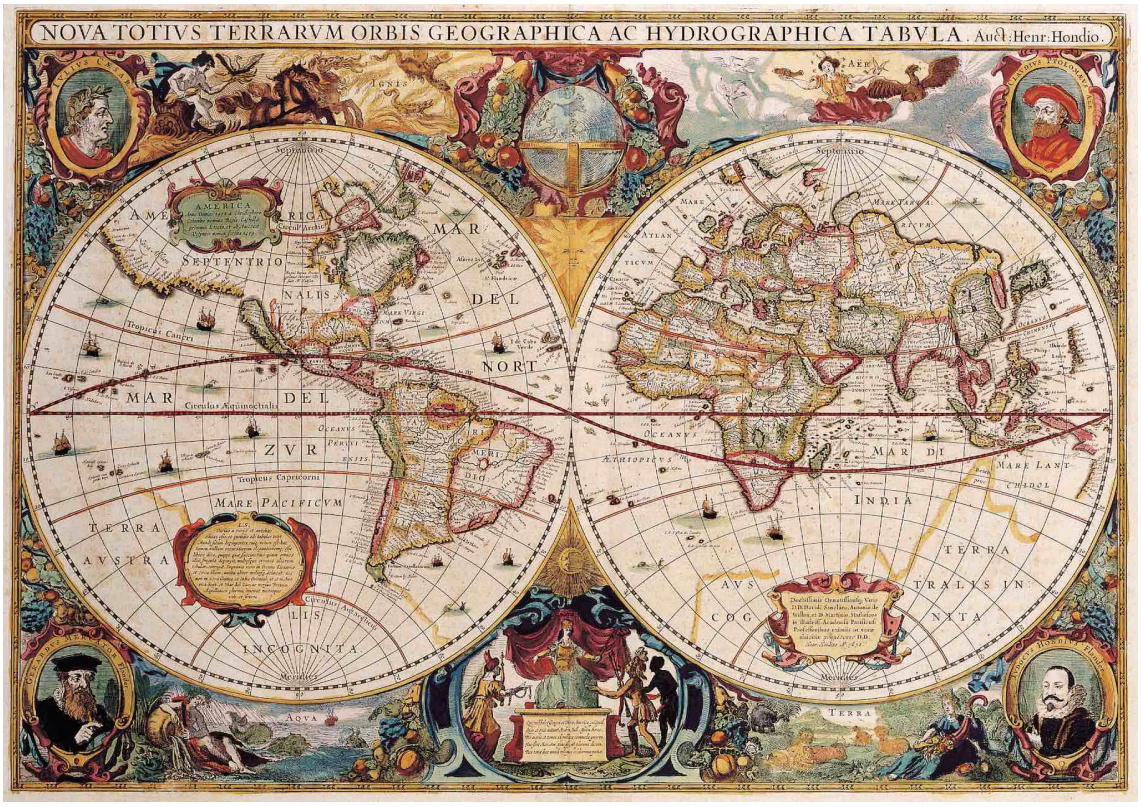

Figure 1.4: Double Hemisphere Map, Henricus Hondius, c 1630. Public domain.

as well as portraits of notable figures Figure 1.4, and the engraving of Leo Belgicus by Jodocus Hondius Figure 1.5 which merges a map of the Netherlands with the heraldic figure of the lion, a cultural reference to the on-going war for independence. These maps are artistic works as well as cartographic tools, and as such have styles representative of their time periods and pleasing to their respective audiences. These works inspire our efforts to create modern maps where the style and embellishments are derived from actual data, as befitting of our modern data-centric culture.

Our motivation for creating our map abstraction tool stems from our desire to give users a way to play with the representation of map data. Users of digital maps already customize them to a degree, such as by choosing layers to display or by incorporating personalized data, however very few of these features incorporate aesthetic or creative aspects beyond those which further the primary practical goals. Additionally, people utilize familiar cultural artifacts for creative or artistic purposes, especially if those artifacts are imbued with cultural or emotional meaning. Maps are firmly in this camp: a country, state, or island's outline is often an instantly recognizable to its residents and can be as much of a rallying symbol as a flag or coat of arms. Therefore we aim to for our project to be used as a tool to create aesthetic value from map data that is personally meaningful.

Executive summary

For this map abstraction project, we use OpenStreetMap cartographic data. Once the user selects a location, we take the data and load it into an undirected graph structure from which the map is displayed. Users can apply a number of different algorithms individually or in combination to the graph data in order to alter the map's structure and make it more abstract. Additionally, we automatically decorate the map's line segments with colored brush strokes. This stylization can be modified algorithmically by the code, or manually by the user.

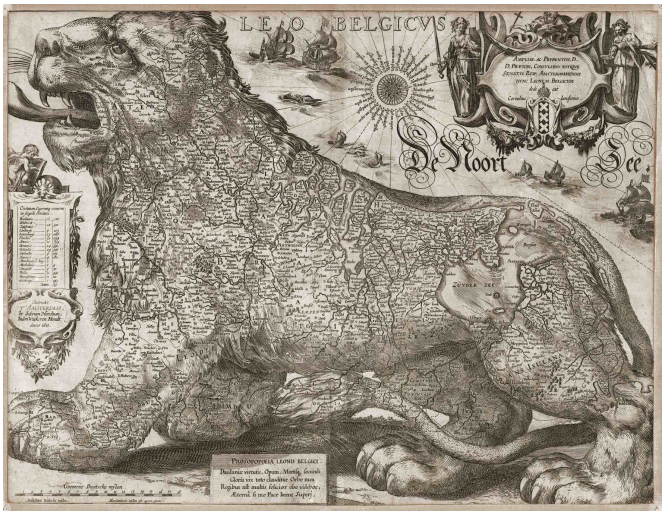

Figure 1.5: Jodocus Hondius, "Leo Belgicus – map of the Low Countries". Public domain.

Overview

This thesis will present six major sections including this introduction. In the Related Work section, we present relevant work in the fields of cartography, abstraction, and map interaction. In the following Concept section we detail the ideas and motivation behind our map-abstraction concept. We also cover our technical contributions and how our work extends the ideas implemented in the Illumap project. In the Realization section, we cover the details of our implementation, the design decisions we made, and the challenges we encountered. In the Evaluation and Results section, we then cover our final implementation and evaluate it in light of our motivations. Finally, we present our conclusions and perspectives on the project.

CHAPTER 2

Related Work

Our project touches upon traditional and modern art, cartography, non-photorealistic computer rendering (NPR), algorithmic three-dimensional shape simplification, and interactive web-based applications. We cover these areas and related work in the rest of this chapter.

Map making traditionally has been a type of artistry and many map makers were also painters, engravers, woodcutters, and printers. For inspiration from the late Middle Ages, we can look to the work of Sebastian Münster Figure 2.1 whose simplified depiction of Italy contains elements, almost symbols, of landscape features rather than actual landforms. We also see the common stylization of borders from that time period, in which the hard black border line contains a colored halo which aids in easily identifying political areas.

Nicolas de Fer's artistically embellished works are informative as to what departing from strict geographic representation can offer, while more modern pieces such as Ed Fairburn's "Croydon" (see Figure 2.3) show that aesthetic objectives can be achieved in an additive way, by modifying the spatial aspect.

Figure 2.1: Carta d'Italia di Sebastian Münster, 1550 [20].

Finally, modern abstractionist art informed our ideas for abstraction algorithms. Piet Mondrian's later neoplastic works such as "Composition with Red, Yellow, and Blue" (Figure 2.2) were the inspiration for our orthogonalization.

In the domain of NPR, we follow research for abstracting images and shapes by replacing low-level detail with coarser and more uniform objects that suggest the original detail. Examples of this form of abstraction include watercolor rendering (e. g. [5]), image-mosaic construction (e. g. [13]), and stroke-based rendering (e. g. [14]).

We use techniques from a number of simplification methods, including progressive meshes [15], quadratic-error surface simplification [11], and line generalization [22, 7, 28, 29]. These algorithms are already used in GIS and map applications such as

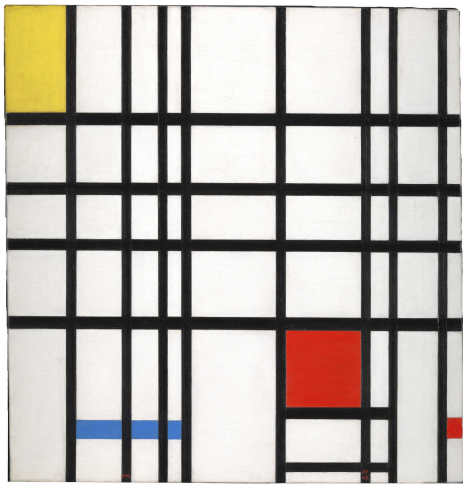

Figure 2.2: "Composition with Red, Yellow and Blue", Piet Mondrian, 1942. Public domain (EU).

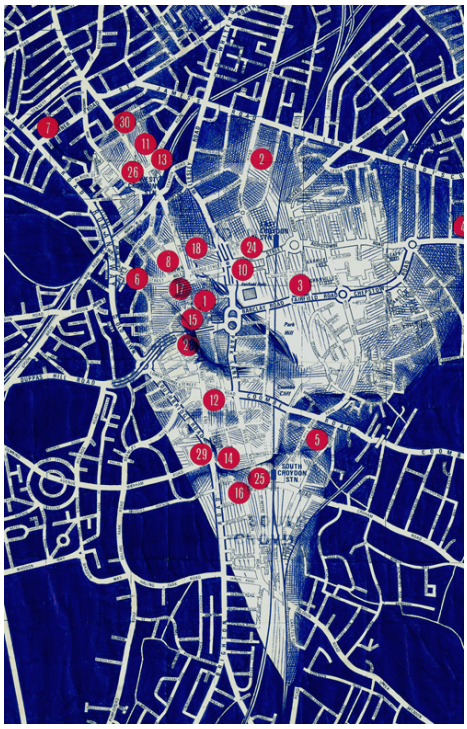

Figure 2.3: Map blended with facial features. "Croydon", Edward Fairburn, 2014 [8].

Mapshaper [1].

The work of Mi et al. [19] is also instructive with their differentiation of shape simplification vs. shape abstraction. Shape simplification removes small-scale detail, while their abstraction technique deconstructs shapes into component parts and removes parts. Their results are impressive, but unfortunately are inapplicable for the type of map data we use. Their technique requires cohesive shapes with associated insides and outsides which our map data, a collection of line segments, does not have. Because of this, we implemented line simplification techniques instead.

Online maps are common these days, but most use tiles of pre-rendered raster images rather than raw vector data. Projects like Polymaps[1] and its successor, Leaflet[2], allow vector overlays of standard raster tiles, while the Kartograph framework[3] allows developers to create interactive SVG maps. While inspirational, ultimately these projects were not appropriate for our use since either their vector support is for only overlays, or they do not support processing of the vector data.

Our line segment decoration, which gives the map its color, is primarily inspired

[1] http://polymaps.org/
[2] http://leafletjs.com/
[3] http://kartograph.org/

by Tarbell's Substrate [27] simulation[4] (see Figure 2.4). This simulation generates random paths which terminate upon intersecting each other, producing images that bear a striking resemblance to cities surrounded by countryside. Our implementation builds on this idea by replacing random paths with actual geographic data, and enabling interactive modification of the map structure.

The design of our code and application architecture comes from a number of different areas, but is heavily influenced by the D3 library by Mike Bostock [2], Addy Osmani's Javascript design patterns [21], and the advice from Douglas Crockford's "Javascript: The Good Parts" [4].

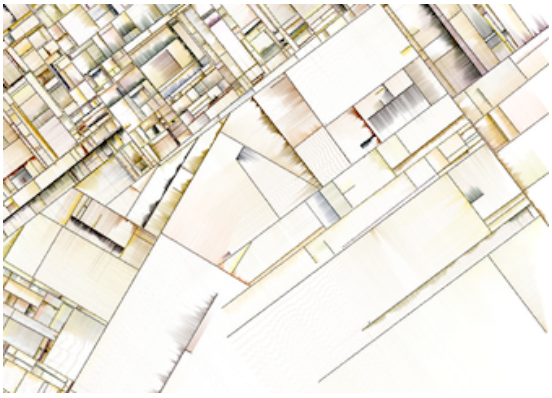

Figure 2.4: Screen capture of a Substrate visualization [27].

We also studied the visually unique styles of different map services. Some of the differences in appearance (as can be seen in the Map Compare[5] screenshot in Figure 2.5) stem from their utilitarian design goals, but they are also influenced by the aesthetics of the map designers. Color pallet, line weight, and fill patterns all contribute to the style of a map and contribute to its visual distinctiveness, independent of the underlying cartographic data.

[4] http://www.complexification.net/gallery/machines/substrate/
[5] http://mc.bbbike.org/mc/

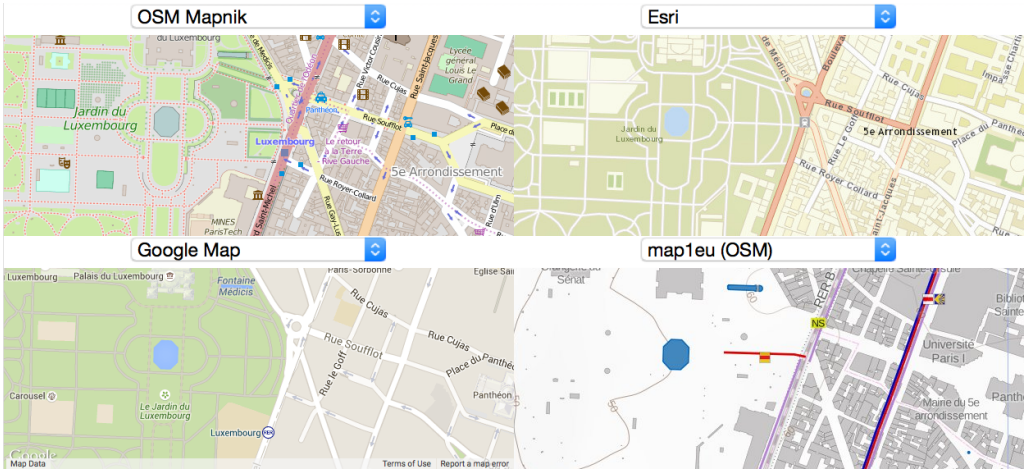

Figure 2.5: Screen capture of a Map Compare comparison [12].

CHAPTER 3

CONCEPT

3.1 Overview

We believe there are a number of potential uses for aesthetically-pleasing abstracted map data. To gauge interest in this type of work, we can look to the acceptance and use of other abstractions as well as data-driven art. The explosive popularity of fractal renderings in the 1980's, particularly the Mandelbrot set [6], demonstrated the public's interest in data-driven art. Abstraction of real-world data is also popular, for example, on mobile devices where users have many options for applications that can process photos to make them appear to be analog works of art such as oil paintings, pencil sketches, or charcoal drawings. Abstracted and stylized map data could be used in static form for posters or desktop backgrounds, while animated abstraction sequences could be used as screensavers or visual-entertainment pieces.

We draw much inspiration from the Illumap [16] project. In particular, we extend Isenberg's approach by: *a*) providing interactive stylization options; *b*) allowing users to select locations via 3rd party tools; *c*) allowing users to change the map data during the abstraction process by panning and zooming; *d*) providing undo functionality; *e*) making the process available via the web; *f*) enabling touch input; *g*) allowing users to save and restore locations and abstractions; and *h*) adding interactive localized abstraction techniques. We elaborate on these extensions and other concepts below.

Because stylization of map data strongly influences user experience [17], we want to give users the opportunity to explore and play with different styles. We take inspiration from Tarbell's Substrate [27], whose color pallet was inspired by Jackson Pollock's work.[6] We include the coloration features found in Isenberg's [16] project and extend them to allow animation of the color pallet and a more natural selection process through direct manipulation.

We also provide abstraction of map features as a major map interaction for the user. Though most maps employ abstraction for utilitarian purposes, we are interested in the aesthetics possibilities that abstraction brings. For instance, users may be able to abstract and deform map geometry to produce patterns that are visually pleasing while still conveying a sense of the original location. Using particular combinations of algorithms may produce patterns that resemble other objects (similar to cloud watching) as in Figure 3.1. Our choice of abstraction algorithms is inspired by the Illumap implementation, as well as online projects such as the interactive particle demonstration of glfx.js.[7] We provide abstraction techniques that users

[6] "1000 color palette stolen from Jackson Pollock" —Jared Tarbell, 2003

[7] http://evanw.github.io/glfx.js/

can run non-interactively (the Ramer-Douglas-Peucker technique [22, 7], progressive meshes [15], and orthogonalization), as well as modified interactive force-direction techniques that allow more of a playful exploration of data abstraction.

In order for the map data to be interesting and personally meaningful to users, we provide them the ability to choose the map starting location. We want users to be able to play with data from locations with which they may be familiar, or locations which are iconic since allowing users to choose their location should give them increased investment in the exploration, and hopefully additional enjoyment and inspiration. To select this initial location, we provide a tie-in with a standard mapping service which lets users search for and pinpoint a location before switching over to our abstraction tool.

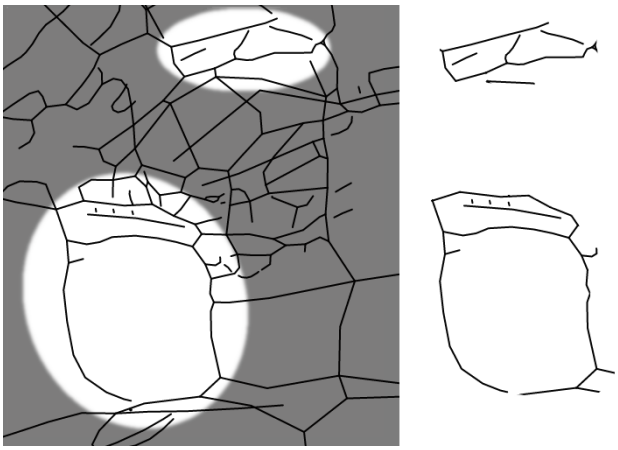

Figure 3.1: Example of potential apophenia or "cloud-watching" with abstracted map data: Face with eyes closed (upper-right) and three-eyed creature (lower-left).

To facilitate exploration and selection of map data, we also provide a "slippy-map" interface within our tool. This allows users to drag the map in order to change location and load previously unavailable data. This interface provides zooming features as well, allowing users to change the detail level interactively. This enhances location discovery by enabling users to easily switch between high-level overviews of areas and low-level detailed inspection of specific spots. It also facilitates switching between known locations since users can smoothly combine high and low-detail panning.

Our goal is to make this project accessible to a wide audience, and easy to use. For this reason, we implement our abstraction tool as a web-application. In comparison, the Illumap project is a stand-alone Java program which must be downloaded and run. Our web-based implementation is highly accessible, since users do not need to configure or install anything beyond a standard web-browser. As a web-app, it also runs on mobile devices via users' browser, giving us access to alternate modalities (e. g. touch input, accelerometer readings, microphones), which we can use to augment the user interface. It also benefits us as developers since we can take advantage of a wide variety of web-based resources such as javascript libraries and map-data servers.

We believe that the previously-stated goals will allow users to focus on the artistic

and playful aspects of the abstraction process. There are some potential implications of our design decisions, however. Most maps color schemes are carefully designed with clarity and accessibility in mind. Because we give users many options for map coloration and appearance, the user runs the risk of making poor display choices. The same is true of the abstraction process: the complete freedom to apply different abstraction techniques in any order means that users may choose aesthetically poor combinations. These potential drawbacks are mitigated by the ease of rechoosing display options, undoing actions, and restarting the abstraction process.

Because our project uses live map data, it requires a working internet connection when first starting or loading new data (triggered when moving or zooming). We implemented and tested offline capability[8], but decided not to expose this functionality to the user since we do not anticipate it benefitting our users. We anticipate many users will be first-time visitors, who require internet access to merely load the application. We also predict that most users will scroll and scale the map which requires an internet connection to load and display new map data.

The abstraction algorithms and map stylization can be processor intensive. Users of slower computers, especially mobile devices, may notice variation in abstraction performance depending on the algorithms selected. Improving performance is definitely possible but not something we focused on for this iteration of the project.

3.2 Techniques

The simplification and abstraction techniques we employ can be applied by the user jointly or independently. The abstraction techniques process map information at the level of simple, connected, undirected polylines stored in a graph structure (rather than polygons or other higher-level structures). We calculate the user's viewing area and detail level, and download the appropriate map data in vector-tile form[9] from the OpenStreetMap service before transforming it into a format that our abstraction algorithms can process. Because we are not concerned with maintaining the original map data, some of our techniques remove or relocate map lines. Below we detail the abstraction techniques we have selected as well as the decoration technique. The example images are modified versions of the raw data shown in section "A" of Figure 3.2, a street map of Schiermonnikoog island in the Netherlands.

[8] Caching for offline use was useful when the OSM vector tiles servers went offline during development.

[9] Description of available vector formats: http://openstreetmap.us/~migurski/vector-datasource/

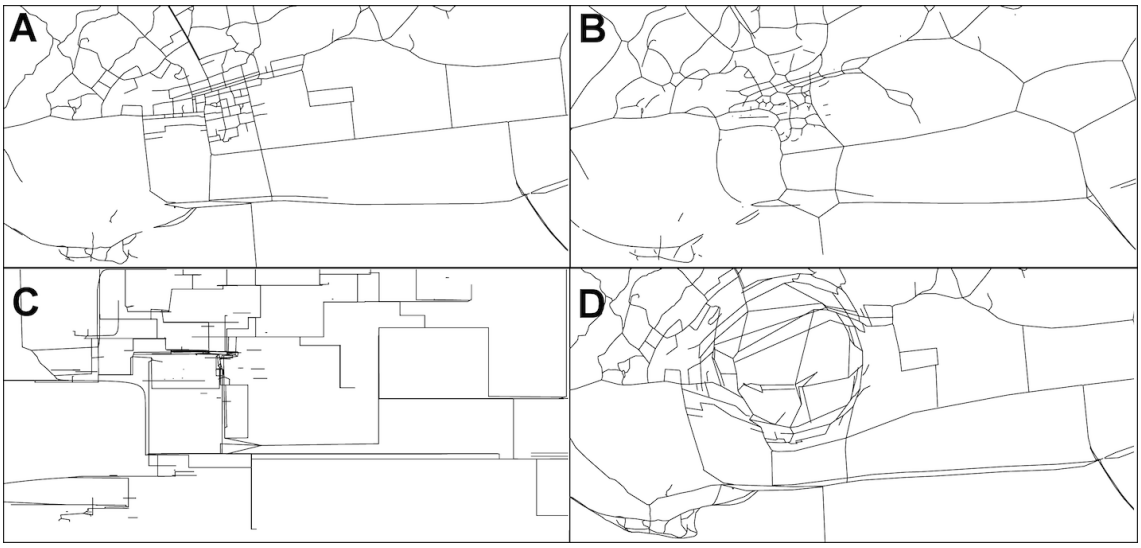

Figure 3.2: Map from OpenStreetMap of Schiermonnikoog island in the Netherlands. This view is prior to any abstraction.

3.2.1 Force-Directed Relaxation

The force-direction algorithm smooths the appearance of the map, by moving each moveable point towards the group-center of its neighbors. It operates on all vertices in the graph during each iteration, but no edges are removed. The algorithm is fairly simple: for each point, average the vectors from the point to each of its neighbors, scale the resulting vector by a constant "spring force" to generate an offset for the point, and, once all of the points' offsets have been calculated, translate each point by its offset. This has the effect of causing all points to move toward the center, and straight ways (lines made up of multiple points) to become curved. This technique also has the effect of highlighting map segments that are disconnected from nearby segments, since the relaxation tends to contract the line segments moving their endpoints away from each other (see the lower right edge of section "B" in Figure 3.2 for an example of a disconnected line segment).

3.2.2 Orthogonalization

Our Orthogonalization algorithm produces map segment layouts reminiscent of Piet Mondrian's work (see section "C" in Figure 3.2). It modifies lines segments' angles, pushing them toward horizontal or vertical alignments based on which orientation they are currently closest to. Like the force-direction technique, orthogonalization operates on all points per iteration, but does not remove any edges.

The algorithm adds offsets to the vertices of each edge, causing the angle of the edge to change. For unconnected angles this is a simple process. Most of the map

edges are connected to at least two neighbors however, so we use an iterative process similar to the force-direction technique, where a vertex's final offset is the average of all the offsets generated when processing each edge of which it is a member. Since this process involves much interaction between edges, we simplify the calculations by using one half of the smaller of the vertical or horizontal component of the vector, and applying it as an offset such that it will rotate the edge toward alignment with the closer coordinate axis. This is much faster, but can introduce oscillation due to combined offsets pushing a vertex beyond the point our algorithm would settle on. To prevent this, we use a dampening factor of 0.35 which slows convergence but also reduces oscillation.

3.2.3 Progressive Meshes

Progressive meshes [15] are normally used in 3D surface mesh simplification. It is one of the techniques we use which eliminates line segments in addition to moving them. It works by collapsing edges, which is essentially merging an edge's vertices into a new single vertex and updating the adjacent edges to connect to the new vertex, as in Figure 3.3.

The collapsed edge is chosen by generating an error metric for each edge in the map graph. The error metric measures how much error would be introduced by collapsing the edge, and the edge with the lowest error value is for collapsing. Our version of progressive mesh simplification differs from the original technique [15, 11] by taking into account that vertices may have a degree <3, something which never occurs in 3D surface meshes. It also uses edge length as the error metric, which is simple and quick to calculate. Since each operation must calculate and sort the lengths of potentially several thousand edges, a more complex error metric would be prohibitively slow to use barring the implementation of a caching mechanism.

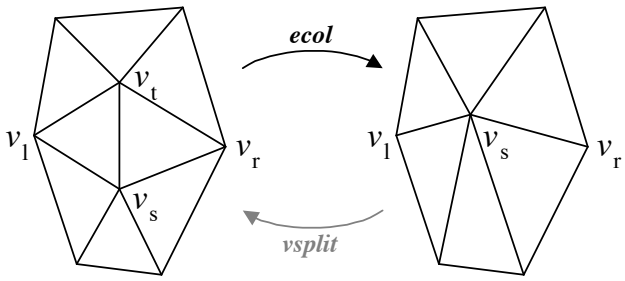

Figure 3.3: Illustration of the edge collapse transformation [15].

3.2.4 Ramer-Douglas-Peucker

The Ramer-Douglas-Peucker technique can be used to simplify map data at the "way" level. Ways are lines made of of a series of connected edges, with each edge connecting

to at most two other edges. The technique works backwards (for our purposes), by starting from the most simplified version of a way (i. e. a single line segment from the first vertex to the last) and generating a sequence of vertex additions that eventually result in the original polyline. At each step, the algorithm selects the vertex that is furthest from the connecting edge. It then adds the vertex, resulting in two edges, and calls itself recursively on each of the new edges.

The algorithm in its original form has several limitations for our use case. First, it produces a hierarchy of operations instead of a linear sequence. It also operates on single polylines rather than collections such as contained in map. Finally, the algorithm works backwards for our purposes, from most simplified (a single edge) to least (the original polyline). Due to these limitations, we used a customized extension of the technique from [16]. We run the algorithm normally, but build a linear sequence of simplification steps by merging the results from each recursive split. The merge is performed such we that vertex collapses with lower error (i. e. vertex distance from simplified edge) are chosen before collapses with higher error. We maintain the recursion results order (i. e. we merge but do not sort recursion results) as we merge so that for any series of abstraction steps the algorithm chooses one with the least error. We finish each merge with the vertex collapse operation between the two segments which gives as a reversed order of operation, from detailed to simplified. Finally, using the same algorithm, we merge the results for each way. This gives us a simplification sequence for the entire map, in the correct order.

3.2.5 Interactive Localized Repulsion

This is an interactive abstraction technique that allows users to apply a repulsive force to an area on the map. When the user drags a mouse or finger across the map, we repeatedly calculate and apply a force on all map nodes away from the pointer. The force is a fixed value which we then attenuate using the inverse-square of the distance between the pointer and the node. The attenuation localizes the most of the effect, giving the users control over which map data is affected by the technique. The result of moving the point in the densest area of the map is illustrated in section "D" of Figure 3.2. Using this technique has the effect of expanding an area, similar to lens magnification techniques (as described in [3]). Because this technique uses dragging, which we also use for panning the map, we allow the user to select which mode (panning or repulsion) they desire via a selector in the user-interface.

3.2.6 Stylization

Along the feature lines of our map, we add decoration in the form of perpendicular colored lines that blend together to simulate a water-color brush-stroke away from the feature line. We place the decoration lines along a single side of the feature line in

Figure 3.4: Example of stylization via line segment decoration.

order to highlight the line as well as avoid overpainting dense areas (see Figure 3.4). The decoration lines extent is proportional to the length of the feature line they come from, which also helps curtail crowding in areas of short dense features. As part of our design to allow aesthetic exploration and experimentation, we allow the user to influence the color-pallet, weight, and orientation of the strokes.

CHAPTER 4

Realization

Our implementation is built upon several design goals. We want the interface to be intuitive and easy to explore, enabling first-time users to discover the effects of our abstraction techniques through playful interaction rather than instruction. To this end, we want effects of user actions to be immediately apparent and reversible. We also want users to be able to personalize their experience through their choice of map location and scale, and through their abstraction and stylization choices. Finally, we want to provide an experience rich and varied enough to capture users' interest beyond the initial discovery phase. On the technical side, we want the application to be easily accessible, run on all major operating systems, and require no prior system configuration. We also want the application to be responsive even on slower systems.

These goals led us to a series of design decisions around the interface, features, and a data processing. In our user interface, we employ standard graphical interface conventions such as buttons and dragging behavior. We also avoid interaction techniques that only work on one type of device. For example, we avoid pointer-hovering or right-clicking for interaction since mobile devices tend not to have them available. We also adjust our interface in consideration of Hick's Law [25], exposing the fewest number of controls possible and moving infrequently used controls, such as preferences, into panels that are normally hidden.

To support our goals of the application running cross-platform and with no setup, we decided to implement a web-based application in Javascript. Almost every internet-connected consumer computing device has web-browsing capability, so we would easily be able to reach a wide audience. Additionally, this would allow us to leverage existing Javascript libraries for building custom user-interfaces and interacting with map data. The libraries available and also features we wanted to implement informed our data processing decisions. We wanted be able to pan and zoom our map interface, which necessitated keeping a history of abstraction steps so that they could be applied to new map data as it came into view. This history could also be used to support a rudimentary undo, by reapplying all but the last abstraction step to the original data. To support panning and zooming, we opted to use GeoJSON vector-tiles[10] which are easy to consume via Javascript and can be downloaded in parallel. This is in contrast to the Illumap [16] project which used OSM XML (OpenStreetMap's native XML format) and downloaded the entire map in one operation. For rendering, we chose to use SVG (Scalable Vector Graphics) rather than Canvas, despite Canvas being more performant due to SVGs being simpler to generate and manipulate.

[10] OpenStreetMap vector-tiles are still an experimental service at the time of writing.

Our design and approach are inspired by that of Illumap [16], but contain significant differences. In addition to using arrays of GeoJSON vector-tiles instead of dynamically-generated OSM XML, we chose to simplify the interface by reducing the number of interface elements available at any given time. We accomplish this by merging some interface elements, moving others into popup dialogs, and eliminating application features which are unnecessary or provided by the browser. For instance, the Illumap application provided two buttons for each type of abstraction, one which ran the abstraction technique once and the other which ran it continuously. We merge these two buttons into one which runs the abstraction continuously while the button is being pressed. We also were able to remove the file management buttons for loading data since our data is now loaded automatically based on map location. By making these changes we were able to reduce the user interface elements on the main page from 35[11] to 10 [12] (see comparison in Figure 4.1).

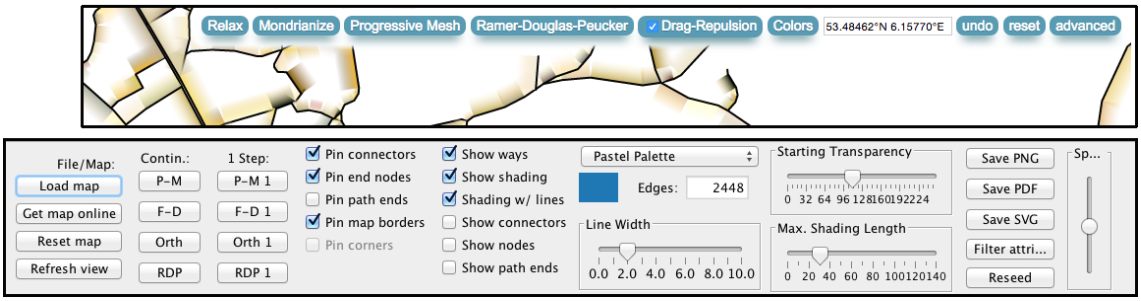

Figure 4.1: Our application controls (top) and Illumap applications controls (bottom).

We reimplemented the existing Illumap simplification techniques however the underlying code is quite different due to the differences between Java, which is strongly-type and object-oriented, and Javascript, which is functional and weakly-typed.

Javascript is a popular language with an active development community. Because of this we had a number of tools available which dealt with graphics and specifically map rendering. Some of the tools we considered were ....

Because few of the current tile-based map rendering engines were able to use map vector data[13] and because none of them were designed to store and re-render modified vectors, we wrote much of the application from scratch. The two libraries we did integrate were graphlib[16], a directed multi-graph library we used store and manipulate map coordinate relations, and D3js[17], an extremely powerful and flexible library that

[11] 17 buttons, 11 checkboxes, 4 sliders, 1 drop-down menu, 1 color indicator, 1 text field.

[12] 9 buttons, 1 text field.

[13] Leaflet[14] and Polymaps[15] are able to render vectors for map overlays, but still use raster tiles for the underlying map image

[16] https://github.com/cpettitt/graphlib/

[17] http://d3js.org/

allows data-driven transformations of documents. D3js also provides asynchronous fetching of server tiles, transformations between map and screen coordinate systems, mouse and touch input detection, and visual data rendering.

There were a number of challenges to realizing our application. The Javascript language has a number of design limitations (as enumerated in [4]), which make scoping tricky and passing context between functions tricky or impossible. The normal workaround is to reduce depth of compartmentalization to make components and state available, or to follow design and architectural patterns that reduce the need for components to address each other. We used a combination of these techniques to overcome these challenges, which also helped us integrate our third-party libraries.

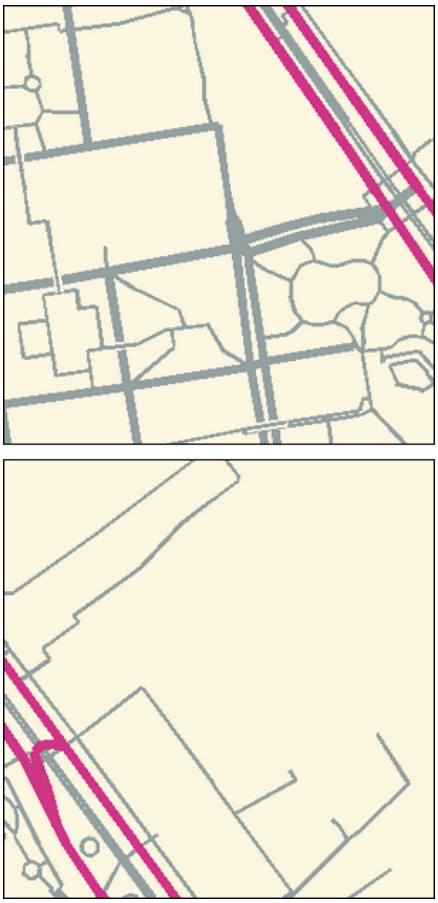

Figure 4.2: In these two tiles, the highlighted path segments share a unique ID and will be merged by our application before abstraction and rendering.

Most slippy-maps download and directly display tiles of data, however since we are storing and displaying modified data, we had to write significant code to tie map movement to downloading, modifying, and rendering. This code also has to track the abstraction steps already taken by a user, and reapply them when fresh data is loaded. Because vector tiles are downloaded asynchronously, coordinating multiple download threads proved challenging since they needed to integrate tile data into a shared data store and trigger the re-abstraction process before refreshing the display. A number of third-party libraries provide solutions (Q[18], Promisejs[19], and Async[20] to name a few), and the latest version of the Javascript standard, ECMAScript 2015 (ES6), includes Promises which provide native support for handling deferred and asynchronous computations. Browser support for promises is incomplete however, and integrating third-party libraries requires learning and introduces complexity. Therefore we wrote our own handler that tracks outstanding tile requests and triggers re-abstraction and rendering at regular intervals or when all requests

[18] http://documentup.com/kriskowal/q
[19] https://www.promisejs.org/
[20] https://github.com/caolan/async

have completed. The tiles are merged into a single data structure before abstraction or rendering, with paths that are split across tiles being rejoined based on the unique feature ids (see example Figure 4.2).

Keeping abstracted maps from changing appearance when panning is a challenge. Since most abstraction methods (progressive meshes being the exception) affect map points differently based on the other map points in view, abstraction outcomes will be different when users pan and bring new data into view. We currently reapply the map abstraction steps to the entire map, but this causes existing map data to change appearance when panning. A better solution might be to bring fresh data into view without reapplying the abstraction steps, however this would break our undo feature since it currently reapplies the abstraction steps to all visible map data.

The final challenge involved the coordinate reference system employed. Map points arrive from OpenStreetMap defined by latitude and longitude coordinates. We store them and perform most abstraction modifications on them in this native format, converting them to x and y screen coordinates (using D3js's Mecator projection transformation) at the rendering stage. Storing and operating on geographic coordinates works well for integrating other geographic data with the abstracted map and requires one less caching layer[21] to maintain, but introduces some complexity and computational overhead since we have do the screen-projection transformation (using D3js) for each display refresh. For interactive abstraction techniques, we also also have to transform mouse coordinates to geographic coordinates before we can calculate any interaction with map points.

We continue to improve and add features to our map abstraction application. Currently it runs on personal computers and mobile touch-based devices. Its capabilities include map abstraction using forced-directed relaxation, orthogonalization, progressive meshes, user-directed repulsion, and a modified Ramer-Douglas-Peucker technique. Users are able to pan and zoom to dynamically load new map data. We also have searching (via 3rd party) and undo features. The application can work offline after being loaded for the first time, though panning and zooming will be limited to map areas that have already been visited and cached.

The current version of our application does have certain limitations. Though offline mode is available, it is not robust and will fail in some cases such as when bad data has been cached. We have focused most of our effort on the network-enabled capabilities of our application, so it would be very difficult to distribute it and map data to an offline user. Performance is also not at a satisfactory level for maps with more than a few hundred points, especially on mobile devices. Even when the underlying algorithms are performant, we do not always refresh the display at regular intervals, leading to the appearance of freezing. Finally, the Javascript code is not as

[21] We currently cache raw tiles and abstracted geographic data, but not data in screen-projection coordinates.

modular as we would like and does not lend itself to being extended or debugged in its current form.

For the next version of the application, we intend to address the previous limitations and add additional features. Offline capabilities will require new caching mechanisms which should in turn enable us to operate internally on screen coordinates rather than geographic coordinates. This should in turn improve our performance and reduce our code complexity. We intend to revisit rendering via Canvas instead of SVG and test whether any potential speed increase is worth the added complexity. We would also like improve the screen updating frequency to give more consistent user feedback. To improve performance, we can work on several areas including leveraging D3js quadtrees to speed up interactive abstractions, precomputing edge tangents for stylization rendering, employing memoization where beneficial, using SVG transformations to move points rather than redrawing them, and tuning areas of code that are called the most frequently. We can dramatically improve undo performance by caching previous abstraction results rather than recomputing them to arrive at a previous state. We would also like to add OSM XML support so that we can draw data directly from the OpenStreetMap servers and not rely on experimental vector tile services. Finally, adding the ability for users to save their abstraction to PNG, SVG, or browser cookie could improve users' experience since they would have something to keep at the end of a session.

CHAPTER 5

Results and Evaluation

At the time of writing, our application (shown in Figure 5.1) was not in a finalized state for user testing, and the vector tile service was unavailable which blocked panning and zooming. Therefore we did not perform user testing and instead gathered informal feedback. The feedback indicated that the incomplete features needed to be finished and improved before the tool could be used as intended. These features included the stylization selection tools, pan-and-zoom, and general performance.

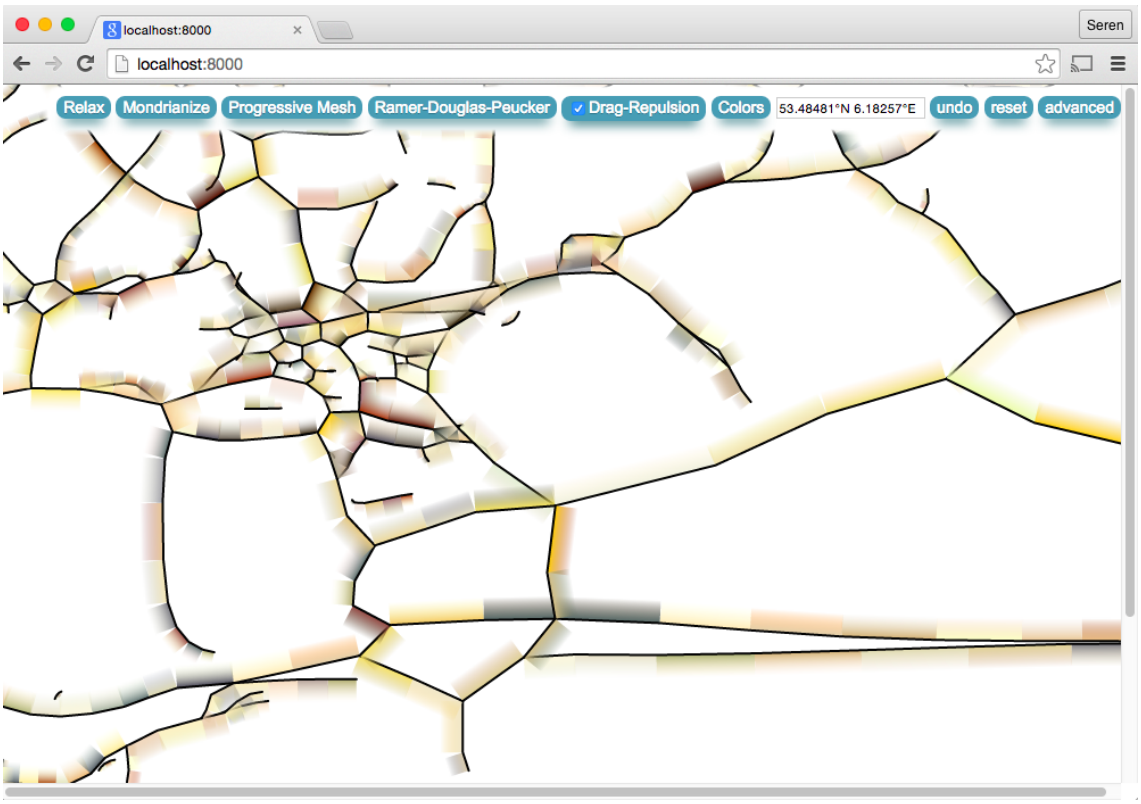

Figure 5.1: Screen capture of web application.

When the project is complete and a tile service available, more formal user and performance testing is in order to evaluate if the application achieves the previously stated project goals, how discoverable the different features were, and how each one performed.

CHAPTER 6

Conclusion and perspectives

In this thesis, we presented an application for abstracting, stylizing, and playing with map data. It is motivated by the Illumap [16] project and the Substrate [27] project which show the aesthetic possibilities of abstracting real or simulated map information. The goal of this work is to provide users with an intuitive and enjoyable means of interacting with and modifying map data, while requiring no training or system preparation. Our implementation improves upon the previous work in Illumap by making it accessible via the web, simplifying the interface, adding interactive abstraction techniques, and implementing a slippy-map interface. We expect this project to be useful for visualizing the effects of different abstraction techniques, for generating stylized depictions of familiar landscapes, and as a leisure time activity for assisted creativity.

We implemented our application using HTML, CSS, and Javascript, and leveraged the D3js and graphlib libraries. Our data is downloaded dynamically from OpenStreetMap's experimental vector tile server. Our feature-set includes five different abstraction techniques, map style customization, jumping to or searching for a location, loading and saving the abstraction state, and the ability to undo abstraction steps. During development we encounter challenges included performance issues, handling asynchronous network operations, preventing abstraction results from changing when importing new data, and handling transformations between geographic coordinates and screen coordinates. We overcame issues around asynchronous operations and coordinate transformations, but leave performance improvements and preventing abstraction shifts to future work.

To properly evaluate the performance of this work, qualitative and quantitative user testing would be useful. Due to time constraints, we only performed informal user surveys and no technical performance evaluations. Additionally, because the project is not yet finished, the feedback is incomplete due to certain features (e. g. undo and mobile use) being prohibitively slow. By the end of the internship, we expect the program to have evolved in both performance and existing features (e. g. line decoration) and new (e. g. abstraction saving and loading), which would address user feedback concerning usability and enjoyment.

There were additional features that would have been desirable, had more time been available. For example, additional interactive abstraction techniques using multitouch could be implemented. Also, using color values from an image as a force-guide could produce interesting abstractions that resembled the guide image.

Acknowledgments

I would like to thank my wife Angela for her infinite patience and academic advice, my infant son Julien who graciously started sleeping through the night earlier than expected, my advisor Tobias Isenberg whose advice, availability, and attention to detail were invaluable and made this thesis possible, and Mike Bostok, the author of D3js, whose many tutorials and extensive help online showed me what is possible with Javascript.

Bibliography

[1] Matthew Bloch and Mark Harrower. MapShaper.org: A Map Generalization Web Service. In *Proc. AutoCarto*. Cartographic and Geographic Information Society, 2006.

[2] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, December 2011.

[3] M. S. T. Carpendale and Catherine Montagnese. A framework for unifying presentation space. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, UIST '01, pages 61–70, New York, NY, USA, 2001. ACM.

[4] Douglas Crockford. *JavaScript : the good parts*. O'Reilly, Beijing Cambridge, 2008.

[5] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-Generated Watercolor. In *Proc. SIGGRAPH*, pages 421–430, New York, 1997. ACM.

[6] A. K. Dewdney. Computer recreations. *Sci Am*, 253(2):16–24, aug 1985.

[7] David H. Douglas and Thomas K. Peucker. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, December 1973.

[8] Edward Fairburn. Sma2003ll ink study over a pocket map, 2014. Website, 2014. Accessed: 2015-07-11.

[9] Carlos A. Furuti. Map projectios: Interrupted maps. Website, 1996. Accessed: 2015-07-13, Copyright 1996, 1997.

[10] Ken Garland. *Mr Beck's underground map*. Capital Transport, Harrow Weald, Middlesex, 1994.

[11] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[12] GmbH Geofabrik. Map compare, website. Web site, 2015. Accessed: 2015-07-20.

[13] Alejo Hausner. Simulating decorative mosaics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 573–580, New York, NY, USA, 2001. ACM.

[14] Aaron Hertzmann. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, 2003.

[15] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 99–108, New York, NY, USA, 1996. ACM.

[16] Tobias Isenberg. Visual abstraction and stylisation of maps. *The Cartographic Journal*, 50(1):8–18, feb 2013.

[17] John Krygier and Denis Wood. *Making maps: a visual guide to map design for GIS*. Guilford Press, 2011.

[18] Viktoras Lukoševičius. Cartographic image of samogitia in the old maps of lithuania, poland and other neighboring countries (1700–1939). *Geodesy and Cartography*, 40(2):75–97, apr 2014.

[19] Xiaofeng Mi, Doug DeCarlo, and Matthew Stone. Abstraction of 2d shapes in terms of parts. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '09, pages 15–24, New York, NY, USA, 2009. ACM.

[20] Sebastian Münster. Carta d'italia. Website, 1550. Accessed: 2015-07-10.

[21] Addy Osmani. *Learning JavaScript design patterns*. O'Reilly Media, Sebastopol, CA, 2012.

[22] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244 – 256, 1972.

[23] Maxwell Roberts. *Underground maps unravelled : explorations in information design*. Maxwell J. Roberts,Distributed by the author Maxwell J. Roberts, Wivenhoe, Essex, UK Wivenhoe, Essex, UK, 2012.

[24] Sameboat. London underground overground dlr crossrail map. Website, 2014. Accessed: 2015-07-10, Licensed under CC BY-SA 4.0 via Wikimedia Commons.

[25] Steven Seow. Information theoretic models of HCI: A comparison of the hick-hyman law and fitts' law. *Human-Comp. Interaction*, 20(3):315–352, sep 2005.

[26] John P. Snyder. *Map Projections - A Working Manual*. United States Government Printing Office, 1987.

[27] Jared Tarbell. Substrate, website and simulation. Web site and simulation, 2003. Accessed: 2015-07-09.

[28] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51, jun 1993.

[29] Sheng Zhou and ChristopherB. Jones. Shape-aware line generalisation with weighted effective area. In *Developments in Spatial Data Handling*, pages 369–380. Springer Berlin Heidelberg, 2005.